Template variables

Template variables help you use data from one part of a template in another part of the template. Use template variables to perform tasks such as respond to user input or finely tune your application's forms.

A template variable can refer to the following:

- a DOM element within a template
- a directive
- an element
- TemplateRef
- a web component

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

Syntax

In the template, you use the hash symbol, `#`, to declare a template variable. The following template variable, `#phone`, declares a `phone` variable on an `<input>` element.

Refer to a template variable anywhere in the component's template. Here, a `<button>` further down the template refers to the `phone` variable.

How Angular assigns values to template variables

Angular assigns a template variable a value based on where you declare the variable:

- If you declare the variable on a component, the variable refers to the component instance.
- If you declare the variable on a standard HTML tag, the variable refers to the element.
- If you declare the variable on an `<ng-template>` element, the variable refers to a `TemplateRef` instance, which represents the template. For more information on `<ng-template>`, see How Angular uses the asterisk, `*`, syntax in Structural directives.
- If the variable specifies a name on the right-hand side, such as `#var="ngModel"`, the variable refers to the directive or component on the element with a matching `exportAs` name.

Using `NgForm` with template variables

In most cases, Angular sets the template variable's value to the element on which it occurs. In the previous example, `phone` refers to the phone number `<input>`. The button's click handler passes the `<input>` value to the component's `callPhone()` method.

The `NgForm` directive demonstrates getting a reference to a different value by referencing a directive's `exportAs` name. In the following example, the template variable, `itemForm`, appears three times separated by HTML.

Without the `ngForm` attribute value, the reference value of `itemForm` would be the HTMLFormElement, `<form>`. There is, however, a difference between a `Component` and a `Directive` in that Angular references a `Component` without specifying the attribute value, and a `Directive` does not change the implicit reference, or the element.

With `NgForm`, `itemForm` is a reference to the NgForm directive with the ability to track the value and validity of every control in the form.

Unlike the native `<form>` element, the `NgForm` directive has a `form` property. The `NgForm` `form` property lets you disable the submit button if the `itemForm.form.valid` is invalid.

Template variable scope

Refer to a template variable anywhere within its surrounding template. Structural directives, such as `*ngIf` and `*ngFor`, or `<ng-template>` act as a template boundary. You cannot access template variables outside of these boundaries.

Define a variable only once in the template so the runtime value remains predictable.

Accessing in a nested template

An inner template can access template variables that the outer template defines.

In the following example, changing the text in the `<input>` changes the value in the `<span>` because Angular immediately updates changes through the template variable, `ref1`.

In this case, there is an implied `<ng-template>` around the `<span>` and the definition of the variable is outside of it. Accessing a template variable from the parent template works because the child template inherits the context from the parent template.

Rewriting the preceding code in a more verbose form explicitly shows the `<ng-template>`.

```
<input #ref1 type="text" [(ngModel)]="firstExample" />

<!-- New template -->
<ng-template [ngIf]="true">
  <!-- Because the context is inherited, the value is available to the new template
-->
  <span>Value: {{ ref1.value }}</span>
</ng-template>
```

However, accessing a template variable from outside the parent template doesn't work.

```
  <input *ngIf="true" #ref2 type="text" [(ngModel)]="secondExample" />
  <span>Value: {{ ref2?.value }}</span> <!-- doesn't work -->
```

The verbose form shows that `ref2` is outside the parent template.

```
<ng-template [ngIf]="true">
  <!-- The reference is defined within a template -->
  <input #ref2 type="text" [(ngModel)]="secondExample" />
</ng-template>
<!-- ref2 accessed from outside that template doesn't work -->
<span>Value: {{ ref2?.value }}</span>
```

Consider the following example that uses `*ngFor`.

```
<ng-container *ngFor="let i of [1,2]">
  <input #ref type="text" [value]="i" />
</ng-container>
{{ ref.value }}
```

Here, `ref.value` doesn't work. The structural directive, `*ngFor` instantiates the template twice because `*ngFor` iterates over the two items in the array. It is impossible to define what the `ref.value` reference signifies.

With structural directives, such as `*ngFor` or `*ngIf`, there is no way for Angular to know if a template is ever instantiated.

As a result, Angular isn't able to access the value and returns an error.

Accessing a template variable within `<ng-template>`

When you declare the variable on an `<ng-template>`, the variable refers to a `TemplateRef` instance, which represents the template.

In this example, clicking the button calls the `log()` function, which outputs the value of `#ref3` to the console. Because the `#ref` variable is on an `<ng-template>`, the value is `TemplateRef`.

The following is the expanded browser console output of the `TemplateRef()` function with the name of `TemplateRef`.

▼ ƒ TemplateRef() name: "TemplateRef" **proto**: Function

{@a template-input-variable} {@a template-input-variables}

Template input variable

A *template input variable* is a variable to reference within a single instance of the template. You declare a template input variable using the `let` keyword as in `let hero`.

There are several such variables in this example: `hero`, `i`, and `odd`.

```
<ng-template #hero let-hero let-i="index" let-odd="isOdd">
  <div [class]="{'odd-row': odd}">{{i}}:{{hero.name}}</div>
</ng-template>
```

The variable's scope is limited to a single instance of the repeated template. Use the same variable name again in the definition of other structural directives.

In contrast, you declare a template variable by prefixing the variable name with `#`, as in `#var`. A template variable refers to its attached element, component, or directive.

Template input variables and template variables names have their own namespaces. The template input variable `hero` in `let hero` is distinct from the template variable `hero` in `#hero`.