Angular 4

Created by :
Sangeeta Joshi

Agenda

- Architectural Overview
- Hello World (Angular CLI)

Angular 1 Vs Angular 2/4

- No $scopes ,No controllers
- Everything is a component

What is Component?

:a self-contained object:

1. which owns its own presentation logic,
2. view,
3. internal state

Example: Button<button>

Advantages of Component Architecture

- Why Component based Architecture ?
  - A component is ***an independent software unit*** that can be composed with the other components to create a software system.
  - Component based web development : ***future of web development***.
  - Reusability
  - allow segmentation within the app to be written independently.
  - Developers can concentrate on business logic only.
- These things are not just features but the requirement of any thick-client web framework.

MVC

- Traditional MVC (n-tiered) Architecture Tries to be Loosely Coupled
- Separation of concern
- App is divided into layers: Model, View, Controller, Service, Persistence, Networking

Problems with MVC

- But MVC has lots of disadvantages:
  - Complexity
  - Fragility
  - Non-reusable
  - Difficult to extend existing functionality

MVVM

- VM : Component class
- View: Template

Angular 4 /2.0

- Angular 2 is the next version of Google's massively popular MV* framework :
- for building complex applications in the browser (and beyond).

  a faster

  more powerful

  Cleaner

  easier to use tool

  a tool that embraced future web standards

  brought ES6 to more developers around the world.

Angular 4 /2.0

- Angular 2 is the next version of Google's massively popular MV* framework :
- for building complex applications in the browser (and beyond).

  a faster

  more powerful

  Cleaner

  easier to use tool

  a tool that embraced future web standards

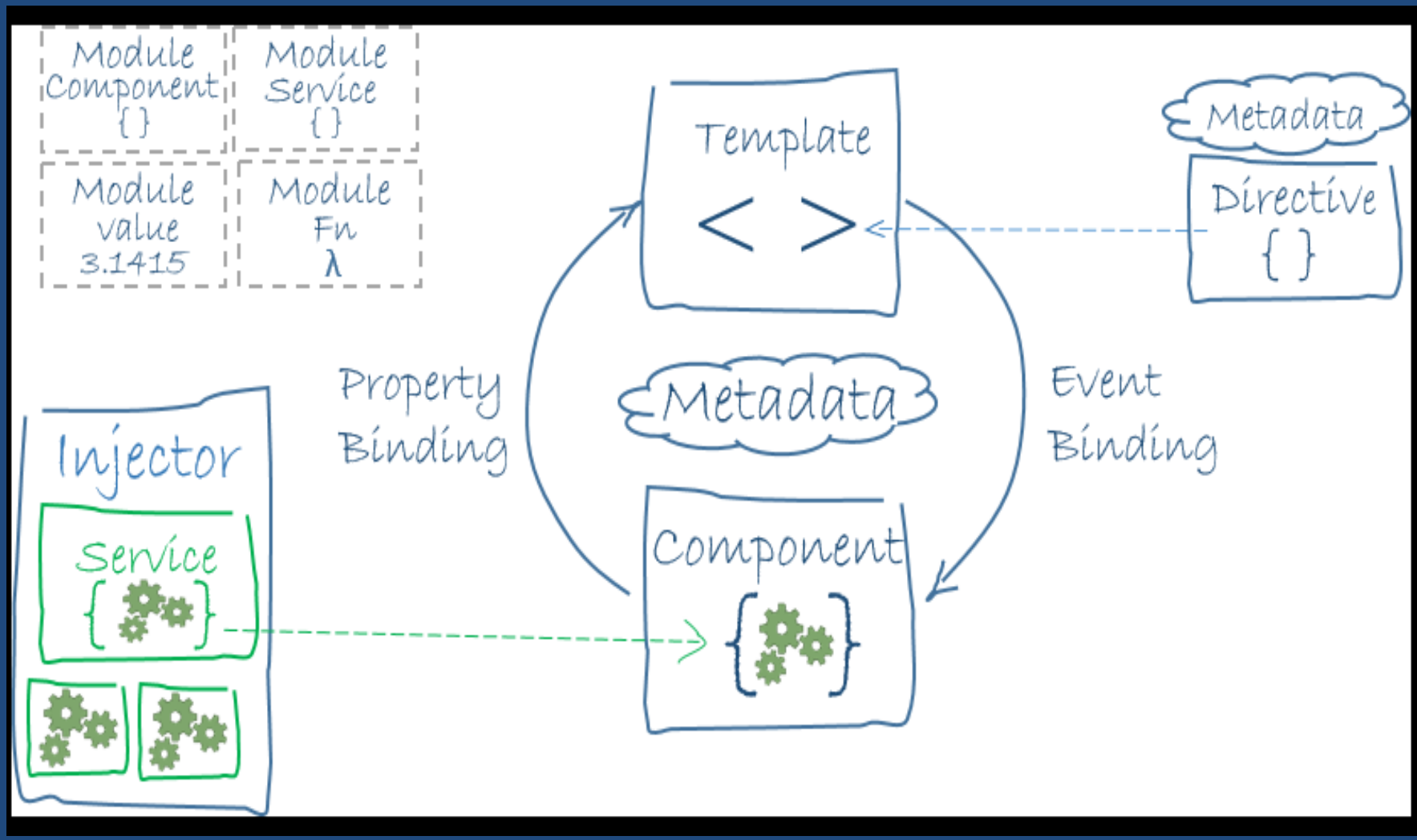
  brought ES6 to more developers around the world.

