Attribute, class, and style bindings

Contents ›

The template syntax provides specialized one-way bindings for scenarios less well-suited to property binding.

> See the live example / download example for a working example containing the code snippets in this guide.

Attribute binding

Set the value of an attribute directly with an **attribute binding**. This is the only exception to the rule that a binding sets a target property and the only binding that creates and sets an attribute.

Usually, setting an element property with a property binding is preferable to setting the attribute with a string. However, sometimes there is no element property to bind, so attribute binding is the solution.

Consider the ARIA and SVG. They are purely attributes, don't correspond to element properties, and don't set element properties. In these cases, there are no property targets to bind to.

Attribute binding syntax resembles property binding, but instead of an element property between brackets, start with the prefix `attr`, followed by a dot (`.`), and the name of the attribute. You then set the attribute value, using an expression that resolves to a string, or remove the attribute when the expression resolves to `null`.

One of the primary use cases for attribute binding is to set ARIA attributes, as in this example:

src/app/app.component.html

```
<!-- create and set an aria attribute for assistive technology -->
<button [attr.aria-label]="actionName">{{actionName}} with Aria</button>
```

> ### `colspan` and `colSpan`
>
> Notice the difference between the `colspan` attribute and the `colSpan` property.

If you wrote something like this:

```
<tr><td colspan="{{1 + 1}}">Three-Four</td></tr>
```

You'd get this error:

```
Template parse errors:
Can't bind to 'colspan' since it isn't a known native property
```

As the message says, the `<td>` element does not have a `colspan` property. This is true because `colspan` is an attribute—`colSpan`, with a capital `S`, is the corresponding property. Interpolation and property binding can set only *properties*, not attributes.

Instead, you'd use property binding and write it like this:

src/app/app.component.html

```
<!-- Notice the colSpan property is camel case -->
<tr><td [colSpan]="1 + 1">Three-Four</td></tr>
```

Class binding

Here's how to set the `class` attribute without a binding in plain HTML:

```
<!-- standard class attribute setting -->
<div class="foo bar">Some text</div>
```

You can also add and remove CSS class names from an element's `class` attribute with a **class binding**.

To create a single class binding, start with the prefix `class` followed by a dot (`.`) and the name of the CSS class (for example, `[class.foo]="hasFoo"`). Angular adds the class when the bound expression is truthy, and it removes the class when the expression is falsy (with the exception of `undefined`, see styling delegation).

To create a binding to multiple classes, use a generic `[class]` binding without the dot (for example, `[class]="classExpr"`). The expression can be a space-delimited string of class names, or you can format it as an object with class names as the keys and truthy/falsy expressions as the values. With object format, Angular will add a class only if its associated value is truthy.

It's important to note that with any object-like expression (`object`, `Array`, `Map`, `Set`, etc), the identity of the object must change for the class list to be updated. Updating the property without changing object identity will have no effect.

If there are multiple bindings to the same class name, conflicts are resolved using styling precedence.

Binding Type	Syntax	Input Type	Example Input Values
Single class binding	`[class.foo]="hasFoo"`	`boolean | undefined | null`	`true`, `false`
Multi-class binding	`[class]="classExpr"`	`string`	`"my-class-1 my-class-2 my-class-3"`
		`{[key: string]: boolean | undefined | null}`	`{foo: true, bar: false}`
		`Array<string>`	`['foo', 'bar']`

The NgClass directive can be used as an alternative to direct `[class]` bindings. However, using the above class binding syntax without `NgClass` is preferred because due to improvements in class binding in Angular, `NgClass` no longer provides significant value, and might eventually be removed in the future.

Style binding

Here's how to set the `style` attribute without a binding in plain HTML:

```
<!-- standard style attribute setting -->
<div style="color: blue">Some text</div>
```

You can also set styles dynamically with a **style binding**.

To create a single style binding, start with the prefix `style` followed by a dot (`.`) and the name of the CSS style property (for example, `[style.width]="width"`). The property will be set to the value of the bound expression, which is normally a string. Optionally, you can add a unit extension like `em` or `%`, which requires a number type.

> Note that a *style property* name can be written in either dash-case, as shown above, or camelCase, such as `fontSize`.

If there are multiple styles you'd like to toggle, you can bind to the `[style]` property directly without the dot (for example, `[style]="styleExpr"`). The expression attached to the `[style]` binding is most often a string list of styles like `"width: 100px; height: 100px;"`.

You can also format the expression as an object with style names as the keys and style values as the values, like `{width: '100px', height: '100px'}`. It's important to note that with any object-like expression (`object`,

`Array`, `Map`, `Set`, etc), the identity of the object must change for the class list to be updated. Updating the property without changing object identity will have no effect.

If there are multiple bindings to the same style property, conflicts are resolved using styling precedence rules.

