Angular 2

Ángel Villalba Fdez-Paniagua

Table of Contents

1. Servicios e inyección de dependencias

Los servicios son una pieza fundamental de Angular, todo valor, función o característica que necesita la aplicación (constantes, lógica de negocio...) se encapsula dentro de los servicios.

Los servicios son simples clases que se usan para interactuar con la BBDD, acceder a la lógica de negocio entre distintos sitios de la aplicación, para la comunicación entre componentes y clases (por ejemplo emitir un evento y escucharlo en distintos sitios de la aplicación sin tener que construir cadenas de eventos o propiedades).

DRY (Don't Repeat Yourself). Si tienes funcionalidad repetida en distintos componentes se debería de centralizar en un servicio. Es más fácil de mantener.

Para crear un servicio se usa el comando:

```
$ ng g s miServicio
```

En el archivo mi-servicio.service.ts aparece `@Injectable()`. Es un decorator que indica que el servicio espera utilizar otros servicios. No es necesario ponerlo si el servicio no usa ningún otro servicio.

```
export class MiServicio {
  // ...
}
```

1.1. Inyección de dependencias

La **inyección de dependencias** es un mecanismo que proporciona nuevas instancias de una clase. La mayoría de las dependencias son servicios y Angular inyecta estos servicios en los componentes que los necesitan.

Se pueden inyectar las dependencias a través de los constructores de los componentes, las directivas o las clases. Para ello se necesita importar lo que queremos inyectar y pasarlo como argumento en el constructor.

```
import { MiServicio } from './mi-servicio.service';
// ...
constructor(miServicio: MiServicio) {}
// ...
```

1.2. Una instancia vs múltiples instancias

Hay que tener en cuenta el comportamiento que debe tener cada componente al inyectarle un servicio, es decir, se necesita que solo sea ese componente el que pueda acceder a los datos y

funciones que provee ese servicio o se necesita que a parte de ese componente haya otros que puedan acceder a los datos y funciones del servicio. Por ejemplo, al implementar una cesta de la compra para un e-commerce, se debería de poder añadir items desde distintos sitios de la aplicación, en este caso tiene sentido que todos los componentes, desde los que se puede añadir items a la cesta, tengan la misma instancia del servicio para que los items que añadas se incluyan en la cesta del usuario. Con el ejemplo anterior, si en lugar de usar una instancia para todos, usamos distintas instancias en cada uno de los componentes desde los que se pueden añadir los items, cada componente añadiria los items en cestas distintas, las cuales no estarían sincronizadas.

En el caso de querer varias instancias del servicio, hay que indicarlo en el array de providers de cada componente que lo vaya a usar.

Por otro lado, si se quiere usar una instancia para varios componentes, basta con añadir el servicio al array de providers de un componente superior que contenga los componentes que lo van a usar. Si se necesita usar en cualquier lugar de la aplicación, la mejor opción es añadirlo en el archivo `app.module.ts`.

```
import { MiServicio } from './mi-servicio.service';

@Component({
  //...
  providers: [MiServicio]
})
// ...
```

Ejemplo de un servicio de Log:

```
$ ng g s log
```

```
export class LogService {
  writeToLog(logMessage: string) {
    console.log(logMessage);
  }
}
```

Para usar este servicio en dos componentes distintos hay que inyectarlo en sus constructores.

```
import { Component } from '@angular/core';
import { LogService } from './log.service';

@Component({
  selector: 'app-cmp-a',
  template: `
    <div>
      <input type="text">
      <button (click)="onLog()">Log</button>
    </div>
  `,
  styles: [],
  providers: [LogService]
})

export class CmpAComponent {
  constructor(private logService: LogService) {}
}
```

Para hacer que escriba en la consola lo que se introduce en el input hay que añadir una referencia en el input para poder pasarle el valor como parámetro en la función onLog. Y hay que implementar la función onLog la cual usara el servicio para mostrar el texto en la consola.

```
<input type="text" #input>
<button (click)="onLog(input.value)">Log</button>
```

```
onLog(value: string) {
  this.logService.writeToLog(value);
}
```

Ahora hay que hacer exactamente lo mismo en el segundo componente, el CmpB:

```
import { Component } from '@angular/core';
import { LogService } from './log.service';

@Component({
  selector: 'app-cmp-b',
  template: `
    <div>
      <input type="text" #input>
      <button (click)="onLog(input.value)">Log</button>
    </div>
  `,
  styles: [],
  providers: [LogService]
})

export class CmpBComponent {
  constructor(private logService: LogService) {}

  onLog(value: string) {
    this.logService.writeToLog(value);
  }
}
```

Cada componente tiene el LogService en su respectivo providers, por lo que son instancias distintas, pero para el log es indiferente, no es necesario que estén sincronizados.

