Gidon Handler Follow
Aug 17, 2020 · 7 min read ★ · Listen

Angular ViewEncapsulation.ShadowDom

Is it the future? and are we there yet?

No doubt, view encapsulation is one of the great features of Angular. Anyone who has worked on a large web project knows that trying to come up with names for CSS selectors and nesting them to avoid collisions is a huge headache, not to mention finding and editing them.
Angular solves this problem with ViewEncapsulation. Component files are conveniently placed next to each other, a unique attribute (e.g [_ngcontent-hef-c18]) is added to every element in a component.html file, and the same attribute to each rule

component will only apply to that component no matter how many <h2> tags are on the page.

This is the the default out of the box method. But we can tell Angular to use a newer, native encapsulation technology:

```
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.scss'],
encapsulation: ViewEncapsulation.ShadowDom
})
```

While this option has been available since the birth of Angular, I have yet to stumble upon an app that uses it.
Worse yet, there is almost no documentation regarding the subject.

In this post we will learn the various effects of using native shadowDom in Angular and understand if we should start using it in our apps.
We will not deep dive into native ShadowDom and web components. Nor is it necessary for this post, However a good understanding of the subject is crucial for any up and coming web developer. Good resources on the subject are: *developer.mozilla.org* and *javascript.info*.

In this post we will be exploring:

1. **ShadowDom with global styles and css and component libraries such as Bootstrap or Angular Material.**
2. **ShadowDom and emulated components working together.**
3. **ShadowDom with content projection (styling,interpolation, property binding and event binding).**
4. **ShadowDom with no template (styling, interpolation, property binding and**




Global Styles

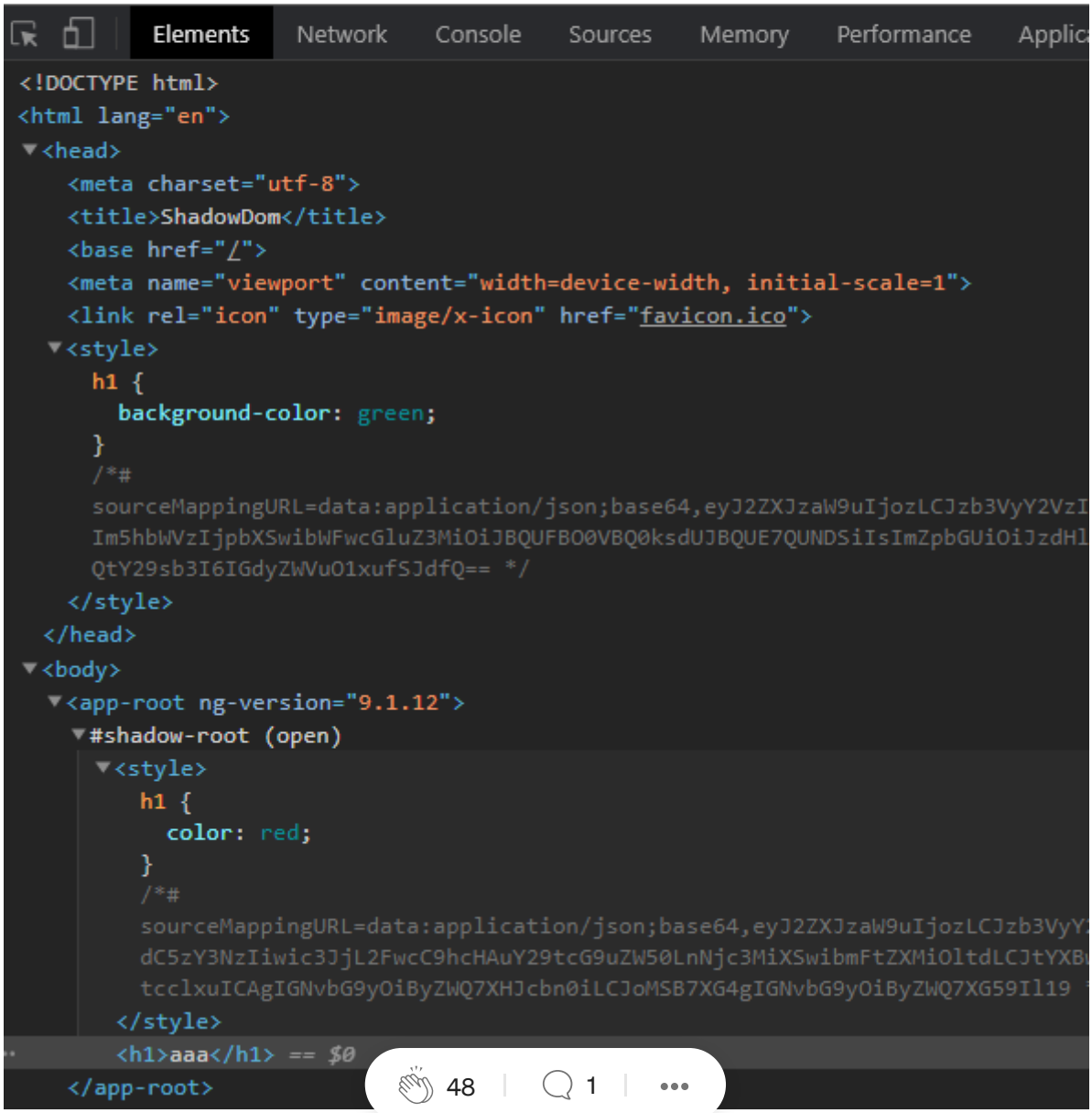
Lets Install Angular, set app.component.ts encapsulation to ShadowDow (as shown above).
We'll create <h1> tag in the app.component.html file.
And in app.component.scss

```
h1{color: red;}
```

