# آموزش Angular6ـ آموزش Service ها در Angular6

تا به اینجای کار، HeroesComponent موجود در Tour of Heroes اطلاعات نادرستی دریافت و نمایش می دهد. بعد از انجام Refactoring در این آموزش، HeroesComponent بر پشتیبانی از View متمرکز خواهد شد. همچنین انجام آزمون واحد (unit-test) با استفاده از سرویس ساختگی (mock service) آسان تر خواهد شد.

## چرا استفاده از service ها مهم است؟

کامپوننت ها نباید به صورت مستقیم اطلاعات را دریافت یا ذخیره کنند، همچنین بدون شک ارائه اطلاعات نادرست به صورت دانسته نباید توسط آن ها صورت بگیرد. این کامپوننت ها باید بر روی ارائه اطلاعات تمرکز داشته باشند و دسترسی به داده ها را به یک service محول کنند.

در این آموزش شما می توانید یک HeroService ایجاد کرده به گونه ای که در تمامی کلاس های نرم افزار بتوانید برای گرفتن hero ها از آن استفاده کنید. برای ایجاد این service به جای استفاده از new ، بر angular dependency injection متکی خواهید بود تا بتوانید آن را در HeroesComponent constructor وارد کنید.

Service ها روشی عالی برای به اشتراک گذاری اطلاعات بین کلاس هایی هستند که همدیگر را نمی شناسند. در اینجا شما اقدام به ایجاد MessageService می کنید و آن در دو مکان وارد می کنید.

1. در HeroService که برای ارسال پیام از این service استفاده می کند.
2. در MessagesComponent که پیام ارسالی را نمایش می دهد.

## آموزش ایجاد HeroService

با استفاده از Angular CLI ، سرویسی به نام hero ایجاد کنید.

```
ng generate service hero
```

این فرمان skeleton HeroService را در src/app/hero.service.ts تولید می کند. این کلاس مشابه کد زیر می تواند باشد.

src/app/hero.service.ts (new service)

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

@Injectable({

 providedIn: 'root',
```

```
})

export class HeroService {

 constructor() { }

}
```

## آموزش @Injectable() services

توجه داشته باشید که این سرویس جدید، نشان Angular Injectable را import می کند و این کلاس را با دکوراتور @Injectable() نمایش می دهد. در این صورت این کلاس به کلاسی تبدیل خواهد شد که در dependency injection system (سیستم تزریق وابستگی) مشارکت می کند. کلاس HeroService، سرویسی تزریق پذیر (Injectable service) را فراهم می کند که این سیستم وابستگی های تزریق شده (injected dependencies) مخصوص به خود را دارد. البته تا به این لحظه این کلاس هنوز عاری از وابستگی است اما به زودی از این حالت در خواهد آمد.

دکوراتور @Injectable() مانند دکوراتور @component() که مربوط به کلاس کامپوننت تان بود، برای این سرویس metadata object را می پذیرد.

## آموزش گرفتن hero data

HeroService می تواند ار همه جا داده دریافت کند مانند: web service,local storage یا mock data source .

حذف دسترسی اطلاعاتی از کامپوننت ها به این معنی است که شما بدون ایجاد کوچک ترین تغییری در کامپوننت ها می توانید در هر زمانی نظرتان را درباره پیاده سازی (implementation) تغییر دهید. این کامپوننت ها نمی دانند که این سرویس چگونه کار می کند. در این آموزش پیاده سازی تا زمان تحویل mock heroes ادامه خواهد یافت.

HEROES و Hero را import کنید.

```
import { Hero } from './hero';

import { HEROES } from './mock-heroes';
```

برای برگشت دادن mock heroes ، getHeroes را اضافه کنید.

```
getHeroes(): Hero[] {

return HEROES;

}
```

## آماده و تامین کردن HeroService

پیش از آنکه Angular بتواند HeroService را در HeroesComponent تزریق کند، شما باید آن را در دسترس dependency injection system قرار دهید (همانطور که در ادامه انجام خواهید داد) . این کار را می توانید با ثبت کردن یک provider انجام دهید. Provider چیزی هست که می تواند سرویسی را تحویل داده یا آن را ایجاد نماید. برای حالتی که ما داریم بررسی می کنیم وظیفه آن آماده کردن کلاس HeroService برای تامین service است.

حالا شما باید مطمئن شوید که HeroService به عنوان provider این سرویس ثبت شده است. کار ثبت را باید از طریق یک injector انجام دهید که این injector شیئی است که مسئول انتخاب و تزریق provider در مکان مورد نیاز است.

به صورت پیشفرض، فرمان ng generate service موجود در Angular command توسط root injector ، با لحاظ کردن فرادادە (متا دیتا)ی provider در دکوراتور Injectable@ ، در سرویس تان provider ثبت می کند.

اگر درست پیش از تعریف کلاس HeroService به عبارت Injectable()@ نگاه کنید، خواهید دید که مقدار Provider برابر با root است:

