Capgemini

Angular 4

Services and Dependency Injection

Dependency Injection

Dependency injection is a design pattern that allows for the removal of hard-coded dependencies, thus making it possible to remove or change them at run time.

```
function Foo(object) {
    this.object = object;
}

Foo.prototype.showDetails = function(data) {
  this.object.display(data);
}

var greeter = {
      display : function(msg){
        alert(msg);
      }
}

var foo = new Foo(greeter);
foo.showDetails("Capgemini");
```

Dependency Injection in Angular 4

DI allows to inject dependencies in different components across applications, without needing to know, how those dependencies are created, or what dependencies they need themselves.

DI can also be considered as framework which helps us out in maintaining assembling dependencies for bigger applications.

Angular 1 has it's own DI system which allows us to annotate services and other components and let the injector find out, what dependencies need to be instantiated. But it has the following limitations.

- Internal Cache
- Namespace Collision
- Built into the framework

Services

Services provided architectural way to encapsulate business logic in a reusable fashion.

Services allow to keep logic out of your components, directives and pipe classes

Services can be injected in the application using Angular's dependency injection (DI).

Angular has In-built service classes like Http, FormBuilder and Router which contains logic for doing specific things that are non component specific.

Custom Services are most often used to create Data Services.

Working with Services in Angular 4

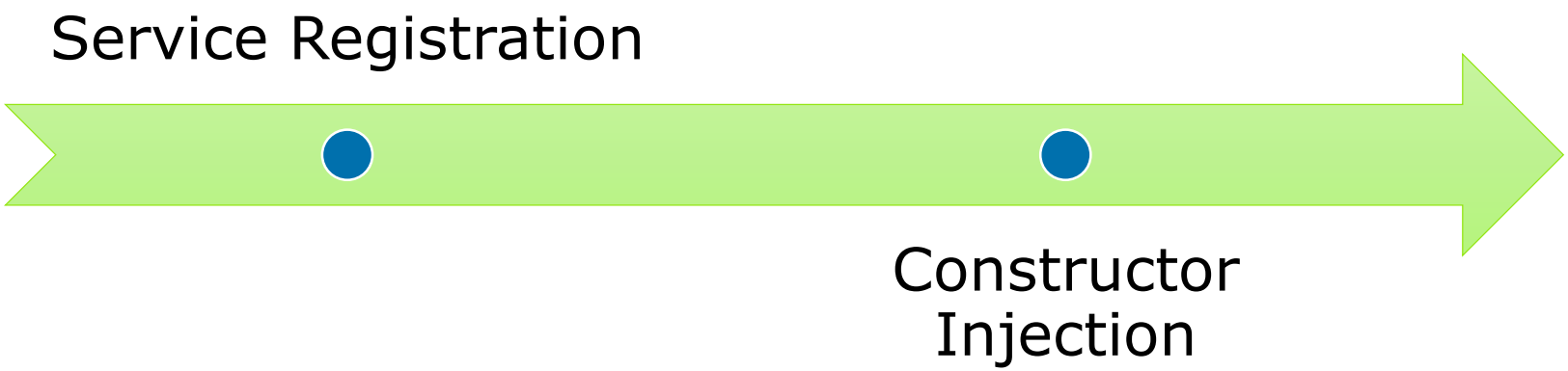
Component can work with service class using two ways

- Creating an instance of the service class
  - Instances are local to the component, so data or other resources cannot be shared
  - Difficult to test the service
- Registering the service with angular using angular Injector
  - Angular injector maintains a container of created service instances
  - The injector creates and manages the single instance or singleton of each registered service.
  - Angular injector provides or injects the service class instance when the component class is instantiated. This process is called dependency injection
  - Angular manages the single instance any data or logic in that instance is shared by all of the classes that use it. This technique is the recommended way to use services because it provides better management of service instances it allow sharing of data and other resources and it's easier to mock the services for testing purposes

Working with Services in Angular 4

Angular has dependency injection support baked into the framework which allows to create component directives in modular fashion.

DI creates instances of objects and inject them into places where they are needed in a two step process.

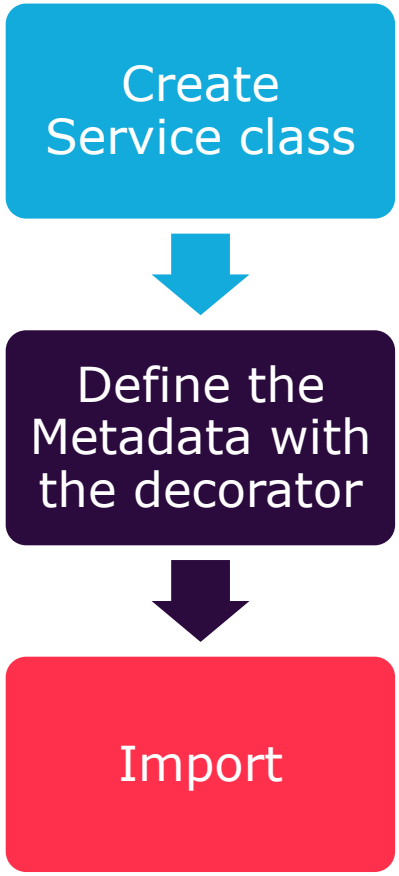


Building a Service

Steps to build a service is similar to build components and a custom pipe.

