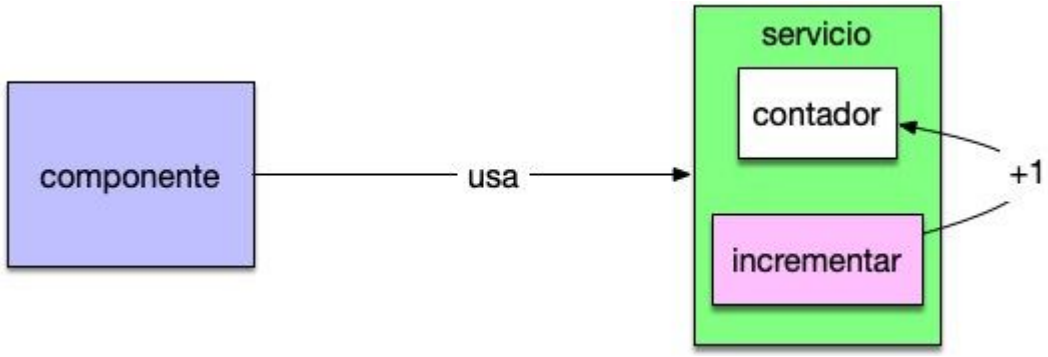
El concepto de Angular Services es uno de los más habituales cuando trabajamos con Angular. Los servicios nos valen para agrupar responsabilidades reutilizables . Uno de los casos más típicos es la llamada a servicios REST . Los servicios cuando trabajabamos con Angular 1.x eran Singletons. Algo que es muy habitual ya que normalmente no almacenan ningún estado , por ejemplo en Spring Framework lo son . ¿ Funcionan de la misma forma en Angular 5/ 6? .Vamos a verlos, recordemos que dar de alta un servicio es relativamente sencillo.

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

@Injectable()
export class Servicio1Service {

  public contador:number=0;

  constructor() { }

  incrementar() {

    this.contador++;
  }

}
```

En este caso definimos un servicio que contiene un simple contador que vamos incrementando .



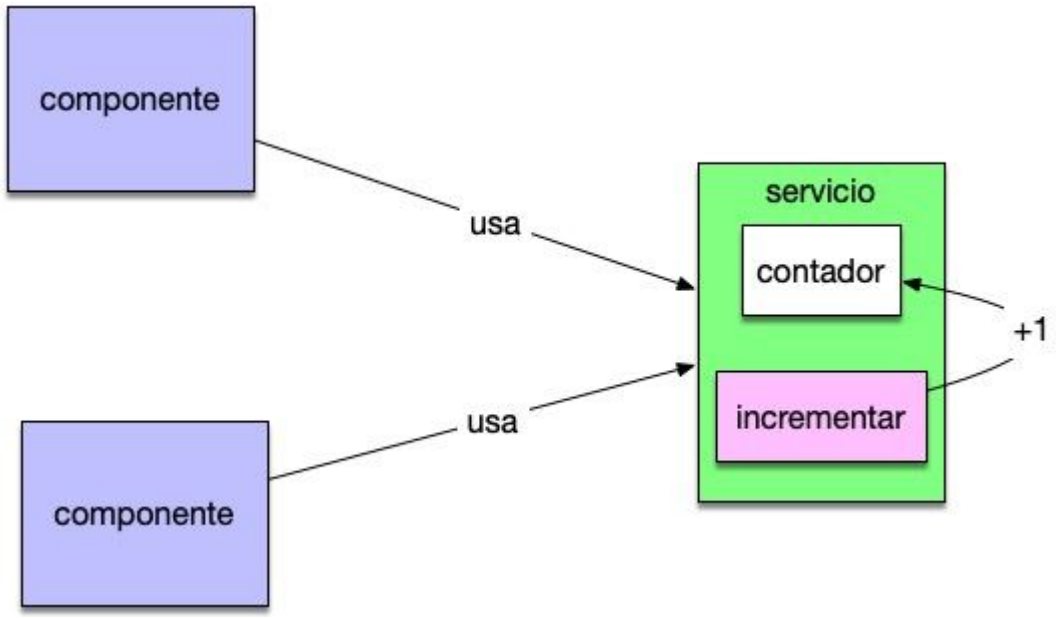
Una vez creado el servicio tenemos que modificar el módulo principal de nuestra aplicación para incluirle como provider y que el framework interno de inyección de dependencia de Angular sea capaz de gestionarle.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { Componente1Component } from
'./componente1/componente1.component';
import { Componente2Component } from
'./componente2/componente2.component';
import { Servicio1Service } from './servicio1.service';

@NgModule({
  declarations: [
    AppComponent,
    Componente1Component,
    Componente2Component
  ],
  imports: [
    BrowserModule
  ],
```

```
  providers: [Servicio1Service],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Angular Services y Componentes

Una vez hecho esto solo nos queda crear un par de componentes que usen este servicio y ver cómo el servicio se comporta como un singleton.



Veamos el código:

```
import { Component, OnInit } from '@angular/core';
import { Servicio1Service } from '../servicio1.service';

@Component({
  selector: 'app-componente1',
```

```
  templateUrl: './componente1.component.html',
  styleUrls: ['./componente1.component.css']
})
export class Componente1Component implements OnInit {

  constructor(private servicio: Servicio1Service) {

  }
  ngOnInit() {
  }

  pulsar() {

    this.servicio.incrementar();

  }
}
```

```
import { Component, OnInit } from '@angular/core';
import { Servicio1Service } from '../servicio1.service';

@Component({
  selector: 'app-componente2',
  templateUrl: './componente2.component.html',
  styleUrls: ['./componente2.component.css'],
  providers:[Servicio1Service]
})
export class Componente2Component implements OnInit {
```

```
  constructor(private servicio:Servicio1Service) { }

  ngOnInit() {
  }

  pulsar() {

    this.servicio.incrementar();

  }

}
```

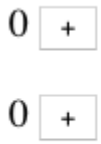
Es momento de ver el contenido html de cada uno de los componentes:

```
{{servicio.contador}}
&amp;lt;input type="button" value="+" (click)="pulsar()"/&amp;gt;
```

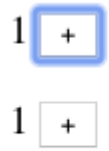
Ambos componentes comparten el código si cargamos en la página principal los dos :