and in global style.scss

```
h1{background-color: green}
```

aaa



As we can, <app-root> is a native web component (shadow-root). Sure is nice not to see the extra attributes on its only tag(<h1>). Its style (color: red) stays in the component and not projected to <head> as in default/emulated mode. But mainly, we see that the global style (background-color: green), which is injected into <head>, does not pierce into the component.

have those to begin with.
We could do this in every conponent.scss file

```
@Import "../style.scss";
h1{color: red;}
```

Import the global css into each and every component. It would work and global style would be available to all shadowDom components, But lets think of a situation where we use Bootstrap. Its about 10,000 lines of global css. And lets say we have a few components on the page, and a list with a hundred list items, and each list item has an icon or two which are also components. We can easily have a thousand or so components on the page and each of those would have a full copy of the bootstrap css. Its way above my pay grade to decide how the browser allocates and treats such situations, and if there are performance implications, but it doesn't look right.

So we know that global styles can not be used in the template of shadowDow components. This is a problem for using css framworks like Twitter Bootstrap or even Angular material (even though they are a component library, they use global style besides the component style).

Now for the interesting part

What would happen if we create a component with shadowDow and put in its template (which means child) a default or "emulated" component?

Lets first recap. If all components are emulated all the css gets put in the <head> and that's simple css selectors with the unique [_ngcomponent_xyz] attribute, No problem. If on the other hand, all components are shadowDom, then we can't use global styles but every component will have its own inner style and will work. But what if a shadow component has an emulated component in its template. The Emulated component gets its style form <head>, but that is blocked by the parent shadow.

Lest create four components.

