Angular 2 Services & Dependency Injection

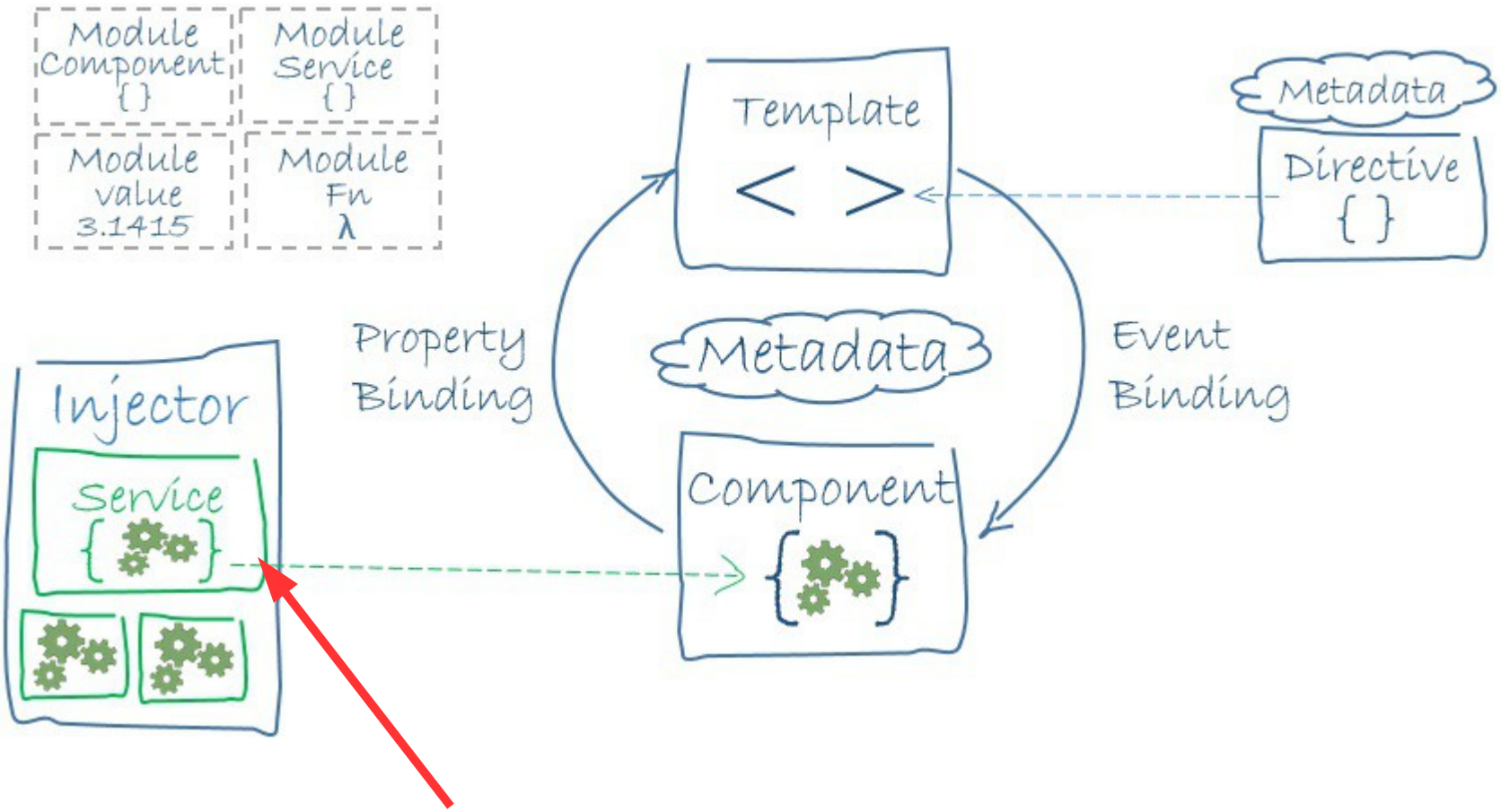
Sang Shin
JPassion.com
"Code with Passion!"

Topics

- Angular 2 Services
- Dependency injection
- @Injectable
- EventEmitter
- Using a pipe within service code

Angular 2 Services

Building Blocks of Angular 2 Application



source: https://angular.io/docs/ts/latest/guide/architecture.html

What is a Angular 2 Service?

- Almost anything can be a service
  - A service is typically a class with a narrow, well-defined purpose
  - Services are reuse'able
- Example services
  - logging service
  - data service (interaction with databases)
  - tax calculator service
  - ...
- There is nothing special about Angular services
  - It is just a class
  - Angular has no definition of a service - there is no service decorator, no place to register a service..

Angular Provides Many Built-in Services

- Http service
  - Let you perform RESTful operations

Where do you place logic? Component or Service?

