CSS Encapsulation in Angular

What Is CSS Encapsulation

CSS Encapsulation allows for scoping one’s styles to a specific component or reusable piece of code. When using component-based architecture, the developer can control how styles are applied to sections of the application. Styles can be applied to a specific component without side effects to other elements. While browsers are starting to natively support the ability to create components and scope their styles via [Custom Elements](https://developer.mozilla.org/en-US/docs/Web/Web_Components) created using the [Shadow DOM](https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM), support is still limited.

Angular comes with CSS encapsulation out of the box. When generating a new project, the default is for the styles.css file at the root of the project to apply globally to the application, and for component styles, such as the styles found in foo.component.css,

1

2

3

4

5

@Component({

selector: 'app-foo',

templateUrl: './foo.component.html',

styleUrls: ['./foo.component.css']

})

typescript

to only apply to the foo component and nowhere else. But that is not the only way styles can be encapsulated in Angular, let us take a closer look.

Angular Encapsulation Options

To explore the differences between each of the view encapsulation options available to us in Angular, consider an app with the following code and structure:

Sample App

*(*[*full code*](https://github.com/martine-dowden/css-encapsulation)*, angular version: 6.1.0, cli version: 6.1.5)*

Using the CLI we generate a project with two components, first and second.

1

2

3

4

$ ng new encapsulation --styles=”scss”

$ cd encapsulation

$ ng g c first

$ ng g c second

bash

This will generate an Angular app that uses SCSS and has the default view encapsulation type of Emulated. Let's add some elements and styles.

Root Elements

index.html (Partial):

1

2

3

4

<body>

<h1>CSS Encapsulation</h1>

<app-root></app-root>

</body>

html

styles.scss

1

2

3

4

5

6

7

8

9

10

11

12

13

p {

color: #333; // dark gray

font-family: sans-serif;

}

h1 {

color: #3888a3; // purple

}

h2 {

border-bottom: solid 1px #896ac8;

color: #896ac8;

}

scss

app.component

app.component.html:

1

2

3

4

<main>

<h2>My Components</h2>

<app-first-component></app-first-component> <app-second-component></app-second-component>

</main>

html

first-component

first.component.html:

1

2

3

4

<section>

<h3>First Component</h3>

<p>Phasellus augue ante, lobortis pulvinar euismod sit amet, auctor vel nisl...</p>

</section>

html

first.component.scss:

1

2

3

4

5

6

7

8

9

10

11

p {

background: #ffe9b5; // pale orange

border-radius: 4px;

font-family: cursive;

padding: 1rem;

}

h3 {

color: #a17200; // brown

text-align: center;

}

scss

second-component

second.component.html:

1

2

3

4

<section>

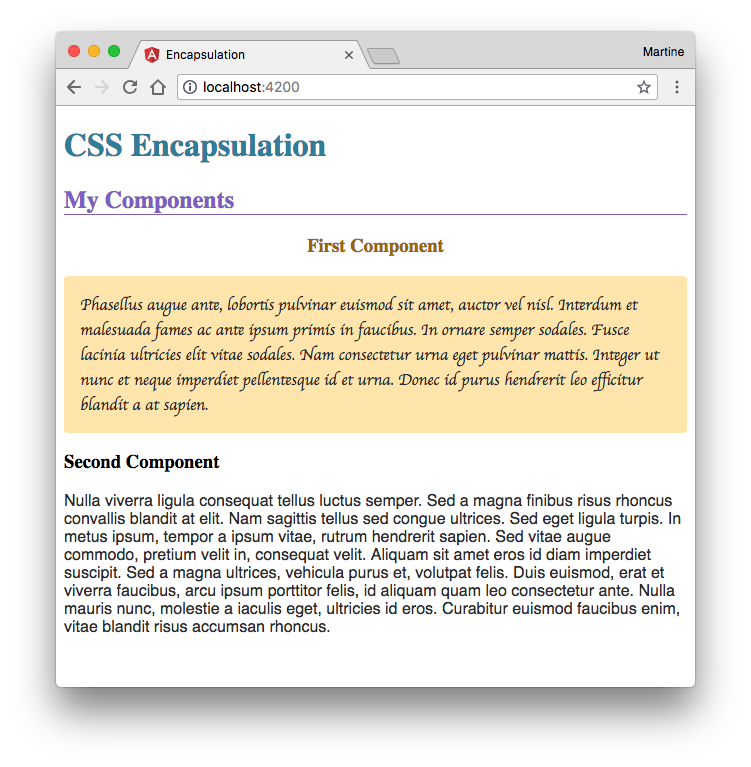
<h3>Second Component</h3>

<p>Nulla viverra ligula consequat tellus luctus semper. Sed a magna finibus... </p>

</section>

html

When using emulated encapsulation, our app looks like this:



Emulated

The default option is “Emulated”. The styles, regardless of whether you are using SASS, SCSS, LESS, Stylus, or plain CSS, is preprocessed. A host element attribute is added to each selector which scopes the styling to the host element.

Looking at the debugger you will notice that the paragraph tag has \_ngcontent-c1 added to it. The same is added to the css class. 

Each DOM element has a \_ngcontent attribute, automatically generated and unique to its host, attached to it. These identify which host the element belongs to and therefore which styles should be applied to it.

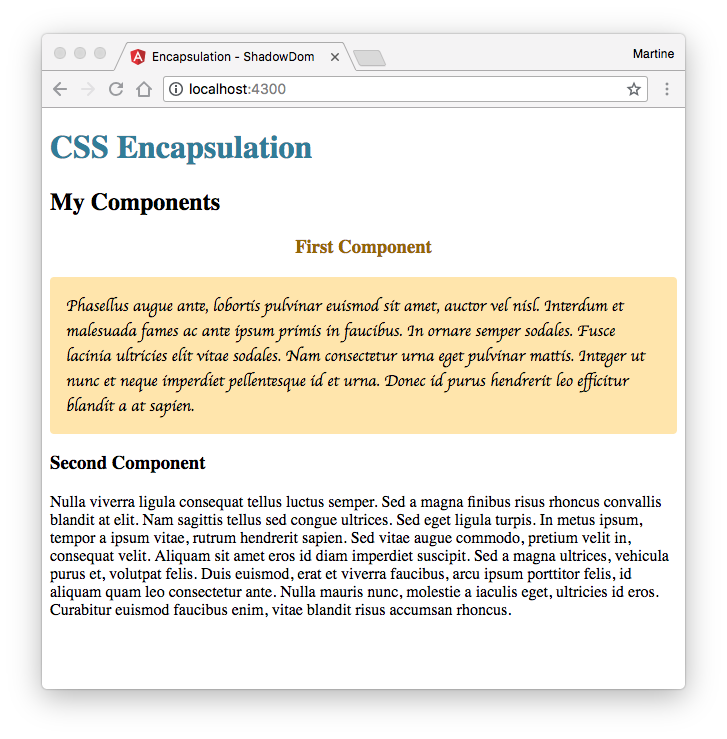
The result is that even though we styled the p and h3 elements in first.component, the styles were not applied to the elements of second.component.

Styles placed in styles.scss however, which is applied globally to the application, were applied to all components including app.component and index.html.