```
@Injectable({
providedIn: 'root',
})
```

زمانی که در سطح root سرویس خود را تامین کردید، Angular تک نمونه ای به اشتراک گذاشته شده از HeroService را ایجاد کرده و آن را در هر کلاسی که نیاز باشد تزریق می کند. ثبت provider در فرادادە injectable@ این امکان را نیز برای Angular فراهم می کند تا در صورتی که مشخص شود که سرویس دیگر به کار نمی آید، آن را برای بهینه کردن نرم افزار حذف کند.

اگر لازم باشد می توانید provider ها را در سطوح مختلفی مانند: HeroesComponent,AppComponent یا AppModule ثبت کنید. برای مثال، می توانید برای تامین و آماده سازی سرویس در سطح ماژول ( module level)، به صورت خودکار با افزودن module=app– این کار را به CLI محول کنید.

ng generate service hero --module=app

برای کسب اطلاعات بیشتر در رابطه با provider ها و injector ها به https://angular.io/guide/dependency-injection مراجعه کنید.

حالا HeroService اماده وارد شدن به HeroesComponent است.

این کد، نمونه ای اصلاح نشده است که به شما این امکان را می دهد تا از HeroService استفاده کنید و آن را تامین کنید. در این نقطه این کد با بازبینی نهایی کد HeroService متفاوت است.

به روز کردن HeroesComponent

فایل کلاس HeroesComponent را باز کنید.

HEROES import را پاک کنید چون دیگر به آن نیازی ندارید و به جای آن HeroService را import کنید.

src/app/heroes/heroes.component.ts (import HeroService)

```
import { HeroService } from '../hero.service';
```

به جای تعریف heroes، یک اعلان ساده را بگذارید.

```
heroes: Hero[];
```

## آموزش تزریق HeroService

پارامتر خصوصی heroService از نوع HeroService را به Constructor اضافه کنید.

```
constructor(private heroService: HeroService) { }
```

این پارامتر به صورت همزمان یک heroService خصوصی را تعریف کرده و آن را به عنوان یک HeroService injection site می شناسد.

زمانی که Angular یک HeroesComponent را ایجاد می کند، Dependency injection system پارامتر heroService را بر روی تک نمونه ای از HeroService تنظیم می کند.

## آموزش افزودن getHeroes()

تابعی را ایجاد کنید که با کمک آن بتوانید hero ها را از سرویس بازیابی کنید.

```
getHeroes(): void {
this.heroes = this.heroService.getHeroes();
}
```

## آموزش فراخوانی ngOnInit

با وجود اینکه می توانید getHeroes() را فراخوانی کنید، اما این کار بهترین راه نیست. Constructor را برای آغازسازی های ساده (simple initialization) مانند ارتباط دادن پارامترهای constructor به property ها نگه دارید.

Constructor نباید کاری انجام دهد و فراتر از آن چیزی که مسلم است نباید تابعی را فرا بخواند که HTTP request ها را در یک remote server به حالتی در آورد که یک سرویس واقعی داده (real data service) آن کار را می کند.

در عوض، getHeroes() را داخل ngOnInit lifecycle hook فراخوانی کرده و اجازه دهید Angular در زمانی مناسب و پس از ساخت نمونه ای از HeroesComponent ، ngOnInit را فراخوانی کند.

```
ngOnInit() {
this.getHeroes();
}
```

## از برنامه run بگیرید

بعد از اینکه مرورگر را refresh می کنید، نرم افزار تان باید مانند قبل اجرا شود و لیستی از hero ها را نشان داده و بعد از اینکه بر روی نام hero ها کلیک کنید جزییات آن ها را نیز باید نشان دهد.

## آموزش Observable data (داده قابل مشاهده)

متد HeroService.getHeroes() دارای امضای همزمان (Synchronous signature) است، به این معنی که HeroService می تواند به صورت همزمان hero ها را دریافت کند. HeroesComponent نتیجه getHeroes() در صورتی استفاده می کند که بتوان hero ها را به صورت همزمان دریافت کرد.