Para que los dos componentes usen la misma instancia de un servicio, este habría que añadirlo en el providers del componente padre (en este caso seria el `app.component`), o en el `app.module.ts`.

```
$ ng g s data
```

```
export class DataService {
  private data: string[] = [];
  constructor() { }

  addData(input: string) {
    this.data.push(input);
  }

  getData() {
    return this.data;
  }
}
```

Este servicio se añade en el providers del `app.component.ts` para que los componentes A y B usen la misma instancia. De esta forma, los datos estarán sincronizados.

```
<h1>Services & Dependency Injection</h1>
<app-cmp-a></app-cmp-a>
<app-cmp-b></app-cmp-b>
```

```
// ...
import { DataService } from './data.service';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  providers: [DataService]
})
// ...
```

Ahora se crean dos botones, uno para guardar datos en un array que se encuentra en el `data.service.ts` y que se mostrarán en ambos componentes al darle al otro botón. Para poder acceder a los datos y poder guardarlos es necesario importar el servicio e inyectarlo en los constructores:

```
import { Component } from '@angular/core';
import { LogService } from './log.service';
import { DataService } from './data.service';

@Component({
  selector: 'app-cmp-a',
  template: `
    <div>
      <input type="text" #input>
      <button (click)="onLog(input.value)">Log</button>
      <button (click)="onStore(input.value)">Store</button>
    </div>
    <hr>
    <div>
      <button (click)="onGet()">Refresh Storage</button>
      <h3>Storage</h3>
      <ul>
        <li *ngFor="let item of items">{{item}}</li>
      </ul>
    </div>
  `,
  styles: []
})
export class CmpAComponent {
  value = '';
  items: string[] = [];

  constructor(private logService: LogService, private dataService: DataService) {
  }

  onLog(value: string) {
    this.logService.writeToLog(value);
  }

  onStore(value: string) {
    this.dataService.addData(value);
  }

  onGet() {
    // El slice evita que se autorefresque cuando le damos a guardar.
    this.items = this.dataService.getData().slice(0);
  }
}
```

Con esto al insertar un nuevo item desde cualquier componente, al darle al botón de `Refresh Storage` de cualquier componente se puede ver que se ha añadido y que ambos componentes tienen visibilidad sobre los items que añade el componente contrario.

1.3. Inyectando servicios en otros servicios

Hasta ahora se han inyectado servicios en componentes, pero hay veces que se necesita insertar un servicio en otro servicio. Siguiendo con el ejemplo anterior, se quiere mostrar por la consola cada item nuevo que se añade en el array del DataService.

Hay que importar e inyectar el LogService en el DataService, y usarlo para mostrar información en la consola de los items que se van añadiendo.

```
import { LogService } from './log.service';

export class DataService {
  private data: string[] = [];
  constructor(private logService: LogService) { }

  addData(input: string) {
    this.data.push(input);
    this.logService.writeToLog('Saved item: ' + input);
  }

  getData() {
    return this.data;
  }
}
```

Esto no funciona, si se prueba se puede ver que da un error en la consola, y esto se debe a que no se le está indicando que a este servicio se le está inyectando un servicio externo. Ahora es el momento de añadir la parte del `@Injectable()`.

```
import { Injectable } from '@angular/core';
import { LogService } from './log.service';

@Injectable()
export class DataService {
  private data: string[] = [];
  constructor(private logService: LogService) { }

  addData(input: string) {
    this.data.push(input);
    this.logService.writeToLog('Saved item: ' + input);
  }

  getData() {
    return this.data;
  }
}
```

