BUILDING COMPONENTS

Ori Calvo, 2017

oric@trainologic.com

https://trainologic.com

Objectives

- Practical information for building real life components
- Template syntax
- Useful directives
- Component interactions

Component

- Controls a fragment of a screen. For example,
  - Side bar
  - Dashboard
  - User details
- Consists of
  - Template
  - Metadata
  - Class
  - Styles

Data Binding

- Special "instructions" inside a template
- Mechanism for synchronizing template with component state
- 4 types of data bindings
  - Interpolation
  - Property binding
  - Event binding
  - Two way data binding

Interpolation

- Display a text inside an HTML element

```
<div class="title">{{title}}</div>
```

- The title property should be of type string
  - Can be null → Angular displays nothing
- Can use complex expression

```
<div class="title">{{contact.name}}</div>
```

- However, be aware of null reference exceptions

Template Expression

- A subset of JavaScript syntax supplemented with a few special operators
- The context is the component
  - title → component.title
- Side effects are prohibited
  - Assignments/new/increment/bitwise
- Supports special operators
  - | ?. !.
- Supports local variables

Safe Navigation Operator .?

- Complex template expression might raise null exception
  - For example, contact.name when contact is null
- Solution, use the ?. special operator
- For example, contact?.name
- Produces empty string in case contact is null

Pipe operator |

- Angular uses toString when interpolating non string value

```
<div class="title">{{obj}}</div>
```

- For plain object it produces [object Object] and for date it produces an ISO date string
- Pipe fixes that (like AngularJS filter)

```
<div class="title">{{birthday | date}}</div>
```

- Can customize the pipe

```
<div class="title">{{birthday | date: 'HH:mm:ss'}}</div>
```

Built-in pipes

date	i18nSelect	json
async	currency	number
i18nPlural	slice	uppercase
titlecase	percent	lowecase

Chaining Pipes

- One pipe's output can be set as an input for another pipe

```
<div class="title">{{birthday | date: 'fullDate' | uppercase}}</div>
```

- Produces the following text

```
FRIDAY, SEPTEMBER 8, 2017
```

Custom Pipe

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

@Pipe({
  name: 'x'
})
export class XPipe implements PipeTransform {
  transform(value: any, args?: any): any {
    return "X" + (value || "") + "X";
  }
}
```

Value from template

Additional parameters

Expression Guidelines

- Expression is executed at every dirty checking cycle
- Therefore is adhere to
  - No side effect
  - Quick execution
  - Simplicity
  - idempotence

Property Binding

- Bind a DOM property component field

```
<button [disabled]="!enabled">Click me</button>
```

- This is a one way data binding
- Alternative syntax (less common)

```
<button bind-disabled="!enabled">Click me</button>
```

- Can also bind to a DOM attribute (string only)

```
<input value="{{title}}">
```

Property Binding

- What happen if you forget the brackets ?

```
<button disabled="!enabled">Click me</button>
```

- Angular treats the string as a constant
- Initializes the target property with the string value
- It does not evaluate the string
- This is one time string initialization
- Probably not what you want

Sanitization

- Angular data binding prevents dangerous values
- By always escaping the assigned value

```
<p>{{title}}</p>
```

- In case title is assigned <h1>Hello</h1>
- The whole text is escaped and displayed as is
- Use [innerHTML] to prevent sanitization
  - Angular still removes dangerous code (script tags)

Attribute binding

- In some cases you want to bind to an attribute that does not have a corresponding property
  - ARIA
  - colspan
- Using the property binding syntax generates error
- Solution, use the special attr syntax

```
<td [attr.colspan]="colspan">1</td>
```

Class binding

- Binding to class attribute resets all classes

```
<div [class]="cls" class="red">Yo yo</div>
```

- Instead, use the special class syntax

```
<div [class.small]="isSmall" class="red">Yo yo</div>
```

- isSmall is expected to be of noolean type

ngClass directive

- Bind to multiple CSS classes using multiple Boolean flags

```
<div [ngClass]="{big: isBig, red: isRed}">Yo yo</div>
```

- Can bind directly to the map object

```
<div [ngClass]="classes">Yo yo</div>
```

Style binding

- Bind to specific element style

```
<div [style.color]="color" [style.font-size.em]="fontSize">Yo yo</div>
```

- Better, use [ngStyle] directive

```
<div [ngStyle]="{color: color, fontSize: fontSizeEm}">Yo yo</div>
```

Event Binding

- Consists of target event name + template statement