It is recommended that every service class use the injectable decorator for clarity and consistency.

@Injectable is a decorator, that informs Angular 4 that the service has some dependencies itself. Basically services in Angular 4 are simple classes with the decorator @Injectable on top of the class, that provides a method to return some items.



Registering a Service

To register a service we must register a provider.

A provider is code that can create or return a service typically the service class itself.

- To register a provider define it as part of the component Metadata, so that Angular injector can inject the service into the component and any of its children.

Injecting a Service

In typescript a constructor is defined with a constructor function.

The constructor function is executed when the component is created.

- It is primarily use for initialization and not for the code that has side effects or takes time to execute.

Dependency can be injected as a parameters to the constructor function.

- When class is constructed the angular injector sets this parameter to the injected instance of the requested service.
- The injected service instance can be assigned to the local variable by adding the private keyword to the constructor parameter, so that it can be accessed anywhere in the class.

Demo

ServiceDemo

Registering-and-Injecting-Service

Providers

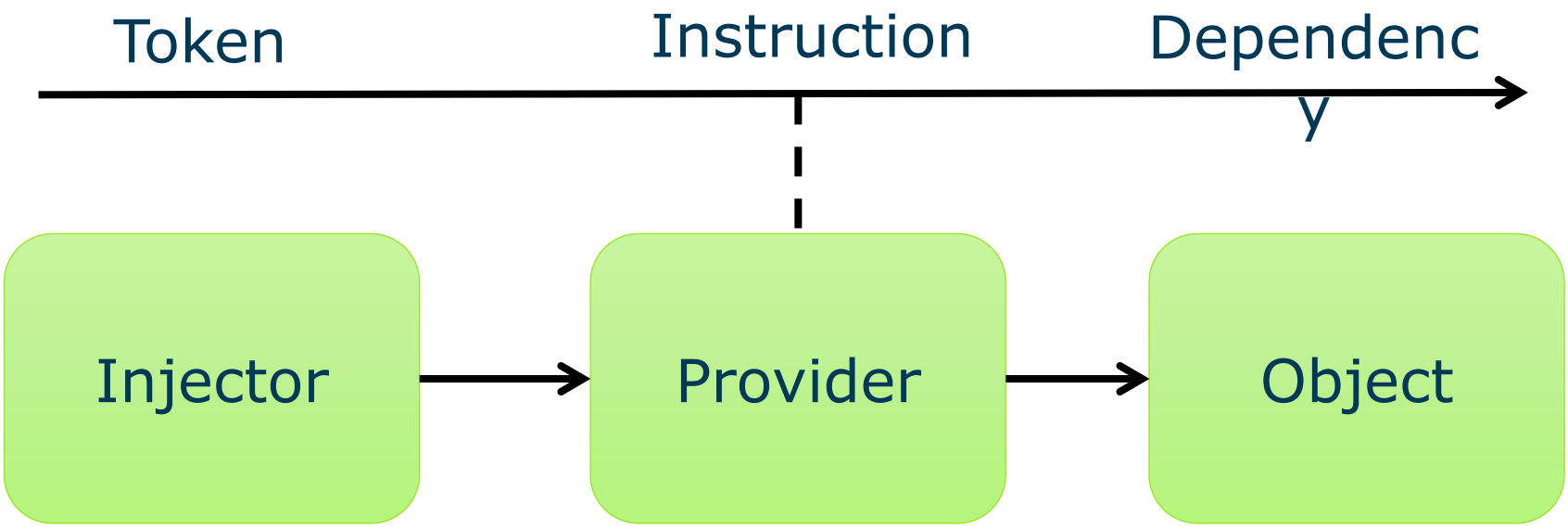
Providers are usually singleton (one instance) objects, that other objects have access to through dependency injection (DI).

A provider describes what the injector should instantiate a given token, so it describes how an object for a certain token is created.

Angular 4 offers the following type of providers:

- A class provider generates/provides an instance of the class (*useClass*).
- A factory provider generates/provides whatever returns when you run a specified function (*useFactory*).
- Aliased Class Provider (useExisting)
- A value provider just returns a value (useValue).

Angular 4 DI System



Demo

ClassProvider

ValueProvider

FactoryProvider

Observables

Observables is like an array whose items arrived asynchronously.

Observable help to manage asynchronous data, such as data coming from a backend service.