1. app.component — shadow




4. grandchild.component — emulated

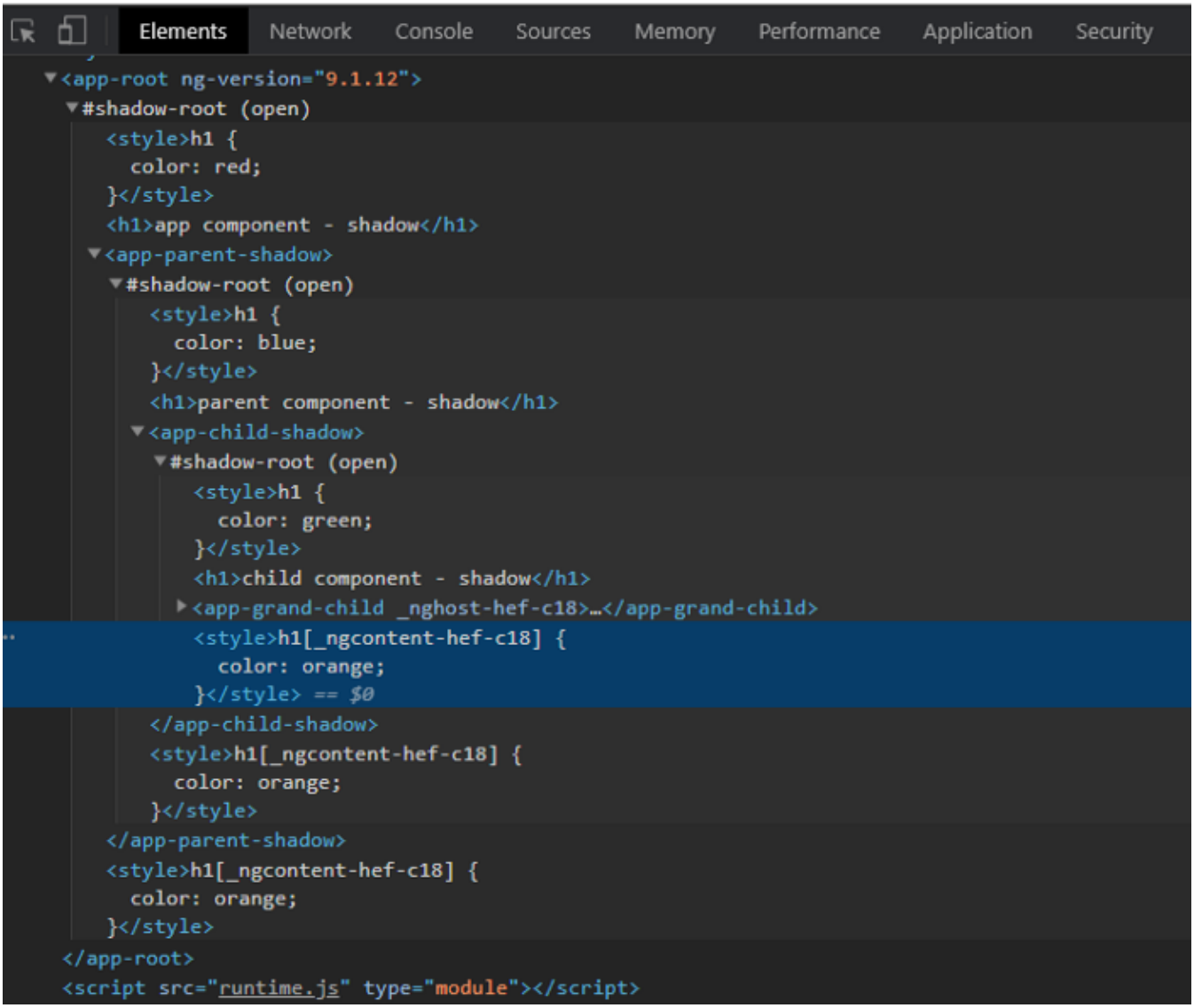
Now we will create the component tree in exactly that order. Each component has an <h1> tag that is given a color in its component.scss file. We would expect the shadow components to work, but the inner most (grandchild), which is regular/emulated should be blocked form his style in the <head> since he is deep down inside three shadowDom components.

app component - shadow

parent component - shadow

child component - shadow

grand-child component - emulated



It works! Grandchild.component gets his style. But how?

GrandChild.component does get its style (color: orange) **added** to <head> (unlike the

component in the module. In the above image, *color: orange* is only set in grandchild.component.scss file, but we see it in the shadow-root of all the components. In our case, grandchild was in the child.component so it got it from there, his direct parent, but even if we put grandchild in the app.component, angular would still create the same HTML. Just that the grandchild would get his css from app.conponent. Always from his parent.

This was just one line of css. What if there were many css rules and many components. I would argue against combining ShadowDom and emulated components.

Pro question: What would happen if grandchild.component had ViewEncapsulation.none instead of shadow?

Angular Material with ShadowDom

We can't talk about Angular without taking about Angular Material. Lets see how setting shadowDow works with Angular Material. This will review, summarize and confirm all the consents we've mentioned.

Lets install Angular Material into our project. The Angular semantics add command makes this a one liner.

```
ng add @angular/material
```

Lets just test it with the MatButtonModule Module. We'll add it to our ngModule imports array. So our test app ngModule looks like this

```
@NgModule({

declarations: [
 AppComponent,
 ParentShadowComponent,
 ChildShadowComponent,
```