```
&amp;lt;div&amp;gt;
  &amp;lt;app-componente1&amp;gt;&amp;lt;/app-componente1&amp;gt;
&amp;lt;app-componente2&amp;gt;&amp;lt;/app-componente2&amp;gt;
&amp;lt;/div&amp;gt;
```

Veremos un interface de usuario como el siguiente:



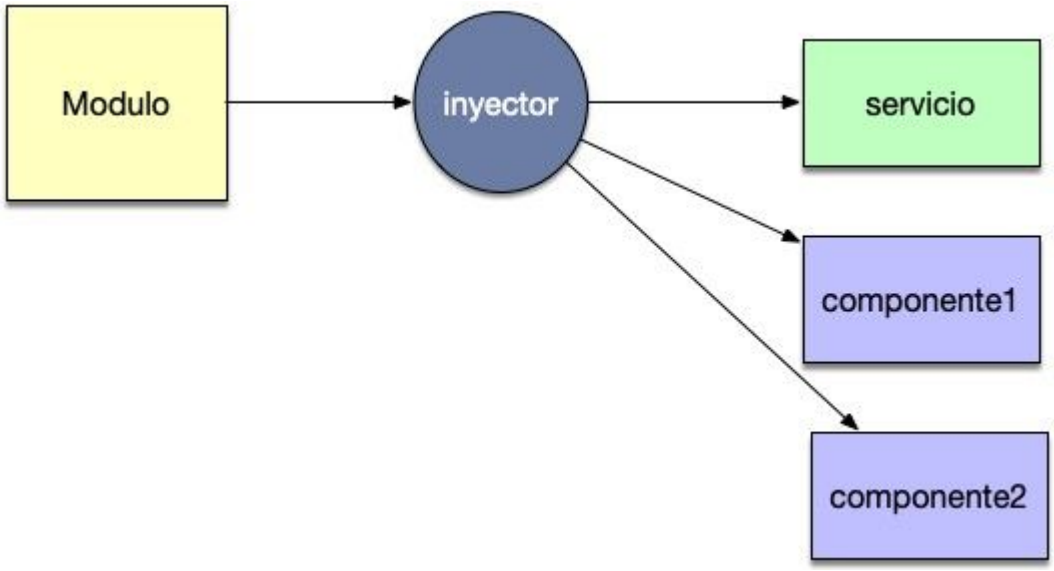
Ambos componentes están asociados el mismo servicio . Por lo tanto cada vez que pulsemos en cualquiera de los botones estaremos incrementando el contador del servicio que debería ser un singleton y ambos números se incrementarán.



Hasta aquí todo es correcto y funciona como esperábamos aquellos que hemos trabajado con Angular 1.x clásico.

Angular Services e Inyectores

Sin embargo Angular es un framework moderno y soporta un sistema de inyección de dependencias más flexible. Es lo que se llama un Inyector jerárquico. Estos son capaces de generar un objeto diferente dependiendo del nivel de jerarquía en el que nos encontremos. Con nuestro ejemplo sencillo las cosas funcionaran de la siguiente forma:

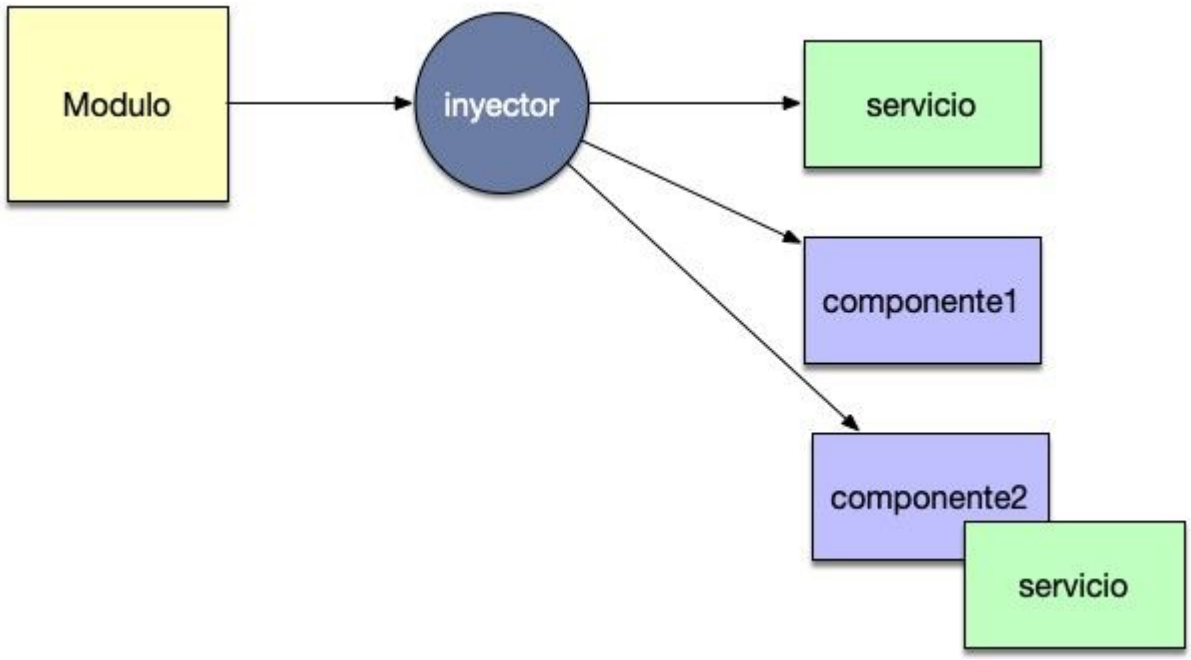


Sin embargo tenemos la opción de modificar como el inyector genera el servicio y decidir que ese servicio esta intimamente ligado a un componente:

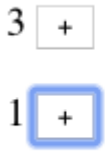
```
import { Component, OnInit } from '@angular/core';
import { Servicio1Service } from '../servicio1.service';

@Component({
  selector: 'app-componente2',
  templateUrl: './componente2.component.html',
  styleUrls: ['./componente2.component.css'],
  providers:[Servicio1Service]
})
export class Componente2Component implements OnInit {

  constructor(private servicio:Servicio1Service) { }

  ngOnInit() {
  }
```

```
  pulsar() {
    this.servicio.incrementar();
  }
}
```

Cómo se puede observar hemos usado la anotación de @Component y la propiedad de providers para inyectar un servicio. En este caso el inyector se comportará de forma diferente y generará otra instancia del servicio a nivel del propio componente.



Si ejecutamos ahora el código en un navegador veremos que cada componente incrementa su contador de forma independiente.



El primero por decirlo de alguna forma con su servicio global y el otro con una instancia especifica para él.Así pues aunque por defecto los servicios en Angular son Singletons no están obligados a serlo y todo depende del sistema de inyección de dependencias jerárquico que soporta.

Otros artículos relacionados

1. Angular ngIf, opciones y componentes
2. Angular ngFor la directiva y sus opciones
3. Angular Modules y el uso de servicios
4. TypeScript interface utilizando Angular DTOs

Links Externos

1. Angular