**Angular 2 - Overview**

## **What is Angular 2?**

Angular 2 is an open source JavaScript framework to build web applications in HTML and JavaScript and has been conceived as a mobile first approach.

## **History**

The beta version of Angular 2 has been released in the March of 2015.

## **Why to use Angular 2?**

* Angular 2 is simpler than Angular 1 and its fewer concepts make it easier to understand.
* You can update the large data sets with minimal memory overhead.
* It will speed up the initial load through server side rendering.

## **Features**

* Angular 2 is faster and easier than Angular 1.
* It supports latest the version of browsers and also supports old browsers including IE9+ and Android 4.1+.
* It is a cross platform framework.
* Angular 2 is mainly focused on mobile apps.
* Code structure is very simplified than the previous version of Angular.

## **Advantages**

* If an application is a heavy load, then Angular 2 keeps it fully UI responsive.
* It uses server side rendering for fast views on mobile.
* It works well with ECMAScript and other languages that compile to JavaScript.
* It uses dependency injection to maintain applications without writing too long code.
* Everything will be the component based approach.

## **Disadvantages**

* Since Angular 2 is a newly introduced framework, there is less online community support.
* It takes time to learn if you are new to Angular 2.

# Angular 2 - Environment

In this chapter let us study about *Angular 2 development environment*.

* Angular uses *TypeScript* which is a primary language for developing of Angular applications.
* The *TypeScript* is a super set of JavaScript which is migrated to TypeScript and code written in TypeScript makes less prone to run time errors.

To setup development environment follow the below steps:

**Step(1):** Create a project folder in your local drive by typing below commands in the command prompt.

mkdir angular2-demo

cd angular2-demo

## **Creating Configuration Files**

**Step(2):** You need to create **tsconfig.json** which is the TypeScript compiler configuration file. It guides the compiler to generate JavaScript files.

{

"compilerOptions": {

"target": "es5",

"module": "system",

"moduleResolution": "node",

"sourceMap": true,

"emitDecoratorMetadata": true,

"experimentalDecorators": true,

"removeComments": false,

"noImplicitAny": false

},

"exclude": [

"node\_modules",

"typings/main",

"typings/main.d.ts"

]

}

**Step(3):** Create a **typings.json** file in your project folder *angular2-demo* as shown below:

**typings.json**

{

"globalDependencies": {

"core-js": "registry:dt/core-js#0.0.0+20160602141332",

"jasmine": "registry:dt/jasmine#2.2.0+20160621224255",

"node": "registry:dt/node#6.0.0+20160621231320"

}

}

A large number of libraries of the JavaScript extends JavaScript environment with features and syntax which is not natively recognized by the TypeScript compiler. The **typings.json** file is used to identify TypeScript definition files in your Angular application.

In the above code, there are three typing files as shown below:

* **core-js**: It brings ES2015/ES6 capabilities to our ES5 browsers.
* **jasmine**: It is the typing for Jasmine test framework.
* **node**: It is used for the code that references objects in the nodejs environment.

These typings are used in development of larger Angular applications.

**Step(4):** Add **package.json** file to your *angular2-demo* project folder with the below code:

**package.json**

{

"name": "angular2-demo",

"version": "1.0.0",

"scripts": {

"start": "concurrent \"npm run tsc:w\" \"npm run lite\" ",

"tsc": "tsc",

"tsc:w": "tsc -w",

"lite": "lite-server",

"typings": "typings",

"postinstall": "typings install"

},

"license": "ISC",

"dependencies": {

"angular2": "2.0.0-beta.7",

"systemjs": "0.19.22",

"es6-promise": "^3.0.2",

"es6-shim": "^0.33.3",

"reflect-metadata": "0.1.2",

"rxjs": "5.0.0-beta.2",

"zone.js": "0.5.15"

},

"devDependencies": {

"concurrently": "^2.0.0",

"lite-server": "^2.1.0",

"typescript": "^1.7.5",

"typings":"^0.6.8"

}

}

The **package.json** will contain the packages that our apps require. These packages are installed and maintained with npm (Node Package Manager). To install *npm* [click here](https://docs.npmjs.com/getting-started/installing-node).

**Step(5):** To install packages, run the below npm command in command prompt.

npm install

Error messages in red may appear while installing npm, just ignore them.

## **Creating Our First Angular Component**

A component is the fundamental concept of Angular. A component is a class that controls a view template - a part of a web page where information to the user is displayed and user feedback is responded. Components are required to build Angular apps.

**Step(6):** Create a sub-folder called *app/* inside your project folder to the place Angular app components. You can use the below command to create the folder:

mkdir app

cd app

**Step(7):** The files which you create need to be saved with **.ts** extension. Create a file called **environment\_app.component.ts** in your *app/* folder with the below code:

**environment\_app.component.ts**

import {Component, View} from "angular2/core";

@Component({

selector: 'my-app'

})

@View({

template: '<h2>My First Angular 2 App</h2>'

})

export class AppComponent {

}

The above code will import the *Component* and *View* package from *angular2/core*.

* The *@Component* is an Angular 2 *decorator* that allows you to associate metadata with the component class.
* The *my-app* can be used as HTML tag to injecting and can be used as a component.
* The *@view* contains a *template* that tells Angular how to render a view.
* The *export* specifies that, this component will be available outside the file.

**Step(8):** Next, create **environment\_main.ts** file with the below code:

**environment\_main.ts**

import {bootstrap} from "angular2/platform/browser"

import {AppComponent} from "./environment\_app.component"

bootstrap(AppComponent);

* The *environment\_main.ts* file tells Angular to load the component.
* To launch the application, we need to import both *Angular's browser bootstrap* function and *root component of the application*.
* After importing, the *bootstrap* is called by passing the *root component type* i.e. *AppComponent*.

**Step(9):** Now create a **index.html** in your project folder *angular2-demo/* with the below code:

**index.html**

<!DOCTYPE html>

<html>

<head>

<title>Hello World</title>

<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>

<script src="https://code.angularjs.org/tools/system.js"></script>

<script src="https://code.angularjs.org/tools/typescript.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>

<script>

System.config({

transpiler: 'typescript',

typescriptOptions: { emitDecoratorMetadata: true },

packages: {'app': {defaultExtension: 'ts'}},

map: { 'app': './angular2/src/app' }

});

System.import('app/environment\_main')

.then(null, console.error.bind(console));

</script>

</head>

<body>

<my-app>Loading...</my-app>

</body>

</html>

Angular will launch the app in the browser with our component and places it in a specific location on *index.html*.

## **Compile and Run**

**Step(10):** To run the application, type the below command in a terminal window:

npm start

The above command runs two parallel node processes as listed below:

* TypeScript compiler in the watch mode
* The **lite-server (static server)** loads the *index.html* in a browser and refreshes the browser as application files change.

After few moments, a browser tab will get open with the following output:

# Angular 2 - Hello World

## **Description**

In the previous chapter, we studied how to setup development environment for Angular 2. In this chapter let us create an example to display *Hello World* text.

## **Example**

The below example describes how to display a simple text in the Angular 2:

<!DOCTYPE html>

<html>

<head>

<title>Hello World</title>

<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>

<script src="https://code.angularjs.org/tools/system.js"></script>

<script src="https://code.angularjs.org/tools/typescript.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>

<script>

System.config({

transpiler: 'typescript',

typescriptOptions: { emitDecoratorMetadata: true },

packages: {'app': {defaultExtension: 'ts'}},

map: { 'app': './angular2/src/app' }

});

System.import('app/hello\_world\_main')

.then(null, console.error.bind(console));

</script>

</head>

<body>

<my-app>Loading...</my-app>

</body>

</html>

The above code includes the following configuration options:

* You can configure the *index.html* file using *typescript* version. The SystemJS transpile the TypeScript to JavaScript before running the application by using the *transpiler* option.
* If you do not transpile to JavaScript before running the application, you could see the compiler warnings and errors which are hidden in the browser.
* The TypeScript generates metadata for each and every class of the code when the *emitDecoratorMetadata* option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impact on the application runtime.
* Angular 2 includes the packages form the *app* folder where files will have the *.ts* extension.
* Next it will load the main component file from the *app* folder. If there is no main component file found, then it will display the error in the console.
* When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the 'app' selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following *TypeScript(.ts)* files which you need to save under the *app* folder.