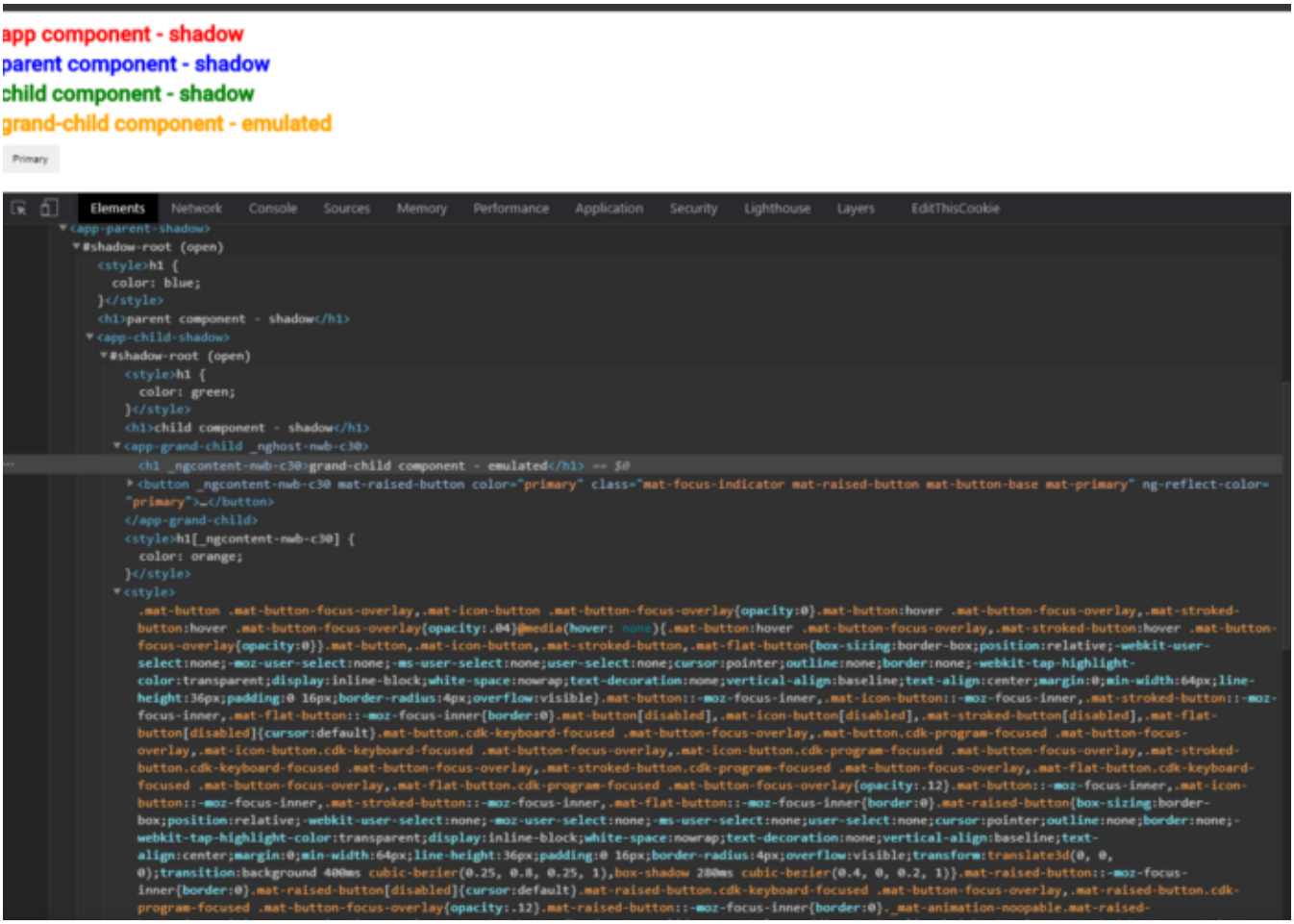

```
 BrowserAnimationsModule,
 MatButtonModule
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

We'll put a mat-button in the inner most grandchild component.

```
<button mat-raised-button color="primary">Primary</button>
```

If we set all components to the default view encapsulation every thing works as expected. The button has all its style, color and ripple.

However, if we set all the components to shadowDom, this is what we get..



style from there can't get them. This is the color and ripple among others. The Style that belong to the mat-button itself, gets injected into each and every components shadowDom just in case the button will be in that component. In our case the button is in the grandchild component so he's getting his style from that component. BTY, Angular Material uses ViewEncapsulation.none, but that doesn't matter in this case because they have quit specifically named and scoped there CSS.

So it really doesn't work. If the Angular Material team did away with the global style and put all the relevant styles in the components that would be a start. But still we would get huge duplication of styles. The real breakthrough would be an option to run the Angular Material components with shadowDom. That way there would be no problem and no duplication, beside the fact that each component would have its own style in side, so ten buttons would each have their own style tags (and not just one in <head> section). But i'm guessing its just a pointer to a reference.

In the next post we will explore the effects ShadowDom has on content projection (ng-content) and ng-templates.