Ahora se necesita usar el LogService desde el nivel más alto de la aplicación para poder usarlo en

cualquier parte de la aplicación, por lo tanto se añade en el `app.module.ts`. No hay que quitar los import de los componentes porque se usan en los constructores. Y no hay que olvidar quitarlo de los providers de los componentes A y B.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
import { AppComponent } from './app.component';
import { CmpAComponent } from './service/cmp-a.component';
import { CmpBComponent } from './service/cmp-b.component';
import { LogService } from './service/log.service';

@NgModule({
  declarations: [
    AppComponent,
    CmpAComponent,
    CmpBComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [LogService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

1.4. Usando servicios para la interacción entre componentes

Los servicios también se pueden usar para que los componentes interactúen entre ellos sin necesidad de implementar cadenas de eventos a través de los componentes de la aplicación.

En este caso se va a añadir la funcionalidad para enviar un dato de un componente a otro y que aparezca automáticamente.

En el DataService hay que crear un EventEmitter para emitir el valor del input al otro componente cuando se le da al botón de enviar.

```
import { Injectable, EventEmitter } from '@angular/core';
import { LogService } from './log.service';

@Injectable()
export class DataService {
  pushedData = new EventEmitter<string>();
  private data: string[] = [];
  constructor(private logService: LogService) { }

  addData(input: string) {
    this.data.push(input);
    this.logService.writeToLog('Saved item: ' + input);
  }

  getData() {
    return this.data;
  }

  pushData(value: string) {
    this.pushedData.emit(value);
  }
}
```

Añadimos en el componente A un botón para enviar el valor del input. Para ello en la función que se ejecuta al pulsar el botón de enviar llamamos a la función que se ha implementado en el DataService y que se encarga de emitir el dato al componente B.

```
import { Component } from '@angular/core';
import { LogService } from './log.service';
import { DataService } from './data.service';

@Component({
  selector: 'app-cmp-a',
  template: `
    <div>
      <input type="text" #input>
      <button (click)="onLog(input.value)">Log</button>
      <button (click)="onStore(input.value)">Store</button>
      <button (click)="onSend(input.value)">Send</button>
    </div>
    <hr>
    <div>
      <button (click)="onGet()">Refresh Storage</button>
      <h3>Storage</h3>
      <ul>
        <li *ngFor="let item of items">{{item}}</li>
      </ul>
    </div>
  `,
  styles: []
})
export class CmpAComponent {
  value = '';
  items: string[] = [];

  constructor(private logService: LogService, private dataService: DataService) {
  }

  onLog(value: string) {
    this.logService.writeToLog(value);
  }

  onStore(value: string) {
    this.dataService.addData(value);
  }

  onGet() {
    // El slice evita que se autorefresque cuando le damos a guardar.
    this.items = this.dataService.getData().slice(0);
  }

  onSend(value: string) {
    this.dataService.pushData(value);
  }
}
```

Y en el componente B hay que subscribirse al evento que se emite en el DataService, es decir que

cuando se detecta el emit, se ejecutará la función de callback en la que se va a actualizar el valor de la variable value con el que llega como argumento en el callback.

```
import { Component, OnInit } from '@angular/core';
import { LogService } from './log.service';
import { DataService } from './data.service';

@Component({
  selector: 'app-cmp-b',
  template: `
    <div>
      <input type="text" #input>
      <button (click)="onLog(input.value)">Log</button>
      <button (click)="onStore(input.value)">Store</button>
    </div>
    <hr>
    <div>
      <button (click)="onGet()">Refresh Storage</button>
      <h3>Storage</h3>
      <ul>
        <li *ngFor="let item of items">{{item}}</li>
      </ul>
      <h3>Received Value</h3>
      <p>{{value}}</p>
    </div>
  `,
  styles: []
})
export class CmpBComponent implements OnInit {
  value = '';
  items: string[] = [];

  constructor(private logService: LogService, private dataService: DataService) {
  }

  onLog(value: string) {
    this.logService.writeToLog(value);
  }

  onStore(value: string) {
    this.dataService.addData(value);
  }

  onGet() {
    // El slice evita que se autorefresque cuando le damos a guardar.
    this.items = this.dataService.getData().slice(0);
  }

  ngOnInit() {
    this.dataService.pushedData.subscribe(
```

```
      data => this.value = data
    );
  }
}
```