**hello\_world\_main.ts**

import {bootstrap} from "angular2/platform/browser"

import {MyHelloWorldClass} from "./hello\_world\_app.component"

bootstrap(MyHelloWorldClass);

Now we will create a component in TypeScript(.ts) file as shown below:

**hello\_world\_app.component.ts**

import {Component, View} from "angular2/core";

@Component({

selector: 'my-app'

})

@View({

template: '<h2>Hello World !!</h2>'

})

export class MyHelloWorldClass {

}

* The *@Component* is a decorator which uses configuration object to create the component.
* The *selector* creates an instance of the component where it finds <my-app> tag in parent HTML.
* The *@view* contains a *template* that tells Angular how to render a view.
* The *export* specifies that the component will be available outside the file.

## **Output**

Let's carry out the following steps to see how above code works:

* Save above HTML code as **index.html** file as how we created in [environment](https://www.tutorialspoint.com/angular2/angular2_environment.htm) chapter and use the above *app* folder which contains *.ts* files.
* Open the terminal window and enter the below command:

npm start

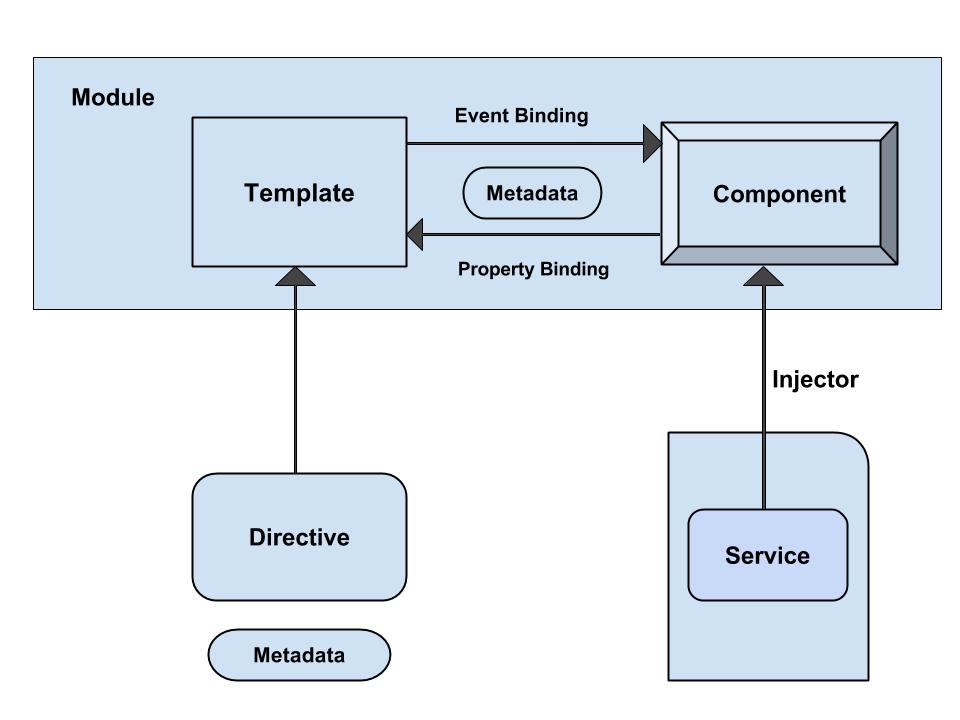
* After few moments, a browser tab should open and displays the output as shown below.

**OR** you can run this file in another way:

* Save above HTML code as **angular2\_hello\_world.html** file in your server root folder.
* Open this HTML file as http://localhost/angular2\_hello\_world.html and output as below gets displayed.

# Angular 2 - Architecture

In this chapter, we are going to discuss the architectural style of Angular 2 framework, for implementing user applications. The following diagram shows architecture of Angular 2:



The architecture of Angular 2 contains following modules:

* **Module**
* **Component**
* **Template**
* **Metadata**
* **Data Binding**
* **Service**
* **Directive**
* **Dependency Injection**

## **Module**

The module component is characterized by a block of code which can be used to perform a single task. You can export the value of something from the code such as a class. The Angular apps are called as modules and build your application using many modules. The basic building block of Angular 2 is a *component* class which can be exported from a module.

Some of the applications will have a component class name as *AppComponent* and you can find it in a file called *app.component.ts*. Use the *export* statement to export component class from module as shown below:

export class AppComponent { }

The *export* statement specifies that it is a module and its *AppComponent* class is defined as public and can be accessible to other modules of the application.

## **Component**

A component is a controller class with a template which mainly deals with a view of the application and logic on the page. It is a bit of code that can be used throughout an application. Component knows how to render itself and configure dependency injection. You can associate CSS styles using component inline styles, style urls and template inline styles to a component.

To register component, we use *@Component* annotation and can be used to break up the application into smaller parts. There will be only one component per DOM element.

## **Template**

The component's view can be defined by using the *template* which tells Angular how to display the component. For instance, below simple template shows how to display the name:

<div>

Your name is : {{name}}

</div>

To display the value, you can put template expression within the interpolation braces.

## **Metadata**

Metadata is a way of processing the class. Consider we have one component called *MyComponent* which will be a class until we tell Angular that it's a component. You can use *metadata* to the class to tell Angular that *MyComponent* is a component. The metadata can be attached to TypeScript by using the *decorator*.

For instance:

@Component({

selector : 'mylist',

template : '<h2>Name is Harry</h2>'

directives : [MyComponentDetails]

})

export class ListComponent{...}

The *@Component* is a decorator which uses configuration object to create the component and its view. The *selector* creates an instance of the component where it finds <mylist> tag in parent HTML. The *template* tells Angular how to display the component. The *directive* decorator is used to represent the array of components or directives.

## **Data Binding**

Data binding is a process of coordinating application data values by declaring bindings between sources and target HTML elements. It combines the template parts with components parts and template HTML is bound with markup to connect both sides. There are four types of data binding:

* **Interpolation**: It displays the component value within the div tags.
* **Property Binding**: It passes the property from the parent to property of the child.
* **Event Binding**: It fires when you click on the components method name.
* **Two-way Binding**: This form binds property and event by using the *ngModel* directive in a single notation.

## **Service**

Services are JavaScript functions that are responsible for doing a specific task only. Angular services are injected using Dependency Injection mechanism. Service includes the value, function or feature which is required by the application. Generally, service is a class which can perform something specific such as logging service, data service, message service, the configuration of application etc. There is nothing much about service in Angular and there is no ServiceBase class, but still services can be treated as fundamental to Angular application.

## **Directive**

The directive is a class that represents the metadata. There are three types of directives:

* **Component Directive**: It creates custom controller by using view and controller and used as custom HTML element.
* **Decorator Directive**: It decorates the elements using additional behavior.
* **Template Directive**: It converts HTML into a reusable template.

## **Dependency Injection**

Dependency Injection is a design pattern that passes an object as dependencies in different components across the application. It creates new a instance of class along with its required dependencies.

You must remember the below points while using Dependency Injection:

* The Dependency Injection is stimulated into the framework and can be used everywhere.
* The *injector* mechanism maintains service instance and can be created using a *provider*.
* The *provider* is a way of creating a service.
* You can register the *providers* along with injectors.

# Angular 2 - Modules

## **Description**

The applications in Angular follow modular structure. The Angular apps will contain many *modules*, each dedicated to the single purpose. Typically module is a cohesive group of code which is integrated with the other modules to run your Angular apps.

A module *exports* some *classes*, *function* and *values* from its code. The *Component* is a fundamental block of Angular and multiple *components* will make up your application.

A module can be a library for another module. For instance, the *angular2/core* library which is a primary Angular library module will be imported by another *component*.

## **Example**

The below example describes use of modules in the Angular 2:

<!DOCTYPE html>

<html>

<head>

<title>Angular 2 Modules</title>

<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>

<script src="https://code.angularjs.org/tools/system.js"></script>

<script src="https://code.angularjs.org/tools/typescript.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>

<script>

System.config({

transpiler: 'typescript',

typescriptOptions: { emitDecoratorMetadata: true },

packages: {'app': {defaultExtension: 'ts'}},

map: { 'app': './angular2/src/app' }

});

System.import('app/modules\_main')

.then(null, console.error.bind(console));

</script>

</head>

<body>

<my-app>Loading...</my-app>

</body>

</html>

The above code includes the following configuration options:

* You can configure the *index.html* file using *typescript* version. The SystemJS transpile the TypeScript to JavaScript before running the application by using the *transpiler* option.
* If you do not transpile to JavaScript before running the application, you could see the compiler warnings and errors which are hidden in the browser.
* The TypeScript generates metadata for each and every class of the code when the *emitDecoratorMetadata* option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impact on the application runtime.
* Angular 2 includes the packages form the *app* folder where files will have the *.ts* extension.
* Next it will load the main component file from the *app* folder. If there is no main component file found, then it will display the error in the console.
* When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the 'app' selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following *TypeScript(.ts)* files which you need to save under the *app* folder.

