View encapsulation

In Angular, a component's styles can be encapsulated within the component's host element so that they don't affect the rest of the application.

The `Component` 's decorator provides the `encapsulation` option which can be used to control how the encapsulation is applied on a *per component* basis.

Choose from the following modes:

- `ViewEncapsulation.ShadowDom` , Angular uses the browser's built-in Shadow DOM API to enclose the component's view inside a ShadowRoot (used as the component's host element) and apply the provided styles in an isolated manner.
- `ViewEncapsulation.Emulated` , Angular modifies the component's CSS selectors so that they are only applied to the component's view and do not affect other elements in the application (*emulating* Shadow DOM behavior). For more details, see Inspecting generated CSS.
- `ViewEncapsulation.None` , Angular does not apply any sort of view encapsulation meaning that any styles specified for the component are actually globally applied and can affect any HTML element present within the application. This mode is essentially the same as including the styles into the HTML itself.

`ViewEncapsulation.ShadowDom` only works on browsers that have built-in support for the shadow DOM (see Can I use - Shadow DOM v1). Not all browsers support it, which is why the `ViewEncapsulation.Emulated` is the recommended and default mode.

{@a inspect-generated-css}

Inspecting generated CSS

When using the emulated view encapsulation, Angular pre-processes all the component's styles so that they are only applied to the component's view.

In the DOM of a running Angular application, elements belonging to components using emulated view encapsulation have some extra attributes attached to them:

<hero-details _nghost-pmm-5> <h2 _ngcontent-pmm-5>Mister Fantastic</h2> <hero-team _ngcontent-pmm-5 _nghost-pmm-6> <h3 _ngcontent-pmm-6>Team</h3> </hero-team> </hero-detail>
There are two kinds of such attributes:

- `_nghost` attributes are added to elements that enclose a component's view and that would be ShadowRoots in a native Shadow DOM encapsulation. This is typically the case for components' host elements.
- `_ngcontent` attributes are added to child element within a component's view, those are used to match the elements with their respective emulated ShadowRoots (host elements with a matching `_nghost` attribute).

The exact values of these attributes are a private implementation detail of Angular. They are automatically generated and you should never refer to them in application code.

They are targeted by the generated component styles, which are injected in the `<head>` section of the DOM:

[_nghost-pmm-5] { display: block; border: 1px solid black; }

h3[_ngcontent-pmm-6] { background-color: white; border: 1px solid #777; }

These styles are post-processed so that each CSS selector is augmented with the appropriate `_nghost` or `_ngcontent` attribute. These modified selectors make sure the styles to be applied to components' views in an isolated and targeted fashion.

Mixing encapsulation modes

As previously mentioned you specify the encapsulation mode in the Component's decorator on a *per component* basis, this means that within your application you can have different components using different encapsulation strategies.

Although possible, this is not recommended. If it is really needed you should be aware of how the styles of components using different encapsulation modes will interact with each other:

- The styles of components with `ViewEncapsulation.Emulated` are added to the `<head>` of the document, making them available throughout the application, but their selectors only affect elements within their respective components' templates.
- The styles of components with `ViewEncapsulation.None` are added to the `<head>` of the document, making them available throughout the application, so are completely global and affect any matching elements within the document.
- The styles of components with `ViewEncapsulation.ShadowDom` are only added to the shadow DOM host, ensuring that they only affect elements within their respective components' views.

Styles of `ViewEncapsulation.Emulated` and `ViewEncapsulation.None` components are also added to the shadow DOM host of each `ViewEncapsulation.ShadowDom` component.

This means that styles for components with `ViewEncapsulation.None` will affect matching elements within shadow DOMs.

This approach may seem counter-intuitive at first, but without it a component with `ViewEncapsulation.None` would be rendered differently within a component using `ViewEncapsulation.ShadowDom`, since its styles would not be available.

Examples

This section shows examples of how the styling of components with different `ViewEncapsulation` interact.

See the to try out these components yourself.

No encapsulation

The first example shows a component that has `ViewEncapsulation.None`. This component colors its template elements red.

Angular adds the styles for this component as global styles to the `<head>` of the document.

As already mentioned Angular also adds the styles to all shadow DOM hosts. Therefore, the styles are available throughout the whole application.

 component with no encapsulation

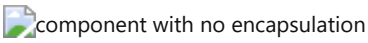
Emulated encapsulation

The second example shows a component that has `ViewEncapsulation.Emulated`. This component colors its template elements green.

Similar to `ViewEncapsulation.None`, Angular adds the styles for this component to the `<head>` of the document, but with "scoped" styles.

Therefore, only the elements directly within this component's template will match its styles. Since the "scoped" styles from the `EmulatedEncapsulationComponent` are very specific, they override the global styles from the `NoEncapsulationComponent`.

In this example, the `EmulatedEncapsulationComponent` contains a `NoEncapsulationComponent`, but `NoEncapsulationComponent` is still styled as expected since the `EmulatedEncapsulationComponent`'s "scoped" styles do not match elements in its template.

 component with no encapsulation

Shadow DOM encapsulation

The third example shows a component that has `ViewEncapsulation.ShadowDom`. This component colors its template elements blue.

Angular adds styles for this component only to the shadow DOM host, so they are not visible outside the shadow DOM.

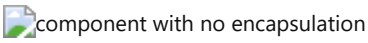
Note that Angular also adds the global styles from the `NoEncapsulationComponent` and `EmulatedEncapsulationComponent` to the shadow DOM host, so those styles are still available to the elements in the template of this component.

In this example, the `ShadowDomEncapsulationComponent` contains both a `NoEncapsulationComponent` and `EmulatedEncapsulationComponent`.

The styles added by the `ShadowDomEncapsulationComponent` component are available throughout the shadow DOM of this component, and so to both the `NoEncapsulationComponent` and `EmulatedEncapsulationComponent`.

The `EmulatedEncapsulationComponent` has specific "scoped" styles, so the styling of this component's template is unaffected.

But since styles from `ShadowDomEncapsulationComponent` are added to the shadow host after the global styles, the `h2` style overrides the style from the `NoEncapsulationComponent`. The result is that the `<h2>` element in the `NoEncapsulationComponent` is colored blue rather than red, which may not be what the component's author intended.

 component with no encapsulation

@reviewed 2021-11-10