22nd July 2019

Shadow DOM styling

Shadow DOM may include both `<style>` and `<link rel="stylesheet" href="…">` tags. In the latter case, stylesheets are HTTP-cached, so they are not redownloaded for multiple components that use same template.

As a general rule, local styles work only inside the shadow tree, and document styles work outside of it. But there are few exceptions.

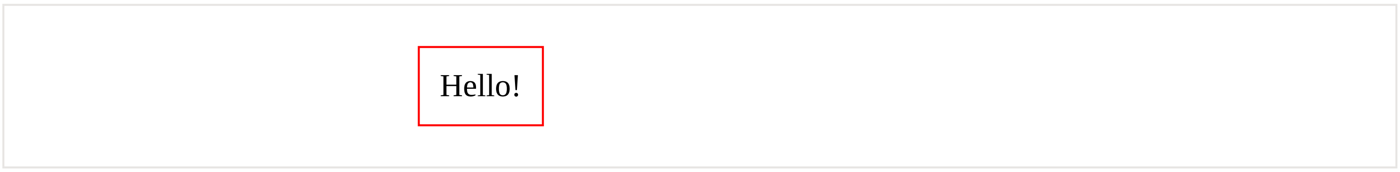
:host

The `:host` selector allows to select the shadow host (the element containing the shadow tree).

For instance, we're making `<custom-dialog>` element that should be centered. For that we need to style the `<custom-dialog>` element itself.

That's exactly what `:host` does:

```
<template id="tmpl">
  <style>
    /* the style will be applied from inside to the custom-dialog element */
    :host {
      position: fixed;
      left: 50%;
      top: 50%;
      transform: translate(-50%, -50%);
      display: inline-block;
      border: 1px solid red;
      padding: 10px;
    }
  </style>
  <slot></slot>
</template>

<script>
customElements.define('custom-dialog', class extends HTMLElement {
  connectedCallback() {
    this.attachShadow({mode: 'open'}).append(tmpl.content.cloneNode(true));
  }
});
</script>

<custom-dialog>
  Hello!
</custom-dialog>
```



Cascading

The shadow host (`<custom-dialog>` itself) resides in the light DOM, so it's affected by document CSS rules.

If there's a property styled both in `:host` locally, and in the document, then the document style takes precedence.

For instance, if in the document we had:

```
<style>
custom-dialog {
  padding: 0;
}
</style>
```

…Then the `<custom-dialog>` would be without padding.

It's very convenient, as we can setup "default" component styles in its `:host` rule, and then easily override them in the document.

The exception is when a local property is labelled `!important` , for such properties, local styles take precedence.

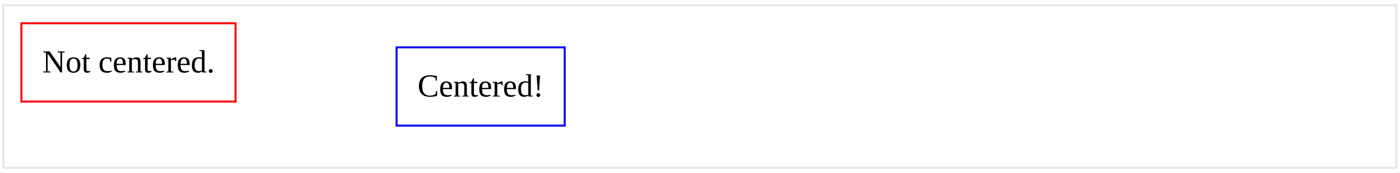
:host(selector)

Same as `:host` , but applied only if the shadow host matches the `selector` .

For example, we'd like to center the `<custom-dialog>` only if it has `centered` attribute:

```
<template id="tmpl">
  <style>
    :host([centered]) {
      position: fixed;
      left: 50%;
      top: 50%;
      transform: translate(-50%, -50%);
      border-color: blue;
    }

    :host {
      display: inline-block;
      border: 1px solid red;
      padding: 10px;
    }
  </style>
  <slot></slot>
</template>

<script>
```

```
customElements.define('custom-dialog', class extends HTMLElement {
  connectedCallback() {
    this.attachShadow({mode: 'open'}).append(tmpl.content.cloneNode(true));
  }
});
</script>


<custom-dialog centered>
  Centered!
</custom-dialog>

<custom-dialog>
  Not centered.
</custom-dialog>
```



Now the additional centering styles are only applied to the first dialog: `<custom-dialog centered>`.

:host-context(selector)

Same as `:host`, but applied only if the shadow host or any of its ancestors in the outer document matches the `selector`.

E.g. `:host-context(.dark-theme)` matches only if there's `dark-theme` class on `<custom-dialog>` on anywhere above it:

```
<body class="dark-theme">
  <!--
    :host-context(.dark-theme) applies to custom-dialogs inside .dark-theme
  -->
  <custom-dialog>...</custom-dialog>
</body>
```

To summarize, we can use `:host`-family of selectors to style the main element of the component, depending on the context. These styles (unless `!important`) can be overridden by the document.

Styling slotted content

Now let's consider the situation with slots.

Slotted elements come from light DOM, so they use document styles. Local styles do not affect slotted content.