```
this.heroes = this.heroService.getHeroes();
```

کد بالا در نرم افزارهای واقعی جواب نمی دهد. فعلا به این دلیل که این سرویس hero های ساختگی را برمیگرداند، شما با این کد سر و کار دارید. اما به زودی این نرم افزار hero ها را از یک remote server به صورت همزمان می گیرد.

HeroService باید منتظر پاسخ سرور بماند به همین دلیل getHeroes() بلافاصله با hero data برنخواهد گشت و مرورگر تا زمانی که سرویس در حال انتظار است مسدود نخواهد شد.

HeroService.getHeroes() باید دارای نوعی از امضای همزمان باشد. این دستور می تواند بازگشت داشته باشد که این بازگشت می تواند شامل Promise یا Observable شود.

در این آموزش، HeroService.getHeroes() تا حدی observable را بازگشت خواهد داد زیرا در هایت برای دریافت hero ها از Angular استفاده خواهد کرد. همچنین HttpClient.get() نیز observable را برگشت می دهد.

## آموزش Observable HeroService

Observable یکی از کلاس های اصلی کتابخانه RxJS می باشد. در آموزش های آینده مربوط به HTTP، شما خواهید آموخت که متدهای HTTPClient مربوط به Angular ، Observable های RxJS را برگشت می دهند. در این آموزش شما دریافت داده از سرور را توسط تابع RxJS of() خواهید آموخت.

فایل HeroService را باز کرده و از RxJS ، سمبل های Observable و of را import کنید.

src/app/hero.service.ts (Observable imports)

```
import { Observable, of } from 'rxjs';
```

به جای متد getHeroes از کد زیر استفاده کنید:

```
getHeroes(): Observable<Hero[]>
{ return of(HEROES);
}
```

Of(HEROES) یک Observable<Hero[]> را بازگشت می دهد که تنها یک مقدار از خود خارج میکند (آرایه mock heroes) .

در آموزش HTTP می توانید HttpClient.get<Hero[]>() را فراخوانی کنید. برگشت این دستور Observable<Hero[]> بوده که آرایه hero ها از بدنه HTTP response را به صورت تنها یک مقدار از خود خارج می کند.

## آموزش استفاده از Subscribe در HeroesComponent

پیش از این متد HeroService.getHeroes ، Hero[] را برگشت می داد. در حال حاضر ، Observable<Hero[]> جای این مقدار برگشتی را گرفته است. در HeroesComponent چاره ای جز کنار آمدن با این تفاوت ندارید. متد getHeroes را پیدا کرده و آن را جایگزین کد زیر بکنید ( جهت مقایسه کد نسخه قبلی نیز زیر کد جدید نشان داده شده است) .

heroes.component.ts (Observable)

```
getHeroes(): void {
this.heroService.getHeroes()
.subscribe(heroes => this.heroes = heroes);
}
```

heroes.component.ts (اصلی)

```
getHeroes(): void { this.heroes = this.heroService.getHeroes();
}
```

تفاوت اصلی در Observable.subscribe() مشاهده می شود.

نسخه قبلی آرایه ای از hero ها را به heroes property مربوط به کامپوننت اختصاص می داد. این عمل اختصاص دادن به صورت همزمان رخ می دهد به این صورت که سرور می تواند بلافاصله hero ها را برگشت دهد یا اینکه مرورگر می تواند تا زمانی که منتظر دریافت پاسخ از سور هست، رابط کاربری را فریز کند.

این عملیات در حالتی که HeroService به صورت واقعی به یک remote server ، request ارسال کند جواب نمی دهد.

نسخه جدید برای بیرون دادن آرایه ای از hero ها منتظر observable می ماند که این عمل می تواند همین لحظه یا چندین دقیقه پس از الان رخ دهد. پس از این عملیات ، subscribe آرایه بیرون داده شده را به callback رد می کند و در نهایت heroes property کامپوننت تنظیم می شود. این عملکرد همزمان زمانی جواب می دهد که HeroService از سرور درخواست hero کند.

## آموزش نمایش دادن پیام ها

پس از اتمام این بخش شما قادر به انجام دادن کارهای زیر خواهید بود:

- messageComponent را اضافه کنید و با کمک آن پیام های نرم افزار را در پایین صفحه نمایش دهید.
- برای ارسال پیام هایی که بناست نمایش داده شوند، MessageService ، app-wide و تزریق پذیر (injectable) را ایجاد کنید.

- MessageService را در HeroService تزریق کنید.
- زمان که به صورت موفقیت آمیز HeroService می تواند Hero ها را دریافت کند، پیامی نمایش داده شود.

## آموزش ایجاد MessagesComponent

با استفاده از CLI ، MessageComponent را ایجاد کنید.