Architectural Overview

Main building blocks of an Angular application

- Modules
- Components
- Templates
- Metadata
- Data binding
- Directives
- Services
- Dependency injection

Architectural Overview



Overview

- Angular - a framework for building client applications in
  : HTML
  : JavaScript
  or a language like
  : TypeScript that compiles to JavaScript.
- The framework consists of several libraries, core and optional.

- For writing Angular applications involoves:
  composing HTML templates with Angularized markup
  writing component classes to manage those templates
  adding application logic in services
  boxing components and services in modules
  Then launching the app by bootstrapping the root module.

Modules

Ngmodules:

- Unit of compilation and distribution of Angular components and pipes.
- the compilation context of its components
- it tells Angular how these components should be compiled.

Modules

- Angular apps are modular
- Angular has its own modularity system called *Angular modules* or *NgModules*
- Every Angular app has at least one module, root module, conventionally named *AppModule*
- While the ***root module*** may be the only module in a small application, most apps have many more ***feature modules***
- An Angular module, whether a root or feature, is a class with an @NgModule decorator

(Decorators are functions that modify JavaScript classes. Angular has many decorators that attach metadata to classes so that it knows what those classes mean and how they should work)

Modules

NgModule is a decorator function that takes a single metadata object whose properties describe the module:

- Declarations
- Imports
- Providers

Bootstrap

Exports

Modules

- Declarations: “View Classes” those belong to this module.

  [view classes: Components,directives,pipes]
- Imports : Other modules needed by components declared in this module
- Providers :
- Bootstrap : It defines the components that are instantiated when a module is bootstrapped
- Exports : Only component mentioned here is added to the compilation context of AppModule

Components

What is a Component?

-A component knows how to interact with its host element.

-A component knows how to interact with its content and view children.

-A component knows how to render itself.

-A component configures dependency injection.

-A component has a well-defined public API of input and output properties.