**modules\_main.ts**

import {bootstrap} from "angular2/platform/browser" //importing bootstrap function

import {MyModulesClass} from "./modules\_app.component" //importing component function

bootstrap(MyModulesClass);

Now we will create a component in TypeScript(.ts) file which we will create component and view for the component.

**modules\_app.component.ts**

import {Component, View} from "angular2/core";

//framework recognizes @Component annotation and knows that we are trying to create a new component

@Component({

selector: 'my-app'

})

@View({

//this template value will be displayed in the browser

template: '<h2>Welcome to Tutorialspoint</h2>'

})

export class MyModulesClass { }

## **Output**

When you run the above code, it will display the text specified within the *template* option which is defined in the *modules\_app.component.ts* file. Let's carry out the following steps to see how above code works:

* Save above HTML code as **index.html** file as how we created in [environment](https://www.tutorialspoint.com/angular2/angular2_environment.htm) chapter and use the above *app* folder which contains *.ts* files.
* Open the terminal window and enter the below command:

npm start

* After few moments, a browser tab should open and displays the output as shown below.

**OR** you can run this file in another way:

* Save above HTML code as **angular2\_modules.html** file in your server root folder.
* Open this HTML file as http://localhost/angular2\_modules.html and output as below gets displayed.

# Angular 2 - Components

## **Description**

The component is a controller class with a template which mainly deals with a view of the application and logic on the page. It is a bit of code can be used throughout an application. The component knows how to render itself and configure dependency injection.

The component contains two important things; one is *view* and another one is *some logic*.

## **Example**

The below example describes use of component in the Angular 2:

<!DOCTYPE html>

<html>

<head>

<title>Angular 2 Component</title>

<!--Load libraries -->

<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>

<script src="https://code.angularjs.org/tools/system.js"></script>

<script src="https://code.angularjs.org/tools/typescript.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>

<script>

System.config({

//transpiler tool converts TypeScript to JavaScript

transpiler: 'typescript',

//emitDecoratorMetadata flag used by JavaScript output to create metadata from the decorators

typescriptOptions: { emitDecoratorMetadata: true },

packages: {'app': {defaultExtension: 'ts'}},

map: { 'app': './angular2/src/app' }

});

System.import('app/component\_main')

.then(null, console.error.bind(console));

</script>

</head>

<!--When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the 'app' selector, locates an element tag named app, and loads the application between those tags.-->

<body>

<app>Loading...</app>

</body></html>

The above code includes the following configuration options:

* You can configure the *index.html* file using *typescript* version. The SystemJS transpile the TypeScript to JavaScript before running the application by using the *transpiler* option.
* If you do not transpile to JavaScript before running the application, you could see the compiler warnings and errors which are hidden in the browser.
* The TypeScript generates metadata for each and every class of the code when the *emitDecoratorMetadata* option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impact on the application runtime.
* Angular 2 includes the packages form the *app* folder where files will have the *.ts* extension.
* Next it will load the main component file from the *app* folder. If there is no main component file found, then it will display the error in the console.
* When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the 'app' selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following *TypeScript(.ts)* files which you need to save under the *app* folder.

**component\_main.ts**

import {bootstrap} from "angular2/platform/browser"; //importing bootstrap function

import {App} from "./component\_app.component" //importing component function

bootstrap(App);

Now we will create a component in TypeScript(.ts) file which we will create component and view for the component.

**component\_app.component.ts**

// component's metadata can be accessed using this primary Angular library

import {Component, View} from "angular2/core";

//framework recognizes @Component annotation and knows that we are trying to create a new component

@Component({

selector: 'app' //specifies selector for HTML element named 'app'

})

@View({

//template property holds component's companion template that tells Angular how to render a view

template: '<h2>Welcome to {{name}}</h2>'

})

export class App {

name : 'Tutorialspoint!!!'

}

## **Output**

When you run the above code, it will display the text specified within the *template* option which is defined in the *component\_app.component.ts* file and holds component's companion template that tells Angular how to render a view.

Let's carry out the following steps to see how above code works:

* Save above HTML code as **index.html** file as how we created in [environment](https://www.tutorialspoint.com/angular2/angular2_environment.htm) chapter and use the above *app* folder which contains *.ts* files.
* Open the terminal window and enter the below command:

npm start

* After few moments, a browser tab should open and displays the output as shown below.

**OR** you can run this file in another way:

* Save above HTML code as **angular2\_components.html** file in your server root folder.
* Open this HTML file as http://localhost/angular2\_components.html and output as below gets displayed.

# Angular 2 - Templates

## **Description**

The component's view can be defined by using the *template* which tells Angular how to display the component. The template describes how the component is rendered on the page.

## **Example**

The below example describes use of template in the Angular 2:

<!DOCTYPE html>

<html>

<head>

<title>Angular 2 Template</title>

<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>

<script src="https://code.angularjs.org/tools/system.js"></script>

<script src="https://code.angularjs.org/tools/typescript.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>

<script>

System.config({

transpiler: 'typescript',

typescriptOptions: { emitDecoratorMetadata: true },

packages: {'app': {defaultExtension: 'ts'}},

map: { 'app': './angular2/src/app' }

});

System.import('app/template\_main')

.then(null, console.error.bind(console));

</script>

</head>

<body>

<my-app>Loading...</my-app>

</body>

</html>

The above code includes the following configuration options:

* You can configure the *index.html* file using *typescript* version. The SystemJS transpile the TypeScript to JavaScript before running the application by using the *transpiler* option.
* If you do not transpile to JavaScript before running the application, you could see the compiler warnings and errors which are hidden in the browser.
* The TypeScript generates metadata for each and every class of the code when the *emitDecoratorMetadata* option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impact on the application runtime.
* Angular 2 includes the packages form the *app* folder where files will have the *.ts* extension.
* Next it will load the main component file from the *app* folder. If there is no main component file found, then it will display the error in the console.
* When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the 'app' selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following *TypeScript(.ts)* files which you need to save under the *app* folder.

**template\_main.ts**

import {bootstrap} from "angular2/platform/browser" //importing bootstrap function

import {MyTemplate} from "./template\_app.component" //importing component function

bootstrap(MyTemplate);

Now we will create a component in TypeScript(.ts) file which we will create component and view for the component.

**template\_app.component.ts**

// component's metadata can be accessed using this primary Angular library

import {Component, View} from "angular2/core";

//framework recognizes @Component annotation and knows that we are trying to create a new component

@Component({

selector: 'my-app' //specifies selector for HTML element named 'app'

})

@View({

//template property holds component's companion template that tells Angular how to render a view

template: '<h2>Welcome to the world of {{val}}</h2>'

})

export class MyTemplate {

val : 'Tutorialspoint!!!'

}

## **Output**

When you run the above code, it will display the text specified within the *template* option which is defined in the *template\_app.component.ts* file and holds component's companion template that tells Angular how to render a view.

Let's carry out the following steps to see how above code works:

* Save above HTML code as **index.html** file as how we created in [environment](https://www.tutorialspoint.com/angular2/angular2_environment.htm) chapter and use the above *app* folder which contains *.ts* files.
* Open the terminal window and enter the below command:

npm start

* After few moments, a browser tab should open and displays the output as shown below.

**OR** you can run this file in another way:

* Save above HTML code as **angular2\_templates.html** file in your server root folder.
* Open this HTML file as http://localhost/angular2\_templates.html and output as below gets displayed.

# Angular 2 - Metadata

## **Description**

Metadata is a way of processing the class. Consider you have one component called *MyComponent* which will be a class until you tell Angular that it's a component. You can use *metadata* to the class to tell Angular that *MyComponent* is a component and metadata can be attached to TypeScript by using the *decorator*.

