Interpolation in Angular

4 Comments / 6 minutes of reading / March 9, 2023

← Data Binding | Angular Tutorial | Property Binding →

In this guide let us learn the interpolation in Angular with examples. We use interpolation to bind a component property, method or to a template reference variable to a string literal in the view. We also call this as ***string interpolation***. It is one way from component to view as the data flow from the component to view.

Table of Contents

What is Interpolation in Angular

Interpolation allows us to include expressions as part of any string literal, which we use in our HTML. The angular evaluates the expressions into a string and replaces it in the original string and updates the view. You can use interpolation wherever you use a string literal in the view.

Angular interpolation is also known by the name ***string interpolation***. Because you incorporate expressions inside another string.

Interpolation syntax

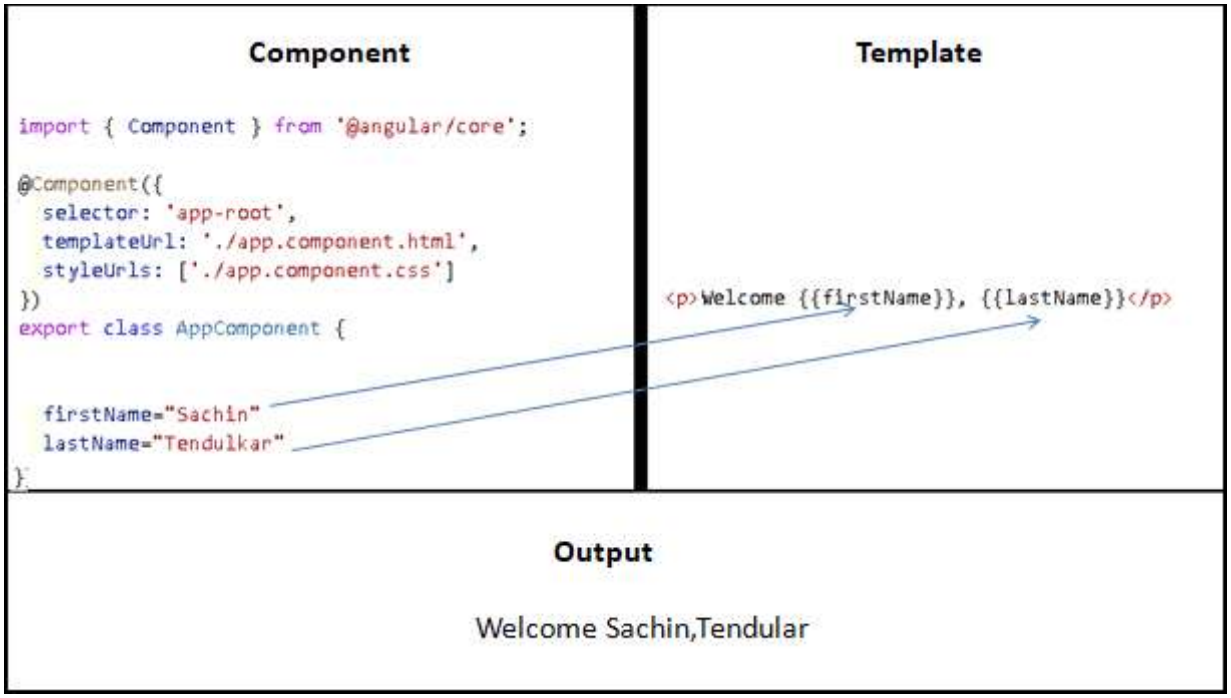
The Angular uses the `{{ }}` (double curly braces) in the template to denote the interpolation. The syntax is as shown below

```
1
2 {{ templateExpression }}
3
```

The content inside the double braces is called `Template Expression`

The Angular first evaluates the `Template Expression` and converts it into a string. Then it replaces `Template expression` with the result in the original string in the HTML.

Whenever the template expression changes, the Angular updates the original string again



Interpolation Example

Create a new angular application using the following command

```
ng new interpolation
```

Open the app.component.html and just add the following code

```
{{title}}
```

Open the app.component.ts and add the following code

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular Interpolation Example';
}
```

Run the app. You will see the "Angular Interpolation Example" on the screen

In the example above the title is the Template Expression. We also have title property in the component. The Angular evaluates {{title}} and replaces it with the value of the title property from the component class.

If the user changes the title in the component class, the Angular updates the view accordingly.

The interpolation is much more powerful than just getting the property of the component. You can use it to invoke any method on the component class or to do some mathematical operations etc.

Notes on Interpolation

Interpolation is one-way binding

Interpolation is one way as values go from the component to the template. When the component values change, the Angular updates the view. But if the values changes in the view components are not updated.

Should not change the state of the app

The Template expression should not change the state of the application. The Angular uses it to read the values from the component and populate the view. If the Template expression changes the component values, then the rendered view would be inconsistent with the model

It means that you cannot make use of the following

- Assignments (=, +=, -=, ...)
- Keywords like new , typeof , instanceof , etc
- Chaining expressions with ; or ,
- The increment and decrement operators ++ and --
- bitwise operators such as | and &

The expression must result in a string

Interpolation expression must result in a string. If we return an object it will not work. If you want to bind the expression that is other than a string (for example – boolean),

then Property Binding is the best option.

Works only on Properties & not attributes

Interpolation and property binding can set only properties, not attributes. For Attributes use attribute binding

Examples of interpolation

You can use interpolation to invoke a method in the component, Concatenate two string, perform some mathematical operations or change the property of the DOM element like color, etc.

Invoke a method in the component

We can invoke the component's methods using interpolation.