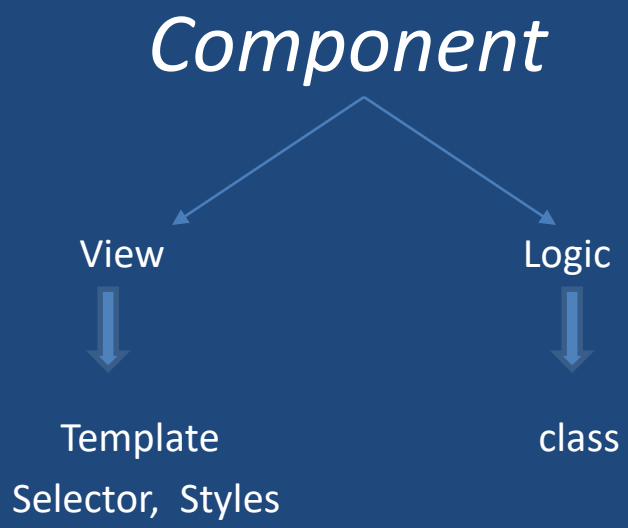
All of these make components in Angular self-describing, so they contain all the information needed to instantiate them.

Components

- In Angular 2, “everything is a component.”
- Components are the main way to build and specify elements and logic on the page
- through both custom elements and attributes that add functionality to our existing components.

Components

- *component* :controls a patch of screen called a *view*



Ex: F:\Demos\Angular2\CLI\MyDemos\my-demo1-app

Components

- A class with component metadata
- Responsible for a piece of the screen referred to as view.
- Template is a form HTML that tells angular how to render the component.
- Metadata tells Angular how to process a class

Components:Templates

Component's template in one of two places:

- *inline using the template property*

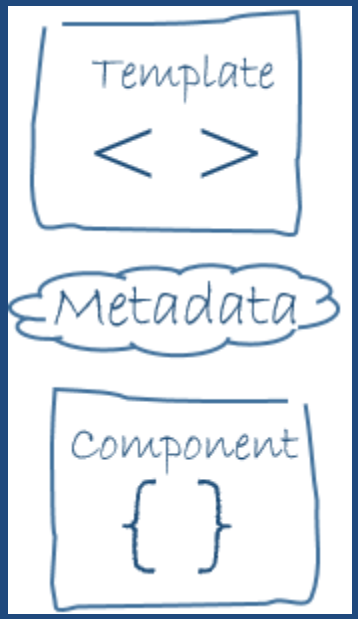
or

- *in a separate HTML file (templateUrl property)*

- The choice between inline and separate HTML is a matter of taste, circumstances, and organization policy.
- In either style, the template data bindings have the same access to component's properties.

@Component

- selector:
- templateUrl:
- providers:



@Input() & @Output

- Input and Output Properties

A component has ***input*** and ***output*** properties, which can be defined:

in the component decorator

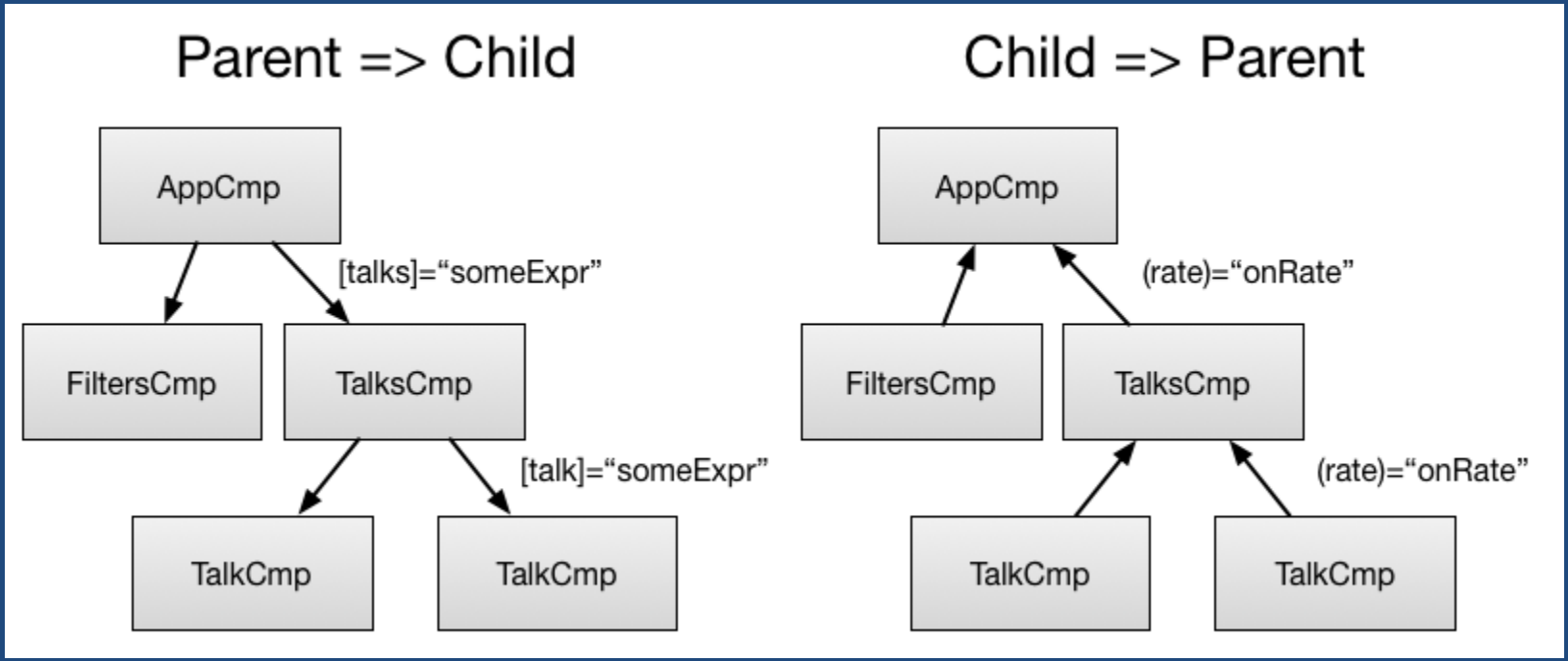
or

using property decorators.

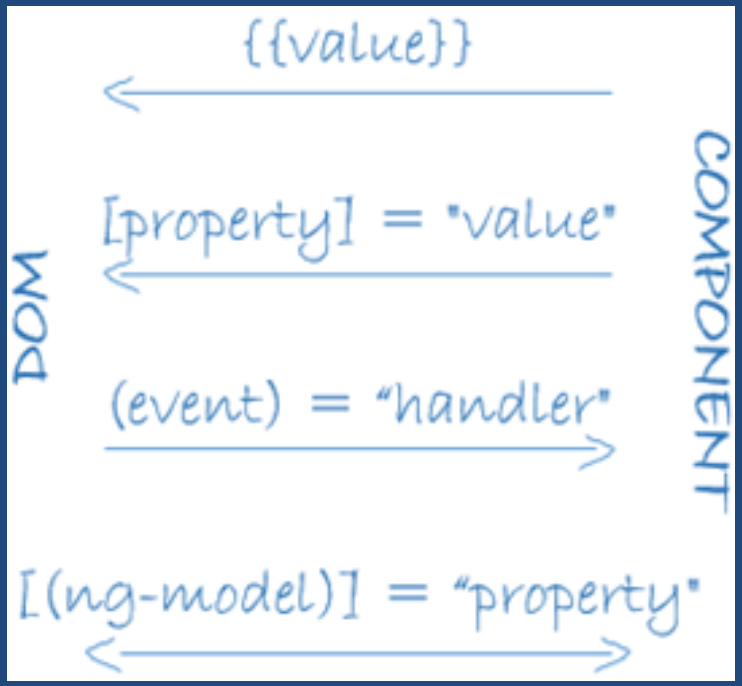
@Component

- Data flows into a component via ***input properties.***
- Data flows out of a component via ***output properties,***
- hence the names: ‘input’ and ‘output’.

