Web Components and Styles Scoping

by Mikhail Bashkirov @bashmish

for the FrontMania Conference
in the beautiful city of Utrecht, 2018

About Me

@bashmish

Work at FrontMen 👔

Frontend developer 😻🖥

Coffee geek ☕😎

Passionate traveller 🚀🌏🌍🌎

Trains and bikes lover 🚂❤🚴

OpenSource enthusiast 👩‍💻👨‍💻

@RadioJSPodcast cofounder 🎙

👉

how I used to look like up until 2018

There are only seven basic plots in the whole world - plots that are recycled again and again in novels, movies, plays and operas.

— The New York Times about Christopher Booker's "The Seven Basic Plots"

Quite recently
in a community near, near you...

SHADOW DOM

Episode I.
The Shadow Menace

Where we recap the history of styles scoping problem and how Shadow DOM appeared to solve it.

Naming conventions

Naming conventions

— BEM

Naming conventions

- BEM
- SMACSS

Naming conventions

- BEM
- SMACSS
- OOCSS

Naming conventions

- BEM
- SMACSS
- OOCSS
- SUIT CSS

Naming conventions

- BEM
- SMACSS
- OOCSS
- SUIT CSS
- AMCSS

Naming conventions

- BEM
- SMACSS
- OOCSS
- SUIT CSS
- AMCSS
- Atomic CSS

BEM - Block Element Modifier

```
<style>
  .block { ... }
  .block__element { ... }
  .block__element--modifier { ... }
  .block--modifier .block__element { ... }
</style>
<div class="block">
  <div class="block__element"></div>
  <div class="block__element block__element--modifier"></div>
</div>
<div class="block block--modifier">
  <div class="block__element"></div>
  <div class="block__element"></div>
</div>
```

BEM - Block Element Modifier

```
<style>
  .block { ... }
  .block__element { ... }
  .block__element--modifier { ... }
  .block--modifier .block__element { ... }
</style>
<div class="block">
  <div class="block__element"></div>
  <div class="block__element block__element--modifier"></div>
</div>
<div class="block block--modifier">
  <div class="block__element"></div>
  <div class="block__element"></div>
</div>
```

BEM - Block Element Modifier

```
<style>
  .block { ... }
  .block__element { ... }
  .block__element--modifier { ... }
  .block--modifier .block__element { ... }
</style>
<div class="block">
  <div class="block__element"></div>
  <div class="block__element block__element--modifier"></div>
</div>
<div class="block block--modifier">
  <div class="block__element"></div>
  <div class="block__element"></div>
</div>
```

BEM - Block Element Modifier

```
<style>
  .block { ... }
  .block__element { ... }
  .block__element--modifier { ... }
  .block--modifier .block__element { ... }
</style>
<div class="block">
  <div class="block__element"></div>
  <div class="block__element block__element--modifier"></div>
</div>
<div class="block block--modifier">
  <div class="block__element"></div>
  <div class="block__element"></div>
</div>
```

BEM - Block Element Modifier

```
<style>
  .block { ... }
  .block__element { ... }
  .block__element--modifier { ... }
  .block--modifier .block__element { ... }
</style>
<div class="block">
  <div class="block__element"></div>
  <div class="block__element block__element--modifier"></div>
</div>
<div class="block block--modifier">
  <div class="block__element"></div>
  <div class="block__element"></div>
</div>
```

BEM - Block Element Modifier

```
<style>
  .block { ... }
  .block__element { ... }
  .block__element--modifier { ... }
  .block--modifier .block__element { ... }
</style>
<div class="block">
  <div class="block__element"></div>
  <div class="block__element block__element--modifier"></div>
</div>
<div class="block block--modifier">
  <div class="block__element"></div>
  <div class="block__element"></div>
</div>
```

BEM + ITCSS = BEMIT

```
<style>
.c-name {} /* component */
.u-name {} /* utility */
.js-name {} /* to bind to it from JavaScript */
.qa-name {} /* for automated UI tests */
/* and others */
</style>
```

Read more in Harry Robert's blog:

- More Transparent UI Code with Namespaces
- BEMIT: Taking the BEM Naming Convention a Step Further

3RD PARTY BLOCK NAME COLLISIONS

<style> "scoped" atttribute

```
<article>
  <p>I have default styles</p>
  <section>
    <style scoped>
      p { ... }
    </style>
    <p>I have extra styles</p>
  </section>
</article>
```

<style> "scoped" atttribute

```
<article>
  <p>I have default styles</p>
  <section>
    <style scoped>
      p { ... }
    </style>
    <p>I have extra styles</p>
  </section>
</article>
```

<style> "scoped" atttribute

```
<article>
  <p>I have default styles</p>
  <section>
    <style scoped>
      p { ... }
    </style>
    <p>I have extra styles</p>
  </section>
</article>
```

<style> "scoped" atttribute

```
<article>
  <p>I have default styles</p>
  <section>
    <style scoped>
      p { ... }
    </style>
    <p>I have extra styles</p>
  </section>
</article>
```

<style> "scoped" atttribute

```
<article>
  <p>I have default styles</p>
  <section>
    <style scoped>
      p { ... }
    </style>
    <p>I have extra styles</p>
  </section>
</article>
```

<style> "scoped" atttribute

```
<article>
  <p>I have default styles</p>
  <section>
    <style scoped>
      p { ... }
    </style>
    <p>I have extra styles</p>
  </section>
</article>
```

Import external module

```
<article>
  <p>I have default styles</p>
  <section>
    <style scoped>
      @import url('path/to/typography-module.css');
    </style>
    <p>I have extra styles</p>
  </section>
</article>
```

Import external module

```
<article>
  <p>I have default styles</p>
  <section>
    <style scoped>
      @import url('path/to/typography-module.css');
    </style>
    <p>I have extra styles</p>
  </section>
</article>
```

Still cascade...

```
<article>
  <style scoped>p { display: none; }</style>
  <p>I'm not visible</p>
  <section>
    <style scoped>p { color: red; }</style>
    <p>I'm still not visible :(</p>
    <section>
      <style scoped>p { display: block; }</style>
      <p>I'm visible and red</p>
    </section>
  </section>
</article>
```

Still cascade...

```
<article>
  <style scoped>p { display: none; }</style>
  <p>I'm not visible</p>
  <section>
    <style scoped>p { color: red; }</style>
    <p>I'm still not visible :(</p>
    <section>
      <style scoped>p { display: block; }</style>
      <p>I'm visible and red</p>
    </section>
  </section>
</article>
```

Still cascade...

```
<article>
  <style scoped>p { display: none; }</style>
  <p>I'm not visible</p>
  <section>
    <style scoped>p { color: red; }</style>
    <p>I'm still not visible :(</p>
    <section>
      <style scoped>p { display: block; }</style>
      <p>I'm visible and red</p>
    </section>
  </section>
</article>
```

Still cascade...

```
<article>
  <style scoped>p { display: none; }</style>
  <p>I'm not visible</p>
  <section>
    <style scoped>p { color: red; }</style>
    <p>I'm still not visible :(</p>
    <section>
      <style scoped>p { display: block; }</style>
      <p>I'm visible and red</p>
    </section>
  </section>
</article>
```

Still cascade...

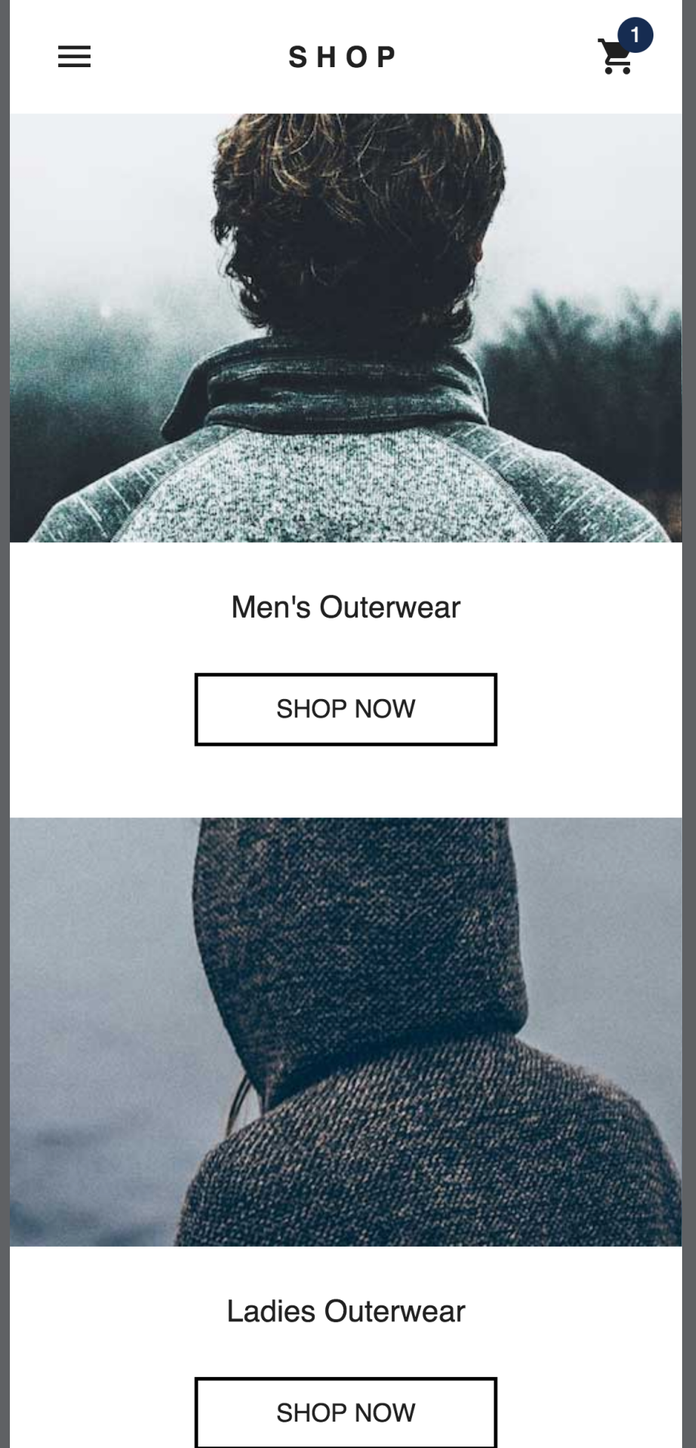
```
<article>
  <style scoped>p { display: none; }</style>
  <p>I'm not visible</p>
  <section>
    <style scoped>p { color: red; }</style>
    <p>I'm still not visible :(</p>
    <section>
      <style scoped>p { display: block; }</style>
      <p>I'm visible and red</p>
    </section>
  </section>
</article>
```

Want to learn more?