## **Example**

The below example describes use of metadata in the Angular 2:

<!DOCTYPE html>

<html>

<head>

<title>Angular 2 Metadata</title>

<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>

<script src="https://code.angularjs.org/tools/system.js"></script>

<script src="https://code.angularjs.org/tools/typescript.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>

<script>

System.config({

transpiler: 'typescript',

typescriptOptions: { emitDecoratorMetadata: true },

packages: {'app': {defaultExtension: 'ts'}},

map: { 'app': './angular2/src/app' }

});

System.import('app/metadata\_main')

.then(null, console.error.bind(console));

</script>

</head>

<body>

<my-app>Loading...</my-app>

</body>

</html>

The above code includes the following configuration options:

* You can configure the *index.html* file using *typescript* version. The SystemJS transpile the TypeScript to JavaScript before running the application by using the *transpiler* option.
* If you do not transpile to JavaScript before running the application, you could see the compiler warnings and errors which are hidden in the browser.
* The TypeScript generates metadata for each and every class of the code when the *emitDecoratorMetadata* option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impact on the application runtime.
* Angular 2 includes the packages form the *app* folder where files will have the *.ts* extension.
* Next it will load the main component file from the *app* folder. If there is no main component file found, then it will display the error in the console.
* When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the 'app' selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following *TypeScript(.ts)* files which you need to save under the *app* folder.

**metadata\_main.ts**

import {bootstrap} from "angular2/platform/browser" //importing bootstrap function

import {MyTemplate} from "./metadata\_app.component" //importing component function

bootstrap(MyTemplate);

Now we will create a component in TypeScript(.ts) file as shown below:

**metadata\_app.component.ts**

import {Component} from "angular2/core";

import {ItemComponent} from './item-list.component';

@Component({

selector: 'my-app',

template: `<my-list></my-list>`,

directives:[ItemComponent]

})

export class MyTemplate {}

* The *@Component* is a decorator which uses configuration object to create the component and its view.
* The *selector* creates an instance of the component where it finds <my-app> tag in parent HTML.
* The *directive* decorator is used to represent the array of components or directives.

The below TypeScript(.ts) file displays the list of items on the output.

**item-list.component.ts**

import {Component} from "angular2/core";

@Component({

selector:'my-list',

template:`<h2>List of Fruits</h2>

<ul>

<li \*ngFor="#myItem of itemList">{{myItem.name}}</li>

</ul>

`

})

export class ItemComponent{

public itemList = [

{name:"Apple"},

{name:"Orange"},

{name:"Grapes"},

];

}

* The *template* tells Angular how to display the component.
* The *\*ngFor* directive is used to loop the list of items from the array of *itemList* object.

## **Output**

Let's carry out the following steps to see how above code works:

* Save above HTML code as **index.html** file as how we created in [environment](https://www.tutorialspoint.com/angular2/angular2_environment.htm) chapter and use the above *app* folder which contains *.ts* files.
* Open the terminal window and enter the below command:

npm start

* After few moments, a browser tab should open and displays the output as shown below.

**OR** you can run this file in another way:

* Save above HTML code as **angular2\_metadata.html** file in your server root folder.
* Open this HTML file as http://localhost/angular2\_metadata.html and output as below gets displayed.

# Angular 2 - Data Binding

## **Description**

Data binding is the synchronization of data between the model and view components. To display the component property, you can put its name in the view template, enclosed in double curly braces. Two-way data binding merges property and event binding in a single notation using the directive *ngModel*.

## **Example**

The below example describes use of data binding in the Angular 2:

<!DOCTYPE html>

<html>

<head>

<title>Data Binding</title>

<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>

<script src="https://code.angularjs.org/tools/system.js"></script>

<script src="https://code.angularjs.org/tools/typescript.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>

<script>

System.config({

transpiler: 'typescript',

typescriptOptions: { emitDecoratorMetadata: true },

packages: {'app': {defaultExtension: 'ts'}},

map: { 'app': './angular2/src/app' }

});

System.import('app/data\_binding\_main')

.then(null, console.error.bind(console));

</script>

</head>

<body>

<my-app>Loading...</my-app>

</body>

</html>

The above code includes the following configuration options:

* You can configure the *index.html* file using *typescript* version. The SystemJS transpile the TypeScript to JavaScript before running the application by using the *transpiler* option.
* If you do not transpile to JavaScript before running the application, you could see the compiler warnings and errors which are hidden in the browser.
* The TypeScript generates metadata for each and every class of the code when the *emitDecoratorMetadata* option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impact on the application runtime.
* Angular 2 includes the packages form the *app* folder where files will have the *.ts* extension.
* Next it will load the main component file from the *app* folder. If there is no main component file found, then it will display the error in the console.
* When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the 'app' selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following *TypeScript(.ts)* files which you need to save under the *app* folder.

**data\_binding\_main.ts**

import {bootstrap} from 'angular2/platform/browser'; //importing bootstrap function

import {AppComponent} from "./data\_binding\_app.component"; //importing component function

bootstrap(AppComponent);

Now we will create a component in TypeScript(.ts) file as shown below:

**data\_binding\_app.component.ts**

import {Component} from 'angular2/core';

@Component({

selector: 'my-app',

template: `

<ul>

<li

\*ngFor="#Item of Items"

(click)="onItemClicked(Item)">

{{ Item.name }}

</li>

</ul>

<input type="text" [(ngModel)]="clickedItem.name">

`

})

export class AppComponent {

public Items = [

{name: "Butter"},

{name: "Milk"},

{name: "Yogurt"},

{name: "Cheese"},

];

public clickedItem = {name: ""};

onItemClicked(Item) {

this.clickedItem = Item;

}

}

* The *@Component* is a decorator which uses configuration object to create the component and its view.
* The *selector* creates an instance of the component where it finds <my-app> tag in parent HTML.
* Next is *\*ngFor* directive creates the view exports which we bind to in the template. The \* is a shorthand for using Angular 2 template syntax with the template tag.
* The local variable *Item* can be referenced in the template and get the index of the array. When you click on the item value, the *onItemClicked()* event will get activate and Angular 2 will bind the model name from the array with the local variable of template.
* The model name *clickedItem* is binded with *name* and displays the item name in the text box when user click on item name from the list.

## **Output**

Let's carry out the following steps to see how above code works:

* Save above HTML code as **index.html** file as how we created in [environment](https://www.tutorialspoint.com/angular2/angular2_environment.htm) chapter and use the above *app* folder which contains *.ts* files.
* Open the terminal window and enter the below command:

npm start

* After few moments, a browser tab should open and displays the output as shown below.

**OR** you can run this file in another way:

* Save above HTML code as **angular2\_data\_binding.html** file in your server root folder.
* Open this HTML file as http://localhost/angular2\_data\_binding.html and output as below gets displayed.

# Angular 2 - Data Display

## **Description**

You can display the data with the help of binding controls in the UI. Angular will display the data by using interpolation and other binding properties such as using binding in HTML template to the Angular component properties.

## **Example**

The below example describes use of displaying data in the Angular 2:

<!DOCTYPE html>

<html>

<head>

<title>Angular 2 Data Display</title>

<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>

<script src="https://code.angularjs.org/tools/system.js"></script>

<script src="https://code.angularjs.org/tools/typescript.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>

<script>

System.config({

transpiler: 'typescript',

typescriptOptions: { emitDecoratorMetadata: true },

packages: {'app': {defaultExtension: 'ts'}},

map: { 'app': './angular2/src/app' }

});

System.import('app/datadisplay\_main')

.then(null, console.error.bind(console));

</script>

</head>

<body>

<my-app>Loading...</my-app>

</body>

</html>

The above code includes the following configuration options:

* You can configure the *index.html* file using *typescript* version. The SystemJS transpile the TypeScript to JavaScript before running the application by using the *transpiler* option.
* If you do not transpile to JavaScript before running the application, you could see the compiler warnings and errors which are hidden in the browser.
* The TypeScript generates metadata for each and every class of the code when the *emitDecoratorMetadata* option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impact on the application runtime.
* Angular 2 includes the packages form the *app* folder where files will have the *.ts* extension.
* Next it will load the main component file from the *app* folder. If there is no main component file found, then it will display the error in the console.
* When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the 'app' selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following *TypeScript(.ts)* files which you need to save under the *app* folder.

