

Angular JS 2.0



Contents

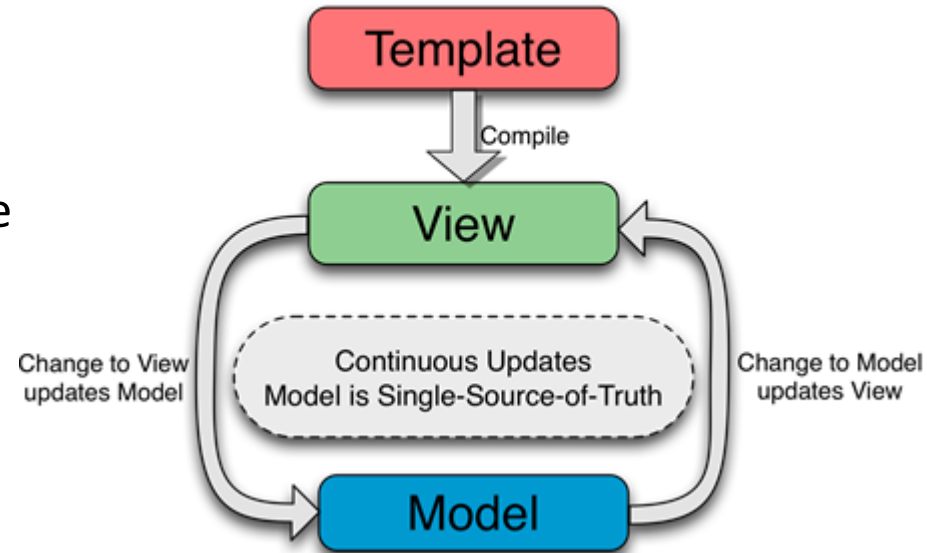


Module	Topic
Module 1	Two Way Binding
Module 2	Directives
Module 3	Inter-Component data transfer
Module 4	Pipes

Two way data binding

- **Two-way binding** means that any **data**-related changes affecting the model are immediately propagated to the matching view(s), and that any changes made in the view(s) (say, by the user) are immediately reflected in the underlying model.

Two-Way Data Binding



Two way data binding in Angular 2



- Two way data binding is by default disabled in Angular 2. However, you can use this feature using ngModel. First you need to import FormsModule in app.module.ts.
- Two way data binding is highly optimized in Angular 2 where it applies on particular component, unlike whole \$scope in Angular 1.x.

```
@Component({  
  selector: 'my-sample',  
  template: `<input type="text" [(ngModel)]="username"> {{username}}`  
}) //ngModel in parenthesis due to event binding & in square brackets due to property binding  
export class SampleComponent {  
  public username: string;  
}
```

- In Angular 2, Directives allows we to attach behavior to DOM elements.
- There are built-in directives to the framework, such as: **NgFor**, **NgIf**, **NgModel**, **NgClass**... and there's an api for creating custom directives.
- Essentially, a directive is something like a component.
- There are three kinds of directives:
 - 1) **Component Directive** – using **@Component()**: Is really a directive with a template.
 - 2) **Structural Directives** – using **@Directive**: Can change the DOM layout by adding and removing DOM elements such as **NgFor**, **NgIf**.
 - 3) **Attribute Directives** – using **@Directive**: doesn't change the DOM but can change the appearance or behavior of an element such as **NgStyle**.

Structural directives are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, or manipulating elements.