Read more about <style scoped>:
"The scoped attribute" on html5doctor.com

We live in components era

```
<shop-app>
  <app-header>
    <paper-icon-button></paper-icon-button>
    SHOP
    <paper-icon-button></paper-icon-button>
  </app-header>
  <shop-home>
    <shop-item>
      Men's Outerwear
      <shop-button></shop-button>
    </shop-item>
    <shop-item>
      Ladies Outerwear
      <shop-button></shop-button>
    </shop-item>
    ...
  </shop-home>
</shop-app>
```



so there was a need for a

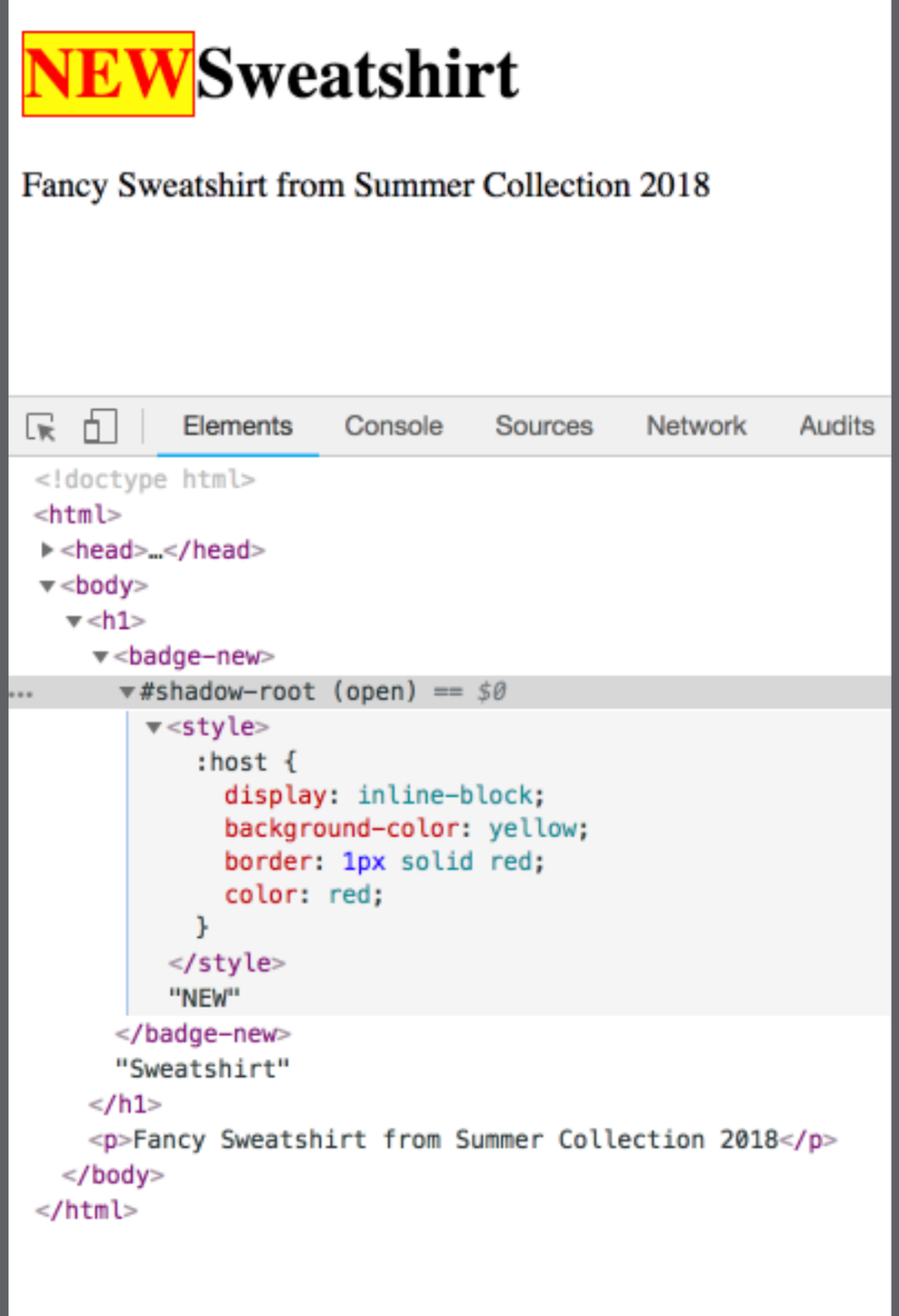
STANDARD

Shadow DOM - a method of establishing and maintaining functional boundaries between DOM trees and how these trees interact with each other within a document, thus enabling better functional encapsulation within the DOM & CSS.

— old definition from w3.org

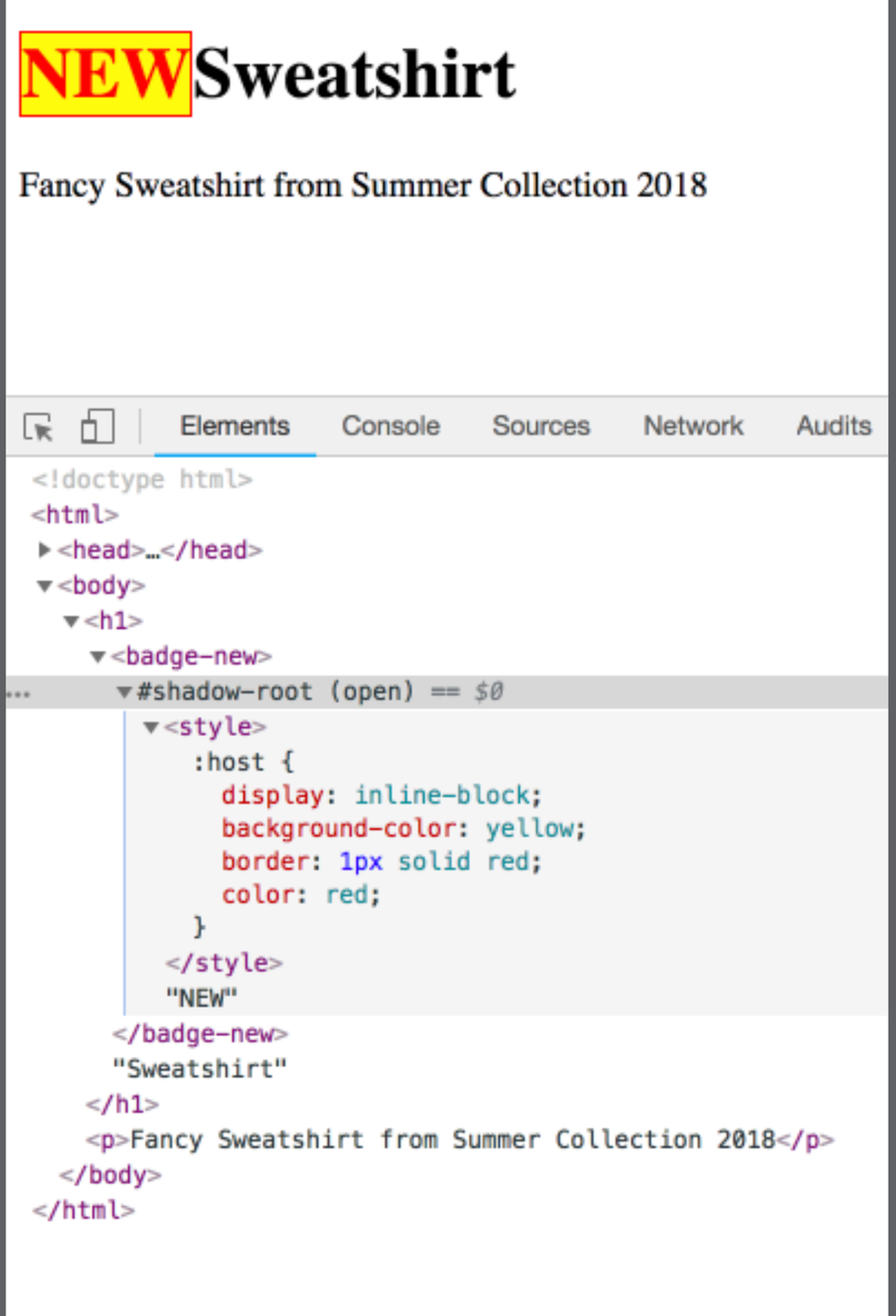
Create a component with Shadow DOM

```
customElements.define('badge-new', class extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```



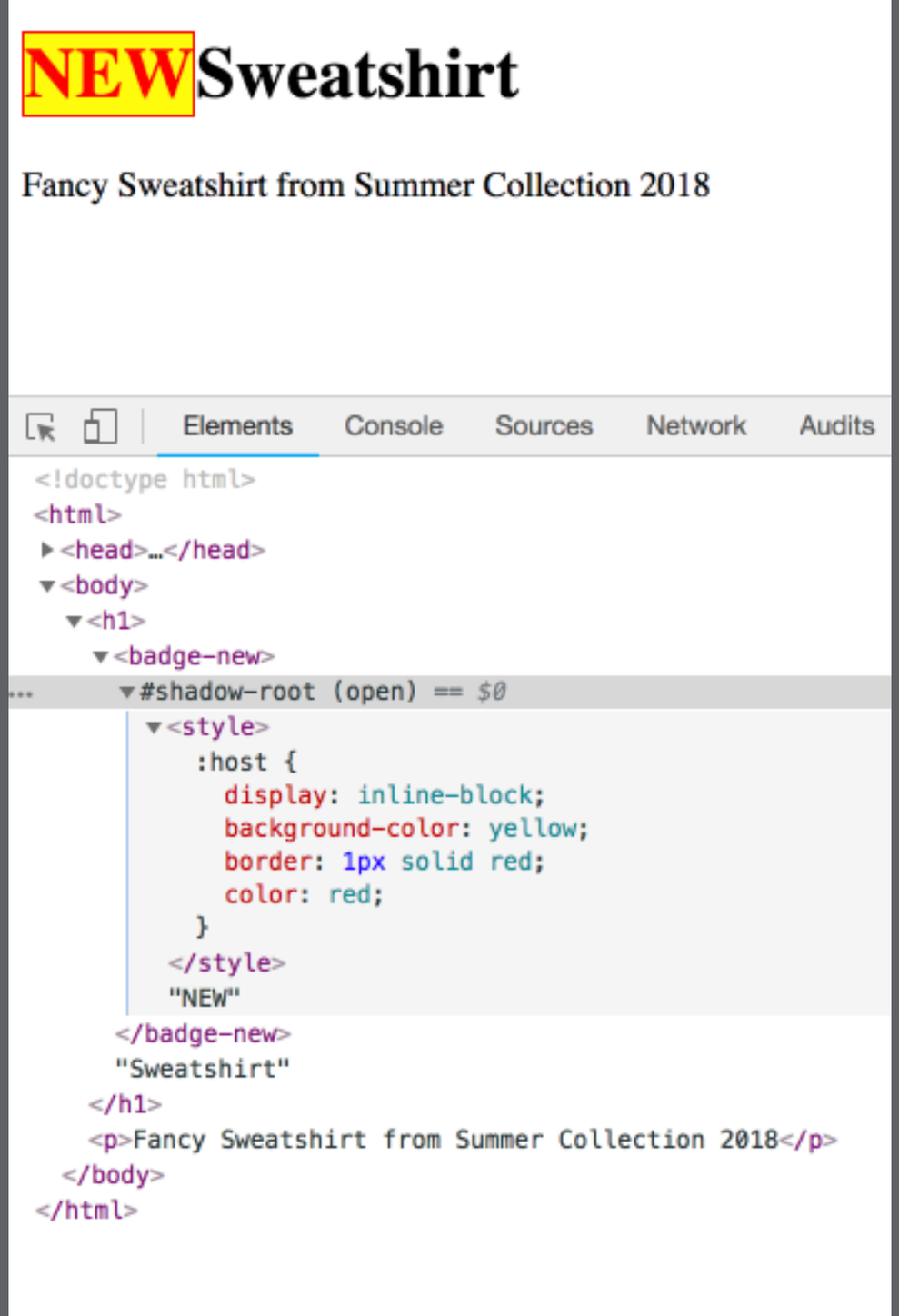
Create a component with Shadow DOM