**datadisplay\_main.ts**

import {bootstrap} from "angular2/platform/browser"

import {MyTemplate} from "./datadisplay\_app.component"

bootstrap(MyTemplate);

Now we will create a component in TypeScript(.ts) file as shown below:

**datadisplay\_app.component.ts**

import {Component, View} from "angular2/core";

@Component({

selector: 'my-app'

})

@View({

template: `

<h2>Showing data using component properties with interpolation</h2>

<h3>Player Name:{{player}}</h3>

<h3>He is famous in: {{sport}}</h3><br>

<h2>Showing data using constructor or variable initialization</h2>

<h3>India capital is: {{capital}}</h3><br>

<h2>Showing data using array property with NgFor</h2>

<h3>My favorite fruit is: {{myfruit}}</h3>

<p>List of Fruits:</p>

<ul>

<li \*ngFor="#fruit of fruits">

{{ fruit }}

</li>

</ul>

`

})

export class MyTemplate {

player: 'M.S. Dhoni ';

sport:'Cricket';

capital: string;

constructor() {

this.capital = 'New Delhi';

}

fruits = ['Apple', 'Orange', 'Mango', 'Grapes'];

myfruit = this.fruits[1];

}

* The *@Component* is a decorator which uses configuration object to create the component.
* The *selector* creates an instance of the component where it finds <my-app> tag in parent HTML.
* The *@view* contains a *template* that tells Angular how to render a view.
* The *export* specifies that the component will be available outside the file.

## **Output**

Let's carry out the following steps to see how above code works:

* Save above HTML code as **index.html** file as how we created in [environment](https://www.tutorialspoint.com/angular2/angular2_environment.htm) chapter and use the above *app* folder which contains *.ts* files.
* Open the terminal window and enter the below command:

npm start

* After few moments, a browser tab should open and displays the output as shown below.

**OR** you can run this file in another way:

* Save above HTML code as **angular2\_data\_display.html** file in your server root folder.
* Open this HTML file as http://localhost/angular2\_data\_display.html and output as below gets displayed.

# Angular 2 - User Input

## **Description**

When the user clicks a button, enters text or clicks a link, these user interactions will trigger DOM events. The below table describes how to bind to these events using Angular event binding syntax.

| **S.N.** | **Event & Description** |
| --- | --- |
| 1 | [**Binding to User Input Events**](https://www.tutorialspoint.com/angular2/binding_user_input.htm) User can input the text or display the text value when you click on it by using the Angular event binding. |
| 2 | [**User Input from $event Object**](https://www.tutorialspoint.com/angular2/user_input_event_object.htm) You can display the input value by binding keyup event and displays the text back what user types onto the screen. |
| 3 | [**User Input from Local Template Variable**](https://www.tutorialspoint.com/angular2/user_input_template_var.htm) You can display the user data by using local template variable. |
| 4 | [**Key Event Filtering**](https://www.tutorialspoint.com/angular2/user_input_key_event_filtering.htm) User can display the data of input box by pressing 'Enter' key on the keyboard. |
| 5 | [**On Blur Event**](https://www.tutorialspoint.com/angular2/user_input_onblur.htm) You can make the input value blur by clicking the mouse outside of the input box on the page. |

# Angular 2 - Forms

## **Description**

In this chapter let us study how to create a *form*. We shall use the following classes and directives in our example.

* The *form-group*, *form-control* and *btn* classes from *Bootstrap*.
* The *[(ngModel)]* for data binding and *NgControl* directive to keep track of control state for us.
* The *NgControl* is one among in *NgForm* directive family which is used for validation and tracking the form elements.
* The *ngSubmit* directive is used for handling the submission of the form.

## **Example**

The below example describes how to create a form in the Angular 2:

<html>

<head>

<title>Contact Form</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">

<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>

<script src="https://code.angularjs.org/tools/system.js"></script>

<script src="https://code.angularjs.org/tools/typescript.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>

<script>

System.config({

transpiler: 'typescript',

typescriptOptions: { emitDecoratorMetadata: true },

packages: {'app': {defaultExtension: 'ts'}},

map: { 'app': './app' }

});

System.import('app/form\_main')

.then(null, console.error.bind(console));

</script>

</head>

<body>

<my-app>Loading...</my-app>

</body>

</html>

The above code includes the following configuration options:

* You can configure the *index.html* file using *typescript* version. The SystemJS transpile the TypeScript to JavaScript before running the application by using the *transpiler* option.
* If you do not transpile to JavaScript before running the application, you could see the compiler warnings and errors which are hidden in the browser.
* The TypeScript generates metadata for each and every class of the code when the *emitDecoratorMetadata* option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impact on the application runtime.
* Angular 2 includes the packages form the *app* folder where files will have the *.ts* extension.
* Next it will load the main component file from the *app* folder. If there is no main component file found, then it will display the error in the console.
* When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the 'app' selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following *TypeScript(.ts)* files which you need to save under the *app* folder.

**form\_main.ts**

import {bootstrap} from 'angular2/platform/browser';

import {AppComponent} from "./forms\_app.component";

bootstrap(AppComponent);

**contact.ts**

export class Contact {

constructor(

public firstname: string,

public lastname: string,

public country: string,

public phone: number

) { }

}

The *Contact* class contains *firstname*, *lastname*, *country* and *phone* which are used in our form.

**forms\_app.component.ts**

import {Component} from 'angular2/core';

import {ContactComponent} from './contact-form.component'

@Component({

selector: 'my-app',

template: '',

directives: [ContactComponent]

})

export class AppComponent { }

* The *@Component* is a decorator which uses configuration object to create the component.
* The *selector* creates an instance of the component where it finds <my-app> tag in parent HTML.
* The *template* that tells Angular what to render as view.
* The above *app.component.ts* will import the *ContactComponent* component and uses *directives* to include the component.

**contact-form.component.ts**

import {Component} from 'angular2/core';

import {NgForm} from 'angular2/common';

import { Contact } from './contact';

@Component({

selector: 'contact-form',

templateUrl: 'app/contact-form.component.html'

})

export class ContactComponent {

countries = ['India', 'Australia', 'England', 'South Africa', 'USA', 'Japan', 'Singapore'];

contact = new Contact('Ravi', 'Shankar', this.countries[0], 6445874544);

submitted = false;

onSubmit() { this.submitted = true; }

active = true;

newContact() {

this.contact = new Contact('', '');

this.active = false;

setTimeout(()=> this.active=true, 0);

}

}

* The NgForm is imported which provides *CSS classes* and *Model states*.
* The *templateUrl* property provides the path to the *contact-form.component.html* file which contains our form elements.
* The *onSubmit()* method will alter the *submitted* value to *true* once invoked and the *newContact()* will create new contact.

**contact-form.component.html**

<div class="container">

<div [hidden]="submitted">

<h2>Contact Form</h2>

<form \*ngIf="active" (ngSubmit)="onSubmit()" #contactForm="ngForm" novalidate>

<div class="form-group">

<label for="firstname">First Name</label>

<input type="text" class="form-control" placeholder="Enter Your First Name" required

[(ngModel)]="contact.firstname"

ngControl="firstname" #firstname="ngForm" >

<div [hidden]="firstname.valid || firstname.pristine" class="alert alert-danger">

firstname is required

</div>

</div>

<div class="form-group">

<label for="lastname">Last Name</label>

<input type="text" class="form-control" placeholder="Enter Your Last Name"

[(ngModel)]="contact.lastname"

ngControl="lastname" >

</div>

<div class="form-group">

<label for="country">Country</label>

<select class="form-control" required

[(ngModel)]="contact.country"

ngControl="country" #country="ngForm" >

<option value="" selected disabled>Select Your Country</option>

<option \*ngFor="#coun of countries" [value]="coun">{{coun}}</option>

</select>

<div [hidden]="country.valid || country.pristine" class="alert alert-danger">

Country is required

</div>

</div>

<div class="form-group">

<label for="phone">Phone Number</label>

<input type="number" class="form-control" placeholder="Enter Your Phone Number"

[(ngModel)]="contact.phone"

ngControl="phone"

>

</div>

<button type="submit" class="btn btn-primary" [disabled]="!contactForm.form.valid">Submit</button>

<button type="button" class="btn btn-primary" (click)="newContact()">New Contact</button>

</form>

</div>

<div [hidden]="!submitted">

<h2>Your contact details :</h2>

<div class="well">

<div class="row">

<div class="col-xs-2">First Name</div>

<div class="col-xs-10 pull-left">{{ contact.firstname }}</div>

</div>

<div class="row">

<div class="col-xs-2">Last Name</div>

<div class="col-xs-10 pull-left">{{ contact.lastname }}</div>

</div>

<div class="row">

<div class="col-xs-2">Country</div>

<div class="col-xs-10 pull-left">{{ contact.country }}</div>

</div>

<div class="row">

<div class="col-xs-2">Phone Number</div>

<div class="col-xs-10 pull-left">{{ contact.phone }}</div>

</div>

<br>

<button class="btn btn-primary" (click)="submitted=false">Edit Contact</button>

</div>

</div>

</div>

* The above code contains the form, on submitting the form the *ngSubmit* directive calls *onSubmit()* method.
* The contact details are displayed when the *submitted* is set to *false*.
* It uses *pristine* and *valid* for validating an form input element.
* The *ngIf* directive checks whether the *active* is set to *true* for displaying the form.

