Property binding

Property binding in Angular helps you set values for properties of HTML elements or directives. Use property binding to do things such as toggle button functionality, set paths programmatically, and share values between components.

See the for a working example containing the code snippets in this guide.

Prerequisites

To get the most out of property binding, you should be familiar with the following:

- Basics of components
- Basics of templates
- Binding syntax

Understanding the flow of data

Property binding moves a value in one direction, from a component's property into a target element property.

For more information on listening for events, see Event binding.

To read a target element property or call one of its methods, see the API reference for ViewChild and ContentChild.

Binding to a property

To bind to an element's property, enclose it in square brackets, `[]`, which identifies the property as a target property. A target property is the DOM property to which you want to assign a value. For example, the target property in the following code is the image element's `src` property.

In most cases, the target name is the name of a property, even when it appears to be the name of an attribute. In this example, `src` is the name of the `<img>` element property.

The brackets, `[]`, cause Angular to evaluate the right-hand side of the assignment as a dynamic expression. Without the brackets, Angular treats the right-hand side as a string literal and sets the property to that static value.

Omitting the brackets renders the string `parentItem`, not the value of `parentItem`.

Setting an element property to a component property value

To bind the `src` property of an `<img>` element to a component's property, place the target, `src`, in square brackets followed by an equal sign and then the property. The property here is `itemImageUrl`.

Declare the `itemImageUrl` property in the class, in this case `AppComponent`.

{@a colspan}

`colspan` and `colSpan`

A common point of confusion is between the attribute, `colspan`, and the property, `colSpan`. Notice that these two names differ by only a single letter.

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' because it isn't a known built-in 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:

Another example is disabling a button when the component says that it `isUnchanged`:

Another is setting a property of a directive:

Yet another is setting the model property of a custom component—a great way for parent and child components to communicate:

Toggling button functionality

To disable a button's functionality depending on a Boolean value, bind the DOM `disabled` property to a property in the class that is `true` or `false`.

Because the value of the property `isUnchanged` is `true` in the `AppComponent`, Angular disables the button.

Setting a directive property

To set a property of a directive, place the directive within square brackets , such as `[ngClass]`, followed by an equal sign and the property. Here, the property is `classes`.

To use the property, you must declare it in the class, which in this example is `AppComponent`. The value of `classes` is `special`.

Angular applies the class `special` to the `<p>` element so that you can use `special` to apply CSS styles.

Bind values between components

To set the model property of a custom component, place the target, here `childItem`, between square brackets `[]` followed by an equal sign and the property. Here, the property is `parentItem`.

To use the target and the property, you must declare them in their respective classes.

Declare the target of `childItem` in its component class, in this case `ItemDetailComponent`.

For example, the following code declares the target of `childItem` in its component class, in this case `ItemDetailComponent`.

Then, the code contains an `@Input()` decorator with the `childItem` property so data can flow into it.

Next, the code declares the property of `parentItem` in its component class, in this case `AppComponent`. In this example the type of `childItem` is `string`, so `parentItem` needs to be a string. Here, `parentItem` has the string value of `lamp`.

With this configuration, the view of `<app-item-detail>` uses the value of `lamp` for `childItem`.

Property binding and security

Property binding can help keep content secure. For example, consider the following malicious content.

The component template interpolates the content as follows:

The browser doesn't process the HTML and instead displays it raw, as follows.

"Template <script>alert("evil never sleeps")</script> Syntax" is the interpolated evil title.

Angular does not allow HTML with `<script>` tags, neither with interpolation nor property binding, which prevents the JavaScript from running.

In the following example, however, Angular sanitizes the values before displaying them.

Interpolation handles the `<script>` tags differently than property binding, but both approaches render the content harmlessly. The following is the browser output of the sanitized `evilTitle` example.

"Template Syntax" is the property bound evil title.

Property binding and interpolation

Often interpolation and property binding can achieve the same results. The following binding pairs do the same thing.

Use either form when rendering data values as strings, though interpolation is preferable for readability. However, when setting an element property to a non-string data value, you must use property binding.

What's next

- Property binding best practices