```
customElements.define('badge-new', class extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```



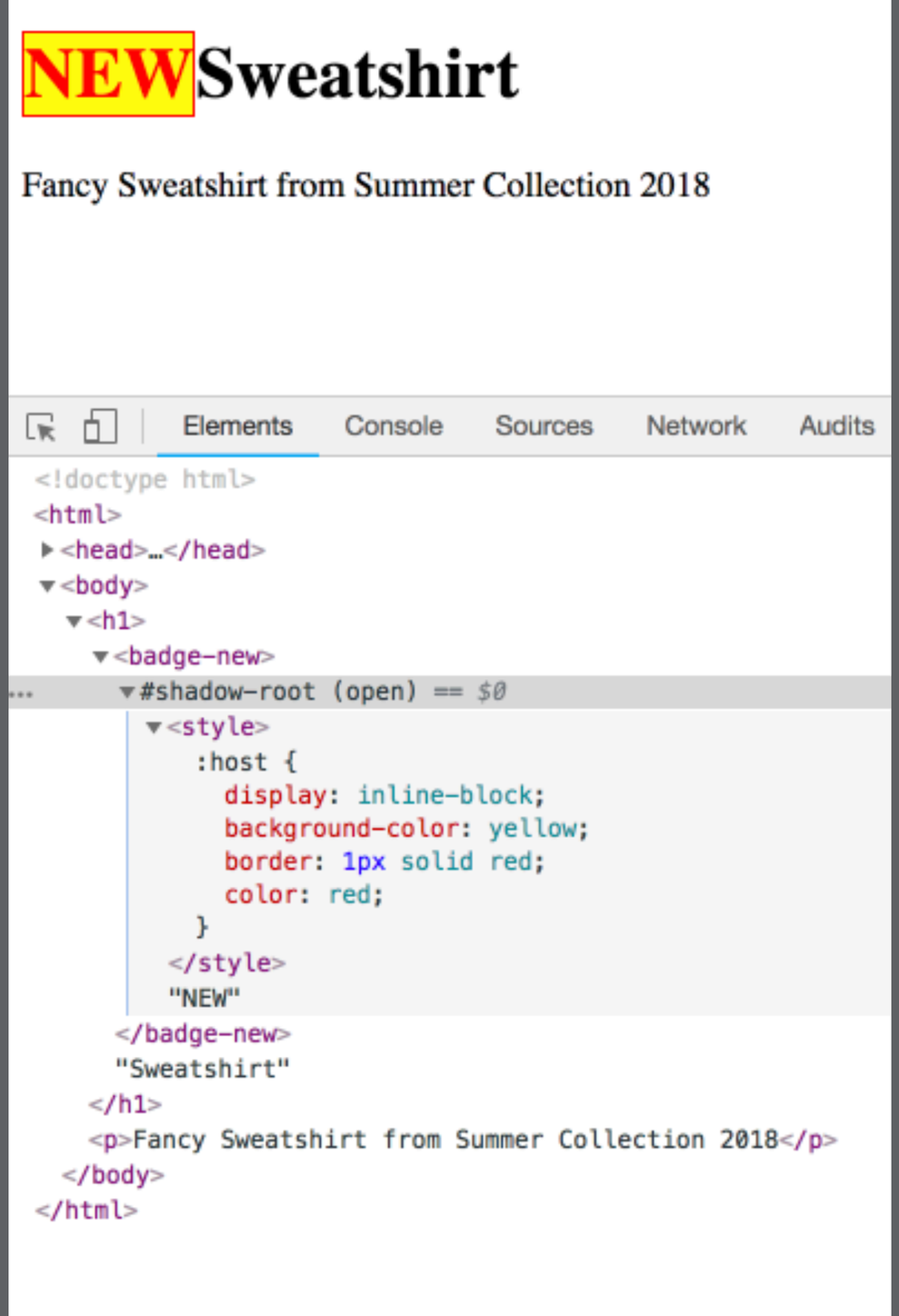
Create a component with Shadow DOM

```
customElements.define('badge-new', class extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```



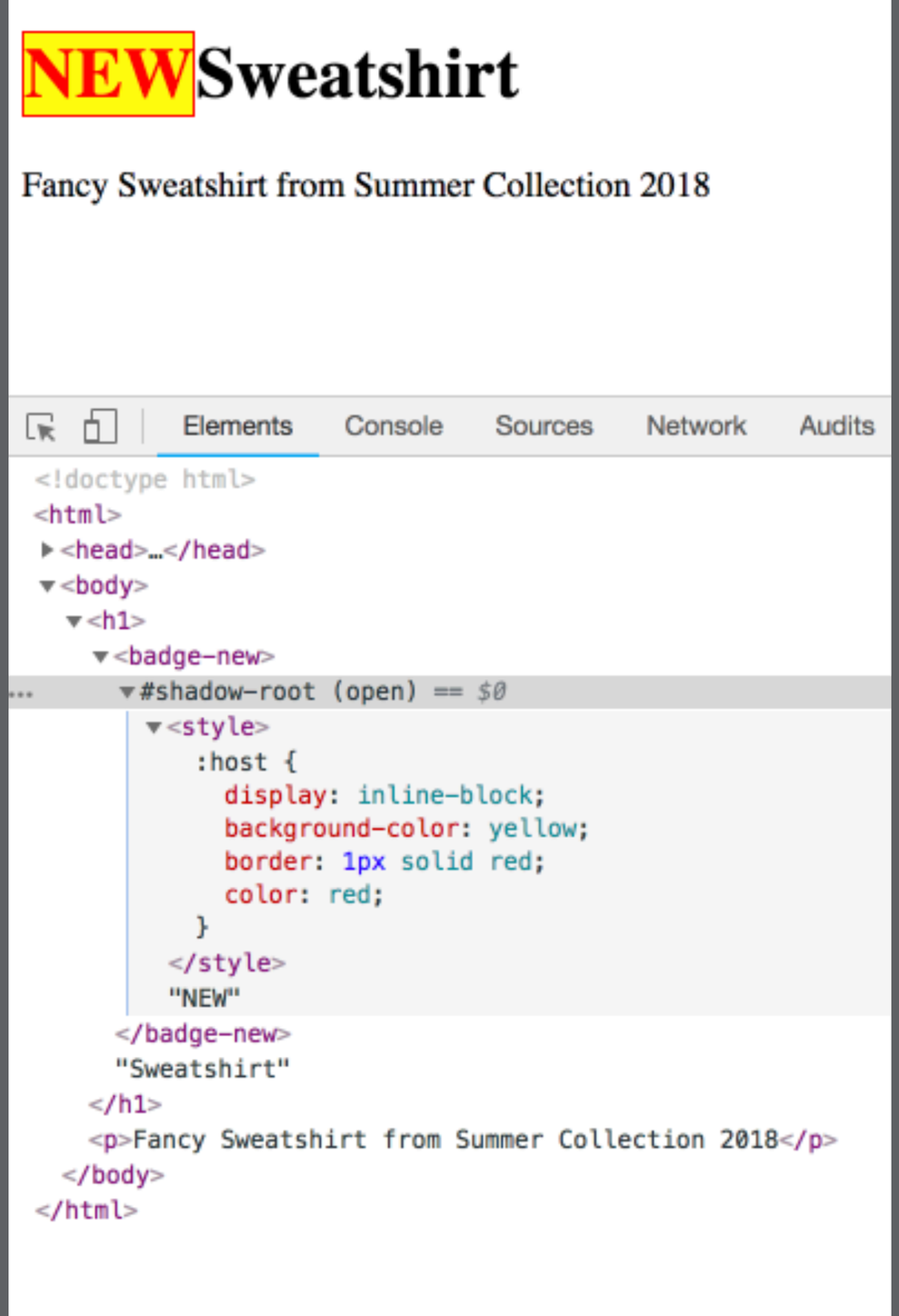
Create a component with Shadow DOM

```
customElements.define('badge-new', class extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```