## **Output**

Let's carry out the following steps to see how above code works:

* Save above HTML code as **index.html** file as how we created in [environment](https://www.tutorialspoint.com/angular2/angular2_environment.htm) chapter and use the above *app* folder which contains *.ts* files.
* Open the terminal window and enter the below command:

npm start

* After few moments, a browser tab should open and displays the output as shown below.

**OR** you can run this file in another way:

* Save above HTML code as **angular2\_forms.html** file in your server root folder.
* Open this HTML file as http://localhost/angular2\_forms.html and output as below gets displayed.

# Angular 2 - Services

## **Description**

Services are JavaScript functions that are responsible for doing a specific task only. Angular services are injected using Dependency Injection mechanism and include the value, function or feature which is required by the application. There nothing much about service in Angular and there is no ServiceBase class, but still services can be treated as fundamental to Angular application.

## **Example**

The below example describes use of services in the Angular 2:

<!DOCTYPE html>

<html>

<head>

<title>Angular 2 Services</title>

<!--Load libraries -->

<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>

<script src="https://code.angularjs.org/tools/system.js"></script>

<script src="https://code.angularjs.org/tools/typescript.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>

<script>

System.config({

transpiler: 'typescript',

typescriptOptions: { emitDecoratorMetadata: true },

packages: {'app': {defaultExtension: 'ts'}},

map: { 'app': './angular2/src/app' }

});

System.import('app/service\_main')

.then(null, console.error.bind(console));

</script>

</head>

<body>

<my-app>Loading...</my-app>

</body>

</html>

The above code includes the following configuration options:

* You can configure the *index.html* file using *typescript* version. The SystemJS transpile the TypeScript to JavaScript before running the application by using the *transpiler* option.
* If you do not transpile to JavaScript before running the application, you could see the compiler warnings and errors which are hidden in the browser.
* The TypeScript generates metadata for each and every class of the code when the *emitDecoratorMetadata* option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impact on the application runtime.
* Angular 2 includes the packages form the *app* folder where files will have the *.ts* extension.
* Next it will load the main component file from the *app* folder. If there is no main component file found, then it will display the error in the console.
* When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the 'app' selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following *TypeScript(.ts)* files which you need to save under the *app* folder.

**metadata\_main.ts**

import {bootstrap} from 'angular2/platform/browser'; //importing bootstrap function

import {AppComponent} from "./app\_service.component"; //importing component function

bootstrap(AppComponent);

Now we will create a component in TypeScript(.ts) file which we will create a view for the component.

**app\_service.component.ts**

import {Component} from 'angular2/core';

import {MyListComponent} from "./service-list.component";

@Component({

selector: 'my-app',

template: `

<country-list></country-list>

`,

directives: [MyListComponent]

})

export class AppComponent {

}

* The *@Component* is a decorator which uses configuration object to create the component and its view.
* The *selector* creates an instance of the component where it finds <my-app> tag in parent HTML.
* Next we create a directive called *MyListComponent* which will be accessed from the *service-list.component* file.

**service-list.component.ts**

import {Component} from "angular2/core";

import {CountryService} from "./country.service";

import {Contact} from "./country";

import {OnInit} from "angular2/core";

@Component({

selector: "country-list",

template: ` List of Countries<br>

<ul>

<li \*ngFor="#cntry of countries">{{ cntry.name }}</li>

</ul>

`,

providers: [CountryService]

})

export class MyListComponent implements OnInit {

public countries : Country[];

constructor(private \_countryService: CountryService) {}

getContacts(){

this.\_countryService.getContacts().then((countries: Country[]) => this.countries = countries);

}

ngOnInit():any{

this.getContacts();

}

}

* The local variable *cntry* can be referenced in the template and get the index of the array. Angular 2 will bind the model name from the array with the local variable of template.
* We have resource called *providers* which registers class, function or value which are in the context of dependency injection. The service called *CountryService* can be injected using @Injectable() in the *country.service.ts* file.
* Next you have implement in the *MyListComponent* class using *OnInit* hook which indicates that Angular is done creating the component. Using constructor call the *\_countryService* and populate the *countries* list.
* The *ngOnInit()* hook is called when done with creating the component and evaluated the inputs.

**country.service.ts**

import {Injectable} from "angular2/core";

import {COUNTRIES} from "./country.contacts";

//@Injectable() specifies class is available to an injector for instantiation and an injector will display an error when trying to instantiate a class that is not marked as @Injectable()

@Injectable()

//CountryService exposes the getContacts() method that returns the data

export class CountryService {

getContacts() {

return Promise.resolve(COUNTRIES); // takes values from country.contacts typescript file

}

}

**country.contacts.ts**

import {Country} from "./country";

//storing array of data in Country

export const COUNTRIES: Country[] =[

{name :"India"},

{name: "Srilanka"},

{name: "South Africa"},

{name: "New Zealand"}

];

**country.ts**

export interface Country{

name: string

}

## **Output**

Let's carry out the following steps to see how above code works:

* Save above HTML code as **index.html** file as how we created in [environment](https://www.tutorialspoint.com/angular2/angular2_environment.htm) chapter and use the above *app* folder which contains *.ts* files.
* Open the terminal window and enter the below command:

npm start

* After few moments, a browser tab should open and displays the output as shown below.

**OR** you can run this file in another way:

* Save above HTML code as **angular2\_services.html** file in your server root folder.
* Open this HTML file as http://localhost/angular2\_services.html and output as below gets displayed.

# Angular 2 - Directives

## **Description**

In this chapter let us study about *Angular 2 directive*. Templates of the Angular are *dynamic*, when these templates are rendered by Angular, it changes the DOM according to the *directive* fed instructions. The directive is a class which contains metadata which will be attached to the class by the *@Directive* decorator.

Angular has three kinds of directives and is briefly explained in below table:

|  |  |
| --- | --- |
| **S.N.** | **Directives & Description** |
| 1 | [**Component**](https://www.tutorialspoint.com/angular2/angular2_components.htm) It is a *directive-with-a-template* and the *@Component* decorator which is indeed a *@Directive* decorator wherein the template-oriented features is extended. |
| 2 | [**Structural directives**](https://www.tutorialspoint.com/angular2/angular2_structural_directive.htm) It alters the layout of the DOM by adding, replacing and removing its elements. |
| 3 | [**Attribute directives**](https://www.tutorialspoint.com/angular2/angular2_attribute_directive.htm) It changes the appearance or behavior of a DOM element. These directives look like regular HTML attributes in templates. |

# Angular 2 - Dependency Injection

## **Description**

Dependency Injection is a design pattern that passes an object as dependencies in different components across the application. It creates a new instance of class along with its required dependencies. The Dependency Injection is stimulated into the framework and can be used everywhere.

You must remember the below points while using Dependency Injection:

* The *injector* mechanism maintains service instance and can be created using a *provider*.
* The *provider* is a way of creating a service.
* You can register the *providers* along with injectors.

## **Example**

The below example describes use of dependency injection in the Angular 2:

<!DOCTYPE html>

<html>

<head>

<title>Angular 2 Dependency Injection</title>

<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>

<script src="https://code.angularjs.org/tools/system.js"></script>

<script src="https://code.angularjs.org/tools/typescript.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>

<script>

System.config({

transpiler: 'typescript',

typescriptOptions: { emitDecoratorMetadata: true },

packages: {'app': {defaultExtension: 'ts'}},

map: { 'app': './angular2/src/app' }

});

System.import('app/dependency\_injection\_main')

.then(null, console.error.bind(console));

</script>

</head>

<body>

<my-app>Loading...</my-app>

</body></html>

The above code includes the following configuration options:

* You can configure the *index.html* file using *typescript* version. The SystemJS transpile the TypeScript to JavaScript before running the application by using the *transpiler* option.
* If you do not transpile to JavaScript before running the application, you could see the compiler warnings and errors which are hidden in the browser.
* The TypeScript generates metadata for each and every class of the code when the *emitDecoratorMetadata* option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impact on the application runtime.
* Angular 2 includes the packages form the *app* folder where files will have the *.ts* extension.
* Next it will load the main component file from the *app* folder. If there is no main component file found, then it will display the error in the console.
* When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the 'app' selector, locates an element tag named app, and loads the application between those tags.