@Component


Parent => Child
AppCmp
[talks]="someExpr"
FiltersCmp
TalksCmp
[talk]="someExpr"
TalkCmp
TalkCmp
Child => Parent
AppCmp
(rate)="onRate"
FiltersCmp
TalksCmp
(rate)="onRate"
TalkCmp
TalkCmp

Data Binding



Interpolation ; Property Binding ; Event Binding ; NgModel

Data Binding

- Angular 2 data binding

Angular 2 data binding

C/D	Attribute	Binding type
—>	{{ value }}	one-way
—>	[property] = “value”	property
<—	(event) = “handler”	event
<—>	[(ng-model)] = “property”	two-way

Custom elements

The main advantages of Custom Element are:

- Defining/Creating new HTML/DOM elements
- Create elements that extend from other elements
- Logically bundle together custom functionality into a single tag
- Extend the API of existing DOM elements
- Very useful in Single Page Applications

Component Interactions

- Pass data from parent to child with input binding &
  - Intercept input property changes with a setter
  - Intercept input property changes with ngOnChanges()
- Parent listens for child event
- Parent interacts with child via local variable
- Parent calls an @ViewChild()
- Parent and children communicate via a service

Services

- As the application grows, it will have multiple components.
- These components may require to work on some common data set.
- Instead of copying and pasting the same code over and over, we'll ***create a single reusable data service*** and inject it into the components that need it.

Dependancy Injection

Why @Injectable()?

@Injectable() marks a class as available to an injector for instantiation.

Generally speaking, an injector will report an error when trying to instantiate a class that is not marked as @Injectable().

:recommended: adding @Injectable() to every service class, even though that don't have dependencies and, therefore, do not technically require it. Here's why:

Future proofing: No need to remember @Injectable() when we add a dependency later.

Consistency: All services follow the same rules, and we don't have to wonder why a decorator is missing.

Configuring Injector

We don't have to create an Angular injector.
Angular creates an application-wide injector for us during the bootstrap process.

Registering providers in an NgModule

```
@NgModule({
  imports: [
    BrowserModule
                ],
  declarations: [
    AppComponent,
    CarComponent,
    HeroesComponent,
    /* . . . */
  ],
  providers: [
    UserService,
    { provide: APP_CONFIG, useValue: HERO_DI_CONFIG }
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Configuring Injector

- import { Component } from '@angular/core';
- import { HeroService } from './hero.service';
- @Component({
- selector: 'my-heroes',
- providers: [HeroService],
- template: `
- <h2>Heroes</h2>
- <hero-list></hero-list>
- `
- })
- export class HeroesComponent { }

Configuring Injector

- When to use the NgModule and when an application component?
- A provider in an NgModule is registered in the root injector. That means that every provider registered within an NgModule will be accessible in the entire application.

- Aa provider registered in an application component is available only on that component and all its children.

- We want the APP_CONFIG service to be available all across the application, but a HeroService is only used within the Heroes feature area and nowhere else.

Pipes

app/exponential-strength.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';
/*
 * Raise the value exponentially
 * Takes an exponent argument that defaults to 1.
 * Usage:
 *   value | exponentialStrength:exponent
 * Example:
 *   {{ 2 |  exponentialStrength:10}}
 *   formats to: 1024
*/
@Pipe({name: 'exponentialStrength'})
export class ExponentialStrengthPipe implements PipeTransform {
  transform(value: number, exponent: string): number {
    let exp = parseFloat(exponent);
    return Math.pow(value, isNaN(exp) ? 1 : exp);
  }
}
```

Life Cycle Methods

- Angular calls lifecycle hook methods on directives and components as it creates, changes, and destroys them.
- A component has a lifecycle managed by Angular itself.
- Angular creates it, renders it, creates and renders its children, checks it when its data-bound properties change, and destroys it before removing it from the DOM.
- Angular offers **lifecycle hooks** that provide visibility into these key life moments and the ability to act when they occur.