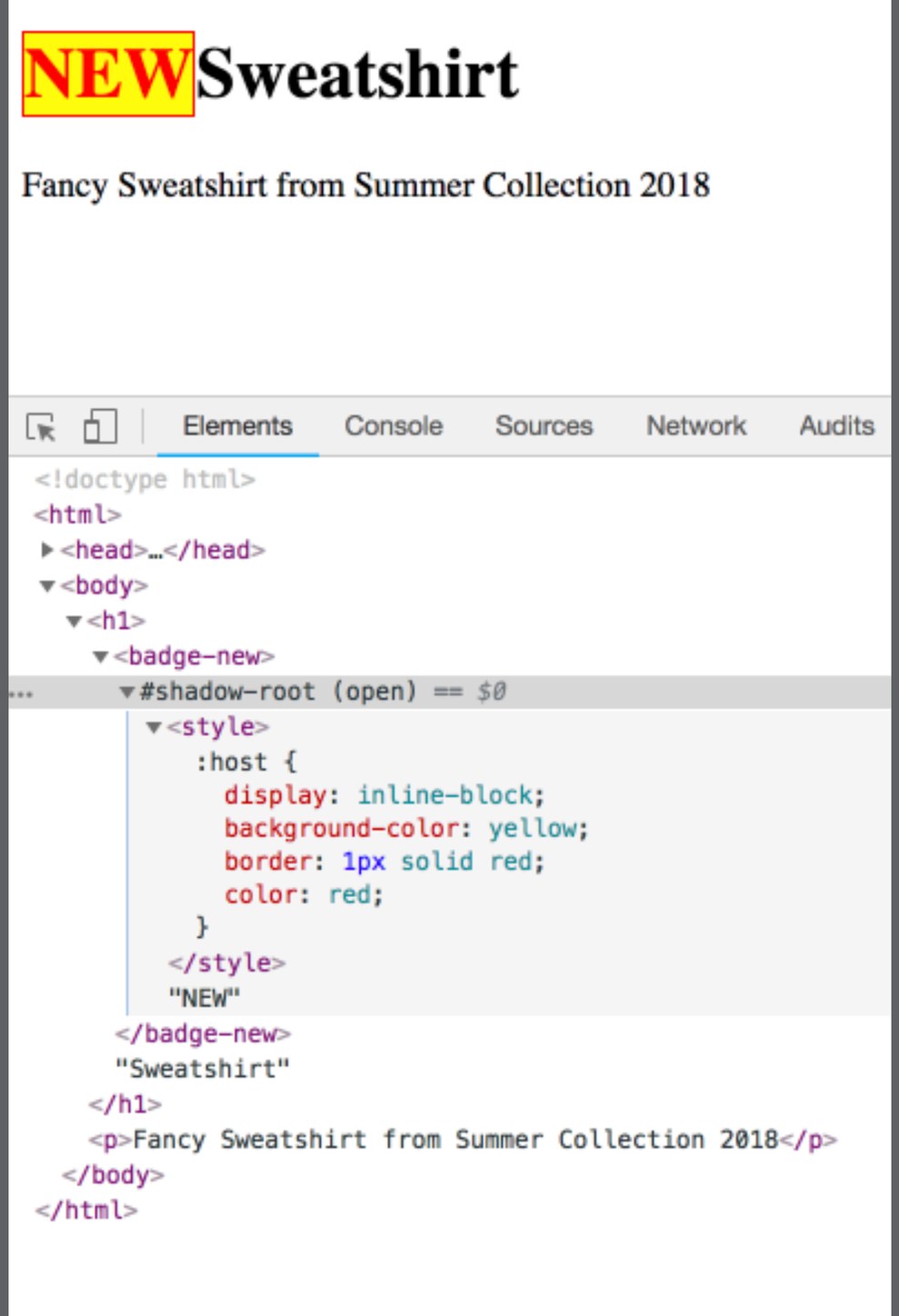
Create a component with Shadow DOM

```
customElements.define('badge-new', class extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```



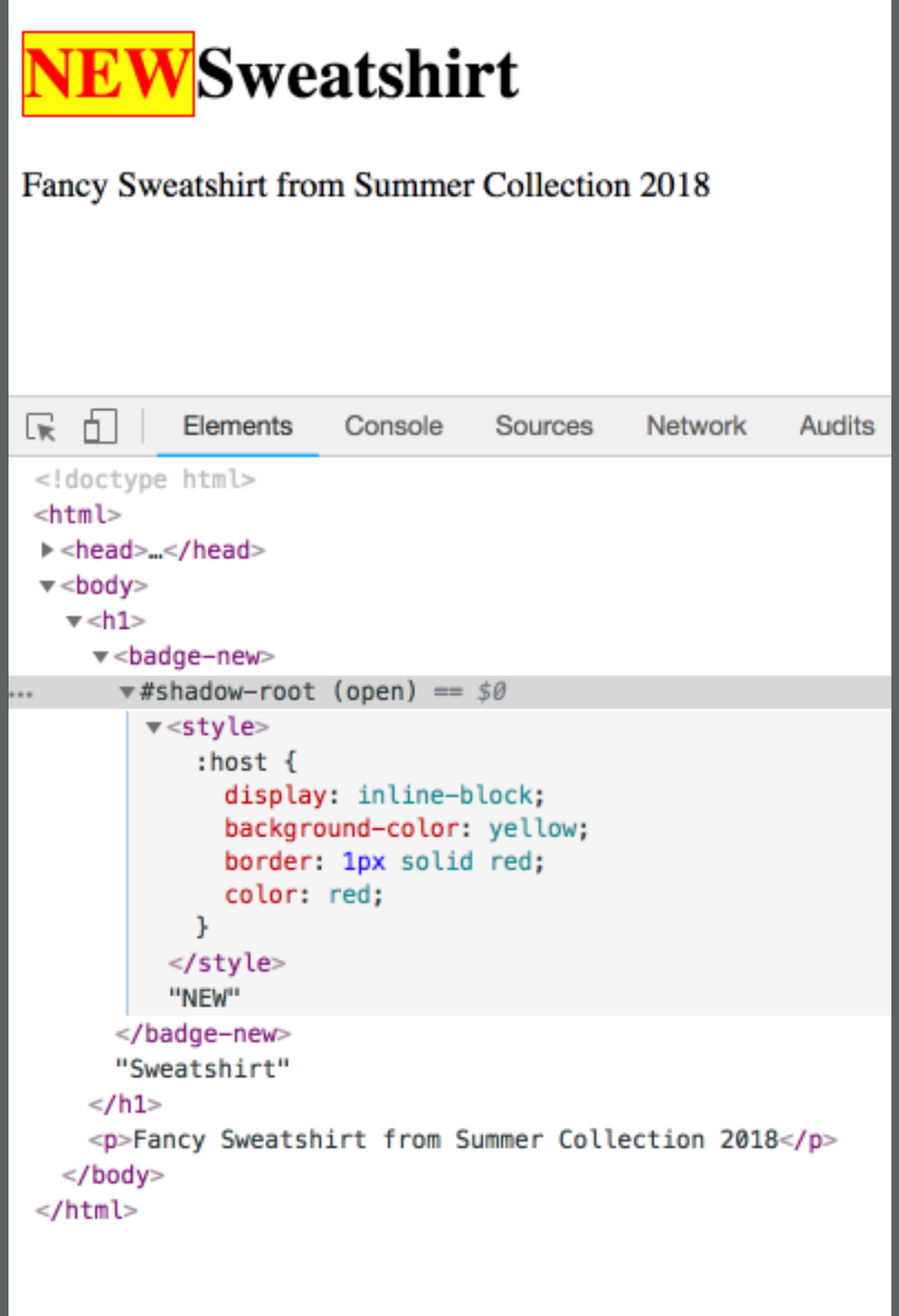
Create a component with Shadow DOM

```
customElements.define('badge-new', class extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```



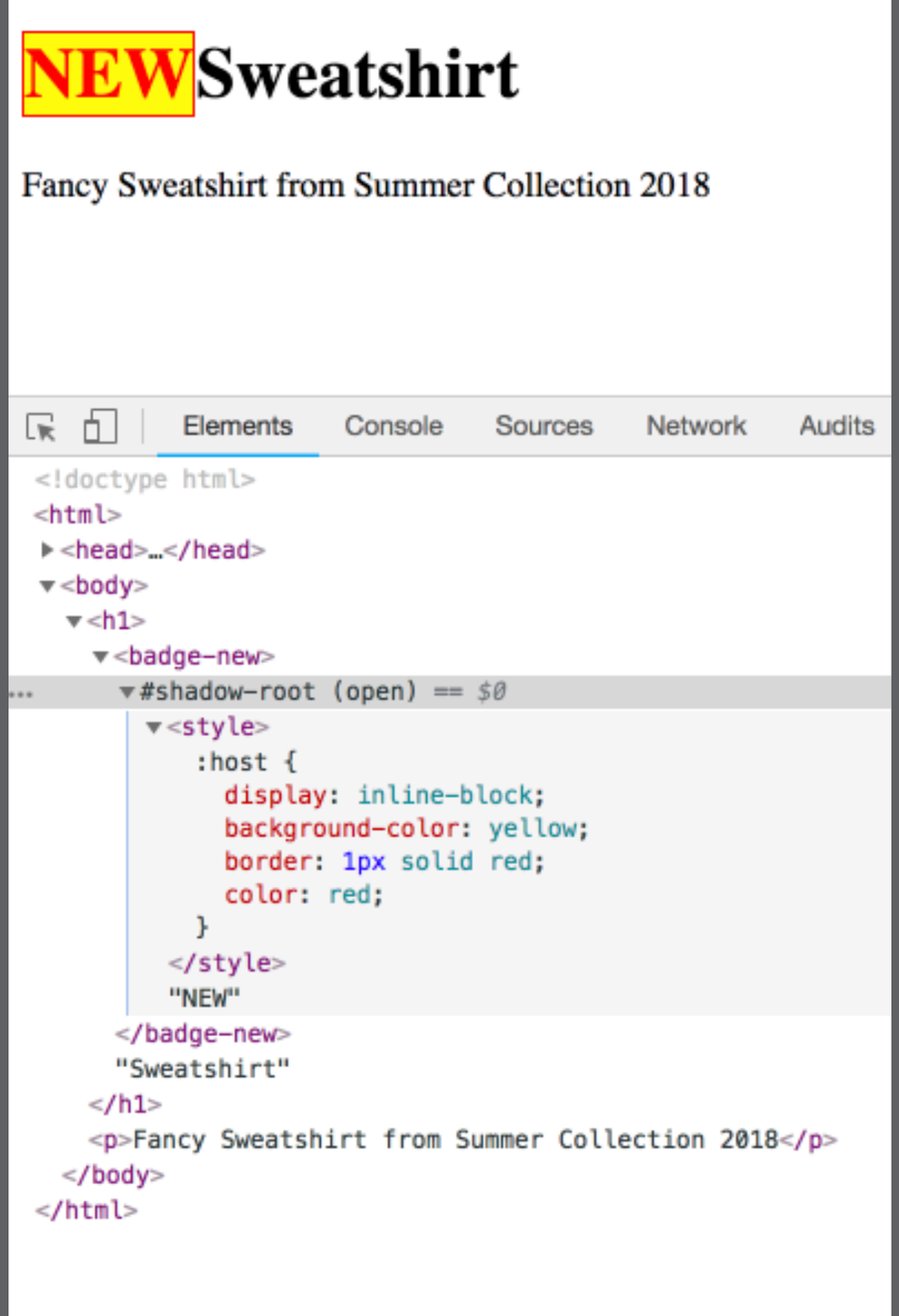
Create a component with Shadow DOM

```
customElements.define('badge-new', class extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```



Create a component with Shadow DOM

```
customElements.define('badge-new', class extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```



Same example with LitElement & lit-html

```
import { LitElement, html } from '@polymer/lit-element';

customElements.define('badge-new', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```

Same example with LitElement & lit-html

```
import { LitElement, html } from '@polymer/lit-element';

customElements.define('badge-new', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```

Same example with LitElement & lit-html

```
import { LitElement, html } from '@polymer/lit-element';

customElements.define('badge-new', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```

Same example with LitElement & lit-html

```
import { LitElement, html } from '@polymer/lit-element';

customElements.define('badge-new', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```

Same example with LitElement & lit-html

```
import { LitElement, html } from '@polymer/lit-element';

customElements.define('badge-new', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```

Same example with LitElement & lit-html

```
import { LitElement, html } from '@polymer/lit-element';

customElements.define('badge-new', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```

Can be on any HTML element (almost)

```
<article>
  #shadow-root
    <style>p { display: none; }</style>
    <p>I'm not visible</p>
    <section>
      #shadow-root
        <style>p { color: red; }</style>
        <p>I'm red</p>
        <section>
          #shadow-root
            <style>p { display: block; }</style>
            <p>I'm visible and have a default color</p>
        </section>
    </section>
</article>
```

Can be on any HTML element (almost)

```
<article>
  #shadow-root
    <style>p { display: none; }</style>
    <p>I'm not visible</p>
    <section>
      #shadow-root
        <style>p { color: red; }</style>
        <p>I'm red</p>
        <section>
          #shadow-root
            <style>p { display: block; }</style>
            <p>I'm visible and have a default color</p>
        </section>
    </section>
</article>
```

Can be on any HTML element (almost)

```
<article>
  #shadow-root
    <style>p { display: none; }</style>
    <p>I'm not visible</p>
    <section>
      #shadow-root
        <style>p { color: red; }</style>
        <p>I'm red</p>
        <section>
          #shadow-root
            <style>p { display: block; }</style>
            <p>I'm visible and have a default color</p>
        </section>
    </section>
</article>
```

Can be on any HTML element (almost)

```
<article>
  #shadow-root
    <style>p { display: none; }</style>
    <p>I'm not visible</p>
    <section>
      #shadow-root
        <style>p { color: red; }</style>
        <p>I'm red</p>
        <section>
          #shadow-root
            <style>p { display: block; }</style>
            <p>I'm visible and have a default color</p>
        </section>
    </section>
</article>
```

Shadow DOM & Co.