```
ng generate component messages
```

CLI فایل های کامپوننت را در پوشه src/app/messages ایجاد می کند و MessagesComponent را در AppModule اعلان می کند.

برای نمایش MessagesComponent تولید شده، قالب AppComponent را تغییر دهید :

/src/app/app.component.html

```
<h1>{{title}}</h1>

<app-heroes></app-heroes>

<app-messages></app-messages>
```

شما می توانید پاراگراف پیشفرض را از MessagesComponent در پایین صفحه ببینید.

## آموزش ایجاد MessageService

MessageService را باز کنید و نوشته های آن را مطابق با کد زیر جایگزین کنید:

/src/app/message.service.ts

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

@Injectable({
  providedIn: 'root',
})
export class MessageService {
  messages: string[] = [];

```

```
9.    add(message: string) {
10.    this.messages.push(message);
11.   }
12.
13.   clear() {
14.    this.messages = [];
15.   }
16. }
```

این سرویس Cache مربوط به Messages و دو متد را نشان می دهد : 1ـ add() کردن یک پیام به cache و clear() کردن cache.

## حالا آن را در HeroService تزریق کنید.

HeroService را مجددا باز کنید و MessageService را import کنید.

/src/app/hero.service.ts (import MessageService)

```
import { MessageService } from './message.service';
```

با استفاده از پارامتری که ویژگی (property) خصوصی messageService را اعلان کند، constructor را تغییر دهید. Angular زمانی که این ویژگی، HeroService را ایجاد می کند، تک شیء MessageService را داخل آن می گذارد.

```
constructor(private messageService: MessageService) { }
```

عبارت بالا، روش کار مرسوم سرویس در سرویس است به این صورت که شما Message Service را در HeroService تزریق می کنید که خود آن در HeroesComponent تزریق می شود.

## نمایش دادن پیام حاصل از HeroService

MessagesComponent همه پیام ها اعم از پیام هایی که HeroService حین دریافت Hero ها ارسال می کند را باید نمایش دهد. MessagesComponent را باز کرده و MessageService را import کنید.

/src/app/messages/messages.component.ts (import MessageService)

```
import { MessageService } from '../message.service';
```

Constructor را با استفاده از پارامتری که ویژگی عمومی MessageService را اعلان می کند، اصلاح کنید. پس از آن که این ویژگی MessagesComponent را ایجاد می کند، Angular تک شیء MessageService را در آن تزریق می کند.

```
constructor(public messageService: MessageService) {}
```

ویژگی MessageService باید عمومی باشد زیرا همانطور که در زیر خواهید دید در قالب باید قید پذیر باشد (قابلیت bind شدن را داشته باشد) .

Angular تنها به ویژگی های کامپوننت های عمومی می تواند چسبانده شود.

## آموزش چسباندن Angular به MessageService

قالب MessagesComponent تولید شده توسط CLI را مانند زیر اصلاح کنید.

src/app/messages/messages.component.html

```
<div *ngIf="messageService.messages.length">


 <h2>Messages</h2>

 <button class="clear"

    (click)="messageService.clear()">clear</button>

 <div *ngFor='let message of messageService.messages'> {{message}} </div>


</div>
```

این قالب به صورت مستقیم به MessageService کامپوننت چسبانده می شود.

- اگر پیامی برای نشان دادن وجود داشته باشد، nglf* تنها محدوده پیام ها را نشان می دهد.
- ngFor* بیانگر لیست پیام هایی است که در المان های تکرار شده  قرار دارند.
- Angular event binding ، رویداد کلیک دکمه را به MessageService.clear() می چسباند.

همانطور که در یکی از tab های "بازبینی نهایی کد" می توانید مشاهده کنید، اگر به messages.component.css استایل خصوصی CSS را بیفزایید، ظاهر پیام ها زیباتر خواهد شد.

مرورگر refresh شده و صفحه لیستی از hero ها را نمایش می دهد. به پایین صفحه رفته تا بتوانید در بخش پیام ها، پیام حاصل از HeroService را ببینید. بر روی دکمه clear کلیک کنید تا بخش پیام ها حذف شود.