Binding Type	Syntax	Input Type	Example Input Values
Single style binding	`[style.width]="width"`	`string | undefined | null`	`"100px"`
Single style binding with units	`[style.width.px]="width"`	`number | undefined | null`	`100`
Multi-style binding	`[style]="styleExpr"`	`string`	`"width: 100px; height: 100px"`
		`{[key: string]: string | undefined | null}`	`{width: '100px', height: '100px'}`
		`Array<string>`	`['width', '100px']`

The NgStyle directive can be used as an alternative to direct `[style]` bindings. However, using the above style binding syntax without `NgStyle` is preferred because due to improvements in style binding in Angular, `NgStyle` no longer provides significant value, and might eventually be removed in the future.

Styling Precedence

A single HTML element can have its CSS class list and style values bound to multiple sources (for example, host bindings from multiple directives).

When there are multiple bindings to the same class name or style property, Angular uses a set of precedence rules to resolve conflicts and determine which classes or styles are ultimately applied to the element.

Styling precedence (highest to lowest)

1. Template bindings
    a. Property binding (for example, `<div [class.foo]="hasFoo">` or `<div [style.color]="color">`)

b. Map binding (for example, `<div [class]="classExpr">` or `<div [style]="styleExpr">`)

c. Static value (for example, `<div class="foo">` or `<div style="color: blue">`)

2. Directive host bindings

a. Property binding (for example, `host: {'[class.foo]': 'hasFoo'}` or `host: {'[style.color]': 'color'}`)

b. Map binding (for example, `host: {'[class]': 'classExpr'}` or `host: {'[style]': 'styleExpr'}`)

c. Static value (for example, `host: {'class': 'foo'}` or `host: {'style': 'color: blue'}`)

3. Component host bindings

a. Property binding (for example, `host: {'[class.foo]': 'hasFoo'}` or `host: {'[style.color]': 'color'}`)

b. Map binding (for example, `host: {'[class]': 'classExpr'}` or `host: {'[style]': 'styleExpr'}`)

c. Static value (for example, `host: {'class': 'foo'}` or `host: {'style': 'color: blue'}`)

The more specific a class or style binding is, the higher its precedence.

A binding to a specific class (for example, `[class.foo]`) will take precedence over a generic `[class]` binding, and a binding to a specific style (for example, `[style.bar]`) will take precedence over a generic `[style]` binding.

src/app/app.component.html

```
<h3>Basic specificity</h3>

<!-- The `class.special` binding will override any value for the `special` class in
`classExpr`.  -->
<div [class.special]="isSpecial" [class]="classExpr">Some text.</div>

<!-- The `style.color` binding will override any value for the `color` property in
`styleExpr`.  -->
<div [style.color]="color" [style]="styleExpr">Some text.</div>
```

Specificity rules also apply when it comes to bindings that originate from different sources. It's possible for an element to have bindings in the template where it's declared, from host bindings on matched directives, and from host bindings on matched components.

Template bindings are the most specific because they apply to the element directly and exclusively, so they have the highest precedence.

Directive host bindings are considered less specific because directives can be used in multiple locations, so they have a lower precedence than template bindings.

Directives often augment component behavior, so host bindings from components have the lowest precedence.

src/app/app.component.html

```
<h3>Source specificity</h3>

<!-- The `class.special` template binding will override any host binding to the
`special` class set by `dirWithClassBinding` or `comp-with-host-binding`.-->
<comp-with-host-binding [class.special]="isSpecial" dirWithClassBinding>Some text.
</comp-with-host-binding>

<!-- The `style.color` template binding will override any host binding to the `color`
property set by `dirWithStyleBinding` or `comp-with-host-binding`. -->
<comp-with-host-binding [style.color]="color" dirWithStyleBinding>Some text.</comp-
with-host-binding>
```

In addition, bindings take precedence over static attributes.

In the following case, `class` and `[class]` have similar specificity, but the `[class]` binding will take precedence because it is dynamic.

src/app/app.component.html

```
<h3>Dynamic vs static</h3>

<!-- If `classExpr` has a value for the `special` class, this value will override the
`class="special"` below -->
<div class="special" [class]="classExpr">Some text.</div>

<!-- If `styleExpr` has a value for the `color` property, this value will override the
`style="color: blue"` below -->
<div style="color: blue" [style]="styleExpr">Some text.</div>
```

Delegating to styles with lower precedence

It is possible for higher precedence styles to "delegate" to lower precedence styles using `undefined` values. Whereas setting a style property to `null` ensures the style is removed, setting it to `undefined` will cause Angular to fall back to the next-highest precedence binding to that style.

For example, consider the following template:

src/app/app.component.html

```
<comp-with-host-binding dirWithHostBinding></comp-with-host-binding>
```

Imagine that the `dirWithHostBinding` directive and the `comp-with-host-binding` component both have a `[style.width]` host binding. In that case, if `dirWithHostBinding` sets its binding to `undefined`, the `width` property will fall back to the value of the `comp-with-host-binding` host binding. However, if `dirWithHostBinding` sets its binding to `null`, the `width` property will be removed entirely.

Super-powered by Google ©2010-2020. Code licensed under an MIT-style License. Documentation licensed under CC BY 4.0.

Version 10.0.10-local+sha.b32126c335.