Contents

[Blog 1](#_Toc35364852)

[TypeScript 1](#_Toc35364853)

[Type Script Playground: 1](#_Toc35364854)

[Key TypeScript Features 1](#_Toc35364855)

[TypeScript Compiler 2](#_Toc35364856)

[Angular Component & Modules 2](#_Toc35364857)

[Angular Module 3](#_Toc35364858)

[Angular Directives 3](#_Toc35364859)

[Binding Technics Interpolation, Property Binding 3](#_Toc35364860)

[Nested Components 4](#_Toc35364861)

## Blog

<https://blogs.msmvps.com/deborahk/angular-2-getting-started-problem-solver/>

Java script Language specification standard is officially called ECMAScript (ES)

* ES 3 - Supported by older browsers
* ES 5 – Supported by all modern browsers
* ES 6 – Renamed as ES 2015
  + Most browsers don’t support ES 2015. So this must be transpired to ES 5 (means ES 2015 code must be compiled by a tool that converts ES 2015 syntax into ES5 syntax so that browsers understands)

## TypeScript

Is a superset of Java script. It compiles typescript into Java script. Type script uses type script Type definition files while converting Types into Java script corresponding types. Type script definition files have extention (\*.d.ts)

### Type Script Playground:

<https://www.typescriptlang.org/play>

### Key TypeScript Features

* + TS is a super set of JavaScript, so it supports standard JavaScript Code
  + Provides static Typing (type safety compile time)
  + Encapsulation through classes and modules
  + Support for constructors, properties, functions
  + Define Interfaces
  + => function support (lambdas)
  + Intelligence and syntax checking

### TypeScript Compiler

TypeScript (.ts files) -> TypeScript compiler (tsc first.ts) -> JavaScript

### Tool / Framework Support

These are some of ktools convert TypeScript into JavaScript

* + - Node.js
    - Sublime
    - Emacs
    - Vi
    - Visual Studio

You can write js code in typescript, in the end its javaScript and TypeScript is superscript of js

### Function Int with return type void

**Int: (s: string, p:string, n:string) => void** = function (str1, str2, str3) {}

### Typing files (\*.d.ts)

TypeScript uses typing files to determine the types that a javascript library uses

More information:

<http://jpapa.me/typings>

### Primitive Types

* any : base type, this holds any value, represents any JavaScript value
* number
* boolean
* string
* arrays
  + example for string array: var names: string[] = [‘kp’,’john’];
* null

can be used for any of primitive types and objects.

Null type is a subtype of all primitives ( except void and undefined )

* undefined
  + var quantity: number; => is of type number but not initialized so it also going to be undefined
  + var company = undefined; => company is of any type and also undefined too
  + undefined is a subtype of all types

### Object Types

Examples:

Functions, class, module, interface, and literal types.

May Contain

Properties

Call Signatures, construct signatures, Index Signatures

## Functions

Arrow function expressions

var myFunc = function (h:number, w:number) {

return h \* w;

};

Above function using arrow function expression

var myFunc = (h: number, w: number) => h\*w;

In this we omitted the function keyword and have compact return statement. Void specifies no return value

Var greetMe = (msg: string) => void

## Angular Component & Modules

Component = Template + Class + Metadata

Template

Is created with HTML and contains what is rendered on page. It contains Angular bindings and directives to power up the view

Class

Is created with TypeScript. Class contains properties and methods. This is also called as ES2015 Module

Metadata

Provides additional information about component to Angular. It is this Metadata that defines this class as Component to Angular.

Metadata is defined with a Decorator.

Decorator is a function that adds metadata to a class, its members or its method arguments.

Decorator is prefixed with an @. Angular has many built-in decorators to provide additional information to Angular

Example:

Import { Component } from '@angular/core';

@Component({

selector: 'pm-root',

template: `<div><h1>{{pageTitle}}</h1>

<div> My First Component </div>

</div>`

})

Since it’s a function, started with (, then its passed object as argument. Object has two properties here, one is selector other is template.

Notice, there is no semicolon after Decorator function as this is similar to attributes used in other programing languages

With components we build nested user interface fragments.

Component is view defined with Template, its associated code defined with Class. Additional information defined with Metadata

Angular Modules

Every Angular application has at least one application Module, called Applications Root module

## Angular Module

Angular modules help to organize our applications into cohesive blocks of functionality. And provides boundaries with in our application

They also provide template resolution environment

It means when a Angular compiler sees a directive in a template, it looks to the angular module for the definition

So we declare the App component in an Angular module so the compiler can find it

We also use the module to bootstrap our component, which is our app component

We also want our application to work properly in the browser, so we add Angular's browser module to our Angular modules import

//We identify the class as Angular module by attaching the @NgModule decorator and passing metadata defining the details of this module. The properties are Array

@NgModule({

imports: [ BrowserModule], -- Define external modules that we want to have available to all our components that belong to this module. These can be Angular / external / our own. Every Browser application must import this module

declarations:[ AppComponent] --This defines which of our components belong to this Module

bootstrap: [AppComponent] --- Defines the startup component of the application. This should contain the selector we use in index.html

})

// We define Angular module using a class

Export class AppModule {

}

## Angular Directives

Directives

are used to add logic to html like if\* and For\* loops

It’s a custom HTML element or attribute used to power up and extend our html

We can use our own-custom directives or use Angular built in directives

Angular built in directives are Structural Directives (SD). SD modifies the structure or layout of the view. (\*) means its structural directive

\*ngIf

\*ngFor

## Binding Technics Interpolation, Property Binding

Interpolation

Is a one way binding from class property to the Template

Can perform operations / calculations

Used to insert text between elements

Can be used to elements property assignments

Ex:

<h1>{{ TemplateExpression/Class property }} </h1>

{{ 'Title: ' + getTitle() }}

<h1 innerText= {{ pageTitle }}> </h1>

{{ 2\*20+7 }}

Property Binding

Allows property of an element to the value of a template expression. Binding target always in [] brackets, representing it property of the element

<img [src]= 'product.imageUrl' />

Notice template property is enclosed in [], for property binding

Similar one using Interpolation would be like

<img src = {{product.imageUrl}} /> -- no [] enclosed to property and no codes around template expression

Only time a nested component property specified as a property binding target (left side of equal) is when that property is decorated with @Input decorator in subcomponent.

Events

(Event-name) = 'action-function-from-Class-Component'

Ex:

(click) = 'toggleImage()'

Two-way Binding

ngModel directive used for two-way binding

## Nested Components

We can use component as directive in html template

We can use component as routing target

Any component can be nested if selector is defined in the component decorator

Passing Data to a Nested Component using @Input decorator

Raising an Event from a Nested Component using @Output Decorator

* OnChanges life cycle event only watches to @input properties

Decorate nested component property with @input decorator to get input from parent Component. Input Decorator is a function

Decorate nested component property with @input decorator, any time it needs to raise events to its parent container. Along with event, it can pass information back to container

Only properties of type EventEmitter should be marked with @Output Decorator

Use the generic argument to define the event payload (like string)

Example: @Output() ratingClicked: EventEmitter<String> = new EventEmitter();

Use the property binding to pass data to the nested component. In below example rating property used property binding to pass product.starRating to nested component from Parent component.

Ex:

<pm-star [rating] = 'product.starRating' (ratingClicked)='onRatingClicked($event)'></pm-star>

Use Event binding to respond to events from the nested component. Above example ratingClicked event is passed from nested component, along with it string message is passed. Use $event to access the message (payload) passed from the nested component