Content projection

This topic describes how to use content projection to create flexible, reusable components.

To view or download the example code used in this topic, see the .

Content projection is a pattern in which you insert, or *project*, the content you want to use inside another component. For example, you could have a `Card` component that accepts content provided by another component.

The following sections describe common implementations of content projection in Angular, including:

- Single-slot content projection. With this type of content projection, a component accepts content from a single source.
- Multi-slot content projection. In this scenario, a component accepts content from multiple sources.
- Conditional content projection. Components that use conditional content projection render content only when specific conditions are met.

{@a single-slot } ## Single-slot content projection

The most basic form of content projection is *single-slot content projection*. Single-slot content projection refers to creating a component into which you can project one component.

To create a component that uses single-slot content projection:

1. Create a component.
2. In the template for your component, add an `<ng-content>` element where you want the projected content to appear.

For example, the following component uses an `<ng-content>` element to display a message.

With the `<ng-content>` element in place, users of this component can now project their own message into the component. For example:

The `<ng-content>` element is a placeholder that does not create a real DOM element. Custom attributes applied to `<ng-content>` are ignored.

{@a multi-slot} ## Multi-slot content projection

A component can have multiple slots. Each slot can specify a CSS selector that determines which content goes into that slot. This pattern is referred to as *multi-slot content projection*. With this pattern, you must specify where you want the projected content to appear. You accomplish this task by using the `select` attribute of `<ng-content>`.

To create a component that uses multi-slot content projection:

1. Create a component.

2. In the template for your component, add an `<ng-content>` element where you want the projected content to appear.
3. Add a `select` attribute to the `<ng-content>` elements. Angular supports selectors for any combination of tag name, attribute, CSS class, and the `:not` pseudo-class.

For example, the following component uses two `<ng-content>` elements.

Content that uses the `question` attribute is projected into the `<ng-content>` element with the `select=[question]` attribute.

ng-content without a select attribute

If your component includes an `<ng-content>` element without a `select` attribute, that instance receives all projected components that do not match any of the other `<ng-content>` elements.

In the preceding example, only the second `<ng-content>` element defines a `select` attribute. As a result, the first `<ng-content>` element receives any other content projected into the component.

{@a conditional }

Conditional content projection

If your component needs to *conditionally* render content, or render content multiple times, you should configure that component to accept an `<ng-template>` element that contains the content you want to conditionally render.

Using an `<ng-content>` element in these cases is not recommended, because when the consumer of a component supplies the content, that content is *always* initialized, even if the component does not define an `<ng-content>` element or if that `<ng-content>` element is inside of an `ngIf` statement.

With an `<ng-template>` element, you can have your component explicitly render content based on any condition you want, as many times as you want. Angular will not initialize the content of an `<ng-template>` element until that element is explicitly rendered.

The following steps demonstrate a typical implementation of conditional content projection using `<ng-template>`.

1. Create a component.
2. In the component that accepts an `<ng-template>` element, use an `<ng-container>` element to render that template, such as:

   This example uses the `ngTemplateOutlet` directive to render a given `<ng-template>` element, which you will define in a later step. You can apply an `ngTemplateOutlet` directive to any type of element. This example

assigns the directive to an `<ng-container>` element because the component does not need to render a real DOM element.

3. Wrap the `<ng-container>` element in another element, such as a `div` element, and apply your conditional logic.

4. In the template where you want to project content, wrap the projected content in an `<ng-template>` element, such as:

   The `<ng-template>` element defines a block of content that a component can render based on its own logic. A component can get a reference to this template content, or `TemplateRef`, by using either the `@ContentChild` or `@ContentChildren` decorators. The preceding example creates a custom directive, `appExampleZippyContent`, as an API to mark the `<ng-template>` for the component's content. With the `TemplateRef`, the component can render the referenced content by using either the `ngTemplateOutlet` directive, or with the `ViewContainerRef` method `createEmbeddedView()`.

5. Create an attribute directive with a selector that matches the custom attribute for your template. In this directive, inject a `TemplateRef` instance.

   In the previous step, you added an `<ng-template>` element with a custom attribute, `appExampleZippyContent`. This code provides the logic that Angular will use when it encounters that custom attribute. In this case, that logic instructs Angular to instantiate a template reference.

6. In the component you want to project content into, use `@ContentChild` to get the template of the projected content.

   Prior to this step, your application has a component that instantiates a template when certain conditions are met. You've also created a directive that provides a reference to that template. In this last step, the `@ContentChild` decorator instructs Angular to instantiate the template in the designated component.

   In the case of multi-slot content projection, use `@ContentChildren` to get a `QueryList` of projected elements.

{@a ngprojectas }

Projecting content in more complex environments

As described in Multi-slot Content Projection, you typically use either an attribute, element, CSS Class, or some combination of all three to identify where to project your content. For example, in the following HTML template, a paragraph tag uses a custom attribute, `question`, to project content into the `app-zippy-multislot` component.

In some cases, you might want to project content as a different element. For example, the content you want to project might be a child of another element. Accomplish this with the `ngProjectAs` attribute.

For instance, consider the following HTML snippet:

This example uses an `<ng-container>` attribute to simulate projecting a component into a more complex structure.

Reminder!

The `ng-container` element is a logical construct that is used to group other DOM elements; however, the `ng-container` itself is not rendered in the DOM tree.

In this example, the content we want to project resides inside another element. To project this content as intended, the template uses the `ngProjectAs` attribute. With `ngProjectAs`, the entire `<ng-container>` element is projected into a component using the `[question]` selector.

@reviewed 2021-09-17