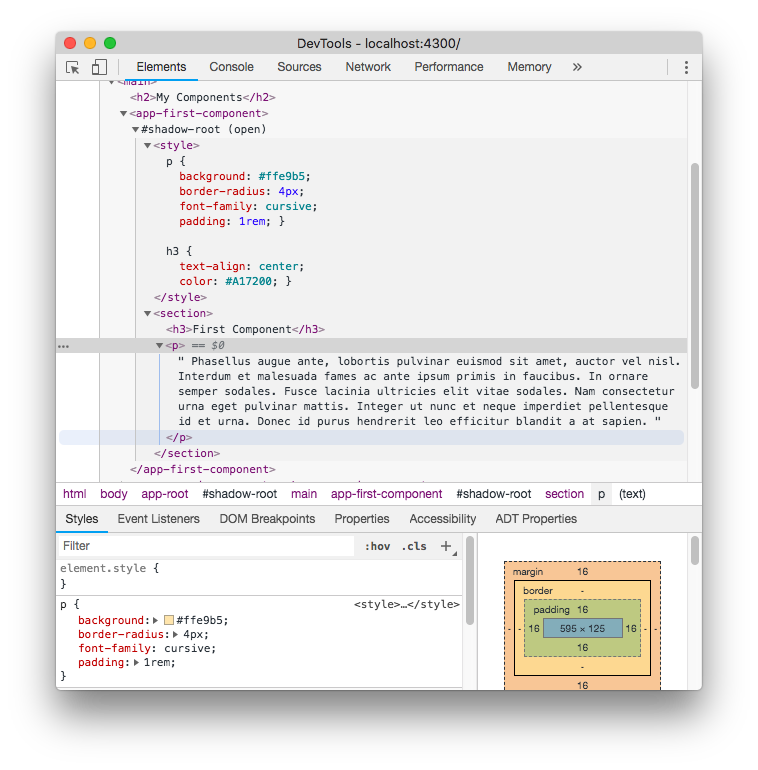
ShadowDom

When ShadowDom view encapsulation is set, the application uses the browser's native shadow DOM implementation. Elements are encapsulated by adding a separate hidden or “shadow” DOM to each element.

When using ShadowDom encapsulation, our app looks like this:

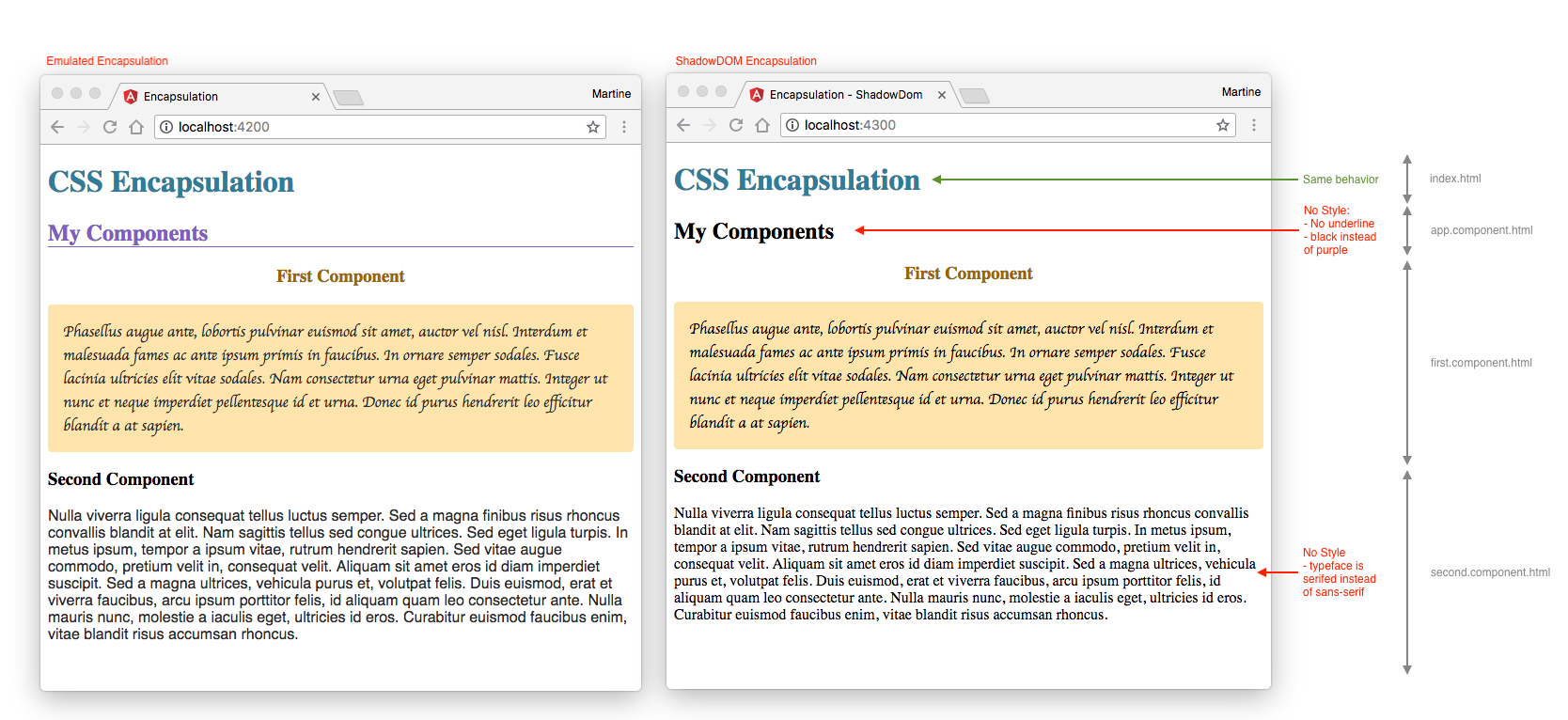


A shadow DOM tree is attached to each component, with the shadow root attached to the component element. Looking at first.component, the component itself (<app-first-component>) is where the shadow root gets attached and elements inside the component (<section>, <h3>, <p>) compose the shadow tree.

This is a screenshot showing dev tools in chrome when using ShadowDom encapsulation. It shows the shadow dom tree for first.component (<app-first-component>). 

Notice that unlike emulated encapsulation, no attributes have been added to the elements or the classes.

Another difference from emulated encapsulation is regarding the global styles, those coming from styles.scss. Notice the “My Components” header, found in app.component.html and the styles of the second component paragraph. They lack the styles previously applied to them through styles.scss. Unlike the “CSS Encapsulation” header, found in index.html which remained styled, components have been completely isolated from the global styles.



Limitations

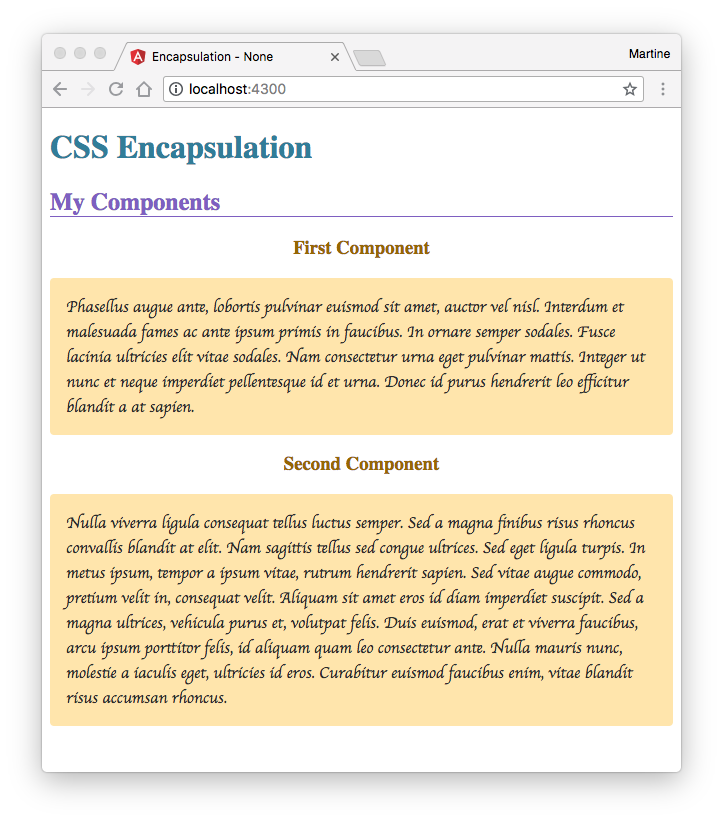
ShadowDom encapsulation has some limitations to be aware of. Not all browsers have Shadow DOM fully implemented. According to [caniuse.com](https://caniuse.com/#search=shadow), Chrome is the only major web browser, along with a couple of mobile browsers, to have both Shadow DOM v1 and v2 fully implemented. When testing in Safari, the application seemed to work, however, it would not load in Firefox. Because of the limited support, Emulated is currently recommended over the use of ShadowDom encapsulation.