Light DOM and <slot>

```
<my-alert-info>
  <!-- your Light DOM -->
  You have 2 new messages.
</my-alert-info>
<script>
customElements.define('my-alert-info', class extends LitElement {
  render() {
    return html`
      <style>
        :host { /* style alert */ }
      </style>
      <slot><!-- Light DOM "You have 2 new messages." is projected to this slot --></slot>
    `;
  }
});
</script>
```

Light DOM and <slot>

```
<my-alert-info>
  <!-- your Light DOM -->
  You have 2 new messages.
</my-alert-info>
<script>
customElements.define('my-alert-info', class extends LitElement {
  render() {
    return html`
      <style>
        :host { /* style alert */ }
      </style>
      <slot><!-- Light DOM "You have 2 new messages." is projected to this slot --></slot>
    `;
  }
});
</script>
```

Light DOM and <slot>

```
<my-alert-info>
  <!-- your Light DOM -->
  You have 2 new messages.
</my-alert-info>
<script>
customElements.define('my-alert-info', class extends LitElement {
  render() {
    return html`
      <style>
        :host { /* style alert */ }
      </style>
      <slot><!-- Light DOM "You have 2 new messages." is projected to this slot --></slot>
    `;
  }
});
</script>
```

Light DOM and <slot>

```
<my-alert-info>
  <!-- your Light DOM -->
  You have 2 new messages.
</my-alert-info>
<script>
customElements.define('my-alert-info', class extends LitElement {
  render() {
    return html`
      <style>
        :host { /* style alert */ }
      </style>
      <slot><!-- Light DOM "You have 2 new messages." is projected to this slot --></slot>
    `;
  }
});
</script>
```

CSS Custom Properties (a.k.a. CSS vars)

```
<my-alert type="info">You have 2 new messages.</my-alert>
<my-alert type="error">Message has not been sent.</my-alert>
<script>
customElements.define('my-alert', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          background-color: var(--alert-bg-color);
          color: white;
        }
        :host([type="info"]) {
          --alert-bg-color: lightblue;
        }
        :host([type="error"]) {
          --alert-bg-color: antiqueWhite;
        }
      </style>
      <slot></slot>
    `;
  }
});
</script>
```

CSS Custom Properties (a.k.a. CSS vars)

```
<my-alert type="info">You have 2 new messages.</my-alert>
<my-alert type="error">Message has not been sent.</my-alert>
<script>
customElements.define('my-alert', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          background-color: var(--alert-bg-color);
          color: white;
        }
        :host([type="info"]) {
          --alert-bg-color: lightblue;
        }
        :host([type="error"]) {
          --alert-bg-color: antiqueWhite;
        }
      </style>
      <slot></slot>
    `;
  }
});
</script>
```

CSS Custom Properties (a.k.a. CSS vars)

```
<my-alert type="info">You have 2 new messages.</my-alert>
<my-alert type="error">Message has not been sent.</my-alert>
<script>
customElements.define('my-alert', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          background-color: var(--alert-bg-color);
          color: white;
        }
        :host([type="info"]) {
          --alert-bg-color: lightblue;
        }
        :host([type="error"]) {
          --alert-bg-color: antiqueWhite;
        }
      </style>
      <slot></slot>
    `;
  }
});
</script>
```

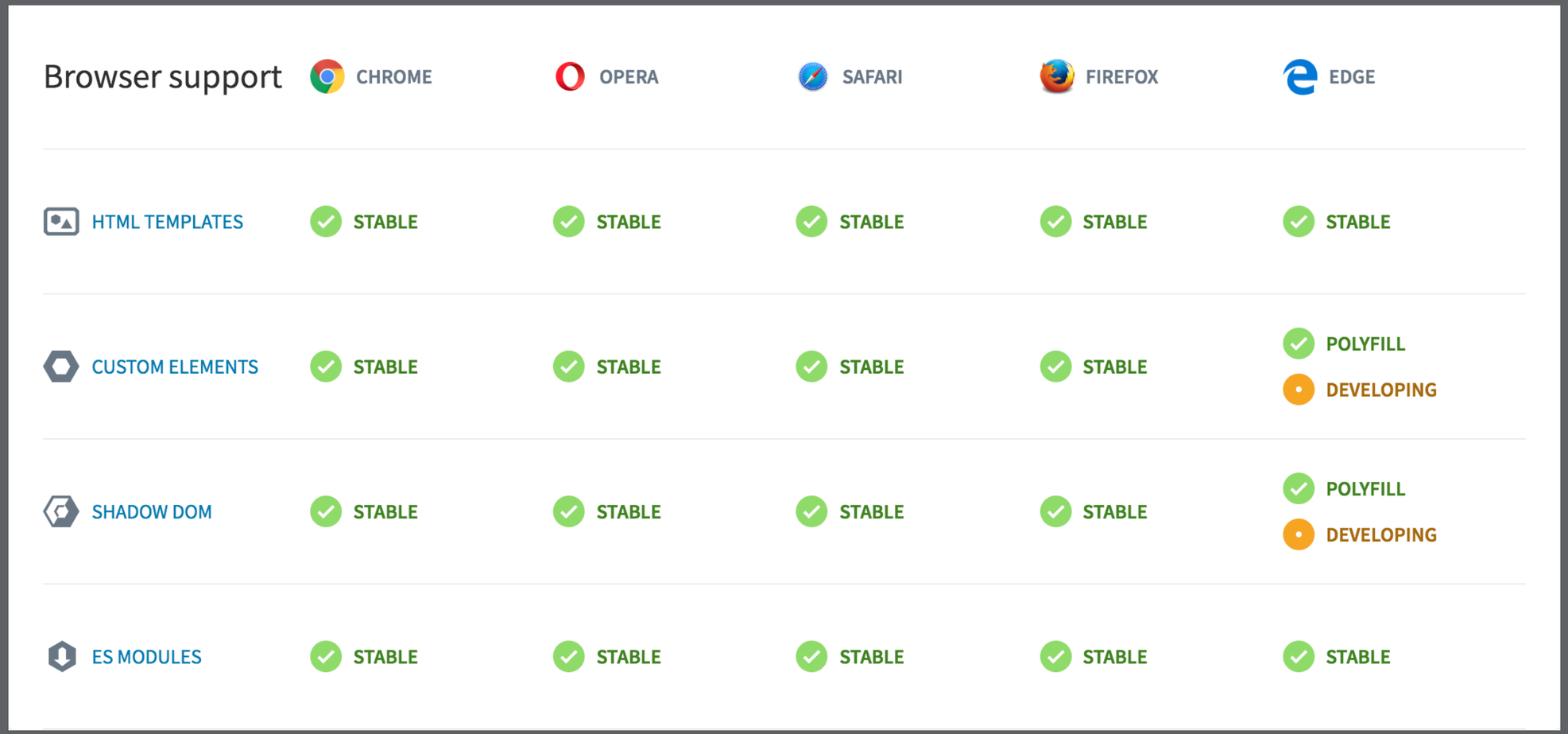
CSS Custom Properties (a.k.a. CSS vars)

```
<my-alert type="info">You have 2 new messages.</my-alert>
<my-alert type="error">Message has not been sent.</my-alert>
<script>
customElements.define('my-alert', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          background-color: var(--alert-bg-color);
          color: white;
        }
        :host([type="info"]) {
          --alert-bg-color: lightblue;
        }
        :host([type="error"]) {
          --alert-bg-color: antiqueWhite;
        }
      </style>
      <slot></slot>
    `;
  }
});
</script>
```

CSS Custom Properties (a.k.a. CSS vars)

```
<my-alert type="info">You have 2 new messages.</my-alert>
<my-alert type="error">Message has not been sent.</my-alert>
<script>
customElements.define('my-alert', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          background-color: var(--alert-bg-color);
          color: white;
        }
        :host([type="info"]) {
          --alert-bg-color: lightblue;
        }
        :host([type="error"]) {
          --alert-bg-color: antiqueWhite;
        }
      </style>
      <slot></slot>
    `;
  }
});
</script>
```

Support in browsers



Users having browsers with Shadow DOM

	Full	Partial	Total
All users	67.75%	12.74%	80.49%
All tracked	70.56%	13.27%	83.83%
Tracked desktop	69.93%	5.48%	*75.41%*
Tracked mobile	71.12%	20.04%	*91.16%*

According to caniuse.com on November 12, 2018.

Partial is mostly thanks to Safari (including iOS) bugs in selectors like **:host > .local-child** and **::slotted()**.

Episode II.
Attack of the Frameworks

Where we compare Shadow DOM with modern frameworks and solutions.

Angular scoped styles

```
@Component({
  selector: 'my-component',
  styles: [`
    .btn { ... }
  `],
  template: `
    <button class="btn"></button>
  `,
})
export class MyComponent {}
```

Angular supports modules

```
@Component({
  selector: 'my-component',
  styles: [`
    @import 'path/to/button.css';
  `],
  template: `
    <button class="btn"></button>
  `,
})
export class MyComponent {}
```

Vue.js <style scoped>

```
<style scoped>
  .btn { ... }
</style>

<template>
  <button class="btn">Push me</button>
</template>
```

Vue.js supports modules

```
<style scoped>
  @import 'path/to/button.css';
</style>

<template>
  <button class="btn">Touch me</button>
</template>
```

Vue.js styles can be non-scoped

```
<!-- App.vue -->
<style>
  body {
    background-color: grey;
  }
</style>
<template>
  <aside class="left">...</aside>
  <article class="main">...</article>
</template>
```

CSS-in-JS

```
import { getClassForStyle } from 'imaginary-css-in-js';

const buttonStyle = {
  backgroundColor: 'blue',
  border: 'none',
  color: 'white',
  fontSize: '1rem',
  padding: '0.5rem',
};

const buttonClass = getClassForStyle(buttonStyle);

document.querySelector('button').classList.add(buttonClass);
```

CSS-in-JS

```
import { getClassForStyle } from 'imaginary-css-in-js';

const buttonStyle = {
  backgroundColor: 'blue',
  border: 'none',
  color: 'white',
  fontSize: '1rem',
  padding: '0.5rem',
};

const buttonClass = getClassForStyle(buttonStyle);

document.querySelector('button').classList.add(buttonClass);
```