There are three built-in structural directives:

- `ngIf`
- `ngSwitch`
- `ngFor`

The ngIf directive conditionally adds or removes content from the DOM based on whether or not an expression is true or false.

```
@Component({  
  selector: 'app-root',  
  template: `

*ngIf="exists"> Hello </p>`  
})  
  
export class AppComponent {  
  exists = true;  
}


```


- ngSwitch is comprised of two directives, an attribute directive and structural directive.
- It's very similar to a switch statement in JavaScript and other programming languages.

```
@Component({
```

```
  selector: 'my-sample',
```

```
  template: `<div [ngSwitch]="color">
```

```
    <p *ngSwitchCase="red">RED</p>
```

```
    <p *ngSwitchCase="green">GREEN</p>
```

```
    <p *ngSwitchDefault>Invalid color</p>
```

```
  </div>` })
```

```
export class SampleComponent { public color = 'green'; }
```


The NgFor directive is a way of repeating a template by using each item of an iterable as that template's context.

```
@Component({  
  selector: 'my-sample',  
  template: `

  
    <li *ngFor="let color of colors">{{color}}</li>  
  </ul>`  
  
  export class SampleComponent {  
    public colors = ['RED', 'GREEN', 'BLUE'];  
  }  
})
```

Attribute Directives

Attribute directive changes the appearance or behavior of a HTML DOM element.

Here are the built-in attribute directives:

➤ *ngClass:*

➤ *ngStyle:*

ngClass Attribute Directive

ngClass attribute directive is used to apply dynamic classes to HTML elements.

```
@Component({  
  selector: 'my-sample',  
  template: `<div [ngClass]="{myClass: class_one}"> This is my division </div>`,  
  styles: ['h4 { color: red }', '.myClass { color: blue; }']  
})  
  
export class SampleComponent {  
  public class_one = true;  
}
```

ngStyle Attribute Directive

- ngStyle directive updates an HTML element styles.
- If you have an inline style instead of css class then you can use ngStyle directive.

```
@Component({  
  selector: 'my-sample',  
  template: `ngStyle...</p>`  
})  
  
export class SampleComponent {  
  public background_color = 'cyan';  
  public style = 'italic';  
}
```

- So far we have seen how to read component property into view & vice versa. Now we are going to discuss how to establish communication between two Angular components & transfer data in between.
- There are two modes to transfer your data across components:
 - 1) Parent to child component
 - 2) Child to parent component

Parent to child component communication



//Parent component

```
@Component({  
  selector: 'my-app',  
  template: `<div class='container'>  
  
<input type="text" #parentTextBox (keyup)="0" >  
  
<my-sample  
[appTextBox]="parentTextBox.value"></my-  
sample>  
  
</div>`  
})  
  
export class AppComponent { }
```

//Child component

```
@Component({  
  selector: 'my-sample',  
  template: `<div>This is child text box  
  
<input type="text" [value]="appTextBox">  
  
</div>`,  
  inputs: ['appTextBox']  
})  
  
export class SampleComponent {  
  public appTextBox: string;  
}
```

Child to parent component communication

//Child component

```
@Component({  selector: 'my-sample',  
  template: `<div>This is child text box  
<input type="text" #childText  
(keyup)="onChange(childText.value)"> </div>`,  
  outputs: ['childEvent']  
})  
  
export class SampleComponent {  
  public childEvent = new EventEmitter<string>();  
  onChange(value: string) {  
    this.childEvent.emit(value);  }}
```

//Parent component

```
@Component({  
  selector: 'my-app',  
  template: `<div class='container'>  
    {{childData}} <my-sample  
    (childEvent)="childData=$event"></my-sample>  
  </div>`  
})  
  
export class AppComponent {  
  public childData: string;  
}
```


- Pipes are used to transform expressions into desired format.
- Angular 2 pipes is same as Angular 1 filters.
- Angular 2 provides several built-in pipes & also you can create your own custom pipes.
- Built-in pipes include:

Async	Currency	Date
Decimal	Json	Percent
Slice	LowerCase	UpperCase

Async & Currency pipes



➤ Async

The Async pipe automatically subscribes to an Observable or a Promise and returns the emitted values as they come in.

```
<ul>
```

```
  <li *ngFor="let item of data | async"> {{ item.name }} 
```

```
  </li>
```

```
</ul>
```