In the example below, slotted `<span>` is bold, as per document style, but does not take `background` from the local style:

```
<style>
  span { font-weight: bold }
</style>

```

```
<user-card>
  <div slot="username"><span>John Smith</span></div>
</user-card>

<script>
customElements.define('user-card', class extends HTMLElement {
  connectedCallback() {
    this.attachShadow({mode: 'open'});
    this.shadowRoot.innerHTML = `
      <style>
      span { background: red; }
      </style>
      Name: <slot name="username"></slot>
    `;
  }
});
</script>
```

Name:
John Smith

The result is bold, but not red.

If we'd like to style slotted elements in our component, there are two choices.

First, we can style the `<slot>` itself and rely on CSS inheritance:

```
<user-card>
  <div slot="username"><span>John Smith</span></div>
</user-card>

<script>
customElements.define('user-card', class extends HTMLElement {
  connectedCallback() {
    this.attachShadow({mode: 'open'});
    this.shadowRoot.innerHTML = `
      <style>
      slot[name="username"] { font-weight: bold; }
      </style>
      Name: <slot name="username"></slot>
    `;
  }
});
</script>
```

Name:
John Smith

Here `<p>John Smith</p>` becomes bold, because CSS inheritance is in effect between the `<slot>` and its contents. But in CSS itself not all properties are inherited.

Another option is to use `::slotted(selector)` pseudo-class. It matches elements based on two conditions:

1. That’s a slotted element, that comes from the light DOM. Slot name doesn’t matter. Just any slotted element, but only the element itself, not its children.
2. The element matches the `selector`.

In our example, `::slotted(div)` selects exactly `<div slot="username">`, but not its children:

```
<user-card>
  <div slot="username">
    <div>John Smith</div>
  </div>
</user-card>

<script>
customElements.define('user-card', class extends HTMLElement {
  connectedCallback() {
    this.attachShadow({mode: 'open'});
    this.shadowRoot.innerHTML = `
      <style>
      ::slotted(div) { border: 1px solid red; }
      </style>
      Name: <slot name="username"></slot>
    `;
  }
});
</script>
```

```
Name:
John Smith
```

Please note, `::slotted` selector can’t descend any further into the slot. These selectors are invalid:

```
::slotted(div span) {
  /* our slotted <div> does not match this */
}

::slotted(div) p {
  /* can't go inside light DOM */
}
```

Also, `::slotted` can only be used in CSS. We can’t use it in `querySelector`.

CSS hooks with custom properties

How do we style internal elements of a component from the main document?

Selectors like `:host` apply rules to `<custom-dialog>` element or `<user-card>`, but how to style shadow DOM elements inside them?

There’s no selector that can directly affect shadow DOM styles from the document. But just as we expose methods to interact with our component, we can expose CSS variables (custom CSS properties) to style it.

Custom CSS properties exist on all levels, both in light and shadow.

For example, in shadow DOM we can use `--user-card-field-color` CSS variable to style fields, and the outer document can set its value:

```
<style>
  .field {
    color: var(--user-card-field-color, black);
    /* if --user-card-field-color is not defined, use black color */
  }
</style>
<div class="field">Name: <slot name="username"></slot></div>
<div class="field">Birthday: <slot name="birthday"></slot></div>
</style>
```

Then, we can declare this property in the outer document for `<user-card>`:

```
user-card {
  --user-card-field-color: green;
}
```

Custom CSS properties pierce through shadow DOM, they are visible everywhere, so the inner `.field` rule will make use of it.

Here's the full example:

```
<style>
  user-card {
    --user-card-field-color: green;
  }
</style>

<template id="tmpl">
  <style>
    .field {
      color: var(--user-card-field-color, black);
    }
  </style>
  <div class="field">Name: <slot name="username"></slot></div>
  <div class="field">Birthday: <slot name="birthday"></slot></div>
</template>

<script>
customElements.define('user-card', class extends HTMLElement {
  connectedCallback() {
    this.attachShadow({mode: 'open'});
    this.shadowRoot.append(document.getElementById('tmpl').content.cloneNode(
  }
});
</script>

<user-card>
  <span slot="username">John Smith</span>
  <span slot="birthday">01.01.2001</span>
</user-card>
```

Name: John Smith
Birthday: 01.01.2001

Summary

Shadow DOM can include styles, such as `<style>` or `<link rel="stylesheet">`.

Local styles can affect:

- shadow tree,
- shadow host with `:host` -family pseudoclasses,
- slotted elements (coming from light DOM), `::slotted(selector)` allows to select slotted elements themselves, but not their children.

Document styles can affect:

- shadow host (as it lives in the outer document)
- slotted elements and their contents (as that's also in the outer document)

When CSS properties conflict, normally document styles have precedence, unless the property is labelled as `!important`. Then local styles have precedence.

CSS custom properties pierce through shadow DOM. They are used as "hooks" to style the component:

1. The component uses a custom CSS property to style key elements, such as `var(--component-name-title, <default value>)`.
2. Component author publishes these properties for developers, they are same important as other public component methods.
3. When a developer wants to style a title, they assign `--component-name-title` CSS property for the shadow host or above.
4. Profit!

Previous lesson Next lesson

Share Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)