- Component class should be lean
  - A component's job is to enable the user experience and nothing more
  - A good component presents properties and methods for data binding with DOM elements and it delegates everything nontrivial to services

Dependency Injection

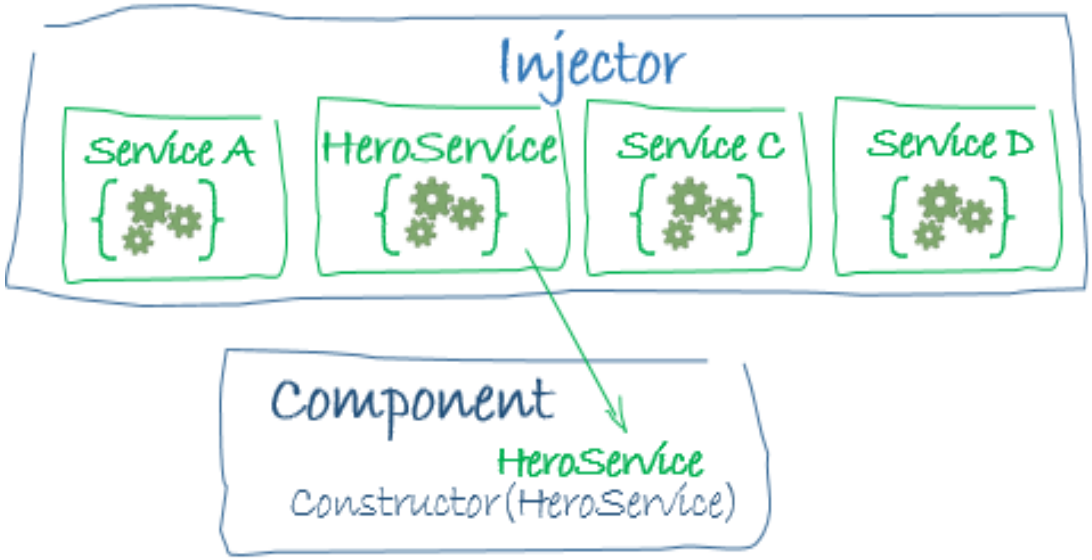
What is Dependency Injection (DI)?

- Dependency injection is a way in which a dependency object gets injected into the depending (target) class
  - Instead of the depending class itself to create the dependency object
  - DI makes code more testable, more maintainable
- Most dependencies are services in Angular application
  - The target could be component, directive, service
- Angular can tell which services a target needs by looking at the types of its constructor parameters (in the target class)
  - If you use private, Angular will also create a private property with that name and assign the injected object

```
constructor(private customService: CustomService, private http: Http) { }
```

What is an injector?

- When Angular creates a component, it first asks an injector for the services that the component requires
- An injector maintains a container of service instances that it created



How does injector create an instance?

- If the injector doesn't have a service instance, how does it know how to make one?
  - It does not until you let it know by registering a service provider using [providers] metadata in the module the target component/directive/service belongs to
  - A provider is something that can create or return a service, typically the service class itself

```
@NgModule({
  declarations: [...],
  imports: [...],
  providers: [MyService],
  bootstrap: [AppComponent]
})
export class AppModule {
}
```

Lab: Create/use Log Service

- Create a LogService using angular-cli
  - ng g service log
  - add logMessage(message:string) method
- Use the LogService in the AppComponent
  - Inject it into the target component
  - Use it in the business logic
- Import the LogService in the module

Lab: Create/use a Data Storage Service

- Create two components – JohnComponent, MaryComponent
- Create a DataService
  - > ng g service data
- Add a button to both components to store and display all strings from the DataService

@Injectable()

Injecting a Service into another Service

- In Angular, you can inject a service only to a class that has some kind of metadata
  - Component already has *@Component* metadata
  - Service might not have any metadata, however
- For a service that does not have metadata, and which needs an injection of another service, *@Injectable()* can be used to provide metadata
  - *@Injectable()* is not needed for a service, which is injected into another component/directive/service but does not have any service injected into it

Lab: @Injectable()

- Inject LogService to DataService
  - Make sure @Injectable() is annotated to the DataService – If you have created DataService via angular-cli, it should be added automatically
  - Try it with and without @Injectable on the DataService

EventEmitter

EventEmitter

- Used by components to emit custom Events (https://angular.io/docs/ts/latest/api/core/index/EventEmitter-class.html)
- emit(value?: T) - use it to emit event

```
this.eventEmitter.emit(message);
```

- subscribe(generatorOrNext?: any, error?: any, complete?: any) : any - use it to subscribe the event

```
this.eventEmitter.subscribe(
    (data) => this.receivedMessage = data
)
```

Lab: Create a publish/subscriber Service

- Create a service called MessageService that can send and receive a message using EventEmitter

```
eventEmitter: EventEmitter<string>;

constructor() {}

sendMessage(message: string){
   this.eventEmitter.emit(message);
}
```

- Send a message from John to Mary

Using a Pipe within Service code

Lab: Using DatePipe in the Service code

- Create DatePipe object and use it

```
import { DatePipe } from '@angular/common';

export class LogService {

  logIt (message: string){
    let datePipe = new DatePipe('en-US');
    let formattedDate = datePipe.transform(new Date(), 'yyyy-MM-dd');
    console.log(formattedDate + ":" + message);
  }

}
```

- Create UpperCasePipe object and use it to convert the message to uppercase

Lab: Inject DatePipe object

- Inject *DatePipe* into the LogService (instead of creating yourself as in the previous slide) so that you can use the filter in the service code

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

@Injectable()
export class LogService {
  constructor(private datePipe: DatePipe) { }

  logIt(message){
    const formattedDate = this.datePipe.transform(new Date(), 'yyyy-MM-dd');
    console.log(formattedDate + ":" + message);
  }
}
```

- You have to register DatePipe in the AppModule

Code with Passion!
JPassion.com