```
<button (click)="change()">Change</button>
```

- Alternative syntax

```
<button on-click="change()">Change</button>
```

- The statement may have side effects

```
<button on-click="text = 'ZZZ'">Change</button>
```

$event

- Get access to the DOM event object

```
<button (click)="change($event)">Change</button>
```

- Inside the component class you can use any parameter name

```
class AppComponent {
  change($event) {
    console.log("change", $event);
  }
}
```

- The this context is the component not the DOM element
  - Use $event.target instead

Two Way Binding

- You write

```
<input [(value)]="name">
```

- Angular transforms that syntax into

```
<input [value]="name" (valueChange)="name=$event">
```

- Since input element has no valueChange event the two way data binding does not work
  - No error/warning is reported

ngModel

- A directive that supports two way binding with input/textarea element

```
<input [(ngModel)]="name">
```

- Don't forget to import the FormsModule

```
@NgModule({
   imports: [
      BrowserModule,
      FormsModule,
   ],
})
export class AppModule {
}
```

Built-in Structural Directives

- Structural directive reshapes the DOM structure
- By adding/removing/manipulates DOM element
- ngIf
- ngFor
- ngSwitch

ngIf

- Conditionally adds/removes components

```
<div *ngIf="contact">Hello, {{contact.name}}</div>
```

- Not the same as show/hide

```
<div [style.display]="show ? 'block' : 'none'"></div>
```

- Using ngIf is considered more efficient with respect to resource consumption
- However, is more challenging when integrating animations

ngSwitch

```
<div>
   <button (click)="smaller()">-</button>
   <button (click)="larger()">+</button>
</div>

<div [ngSwitch]="page">
   <div    *ngSwitchCase="'Small'">Small</div>
   <div    *ngSwitchCase="'Normal'">Normal</div>
   <div    *ngSwitchCase="'Large'">Large</div>
   <div  *ngSwitchDefault>Default</div>
</div>
```

ngFor

- A repeater directive

```
<ul>
  <li *ngFor="let contact of contacts">
    <span>{{contact.name}}</span>
  </li>
</ul>
```

- The expression assigned to ngFor is not a template expression but rather an Angular microsyntax

ngFor index

```
<ul>
  <li *ngFor="let contact of contacts; let index=index">
    <span>{{contact.name}}</span>
    <button (click)="remove(index)">Delete</button>
  </li>
</ul>
```

```
class AppComponent {
  contacts: Contact[];

  constructor() {
    this.contacts = [...];
  }

  remove(index: number) {
    this.contacts.splice(index, 1);
  }
}
```

Template Reference Variable

- A parent component may request direct access to a DOM element/child component/directive instance

```
<video #myVideo></video>
```

```
export class AppComponent {
   @ViewChild("myVideo") video;

   ngOnInit() {
      console.log(this.video);
   }
}
```

Bound to Components

- Up until now we bound to DOM element properties and events
- The same syntax can be used with components/directives
- You must use the @Input/@Output syntax

Component Property

```
export class ClockComponent {
    @Input() format: string = "HH:mm:ss";

    time: Date;

    constructor() {
        this.time = new Date();
    }
}
```

```
<app-clock format="HH:mm"></app-clock>

<app-clock [format]="format"></app-clock>
```

```
<span>{{time | date: format}}</span>
```

Component Event

```
export class ClockComponent {
    time: Date;
    @Output() tick: EventEmitter<Date> = new EventEmitter<Date>();
    private intervalId;

    ngOnInit() {
        this.intervalId = setInterval(()=> {
            this.time = new Date();

            this.tick.emit(this.time);
        }, 1000);
    }
}
```

```
export class AppComponent {
    onTick(time: Date) {
        console.log("onTick", time);
    }
}
```

```
<app-clock (tick)="onTick($event)"></app-clock>
```

Aliasing Input/Output

- You can differentiate between component internal and public name

```
export class ClockComponent {
   @Input("xxx") format: string = "HH:mm:ss";

   @Output("t") tick: EventEmitter<Date> = new EventEmitter<Date>();
}
```

```
@Component({
   …
   inputs: ['format: xxx'],
   outputs: ['tick: t']
})
export class ClockComponent {
   format: string = "HH:mm:ss";

   tick: EventEmitter<Date> = new EventEmitter<Date>();
}
```

Summary

- Angular has nice template syntax
  - Some consider it a bit tricky
- This is the core of Angular data binding
  - [src]
  - (click)
  - [(ngModel)]