➤ Currency

The Currency pipe allows to format numbers in different currencies.

```
{{ 21.45 | currency:'CAD' }}           // CAD 21.45
```

```
{{21.45 | currency:'USD':true }}        // $21.45
```

```
{{21.45 | currency:'EUR':false:'2.3-4' }} // EUR 21.450
```

➤ Date

The Date pipe formats the supplied date.

```
{{ someDate | date:'medium' }}    // May 23, 2017, 9:57:09 AM
```

```
{{ someDate | date:'fullDate' }}   // Tuesday, May 23, 2017
```

```
{{ someDate | date:'yy' }}         // 17
```

```
{{ someDate | date:'Hm' }}         // 0957
```

For more info, refer: <https://angular.io/docs/ts/latest/api/common/index/DatePipe-pipe.html>

➤ Decimal

The Decimal pipe formats decimal values.

```
{{ 21.45 | number:'4.3-5' }}    // 0,021.450
```

In the above example ('4.3-5'), 4 is for the minimum number of integer digits, 3 is for the minimum number of fraction digits and 5 is for the maximum number of fraction digits.

Json & Lowercase/Uppercase pipes



➤ Json

The Json pipe is useful for debugging and displays an object as a Json string. It uses JSON.stringify behind the scenes.

```
{{ someObject | json }} // { "id": 111, "title": "Welcome to Angular 2" }
```

➤ LowerCase & UpperCase

Covert text to either lower case or upper case with the respective pipe.

```
{{ 'Angular' | uppercase }} // ANGULAR
```

```
{{ 'Angular' | lowercase }} // angular
```

Percent & Slice pipes

➤ Percent

The Percent pipe transforms a number into it's percentage value.

```
{{ 0.01 | percent }}           // 1%
```

```
{{ 0.01 | percent:'3.2-3' }}   // 001.00%
```

➤ Slice

Slice pipe creates a subset list or string.

```
{{ 'Angular' | slice:3:6 }}     // ula
```

```
<ul>
```

```
  <li *ngFor="let item of [12, 44, 2, 61] | slice:2:4"> {{ item }} // 2, 61
```

```
  </li>
```

```
</ul>
```

- Angular 2 allows us to create our own Pipes. It is called as 'custom pipe'.
- Custom pipe is a class which uses decorator @Pipe.
- Custom pipe class must implements an interface 'PipeTransform' which has a single method 'transform':

```
export interface PipeTransform {  
    transform(value: any, ...args: any[]): any;  
}
```

Custom Pipe example

Program to create a custom pipe 'even' that filters only even numbers from an array.

```
import { Pipe, PipeTransform } from '@angular/core';
```

```
@Pipe({name: 'even'})
```

```
export class EvenNumberPipe implements PipeTransform {
```

```
  transform(value: Array<number>): Array<number> {
```

```
    let evenArray = [];
```

```
    for(let item of value) {
```

```
      if (item % 2 === 0) {    evenArray.push(item);} 
```

```
  }
```

```
  return evenArray;
```

```
}
```

```
}
```

Usage:

```
{{ [12, 55, 81, 34, 522] | even }} //12, 34, 522
```


Thank You!

US – Corporate Headquarters

1248 Reamwood Avenue,
Sunnyvale, CA 94089
Phone: (408) 743 4400

343 Thornall St 720
Edison, NJ 08837
Phone: (732) 395 6900

UK

20 Broadwick Street
Soho, London
W1F 8HT, UK

89 Worship Street
Shoreditch,
London EC2A 2BF, UK
Phone: (44) 2079 938 955

India

Mumbai
4th Floor, Nomura
Powai , Mumbai 400 076

Pune
5th Floor, Amar Paradigm
Baner, Pune 411 045

Kolkata
2B, 12th Floor, Tower 'C'
Rajarhat, Kolkata 700 156

Bangalore
4th Floor, Kabra Excelsior,
80 Feet Main Road,
Koramangala 1st Block,
Bengaluru (Bangalore) 560034

Gurgaon
A/373rd Floor, Sigma Center
Gurgaon, Haryana 122 011s