Native

Native view encapsulation is now deprecated. Similarly, to ShadowDom encapsulation, it uses the browser’s native shadow DOM implementation, however, it uses version 0 which is now also deprecated.

None

None, as its name implies, removes all encapsulation. Styles in any of the style sheets, regardless of being applied directly to a component, will be applied globally to the application.



Notice second.component. It is now styled identically to first.component. Global styles (styles.scss)have been applied normally, however, component specific styles, such as the styles in first.component have been applied to both components.

Inheritance

To demonstrate inheritance, we will add some styles to second.component.

second.component.scss

1

2

3

4

p {

background: #caf2ff; // light blue

font-family: monospace;

}

scss

As a reminder first.component.scss has the following styles

first.component.scss

1

2

3

4

5

6

7

8

9

10

11

p {

background: #ffe9b5; // pale orange

border-radius: 4px;

font-family: cursive;

padding: 1rem;

}

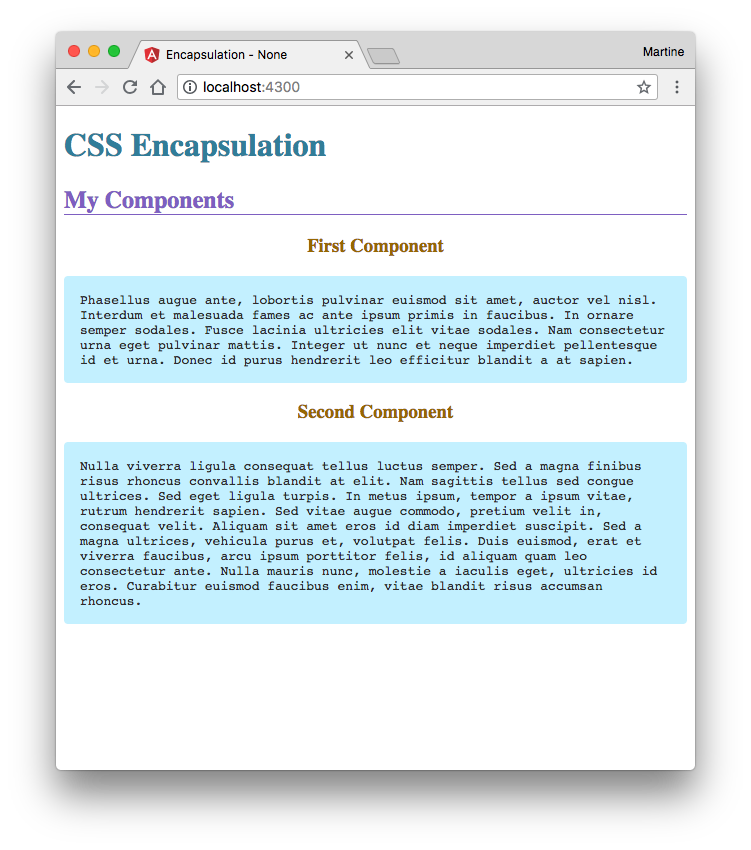
h3 {

color: #a17200; // brown

text-align: center;

}

scss



When styles were added to second.component, the <p> elements of both first.component and second.component took on the styles set in second.component. Since we did not add any styling to the header tag, the <h3> tags kept the styling from first.component.