```
1
2  //Template
3
4  {{getTitle()}}
5
6
7  //Component
8  title = 'Angular Interpolation Example';
9  getTitle(): string {
10     return this.title;
11  }
12
```

Concatenate two string

```
<p>Welcome to {{title}}</p>
<p>{{ 'Hello & Welcome to '+ ' Angular Interpolation '}}</p>
<p>Welcome {{firstName}}, {{lastName}}</p>
<p>Welcome {{getFirstName()}}, {{getLastName()}}</p>
```

Perform some mathematical operations

```
<h2>Mathematical Operations</h2>

<p>100x80 = {{100*80}}</p>
<p>Largest: {{max(100, 200)}}</p>

//Component
max(first: number, second: number): number {
  return Math.max(first, second);
}
```

Bind to an element property

We can use it to bind to a property of the HTML element, a component, or a directive. in the following code, we bind to the `style.color` property of the `<p>` element. We can bind to any property that accepts a string.

```
<p>Show me <span class = "{{giveMeRed}}">red</span></p>
<p style.color={{giveMeRed}}>This is red</p>
```

```
4
```

Bind to an image source

```
1
2 <div><img src="{{itemImageUrl}}"></div>
3
```

href

```
1
2 <a href="/product/{{productID}}">{{productName}}</a>
3
```

Use a template reference variable

You can also use the template reference variable. The following example creates a template variable `#name` to an input box. You can use it get the value of the input field `{{name.value}}`

```
1
2 <label>Enter Your Name</label>
3 <input (keyup)="0" #name>
4 <p>Welcome {{name.value}} </p>
5
```

We also use `(keyup)="0"` on the `input` element. It does nothing but it forces the angular run the change detection, which in turn updates the view.

> The Angular updates the view, when it runs the change detection. The change detection runs only in response to asynchronous events, such as the arrival of HTTP responses, raising of events, etc. In the example above whenever you type

on the input box, it raises the `keyup` event. It forces the angular run the change detection, hence the view gets the latest values.

```
1
2  //Template
3
4  <p>items</p>
5  <ul>
6    <li *ngFor="let d of items">
7      {{ d.name }}
8      </li>
9  </ul>
10
11 //Component
12   items= [
13     new item(1, 'Mobile'),
14     new item(2, 'Laptop'),
15     new item(3, 'Desktop'),
16     new item(4, 'Printer')
17   ]
18
19 class item {
20   code:string
21   name:string
22
23   constructor(code,name) {
24     this.code=code;
25     this.name=name
26   }
27 }
28
29
```

Cross-site Scripting or XSS

Angular Sanitizes everything before inserting into DOM, thus preventing Cross-Site Scripting Security bugs (XSS). For example values from the component property, attribute, style, class binding, or interpolation, etc are sanitized. The `script` in the following example is not invoked but shown as it is.

```
1
2 //Template
3 <p>{{script}}</p>
4 <p>{{div}}</p>
5
6 //Component
7 script ='<script>alert("You are hacked")</script>'
8 div='<div>this is a div</div>';
9
```

NgNonBindable

Use ngNonBindable to tell Angular not to compile or bind the contents of the current DOM element. I.e any expression is not evaluated but shown as it is.

```
1
2 <p>Evaluate: {{variable}}</p>
3 <p ngNonBindable>Do not evaluate: {{variable}}</p>
4
5
6 <p>Angular uses {{ variable }} syntax for Interpolation</p>
7 <p ngNonBindable>Angular uses {{ variable }} syntax for Interpolation</p>
8
```

Use Pipes

You can make use of Angular Pipes to transform the expression result. Like converting to an uppercase, date formats, adding currency symbols, etc

```
<p>uppercase pipe: {{title | uppercase}}</p>
<p>pipe chain: {{title | uppercase | lowercase}}</p>
<p>json pipe: {{items | json}}</p>
```

The safe navigation operator (?)

You can make use of a safe navigation operator (?) to guards against `null` and `undefined` values.

The following code results in an error because there is no `nullItem`

```
<p>The item name is: {{nullItem.Name}}</p>

TypeError: Cannot read property 'itemName' of undefined
```

Use a safe navigation operator and the error goes away. Angular replace it with an empty string

```
<p>The null item name is {{nullItem?.itemName}}</p>
```

The non-null assertion operator

Typescript enforces the strict null checking if you enable the `--strictNullChecks` flag in your `tsconfig.json` . Under strict null check any variable not defined or null results in a compiler error. The type checker also throws an error if it can't determine whether a variable will be null or undefined at runtime