



Angular services

GFT INTERNAL TRAINING

INNOVATE. TRANSFORM. DELIVER.

Agenda

- 1. services**
- 2. dependency injection**
- 3. service example**
- 4. services with promises and observables**
- 5. services benefits**
- 6. injecting services**
- 7. explicit injection**
- 8. tree-shakeable services**

services

services

- **Service:** any value, function or feature that an app needs
- anything can be a service
- typically a class with a narrow, well-defined purpose as:
 - logging service
 - data service
 - message bus
 - tax calculator
 - application configuration



<https://angular.io/guide/architecture#services>

services in angular



- there is nothing specifically Angular about services
- Angular has no definition of a service
- there is no service base class, and no place to register a service
- yet services are fundamental to any Angular application
- components are usually big consumers of services

<https://angular.io/guide/architecture#services>

dependency injection

Dependency injection (DI)

- is an important application design pattern whereby one object supplies the dependencies of another object, where a dependency is an object that can be used. Finally, the dependency will be passed to a dependent object (the caller/client, the object that would need it)
- its main purpose is to decouple the both parts; in other words, that a change in one part would not imply a change in the other part

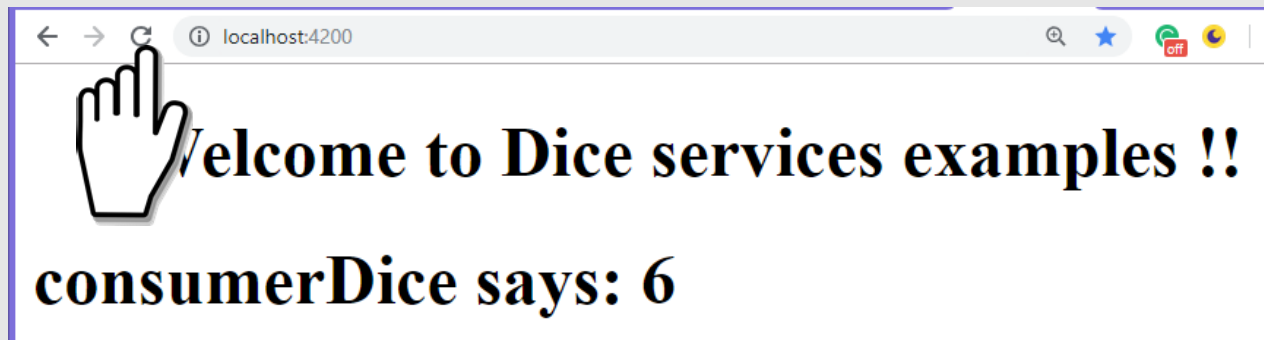
<https://angular.io/guide/dependency-injection>

Angular DI

- Angular has its own dependency injection framework
- if we have a component that depends on a service, we have not create the service ourselves; instead, we will request it in the constructor, and Angular DI engine will provide it for us. Doing that, we can depend on interfaces rather than concrete types

services example

let`s make a dice



1) create service class

dice.service.ts

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

```
@Injectable()
```



lets Angular know that a *class* can be used with the dependency injector

```
export class DiceService {
```

```
  constructor() { }
```

```
  throwDice(){
```

```
    return Math.floor(6*Math.random()+1);
```

```
  }
```

```
}
```

2) add to providers

app.module.ts

```
@NgModule({  
  declarations: [  
    AppComponent,  
    ConsumerDiceComponent  
  ],  
  imports: [  
    BrowserModule  
  ],  
  providers: [ DiceService ],  
  bootstrap: [AppComponent]  
})  
  
export class AppModule { }
```

this approach will work in any Angular version, but since Angular 6 the best practice is to specify the provider in the service class

configure Angular DI with the services it will instantiate on demand

3) use it in component

```
@Component({
  selector: 'app-consumer-dice',
  template: '<h1>consumerDice says: {{throwResult}}</h1>'
})
export class ConsumerDiceComponent implements OnInit {

  throwResult: number;

  constructor( private diceSrv: DiceService ) { }

  ngOnInit() {
    this.throwResult = this.diceSrv.throwDice();
  }
}
```

angular DI will create a property with the service

4) refactor

dice.service.ts

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

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

  constructor() { }

  throwDice(){
    return Math.floor(6*Math.random()+1);
  }
}
```

also delete DiceService reference
from app.module.ts

learn by doing

- create a service with two methods:
 - dameFrase() -> will return a random text in Spanish
 - getPhrase() -> will return a random text in English
- create a component that uses both methods

Muestra una frase del día

A buenas horas mangas verdes

Show a daily phrase

When the going gets tough, the tough get going

services with promises and observables

promises and observables

- often a service involves asynchronous programming such as http requests to a remote server that could also fail
- in that case it's a good practice to return a promise or an observable

learn by doing

- create a component that will use the service showed in the next slide

Input your licence plate number:

Show me if i`m allowed to drive in the city center

Sorry NOT allowed to drive in the city center

Input your licence plate number:

Show me if i`m allowed to drive in the city center

Congratulations!!! allowed to drive in the city center