The order in which the child components are declared in parent determine style inheritance; styles from components declared later in the parent file will supersede those set for the same elements at the same level in earlier components. If we invert the order in which the components are placed in app.component.html so that second.component is at the top, declared first, and first.component is at the bottom, declared second. The paragraphs of both components will be styled using the css from second.component. Header tags will continue to look the same.

app.component.html

1

2

3

4

5

<main>

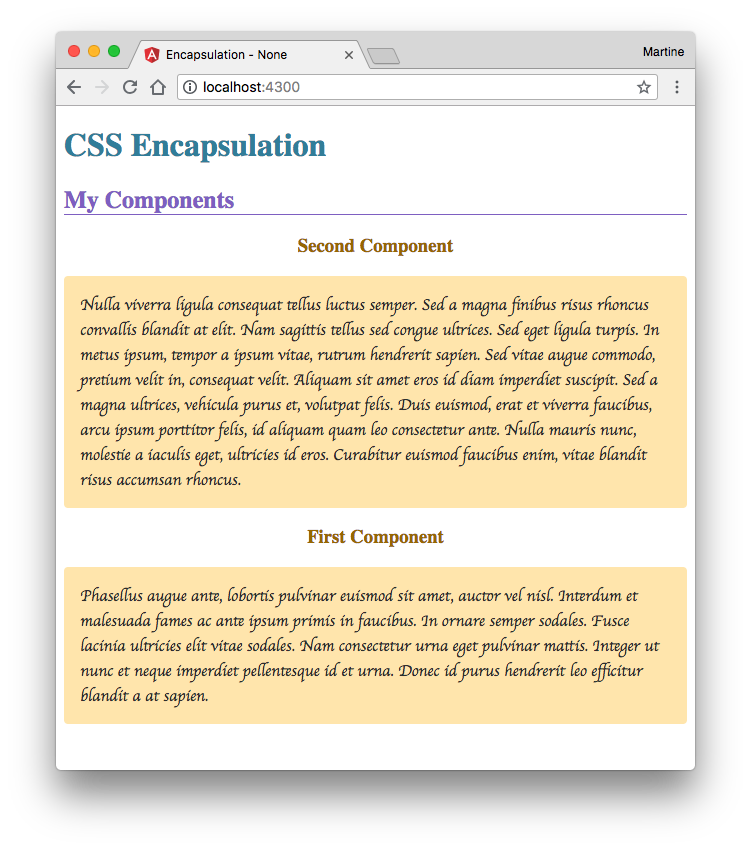
<h2>My Components</h2>

<app-second-component></app-second-component>

<app-first-component></app-first-component>

</main>

html



Because first.component is declared second, its styles are now being applied to the paragraph tags.

How to Change Encapsulation Types

There are 2 ways to change encapsulation type in an Angular application, globally for all components once a project has already been created, and individually, by component.

Globally

In the main.ts file change:

1

platformBrowserDynamic().bootstrapModule(AppModule);

typescript

to

1

2

3

4

5

platformBrowserDynamic().bootstrapModule(AppModule, [

{

defaultEncapsulation: ViewEncapsulation.None

}

]);

typescript

defaultEncapsulation options are:

* ViewEncapsulation.Emulated
* ViewEncapsulation.Native
* ViewEncapsulation.ShadowDom
* ViewEncapsulation.None

Don’t forget to import ViewEncapsulation from @angular/core

1

import { enableProdMode, ViewEncapsulation } from "@angular/core";

typescript

Individual Components

To change the encapsulation strategy for a specific component, in the component.ts file as part of the component declaration we will add the encapsulation type:

1

2

3

4

5

6

@Component({

selector: 'app-first',

templateUrl: './first.component.html',

styleUrls: ['./first.component.scss'],

encapsulation: ViewEncapsulation.None

})

typescript

Don’t forget to import ViewEncapsulation from @angular/core

1

import { Component, OnInit, ViewEncapsulation } from "@angular/core";

typescript

CLI

To accomplish the equivalent generating a component using the CLI:

1

$ ng g c first --view-encapsulation="None"

bash

--view-encapsulation options are:

* Emulated
* Native
* ShadowDom
* None

Closing Remarks

We have now looked at the 4 ways of encapsulating styles in an angular application: Emulated, Native, ShadowDom, and None, how they work, and their differences. The code used in this guide can be found on [github](https://github.com/martine-dowden/css-encapsulation), feel free to play with it and experiment for yourself. Happy Coding!