CSS-in-JS

```
import { getClassForStyle } from 'imaginary-css-in-js';

const buttonStyle = {
  backgroundColor: 'blue',
  border: 'none',
  color: 'white',
  fontSize: '1rem',
  padding: '0.5rem',
};

const buttonClass = getClassForStyle(buttonStyle);

document.querySelector('button').classList.add(buttonClass);
```

CSS-in-JS

```
import { getClassForStyle } from 'imaginary-css-in-js';

const buttonStyle = {
  backgroundColor: 'blue',
  border: 'none',
  color: 'white',
  fontSize: '1rem',
  padding: '0.5rem',
};

const buttonClass = getClassForStyle(buttonStyle);

document.querySelector('button').classList.add(buttonClass);
```

CSS-in-JS

```
import { getClassForStyle } from 'imaginary-css-in-js';

const buttonStyle = {
  backgroundColor: 'blue',
  border: 'none',
  color: 'white',
  fontSize: '1rem',
  padding: '0.5rem',
};

const buttonClass = getClassForStyle(buttonStyle);

document.querySelector('button').classList.add(buttonClass);
```

CSS-in-JS

```
import { getClassForStyle } from 'imaginary-css-in-js';

const buttonStyle = {
  backgroundColor: 'blue',
  border: 'none',
  color: 'white',
  fontSize: '1rem',
  padding: '0.5rem',
};

const buttonClass = getClassForStyle(buttonStyle);

document.querySelector('button').classList.add(buttonClass);
```

CSS-in-JS ❤️ React

CSS-in-JS ❤️ React

— aesthetic

CSS-in-JS ❤️ React

— aesthetic

— aphrodite

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs
- emotion

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs
- emotion
- fela

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs
- emotion
- fela
- glam

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs
- emotion
- fela
- glam
- glamor

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs
- emotion
- fela
- glam
- glamor
- glamorous

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs
- emotion
- fela
- glam
- glamor
- glamorous
- JSS

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs
- emotion
- fela
- glam
- glamor
- glamorous
- JSS
- radium

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs
- emotion
- fela
- glam
- glamor
- glamorous
- JSS
- radium
- styled-components

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs
- emotion
- fela
- glam
- glamor
- glamorous
- JSS
- radium
- styled-components
- styletron

CSS-in-JS ❤️ React

— aesthetic
— aphrodite
— cxs
— emotion
— fela
— glam
— glamor
— glamorous
— JSS
— radium
— styled-components
— styletron
— CSS Modules (conceptually NOT, but technically more like YES)

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs
- emotion
- fela
- glam
- glamor
- glamorous
- JSS
- radium
- styled-components
- styletron
- CSS Modules (conceptually NOT, but technically more like YES)
- JSS even implements other interfaces

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs
- emotion
- fela
- glam
- glamor
- glamorous
- JSS
- radium
- styled-components
- styletron
- CSS Modules (conceptually NOT, but technically more like YES)
- JSS even implements other interfaces
  - aphrodite-jss

CSS-in-JS ❤️ React

- aesthetic
- aphrodite
- cxs
- emotion
- fela
- glam
- glamor
- glamorous
- JSS
- radium
- styled-components
- styletron
- CSS Modules (conceptually NOT, but technically more like YES)
- JSS even implements other interfaces
  - aphrodite-jss
  - styled-jss

TOO MANY SOLUTIONS

TOO MANY

INCOMPATIBLE

SOLUTIONS

Want to learn more?

How they differ and why you can find out here:

github.com/tuchk4/awesome-css-in-js

styled-components 💅

```
import React from 'react';
import styled from 'styled-components';

const Button = styled.button`
  background-color: blue;
  border: none;
  color: white;
  font-size: 1rem;
  padding: 0.5rem;
`;

function MyComponent() {
  return <Button>Push me</Button>;
}
```

styled-components 💅

```
import React from 'react';
import styled from 'styled-components';

const Button = styled.button`
  background-color: blue;
  border: none;
  color: white;
  font-size: 1rem;
  padding: 0.5rem;
`;

function MyComponent() {
  return <Button>Push me</Button>;
}
```

styled-components 💅

```
import React from 'react';
import styled from 'styled-components';

const Button = styled.button`
  background-color: blue;
  border: none;
  color: white;
  font-size: 1rem;
  padding: 0.5rem;
`;

function MyComponent() {
  return <Button>Push me</Button>;
}
```

styled-components 💅

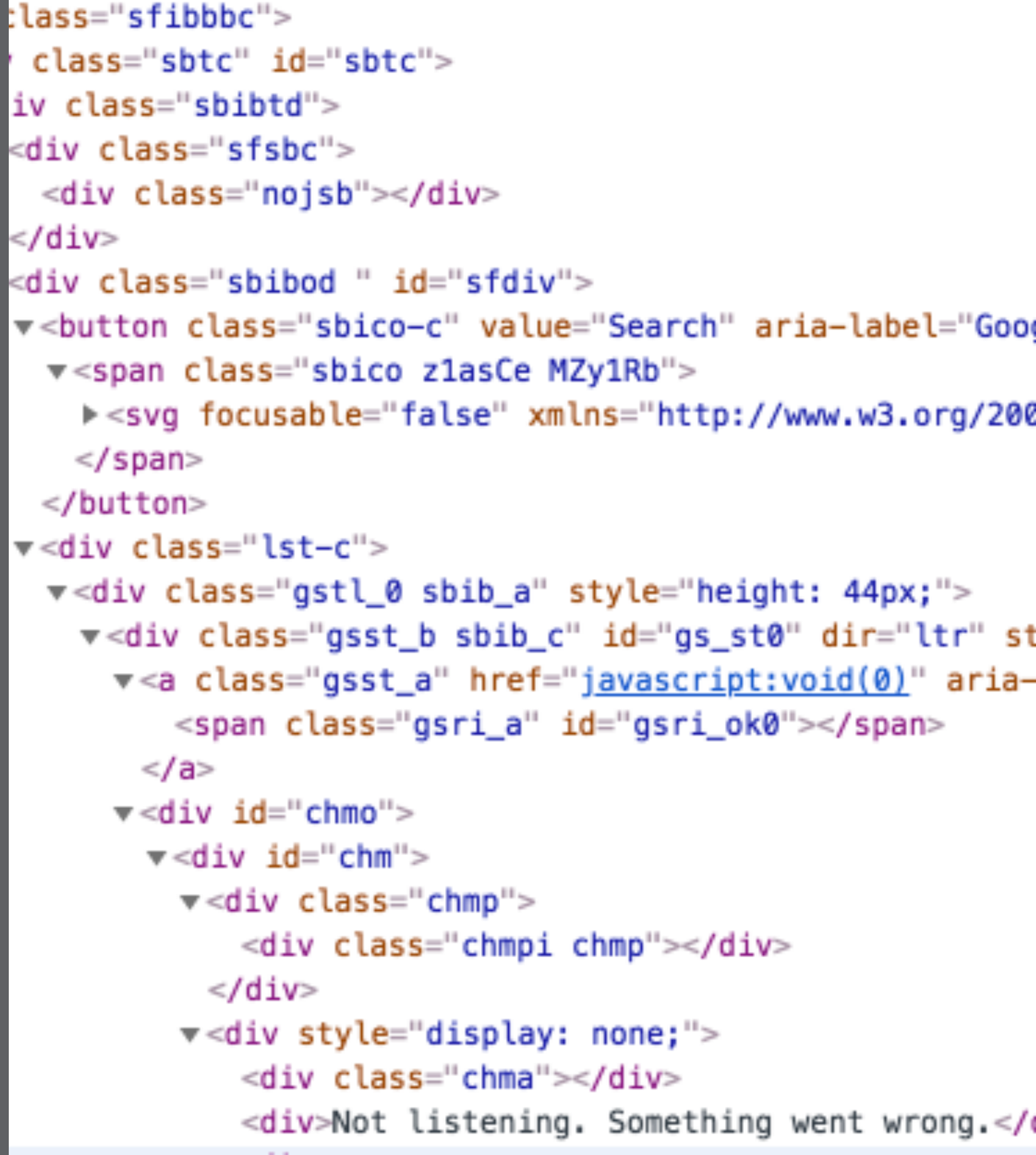
```
import React from 'react';
import styled from 'styled-components';

const Button = styled.button`
  background-color: blue;
  border: none;
  color: white;
  font-size: 1rem;
  padding: 0.5rem;
`;

function MyComponent() {
  return <Button>Push me</Button>;
}
```

Debugging is often hard 😓

Especially:
insertRule DevTools issue



Does this guy

dislike

CSS-in-JS because

he has to #usetheplatform?

You adore any technology up until you worked with it long enough time. 🤷‍♂️

INTEROPERABILITY IS KEY

First attempts to standardize

First attempts to standardize

— **fela** author Robin Frischmann attempts to make proper abstraction layers

First attempts to standardize

— **fela** author Robin Frischmann attempts to make proper abstraction layers

  — fela - the core

First attempts to standardize

- **fela** author Robin Frischmann attempts to make proper abstraction layers
  - fela - the core
  - fela-dom - rendering to DOM

First attempts to standardize

- **fela** author Robin Frischmann attempts to make proper abstraction layers
  - fela - the core
  - fela-dom - rendering to DOM
  - bindings: fela-react, fela-preact, fela-inferno, cycle-fela

First attempts to standardize

- **fela** author Robin Frischmann attempts to make proper abstraction layers
  - fela - the core
  - fela-dom - rendering to DOM
  - bindings: fela-react, fela-preact, fela-inferno, cycle-fela
  - fela-native - for React Native

First attempts to standardize

- **fela** author Robin Frischmann attempts to make proper abstraction layers
  - fela - the core
  - fela-dom - rendering to DOM
  - bindings: fela-react, fela-preact, fela-inferno, cycle-fela
  - fela-native - for React Native
- **JSS** author Oleg Slobodskoi attempts to standardize distribution format

First attempts to standardize

- **fela** author Robin Frischmann attempts to make proper abstraction layers
  - fela - the core
  - fela-dom - rendering to DOM
  - bindings: fela-react, fela-preact, fela-inferno, cycle-fela
  - fela-native - for React Native
- **JSS** author Oleg Slobodskoi attempts to standardize distribution format
  - Interoperable Style Transfer Format

First attempts to standardize

- **fela** author Robin Frischmann attempts to make proper abstraction layers
  - fela - the core
  - fela-dom - rendering to DOM
  - bindings: fela-react, fela-preact, fela-inferno, cycle-fela
  - fela-native - for React Native
- **JSS** author Oleg Slobodskoi attempts to standardize distribution format
  - Interoperable Style Transfer Format
  - Unique Value Proposition of CSSinJS and the Future @ ReactiveConf 2017

Want to learn more?

My experiments with CSS-in-JS in Web Components: github.com/bashmish/wc-css-in-js-sandbox

Episode III.
Revenge of the CSS

Where we loose our hope
for Shadow DOM and Web Components.

Web components are a complete waste of time and effort at this point. The champions of web components are too invested to admit it, but there are so many problems with the current spec. They haven't learned anything from the current state of the art of building UI for the web.