```
export class LicencePlateAuthorizationService {  
  //these could be configured  
  oddAllowed: boolean = true;  
  evenAllowed: boolean = false;  
  isAuthorized: boolean;  
  
  authorize(numberPlate): Promise<string> {  
    if (numberPlate%2==0){  
      this.isAuthorized = this.evenAllowed;  
    } else {  
      this.isAuthorized = this.oddAllowed;  
    }  
    if (this.isAuthorized) {  
      return Promise.resolve("allowed to drive in the city center");  
    } else {  
      return Promise.reject("NOT allowed to drive in the city center");  
    }  
  };  
}
```

services benefits


good practices

Organize logic into services has the following benefits:

- code organization
- legibility
- reusability
- testability

injecting services

where a service can be provided



In the
AppModule

At module
level

At
component
level

- if multiple imported modules define the same provider, the last module wins
- dependencies are singletons within the scope of an injector

since Angular 9

New options for 'providedIn'

When you create an `@Injectable` service in Angular, you must choose where it should be added to the injector. In addition to the previous `root` and module options, you have two additional options.

- `platform` — Specifying `providedIn: 'platform'` makes the service available in a special singleton platform injector that is shared by all applications on the page.
- `any` — Provides a unique instance in every module (including lazy modules) that injects the token.

<https://blog.angular.io/version-9-of-angular-now-available-project-ivy-has-arrived-23c97b63cfa3#1e11>

service at component level

```
@Component({
  selector: 'app-consumer-dice-component-level',
  template: '<h1>consumerDice component level says: {{throwResult}}</h1>',
  providers: [ DiceService ]
})
export class ConsumerDiceComponentLevelComponent implements OnInit {

  throwResult: number;

  constructor( private diceSrv: DiceService ) { }

  ngOnInit() {
    this.throwResult = this.diceSrv.throwDice();
  }
}
```

explicit injection

using injector explicitly

```
@Component({
  selector: 'app-consumer-dice-injector',
  template: '<h1>consumerDice with explicit injector says: {{throwResult}}</h1>'
})
export class ConsumerDiceInjectorComponent implements OnInit {

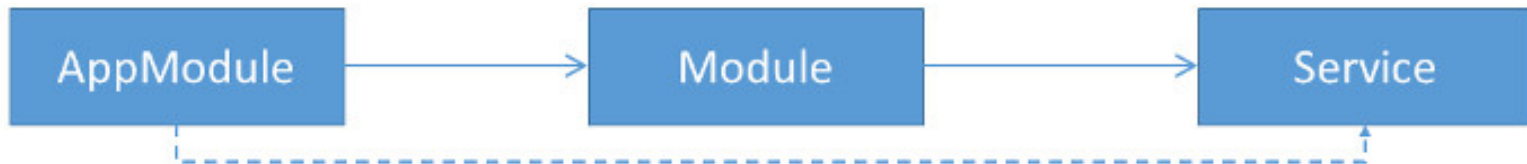
  throwResult: number;
  diceSrv: DiceService;

  constructor( private injector: Injector) { }

  ngOnInit() {
    this.diceSrv = this.injector.get(DiceService);
    this.throwResult = this.diceSrv.throwDice();
  }
}
```

tree-shakeable services

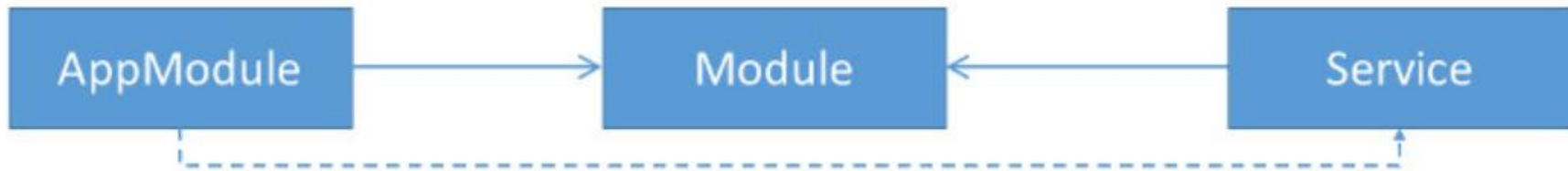
Classic service-provider (since Angular 2)



```
@NgModule({
  declarations: [ AppComponent, ConsumerDiceComponent ],
  imports: [ BrowserModule, FormsModule ],
  providers: [ DiceService, LicencePlateAuthorizationService ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

**This services are NOT
treeshakable**

Treeshakable providers (since Angular 6)



It looks like this in code:

```
@Injectable({ providedIn: 'root'})  
export class SimpleFlightCancellingService { [...] }
```

<https://jaxenter.com/new-angular6-143995.html>

Shaping the future of digital business

GFT Internal Technical Training

Eduardo García Ibaseta

eduardo.garcia-ibaseta@gft.com
+34 935 639476

GFT IT Consulting, S.L.

Av. Alcalde Barnils, 69-71

08174 Sant Cugat del Vallès (BARCELONA)