Let's create the *TypeScript(.ts)* files and save them in the *app* folder.

**dependency\_injection\_main.ts**

import {bootstrap} from 'angular2/platform/browser'; //importing bootstrap function

import {AppComponent} from "./fruit-component"; //importing component function

bootstrap(AppComponent);

**fruit-component.ts**

import {Component} from 'angular2/core';

import {MyListComponent} from './play-component';

//@Component is a decorator that uses configuration object to create the component and its view.

@Component({

selector: 'my-app', //The selector creates an instance of the component where it finds 'my-app' tag in parent HTML

template:`<my-list></my-list>`,

directives:[MyListComponent] //'MyListComponent' is the root component of fruits that governs child components

})

export class AppComponent { }

**play-component.ts**

import {Component} from "angular2/core";

import {FruitService} from "./fruits.service";

import {Fruits} from "./fruits";

import {OnInit} from "angular2/core";

@Component({

selector: "my-list",

template: ` List of Fruits<br>

<ul>

<li \*ngFor="#list of fruits"> //The NgFor directive instantiates a template once per item from an iterable

{{list.id}} - {{ list.name }}

</li>

</ul>

`,

providers: [FruitService] //providers are part of @Component metadata

})

//The 'MyListComponent' should get list of fruits from the injected 'FruitService'

export class MyListComponent implements OnInit {

public fruits : Country[];

//Using constructor, call the \_fruitService and populate the fruits list

constructor(private \_fruitService: FruitService) {}

getContacts(){

this.\_fruitService.getContacts().then((fruits: Country[]) => this.fruits = fruits);

}

//The 'ngOnInit()' hook is called when done with creating the component and evaluated the inputs

ngOnInit():any{

this.getContacts();

}

}

**fruits.service.ts**

import {Injectable} from "angular2/core";

import {COUNTRIES} from "./fruits.lists";

//It is used for meta data generation and specifies that class is available to an injector for instantiation

@Injectable()

//'FruitService' exposes 'getContacts()' method that returns list of data

export class FruitService {

getContacts() {

return Promise.resolve(COUNTRIES); // takes values from fruits.lists.ts file

}

}

**fruits.lists.ts**

import {Fruits} from "./fruits";

//storing array of data in Fruits

export const COUNTRIES: Fruits[] =[

{"id": 1, name :"Apple"},

{"id": 2, name: "Grapes"},

{"id": 3, name: "Orange"},

{"id": 4, name: "Banana"}

];

**fruits.ts**

export interface Fruits{

id: number;

name: string;

}

## **Output**

Let's carry out the following steps to see how above code works:

* Save above HTML code as **index.html** file as how we created in [environment](https://www.tutorialspoint.com/angular2/angular2_environment.htm) chapter and use the above *app* folder which contains *.ts* files.
* Open the terminal window and enter the below command:

npm start

* After few moments, a browser tab should open and displays the output as shown below.

**OR** you can run this file in another way:

* Save above HTML code as **angular2\_dependency\_injection.html** file in your server root folder.
* Open this HTML file as http://localhost/angular2\_dependency\_injection.html and output as below gets displayed.

# AngularJS Interview Questions

Dear readers, these **AngularJS Interview Questions** have been designed specially to get you acquainted with the nature of questions you may encounter during your interview for the subject of **AngularJS**. As per my experience good interviewers hardly plan to ask any particular question during your interview, normally questions start with some basic concept of the subject and later they continue based on further discussion and what you answer:

**What is AngularJS?**

AngularJS is a framework to build large scale and high performance web application while keeping them as easy-to-maintain. Following are the features of AngularJS framework.

* AngularJS is a powerful JavaScript based development framework to create RICH Internet Application (RIA).
* AngularJS provides developers options to write client side application (using JavaScript) in a clean MVC (Model View Controller) way.
* Application written in AngularJS is cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
* AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache License version 2.0.

**What is data binding in AngularJS?**

Data binding is the automatic synchronization of data between model and view components. ng-model directive is used in data binding.

**What is scope in AngularJS?**

Scopes are objects that refer to the model. They act as glue between controller and view.

**What are the controllers in AngularJS?**

Controllers are JavaScript functions that are bound to a particular scope. They are the prime actors in AngularJS framework and carry functions to operate on data and decide which view is to be updated to show the updated model based data.

**What are the services in AngularJS?**

AngularJS come with several built-in services. For example $https: service is used to make XMLHttpRequests (Ajax calls). Services are singleton objects which are instantiated only once in app.

**What are the filters in AngularJS?**

Filters select a subset of items from an array and return a new array. Filters are used to show filtered items from a list of items based on defined criteria.

Explain directives in AngularJS.

Directives are markers on DOM elements (such as elements, attributes, css, and more). These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives (ng-bind, ng-model, etc) to perform most of the task that developers have to do.

Explain templates in AngularJS.

Templates are the rendered view with information from the controller and model. These can be a single file (like index.html) or multiple views in one page using "partials".

**What is routing in AngularJS?**

It is concept of switching views. AngularJS based controller decides which view to render based on the business logic.

**What is deep linking in AngularJS?**

Deep linking allows you to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.

**What are the advantages of AngularJS?**

Following are the advantages of AngularJS.

* AngularJS provides capability to create Single Page Application in a very clean and maintainable way.
* AngularJS provides data binding capability to HTML thus giving user a rich and responsive experience.
* AngularJS code is unit testable.
* AngularJS uses dependency injection and make use of separation of concerns.
* AngularJS provides reusable components.
* With AngularJS, developer writes less code and gets more functionality.
* In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.
* AngularJS applications can run on all major browsers and smart phones including Android and iOS based phones/tablets.

**What are the disadvantages of AngularJS?**

Following are the disadvantages of AngularJS.

* **Not Secure** − Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure.
* **Not degradable** − If your application user disables JavaScript then user will just see the basic page and nothing more.

Which are the core directives of AngularJS?

Following are the three core directives of AngularJS.

* **ng-app** − This directive defines and links an AngularJS application to HTML.
* **ng-model** − This directive binds the values of AngularJS application data to HTML input controls.
* **ng-bind** − This directive binds the AngularJS Application data to HTML tags.

**Explain AngularJS boot process.**

When the page is loaded in the browser, following things happen:

* HTML document is loaded into the browser, and evaluated by the browser. AngularJS JavaScript file is loaded; the angular *global* object is created. Next, JavaScript which registers controller functions is executed.
* Next AngularJS scans through the HTML to look for AngularJS apps and views. Once view is located, it connects that view to the corresponding controller function.
* Next, AngularJS executes the controller functions. It then renders the views with data from the model populated by the controller. The page gets ready.

**What is MVC?**

**M**odel **V**iew **C**ontroller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

* **Model** − It is the lowest level of the pattern responsible for maintaining data.
* **View** − It is responsible for displaying all or a portion of the data to the user.
* **Controller** − It is a software Code that controls the interactions between the Model and View.

**Explain ng-app directive.**

ng-app directive defines and links an AngularJS application to HTML. It also indicate the start of the application.

**Explain ng-model directive.**

ng-model directive binds the values of AngularJS application data to HTML input controls. It creates a model variable which can be used with the html page and within the container control( for example, div) having ng-app directive.

**Explain ng-bind directive.**

ng-bind directive binds the AngularJS Application data to HTML tags. ng-bind updates the model created by ng-model directive to be displayed in the html tag whenever user input something in the control or updates the html control's data when model data is updated by controller.

**Explain ng-controller directive.**

ng-controller directive tells AngularJS what controller to use with this view. AngularJS application mainly relies on controllers to control the flow of data in the application. A controller is a JavaScript object containing attributes/properties and functions. Each controller accepts $scope as a parameter which refers to the application/module that controller is to control.

**How AngularJS integrates with HTML?**

AngularJS being a pure javaScript based library integrates easily with HTML.

**Step 1** − Include angularjs javascript libray in the html page

<head>

<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

</head>

**Step 2** − Point to AngularJS app

Next we tell what part of the HTML contains the AngularJS app. This done by adding the *ng-app* attribute to the root HTML element of the AngularJS app. You can either add it to *html* element or *body* element as shown below:

<body ng-app = "myapp">

</body>

**Explain ng-init directive.**

ng-init directive initializes an AngularJS Application data. It is used to put values to the variables to be used in the application.

**Explain ng-repeat directive.**

ng-repeat directive repeats html elements for each item in a collection.