— @mjackson tweet on October 12, 2018

everything comes with a

PRICE

... NO SUPPORT FOR SSR

ATM NO SUPPORT FOR SSR

Want to learn more?

SkateJS author Trey Shugart wrote:

- a blog post Server-side rendering web components
- a module for SkateJS @skatejs/ssr

In short: we need a standard 🙈

SSR  CSR

≠

good  bad

Polyfills limitations (👋 IE11 and Edge)

Polyfills limitations (👋 IE11 and Edge)

— Only 1 value is allowed per CSS custom property for a given shadowRoot 🤪

Polyfills limitations (👋 IE11 and Edge)

— Only 1 value is allowed per CSS custom property for a given shadowRoot 🤪

— Dynamically created styles are not supported 😬

Polyfills limitations (👋 IE11 and Edge)

— Only 1 value is allowed per CSS custom property for a given shadowRoot 🤪

— Dynamically created styles are not supported 😬

— External stylesheets are not supported 🤯

Polyfills limitations (👋 IE11 and Edge)

— Only 1 value is allowed per CSS custom property for a given shadowRoot 🤪

— Dynamically created styles are not supported 😬

— External stylesheets are not supported 🤯

— Components are not inspectable in IE11 🤬

Polyfills limitations (👋 IE11 and Edge)

— Only 1 value is allowed per CSS custom property for a given shadowRoot 🤪

— Dynamically created styles are not supported 😬

— External stylesheets are not supported 🤯

— Components are not inspectable in IE11 🤬

— ... more on github.com/webcomponents/shadycss

Are you used to normalize.css?

... or an alternative?

😟

No longer possible in the way you are used to.

Want to fix browser's [hidden] defaults?

```
<!-- browser defaults -->
<style>
  [hidden] { /* specificity 010 */
    display: none;
  }
</style>

<!-- your app -->
<style>
  .my-element { /* specificity 010, but defined later */
    display: flex;
    /* ...other properties */
  }
</style>

<div class="my-element" hidden><!-- I am visible 😞  --></div>
```

Want to fix browser's [hidden] defaults?

```
<!-- browser defaults -->
<style>
  [hidden] { /* specificity 010 */
    display: none;
  }
</style>

<!-- your app -->
<style>
  .my-element { /* specificity 010, but defined later */
    display: flex;
    /* ...other properties */
  }
</style>

<div class="my-element" hidden><!-- I am visible 😞  --></div>
```

Want to fix browser's [hidden] defaults?

```
<!-- browser defaults -->
<style>
  [hidden] { /* specificity 010 */
    display: none;
  }
</style>

<!-- your app -->
<style>
  .my-element { /* specificity 010, but defined later */
    display: flex;
    /* ...other properties */
  }
</style>

<div class="my-element" hidden><!-- I am visible 😞  --></div>
```

Want to fix browser's [hidden] defaults?

```
<!-- browser defaults -->
<style>
  [hidden] { /* specificity 010 */
    display: none;
  }
</style>

<!-- your app -->
<style>
  .my-element { /* specificity 010, but defined later */
    display: flex;
    /* ...other properties */
  }
</style>

<div class="my-element" hidden><!-- I am visible 😞  --></div>
```

Want to fix browser's [hidden] defaults?

```
<!-- browser defaults -->
<style>
  [hidden] { /* specificity 010 */
    display: none;
  }
</style>

<!-- your app -->
<style>
  .my-element { /* specificity 010, but defined later */
    display: flex;
    /* ...other properties */
  }
</style>

<div class="my-element" hidden><!-- I am visible 😞  --></div>
```

Want to fix browser's [hidden] defaults?

```
<!-- browser defaults -->
<style>
  [hidden] { /* specificity 010 */
    display: none;
  }
</style>

<!-- your app -->
<style>
  .my-element { /* specificity 010, but defined later */
    display: flex;
    /* ...other properties */
  }
</style>

<div class="my-element" hidden><!-- I am visible 😞  --></div>
```

Want to fix browser's [hidden] defaults?

```
<!-- browser defaults -->
<style>
  [hidden] { /* specificity 010 */
    display: none;
  }
</style>

<!-- your app -->
<style>
  .my-element { /* specificity 010, but defined later */
    display: flex;
    /* ...other properties */
  }
</style>

<div class="my-element" hidden><!-- I am visible 😞  --></div>
```

Want to fix browser's [hidden] defaults?

```
<!-- browser defaults -->
<style>
  [hidden] { /* specificity 010 */
    display: none;
  }
</style>

<!-- your app -->
<style>
  .my-element { /* specificity 010, but defined later */
    display: flex;
    /* ...other properties */
  }
</style>

<div class="my-element" hidden><!-- I am visible 😞  --></div>
```

Want to fix browser's [hidden] defaults?

```
<!-- browser defaults -->
<style>
  [hidden] { /* specificity 010 */
    display: none;
  }
</style>

<!-- your app -->
<style>
  .my-element { /* specificity 010, but defined later */
    display: flex;
    /* ...other properties */
  }
</style>

<div class="my-element" hidden><!-- I am visible 😞  --></div>
```

Why not just change this globally?

```
[hidden] {
  display: none !important;
}
```

You can't. Instead...

In every second component:

```
<my-average-component>
  #shadow-root
    <style>
      [hidden] {
        display: none !important;
      }
    </style>
</my-average-component>
```

Is [hidden] an anti-pattern?

Different opinions:

Louis Lazaris (impressivewebs.com) promotes it:

👍 HTML5's Global hidden Attribute

Monica Dinculescu (ex Polymer team member) opinion:

👎 [hidden] is a lie

:slotted() selector is way too limited

```
<my-fieldset>
  #shadow-root
    <style>
      ::slotted(fieldset) { /* works */ }
      ::slotted(legend) { /* does not work */ }
    </style>
    <slot>
      <!-- start: projected content -->
      <fieldset>
        <legend></legend>
      </fieldset>
      <!-- end: projected content -->
    </slot>
  <fieldset>
    <legend>My form fieldset</legend>
  </fieldset>
</my-fieldset>
```

:slotted() selector is way too limited

```
<my-fieldset>
  #shadow-root
    <style>
      ::slotted(fieldset) { /* works */ }
      ::slotted(legend) { /* does not work */ }
    </style>
    <slot>
      <!-- start: projected content -->
      <fieldset>
        <legend></legend>
      </fieldset>
      <!-- end: projected content -->
    </slot>
  <fieldset>
    <legend>My form fieldset</legend>
  </fieldset>
</my-fieldset>
```

:slotted() selector is way too limited

```
<my-fieldset>
  #shadow-root
    <style>
      ::slotted(fieldset) { /* works */ }
      ::slotted(legend) { /* does not work */ }
    </style>
    <slot>
      <!-- start: projected content -->
      <fieldset>
        <legend></legend>
      </fieldset>
      <!-- end: projected content -->
    </slot>
  <fieldset>
    <legend>My form fieldset</legend>
  </fieldset>
</my-fieldset>
```

:slotted() selector is way too limited

```
<my-fieldset>
  #shadow-root
    <style>
      ::slotted(fieldset) { /* works */ }
      ::slotted(legend) { /* does not work */ }
    </style>
    <slot>
      <!-- start: projected content -->
      <fieldset>
        <legend></legend>
      </fieldset>
      <!-- end: projected content -->
    </slot>
  <fieldset>
    <legend>My form fieldset</legend>
  </fieldset>
</my-fieldset>
```

:slotted() selector is way too limited

```
<my-fieldset>
  #shadow-root
    <style>
      ::slotted(fieldset) { /* works */ }
      ::slotted(legend) { /* does not work */ }
    </style>
    <slot>
      <!-- start: projected content -->
      <fieldset>
        <legend></legend>
      </fieldset>
      <!-- end: projected content -->
    </slot>
  <fieldset>
    <legend>My form fieldset</legend>
  </fieldset>
</my-fieldset>
```

:slotted() selector is way too limited

```
<my-fieldset>
  #shadow-root
    <style>
      ::slotted(fieldset) { /* works */ }
      ::slotted(legend) { /* does not work */ }
    </style>
    <slot>
      <!-- start: projected content -->
      <fieldset>
        <legend></legend>
      </fieldset>
      <!-- end: projected content -->
    </slot>
  <fieldset>
    <legend>My form fieldset</legend>
  </fieldset>
</my-fieldset>
```

:slotted() selector is way too limited

```
<my-fieldset>
  #shadow-root
    <style>
      ::slotted(fieldset) { /* works */ }
      ::slotted(legend) { /* does not work */ }
    </style>
    <slot>
      <!-- start: projected content -->
      <fieldset>
        <legend></legend>
      </fieldset>
      <!-- end: projected content -->
    </slot>
  <fieldset>
    <legend>My form fieldset</legend>
  </fieldset>
</my-fieldset>
```

:slotted() selector is way too limited

```
<my-fieldset>
  #shadow-root
    <style>
      ::slotted(fieldset) { /* works */ }
      ::slotted(legend) { /* does not work */ }
    </style>
    <slot>
      <!-- start: projected content -->
      <fieldset>
        <legend></legend>
      </fieldset>
      <!-- end: projected content -->
    </slot>
  <fieldset>
    <legend>My form fieldset</legend>
  </fieldset>
</my-fieldset>
```

Shadow DOM is not a silver bullet

Please keep in mind that Shadow DOM:

Shadow DOM is not a silver bullet

Please keep in mind that Shadow DOM:

1. Doesn't scope @font-face or @keyframe

Shadow DOM is not a silver bullet

Please keep in mind that Shadow DOM:

1. Doesn't scope @font-face or @keyframe
2. Doesn't scope SVG symbols

```
<svg><symbol id="my-icon"></symbol></svg>
```

Shadow DOM is not a silver bullet

Please keep in mind that Shadow DOM:

1. Doesn't scope @font-face or @keyframe
2. Doesn't scope SVG symbols
   `<svg><symbol id="my-icon"></symbol></svg>`
3. Doesn't limit JavaScript access to document or window

Shadow DOM is not a silver bullet

Please keep in mind that Shadow DOM:

1. Doesn't scope @font-face or @keyframe
2. Doesn't scope SVG symbols
   `<svg><symbol id="my-icon"></symbol></svg>`
3. Doesn't limit JavaScript access to document or window
4. Doesn't prevent injecting an entire CSS framework into each Shadow DOM

Shadow DOM is not a silver bullet

Please keep in mind that Shadow DOM:

1. Doesn't scope @font-face or @keyframe
2. Doesn't scope SVG symbols
   `<svg><symbol id="my-icon"></symbol></svg>`
3. Doesn't limit JavaScript access to document or window
4. Doesn't prevent injecting an entire CSS framework into each Shadow DOM