Observables are proposed feature for ES 2016 the next version of JavaScript. To use observables now angular uses a third party library called reactive extensions.

Observables are used with in angular itself including angular's event system and its http client service

A method can be subscribed to an observable to receive asynchronous notifications as new data arrives.

Introducing RxJs

RxJs stands for Reactive Extensions for Javascript, and its an implementation of Observables for Javascript.

It is a ReactiveX library for JavaScript.

It provides an API for asynchronous programming with observable streams.

ReactiveX is a combination of the best ideas from the Observer pattern, the Iterator pattern, and functional programming.

```
var source =
Rx.Observable.interval(1000).map(num=>['1','2','3','A','4','5','6'][num])
;
var result = source.map(x=>parseInt(x)).filter(x=> !isNaN(x));
result.subscribe(x=>console.log(x));
```

Demo

Observables

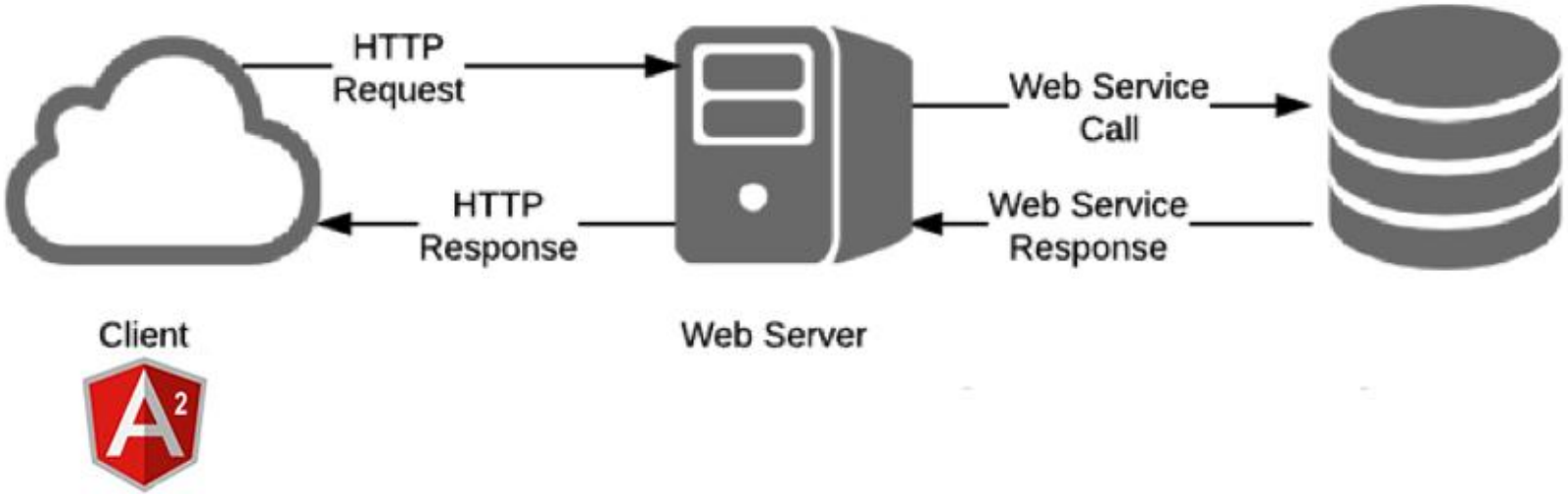
RenderingObservableWithpipe

Angular 4 HTTP

Angular applications often obtain data using http

Application issues http get requests to a web server which returns http response to the application.

Application then processes that data



Http Class

Performs http requests using `XMLHttpRequest` as the default backend.

Http is available as an injectable class.

Calling request returns an Observable which will emit a single Response when a response is received.

To work with Http Class

- Include Angular 4 Http script (http.dev.js) in index.html
- Include script tag for the reactive extensions (Rx.js) in index.html
- Register HTTP_PROVIDERS
- Import RxJS

Catch Operator

Reacts to the error case of an Observable.

Need to return a new Observable to continue with

```
export class UserProxy{
    constructor(private http :Http){}
    load(){
        return this.http
            .get('http://api.randomuser.me/10')
            .map(res =>res.json())
            .catch(this.logAndPassOn);
}
private logAndPassOn (error: Error) {
    console.error(error);
    return Observable.throw(error);
}
```

Communication with JSONP

Angular provides us with a JSONP services which has the same API surface as the Http.

Only difference that it restricts us to use GET requests only.

JSONP service requires the JSONP_PROVIDERS.

Server Simulation

To enable our server simulation, we replace the XHRBackend service with the in-memory web api backend.

The in-memory api must to implements ConnectionBackend

```
bootstrap(App,[ HTTP_PROVIDERS,
   // in-memory web api providers
   provide( XHRBackend, { useClass: InMemoryBackend
} )
]);
```

Demo

HttpDemo