## بازبینی نهایی کد

در ادامه فایل های بحث شده در این صفحه را می توانید مشاهده کنید و نرم افزار شما باید مشابه این کد باشد:

src/app/hero.service.ts

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

import { Observable, of } from 'rxjs';

import { Hero } from './hero';
import { HEROES } from './mock-heroes';
import { MessageService } from './message.service';

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

  constructor(private messageService: MessageService) { }

  getHeroes(): Observable<Hero[]> {
    // TODO: send the message _after_ fetching the heroes
    this.messageService.add('HeroService: fetched heroes');
    return of(HEROES);
  }
}
```

src/app/message.service.ts

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

```

```
@Injectable({
providedIn: 'root',
})
export class MessageService {
messages: string[] = [];

add(message: string) {
this.messages.push(message);
}

clear() {
this.messages = [];
}
}
```

src/app/heroes/heroes.component.ts

```
import { Component, OnInit } from '@angular/core';

import { Hero } from '../hero';
import { HeroService } from '../hero.service';

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

selectedHero: Hero;

heroes: Hero[];

constructor(private heroService: HeroService) { }
```

```
18.
19. ngOnInit() {
20. this.getHeroes();
21. }
22.
23. onSelect(hero: Hero): void {
24. this.selectedHero = hero;
25. }
26.
27. getHeroes(): void {
28. this.heroService.getHeroes()
29. .subscribe(heroes => this.heroes = heroes);
30. }
31. }
```

src/app/messages/messages.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { MessageService } from '../message.service';
3.
4. @Component({
5. selector: 'app-messages',
6. templateUrl: './messages.component.html',
7. styleUrls: ['./messages.component.css']
8. })
9. export class MessagesComponent implements OnInit {
10.
11. constructor(public messageService: MessageService) {}
12.
13. ngOnInit() {
14. }
15.
16. }
```

src/app/messages/messages.component.html

```
<div *ngIf="messageService.messages.length">
 <h2>Messages</h2>
 <button class="clear"
(click)="messageService.clear()">clear</button>
<div *ngFor='let message of messageService.messages'> {{message}} </div>
 </div>
```

src/app/messages/messages.component.css

```
/* MessagesComponent's private CSS styles */
h2 {
color: red;
font-family: Arial, Helvetica, sans-serif;
font-weight: lighter;
}
body {
margin: 2em;
}
body, input[text], button {
color: crimson;
font-family: Cambria, Georgia;
}

button.clear {
font-family: Arial;
background-color: #eee;
border: none;
padding: 5px 10px;
border-radius: 4px;
cursor: pointer;
```

```
cursor: hand;
}
button:hover {
background-color: #cfd8dc;
}
button:disabled {
background-color: #eee;
color: #aaa;
cursor: auto;
}
button.clear {
color: #888;
margin-bottom: 12px;
}
```

src/app/app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { HeroesComponent } from './heroes/heroes.component';
import { HeroDetailComponent } from './hero-detail/hero-detail.component';
import { MessagesComponent } from './messages/messages.component';

@NgModule({
declarations: [
AppComponent,
HeroesComponent,
HeroDetailComponent,
MessagesComponent
],
imports: [
BrowserModule,
FormsModule
],
```

```
20. providers: [
21. // no need to place any providers due to the `providedIn` flag...
22. ],
23. bootstrap: [ AppComponent ]
24. })
25. export class AppModule { }
```

src/app/app.component.html

```
<h1>{{title}}</h1>
<app-heroes></app-heroes>
<app-messages></app-messages>
```

## خلاصه

- دسترسی داده ها را به کلاس HeroService ریفاکتور کردید.
- HeroService را به عنوان provider سرویس خود در سطح ریشه (Root level) ثبت کردید به گونه ای که بتوان آن را در هر جایی از نرم افزار تزریق کرد.
- برای تزریق Angular از Angular dependency injection استفاده کردید.
- به متد HeroService get data یک امضای همزمان دادید.
- با observable و کتاب خانه RxJS Observable آشنا شدید.
- برای برگشت دادن observable مربوط به mock hero ها از RxJS of () استفاده کردید (observable<Hero[]>) .
- هوک چرخه عمر ngonInit کامپوننت (component's ngOnInit lifecycle hook) متد HeroService را فراخوانی می کند نه constructor را.
- بین کلاس هایی که ارتباط ضعیفی وجود داشت، MessageService ایجاد کردید.
- HeroService که در یکی از کامپوننت ها تزریق شد، خود توسط سرویس تزریقی دیگر به نام MessageService ایجاد شد.