**What are AngularJS expressions?**

Expressions are used to bind application data to html. Expressions are written inside double braces like {{ expression}}. Expressions behave in same way as ng-bind directives. AngularJS application expressions are pure JavaScript expressions and outputs the data where they are used.

**Explain uppercase filter.**

Uppercase filter converts a text to upper case text.

In below example, we've added uppercase filter to an expression using pipe character. Here we've added uppercase filter to print student name in all capital letters.

Enter first name:<input type = "text" ng-model = "student.firstName">

Enter last name: <input type = "text" ng-model = "student.lastName">

Name in Upper Case: {{student.fullName() | uppercase}}

**Explain lowercase filter.**

Lowercase filter converts a text to lower case text.

In below example, we've added lowercase filter to an expression using pipe character. Here we've added lowercase filter to print student name in all lowercase letters.

Enter first name:<input type = "text" ng-model = "student.firstName">

Enter last name: <input type = "text" ng-model = "student.lastName">

Name in Upper Case: {{student.fullName() | lowercase}}

**Explain currency filter.**

Currency filter formats text in a currency format.

In below example, we've added currency filter to an expression returning number using pipe character. Here we've added currency filter to print fees using currency format.

Enter fees: <input type = "text" ng-model = "student.fees">

fees: {{student.fees | currency}}

**Explain filter filter.**

filter filter is used to filter the array to a subset of it based on provided criteria.

In below example, to display only required subjects, we've used subjectName as filter.

Enter subject: <input type = "text" ng-model = "subjectName">

Subject:

<ul>

<li ng-repeat = "subject in student.subjects | filter: subjectName">

{{ subject.name + ', marks:' + subject.marks }}

</li>

</ul>

**Explain orderby filter.**

orderby filter orders the array based on provided criteria.

In below example, to order subjects by marks, we've used orderBy marks.

Subject:

<ul>

<li ng-repeat = "subject in student.subjects | orderBy:'marks'">

{{ subject.name + ', marks:' + subject.marks }}

</li>

</ul>

**Explain ng-disabled directive.**

ng-disabled directive disables a given control.

In below example, we've added ng-disabled attribute to a HTML button and pass it a model. Then we've attached the model to an checkbox and can see the variation.

<input type = "checkbox" ng-model = "enableDisableButton">Disable Button

<button ng-disabled = "enableDisableButton">Click Me!</button>

**Explain ng-show directive.**

ng-show directive shows a given control.

In below example, we've added ng-show attribute to a HTML button and pass it a model. Then we've attached the model to a checkbox and can see the variation.

<input type = "checkbox" ng-model = "showHide1">Show Button

<button ng-show = "showHide1">Click Me!</button>

**Explain ng-hide directive.**

ng-hide directive hides a given control.

In below example, we've added ng-hide attribute to a HTML button and pass it a model. Then we've attached the model to a checkbox and can see the variation.

<input type = "checkbox" ng-model = "showHide2">Hide Button

<button ng-hide = "showHide2">Click Me!</button>

**Explain ng-click directive.**

ng-click directive represents a AngularJS click event.

In below example, we've added ng-click attribute to a HTML button and added an expression to updated a model. Then we can see the variation.

<p>Total click: {{ clickCounter }}</p></td>

<button ng-click = "clickCounter = clickCounter + 1">Click Me!</button>

l

**How angular.module works?**

angular.module is used to create AngularJS modules along with its dependent modules. Consider the following example:

var mainApp = angular.module("mainApp", []);

Here we've declared an application **mainApp** module using angular.module function. We've passed an empty array to it. This array generally contains dependent modules declared earlier.

**How to validate data in AngularJS?**

AngularJS enriches form filling and validation. We can use $dirty and $invalid flags to do the validations in seamless way. Use novalidate with a form declaration to disable any browser specific validation.

Following can be used to track error.

* **$dirty** − states that value has been changed.
* **$invalid** − states that value entered is invalid.
* **$error** − states the exact error.

**Explain ng-include directive**.

Using AngularJS, we can embed HTML pages within a HTML page using ng-include directive.

<div ng-app = "" ng-controller = "studentController">

<div ng-include = "'main.htm'"></div>

<div ng-include = "'subjects.htm'"></div>

</div>

**How to make an ajax call using Angular JS?**

AngularJS provides $https: control which works as a service to make ajax call to read data from the server. The server makes a database call to get the desired records. AngularJS needs data in JSON format. Once the data is ready, $https: can be used to get the data from server in the following manner:

function studentController($scope,$https:) {

var url = "data.txt";

$https:.get(url).success( function(response) {

$scope.students = response;

});

}

**What is use of $routeProvider in AngularJS?**

$routeProvider is the key service which set the configuration of urls, maps them with the corresponding html page or ng-template, and attaches a controller with the same.

**What is $rootScope?**

Scope is a special JavaScript object which plays the role of joining controller with the views. Scope contains the model data. In controllers, model data is accessed via $scope object. $rootScope is the parent of all of the scope variables.

**What is scope hierarchy in AngularJS?**

Scopes are controllers specific. If we define nested controllers then child controller will inherit the scope of its parent controller.

<script>

var mainApp = angular.module("mainApp", []);

mainApp.controller("shapeController", function($scope) {

$scope.message = "In shape controller";

$scope.type = "Shape";

});

mainApp.controller("circleController", function($scope) {

$scope.message = "In circle controller";

});

</script>

Following are the important points to be considered in above example.

* We've set values to models in shapeController.
* We've overridden message in child controller circleController. When "message" is used within module of controller circleController, the overridden message will be used.

**What is a service?**

Services are JavaScript functions and are responsible to do specific tasks only. Each service is responsible for a specific task for example, $https: is used to make ajax call to get the server data. $route is used to define the routing information and so on. Inbuilt services are always prefixed with $ symbol.

**What is service method?**

Using service method, we define a service and then assign method to it. We've also injected an already available service to it.

mainApp.service('CalcService', function(MathService){

this.square = function(a) {

return MathService.multiply(a,a);

}

});

**What is factory method?**

Using factory method, we first define a factory and then assign method to it.

var mainApp = angular.module("mainApp", []);

mainApp.factory('MathService', function() {

var factory = {};

factory.multiply = function(a, b) {

return a \* b

}

return factory;

});

**What are the differences between service and factory methods?**

factory method is used to define a factory which can later be used to create services as and when required whereas service method is used to create a service whose purpose is to do some defined task.

**Which components can be injected as a dependency in AngularJS?**

AngularJS provides a supreme Dependency Injection mechanism. It provides following core components which can be injected into each other as dependencies.

* value
* factory
* service
* provider
* constant

**What is provider?**

provider is used by AngularJS internally to create services, factory etc. during config phase(phase during which AngularJS bootstraps itself). Below mention script can be used to create MathService that we've created earlier. Provider is a special factory method with a method get() which is used to return the value/service/factory.

//define a module

var mainApp = angular.module("mainApp", []);

...

//create a service using provider which defines a method square to return square of a number.

mainApp.config(function($provide) {

$provide.provider('MathService', function() {

this.$get = function() {

var factory = {};

factory.multiply = function(a, b) {

return a \* b;

}

return factory;

};

});

});

**What is constant?**

constants are used to pass values at config phase considering the fact that value cannot be used to be passed during config phase.

mainApp.constant("configParam", "constant value");

**Is AngularJS extensible?**

Yes! In AngularJS we can create custom directive to extend AngularJS existing functionalities.

Custom directives are used in AngularJS to extend the functionality of HTML. Custom directives are defined using "directive" function. A custom directive simply replaces the element for which it is activated. AngularJS application during bootstrap finds the matching elements and do one time activity using its compile() method of the custom directive then process the element using link() method of the custom directive based on the scope of the directive.

On which types of component can we create a custom directive?

AngularJS provides support to create custom directives for following type of elements.

* **Element directives** − Directive activates when a matching element is encountered.
* **Attribute** − Directive activates when a matching attribute is encountered.
* **CSS** − Directive activates when a matching css style is encountered.
* **Comment** − Directive activates when a matching comment is encountered.

**What is internationalization?**

Internationalization is a way to show locale specific information on a website. For example, display content of a website in English language in United States and in Danish in France.

How to implement internationalization in AngularJS?

AngularJS supports inbuilt internationalization for three types of filters currency, date and numbers. We only need to incorporate corresponding js according to locale of the country. By default it handles the locale of the browser. For example, to use Danish locale, use following script

<script src = "https://code.angularjs.org/1.2.5/i18n/angular-locale\_da-dk.js"></script>