When to use square brackets [ ] in directives @Inputs and when not?

Asked 2 years, 4 months ago Active 1 month ago Viewed 18k times

40

9

I'm confused a little.

See this simple directive:

```
@Directive({
     selector: '[myDirective]'
   })
   export class MyDirective {

     private text: string;
     private enabled: boolean;

     @Input() myDirective:string;

     @Input('myText')
     set myText(val: string) {
       this.text = val;
     }

     @Input('myEnabled')
     set myEnabled(val: boolean) {
       this.enabled = val;
     }

     ngOnInit() {

       console.log("myDirective string: " + this.myDirective);
       console.log("myText string: " + this.text);
       console.log("myEnabled boolean: " + this.enabled);
   }
}
```

if my html will look like this:

```
<div [myDirective]="myDefaultText" [myEnabled]="true"  [myText]="abc"></div>
```

The output will be:

```
myDirective string: myDefaultText real value  // good
myEnabled boolean: true                       // good
myText string: undefined                      // Why?
```

If I REMOVE the [] from `myText` :

```
<div [myDirective]="myDefaultText" [myEnabled]="true"  myText="abc"></div>
```

The output will be:

```
myDirective string: myDefaultText real value  // good
myEnabled boolean: true                       // good
myText string: abc                            // GOOD
```

I can also remove the `[]` from `myEnabled` and it will work too. So here is my confusion - when I need to use square brackets `[]` and when not, while I want the user who is going to use `myDirective` will never need to wonder if or if not, I think the square brackets `[]` should always be there. Aren't they?

angular typescript angular2-directives

edited Apr 26 '17 at 12:16

Roman C
44.8k 15 47 121

asked Apr 26 '17 at 11:53

AngularOne
903 5 17 37

4 Answers

28

When you use `[]` to bind to an `@Input()` , it's basically a template expression.

The same way displaying `{{abc}}` wouldn't display anything (unless you actually had a variable called `abc`).

If you have a string `@Input()` , and you want to bind it to a constant string, you could bind it like this: `[myText]=" 'some text' "` , or in short, like a normal HTML attribute: `myText="some text"` .

The reason `[myEnabled]="true"` worked is because `true` is a valid template expression which of course evaluates to the boolean `true` .

answered Apr 26 '17 at 12:00

Amit
2,656 13 22

1 I didn't even think that's what's actually happening, are you sure? The asker mentioned that `[myDirective]="myDefaultText"` gave the input "myDefaultText real **value**", I though he meant the actual value of `myDefaultText` , which means it's still treated as an expression. – Amit May 20 '17 at 17:09

1 I was wrong. Missed the " real value". – Royi Namir May 20 '17 at 18:23

7 This is covered in the current Angular 4 documentation on this page. – 2Aguy Oct 5 '17 at 13:55

9

The brackets tell Angular to evaluate the template expression. If you omit the brackets, Angular treats the string as a constant and initializes the target property with that string. It does not evaluate the string!

Don't make the following mistake:

```
<!-- ERROR: HeroDetailComponent.hero expects a
     Hero object, not the string "currentHero" -->
<hero-detail hero="currentHero"></hero-detail>
```

check: https://angular.io/docs/ts/latest/guide/template-syntax.html#!#property-binding

answered Apr 26 '17 at 12:03

sainu
1,734 1 15 30

6

I think I understand where your confusion is coming from. When you say `[myText]="abc"` you are expecting `myText` is a property defined in my component whose value I want to initialize to `abc` . But this is not correct. But first, let's understand a little more about HTML.

In HTML you can define an element like this.

```
<input type="text" value="Bob">
```

input is an element whose `attributes` are type and value. When your browser parses this, it will create a DOM entry (an object) for this element. The DOM entry will have some `properties` like align, baseURI, childNodes, children etc. So, that's the difference between

HTML attributes and DOM properties See reference. Sometimes the attribute and property have same names which causes confusion. For above input tag, it has the attribute `value` = Bob and also has a property `value` that will have the value of whatever you type in the text box. In summary, attribute is what you define about the tag, and property is what gets generated in the DOM tree.

The reason why you need to know this is because, in the world of Angular, the only role of attributes is to initialize element and directive state. When you write a data binding, you're dealing exclusively with properties and events of the target object. HTML attributes effectively disappear.

All that means is that if you write `<img [src]="heroImageUrl">` it means that `src` is NOT an attribute, but is a `property` defined inside the DOM of img. And the right-hand side `heroImageUrl` is a template expression.

The simple difference between `[myText]="abc"` and `myText="abc"` is that in former you are asking angular to set the PROPERTY myText, where in latter you are creating an ATTRIBUTE called myText, and this attribute will have its own DOM property. Angular does not deal with attributes.

So to summarize, in `<div [myDirective]="myDefaultText" [myEnabled]="true"  [myText]="abc"></div>` you essentially are saying that:

1. apply the directive `myDirective` to my div element.
2. bind the variable `myEnabled` to the expression on the right. The expression says `true` , so the value of myEnabled is true.
3. bind the variable `myText` to the expression on the right. The expression says `abc` . Is there any abc defined? No, so the expression evaluated to undefined.

edited Jul 31 at 1:23

answered Jan 15 at 19:22

Rash

4,084 1 30 50

This is the only answer that actually explains the reason rather than just showing some side effects. Kudos. – Nathaniel Johnson Aug 29 at 18:28

5

binding `[]` is for objects, without it the value is string. Be careful about types.

In the code

```
<div [myDirective]="myDefaultText" [myEnabled]="true"  [myText]="abc"></div>
```

you have tried to bind the object, but the object is not available, thus it's value is `undefined` . On the other hand if you remove binding then the object is gone, you have only a `string` value assigned to the property.

answered Apr 26 '17 at 12:11

Roman C
44.8k 15 47 121

2 Make sure not to do `myEnabled="false"` because that will be evaluated as a string and since `"false"` is actually a truthy value it won't behave as you expected. – Simon_Weaver Feb 9 '18 at 20:13

Where did you see that? – Roman C Feb 10 '18 at 13:07