5. ~~Doesn't brew ☕ (sadly...)~~

@apply-gate

```
<style>
  .customize {
    --my-shadow-element-mixin: { /* customize mixin from the outside */
      prop1: custom_value;
      prop2: custom_value;
      prop3: custom_value;
    };
  }
</style>
<my-component class="customize">
  #shadow-root
    <style>
      :host {
        --my-shadow-element-mixin: { /* setup default mixin */
          prop1: default_value;
          prop2: default_value;
          prop3: default_value;
        };
      }
      .my-shadow-element {
        @apply --my-shadow-element-mixin; /* apply mixin to an element in the shadow root */
      }
    </style>
    <div class="my-shadow-element"></div>
</my-component>
```

@apply-gate

```
<style>
  .customize {
    --my-shadow-element-mixin: { /* customize mixin from the outside */
      prop1: custom_value;
      prop2: custom_value;
      prop3: custom_value;
    };
  }
</style>
<my-component class="customize">
  #shadow-root
    <style>
      :host {
        --my-shadow-element-mixin: { /* setup default mixin */
          prop1: default_value;
          prop2: default_value;
          prop3: default_value;
        };
      }
      .my-shadow-element {
        @apply --my-shadow-element-mixin; /* apply mixin to an element in the shadow root */
      }
    </style>
    <div class="my-shadow-element"></div>
</my-component>
```

@apply-gate

```
<style>
  .customize {
    --my-shadow-element-mixin: { /* customize mixin from the outside */
      prop1: custom_value;
      prop2: custom_value;
      prop3: custom_value;
    };
  }
</style>
<my-component class="customize">
  #shadow-root
    <style>
      :host {
        --my-shadow-element-mixin: { /* setup default mixin */
          prop1: default_value;
          prop2: default_value;
          prop3: default_value;
        };
      }
      .my-shadow-element {
        @apply --my-shadow-element-mixin; /* apply mixin to an element in the shadow root */
      }
    </style>
    <div class="my-shadow-element"></div>
</my-component>
```

@apply-gate

```
<style>
  .customize {
    --my-shadow-element-mixin: { /* customize mixin from the outside */
      prop1: custom_value;
      prop2: custom_value;
      prop3: custom_value;
    };
  }
</style>
<my-component class="customize">
  #shadow-root
    <style>
      :host {
        --my-shadow-element-mixin: { /* setup default mixin */
          prop1: default_value;
          prop2: default_value;
          prop3: default_value;
        };
      }
      .my-shadow-element {
        @apply --my-shadow-element-mixin; /* apply mixin to an element in the shadow root */
      }
    </style>
    <div class="my-shadow-element"></div>
</my-component>
```

@apply-gate

```
<style>
  .customize {
    --my-shadow-element-mixin: { /* customize mixin from the outside */
      prop1: custom_value;
      prop2: custom_value;
      prop3: custom_value;
    };
  }
</style>
<my-component class="customize">
  #shadow-root
    <style>
      :host {
        --my-shadow-element-mixin: { /* setup default mixin */
          prop1: default_value;
          prop2: default_value;
          prop3: default_value;
        };
      }
      .my-shadow-element {
        @apply --my-shadow-element-mixin; /* apply mixin to an element in the shadow root */
      }
    </style>
    <div class="my-shadow-element"></div>
</my-component>
```

@apply-gate

```
<style>
  .customize {
    --my-shadow-element-mixin: { /* customize mixin from the outside */
      prop1: custom_value;
      prop2: custom_value;
      prop3: custom_value;
    };
  }
</style>
<my-component class="customize">
  #shadow-root
    <style>
      :host {
        --my-shadow-element-mixin: { /* setup default mixin */
          prop1: default_value;
          prop2: default_value;
          prop3: default_value;
        };
      }
      .my-shadow-element {
        @apply --my-shadow-element-mixin; /* apply mixin to an element in the shadow root */
      }
    </style>
    <div class="my-shadow-element"></div>
</my-component>
```

@apply-gate

```
<style>
  .customize {
    --my-shadow-element-mixin: { /* customize mixin from the outside */
      prop1: custom_value;
      prop2: custom_value;
      prop3: custom_value;
    };
  }
</style>
<my-component class="customize">
  #shadow-root
    <style>
      :host {
        --my-shadow-element-mixin: { /* setup default mixin */
          prop1: default_value;
          prop2: default_value;
          prop3: default_value;
        };
      }
      .my-shadow-element {
        @apply --my-shadow-element-mixin; /* apply mixin to an element in the shadow root */
      }
    </style>
    <div class="my-shadow-element"></div>
</my-component>
```

@apply-gate

```
<style>
  .customize {
    --my-shadow-element-mixin: { /* customize mixin from the outside */
      prop1: custom_value;
      prop2: custom_value;
      prop3: custom_value;
    };
  }
</style>
<my-component class="customize">
  #shadow-root
    <style>
      :host {
        --my-shadow-element-mixin: { /* setup default mixin */
          prop1: default_value;
          prop2: default_value;
          prop3: default_value;
        };
      }
      .my-shadow-element {
        @apply --my-shadow-element-mixin; /* apply mixin to an element in the shadow root */
      }
    </style>
    <div class="my-shadow-element"></div>
</my-component>
```

@apply-gate

```
<style>
  .customize {
    --my-shadow-element-mixin: { /* customize mixin from the outside */
      prop1: custom_value;
      prop2: custom_value;
      prop3: custom_value;
    };
  }
</style>
<my-component class="customize">
  #shadow-root
    <style>
      :host {
        --my-shadow-element-mixin: { /* setup default mixin */
          prop1: default_value;
          prop2: default_value;
          prop3: default_value;
        };
      }
      .my-shadow-element {
        @apply --my-shadow-element-mixin; /* apply mixin to an element in the shadow root */
      }
    </style>
    <div class="my-shadow-element"></div>
</my-component>
```

Tab Completion

I'm Tab Atkins Jr, and I wear many hats. I work for Google on the Chrome browser as a Web Standards Hacker. I'm also a member of the CSS Working Group, and am either a member or contributor to several other working groups in the W3C. You can contact me here.

Listing of All Posts

Why I Abandoned @apply

Apr 24

Last updated: Monday, May 8 2017

Some time ago, I created the modern spec for CSS Variables, which lets you store uninterpreted values into "custom properties", then substitute those values into real properties later on.

This worked great, and was particularly convenient for Shadow DOM, which wanted a simple, controllable way to let the outside page twiddle the styling of a component, but without giving them full access to the internals of the component. The component author could just use a couple of `var()` functions, with default values so that it worked automatically, and then the component user could set custom properties on the shadow host and let them inherit into the component.

Alternative approach to @apply?

CSS Shadow Parts

- 1st public draft was published yesterday 🤓
- not even under flag in Canary 🤷‍♂️

Explainer by Monica Dinculescu:
meowni.ca/posts/part-theme-explainer/

How to mimic CSS Shadow Parts now? Watch this talk:
Vaadin Elements pre-Polymer Summit 2017 meetup by Jouni Koivuviita

HTML Imports (and Polymer) legacy

```
<!-- iron-flex.html: reusable style module -->
<dom-module id="iron-flex">
  <template>
    <style>
      .layout.horizontal, .layout.vertical {
        display: flex;
      }
      /* ... */
    </style>
  </template>
</dom-module>

<!-- my-component.html: your component module -->
<link rel="import" href="path/to/iron-flex.html">
<style include="iron-flex">/* your component styles */</style>
```

HTML Imports (and Polymer) legacy

```
<!-- iron-flex.html: reusable style module -->
<dom-module id="iron-flex">
  <template>
    <style>
      .layout.horizontal, .layout.vertical {
        display: flex;
      }
      /* ... */
    </style>
  </template>
</dom-module>

<!-- my-component.html: your component module -->
<link rel="import" href="path/to/iron-flex.html">
<style include="iron-flex">/* your component styles */</style>
```

HTML Imports (and Polymer) legacy

```
<!-- iron-flex.html: reusable style module -->
<dom-module id="iron-flex">
  <template>
    <style>
      .layout.horizontal, .layout.vertical {
        display: flex;
      }
      /* ... */
    </style>
  </template>
</dom-module>

<!-- my-component.html: your component module -->
<link rel="import" href="path/to/iron-flex.html">
<style include="iron-flex">/* your component styles */</style>
```

HTML Imports (and Polymer) legacy

```
<!-- iron-flex.html: reusable style module -->
<dom-module id="iron-flex">
  <template>
    <style>
      .layout.horizontal, .layout.vertical {
        display: flex;
      }
      /* ... */
    </style>
  </template>
</dom-module>

<!-- my-component.html: your component module -->
<link rel="import" href="path/to/iron-flex.html">
<style include="iron-flex">/* your component styles */</style>
```

HTML Imports (and Polymer) legacy

```
<!-- iron-flex.html: reusable style module -->
<dom-module id="iron-flex">
  <template>
    <style>
      .layout.horizontal, .layout.vertical {
        display: flex;
      }
      /* ... */
    </style>
  </template>
</dom-module>

<!-- my-component.html: your component module -->
<link rel="import" href="path/to/iron-flex.html">
<style include="iron-flex">/* your component styles */</style>
```

HTML Imports (and Polymer) legacy

```
<!-- iron-flex.html: reusable style module -->
<dom-module id="iron-flex">
  <template>
    <style>
      .layout.horizontal, .layout.vertical {
        display: flex;
      }
      /* ... */
    </style>
  </template>
</dom-module>

<!-- my-component.html: your component module -->
<link rel="import" href="path/to/iron-flex.html">
<style include="iron-flex">/* your component styles */</style>
```

HTML Imports (and Polymer) legacy

```
<!-- iron-flex.html: reusable style module -->
<dom-module id="iron-flex">
  <template>
    <style>
      .layout.horizontal, .layout.vertical {
        display: flex;
      }
      /* ... */
    </style>
  </template>
</dom-module>

<!-- my-component.html: your component module -->
<link rel="import" href="path/to/iron-flex.html">
<style include="iron-flex">/* your component styles */</style>
```

Style modules via HTML Imports in real life

```
<link rel="import" href="../font-roboto/roboto.html"><!-- @font-face from Google Fonts -->
<link rel="import" href="../paper-styles/default-theme.html"><!-- CSS vars -->
<link rel="import" href="../paper-styles/color.html"><!-- CSS vars -->
<link rel="import" href="../paper-styles/classes/global.html"><!-- normalize tags -->
<link rel="import" href="../paper-styles/classes/shadow.html"><!-- elevation classes -->
<link rel="import" href="../paper-styles/classes/typography.html"><!-- typography classes -->
<link rel="import" href="../paper-styles/classes/element-styles/paper-item-styles.html"><!-- CSS "component" item -->
<link rel="import" href="../paper-styles/classes/element-styles/paper-material-styles.html"><!-- CSS "component" card -->
<link rel="import" href="../iron-flex-layout/iron-flex-layout-classes.html"><!-- layout/grid classes -->

<dom-module id="my-component">
  <template>
    <style include="paper-item-styles paper-material-styles iron-flex">
      /* your component styles */
    </style>
    <!-- your component html -->
  </template>
</dom-module>
```

Style modules via HTML Imports in real life

```
<link rel="import" href="../font-roboto/roboto.html"><!-- @font-face from Google Fonts -->
<link rel="import" href="../paper-styles/default-theme.html"><!-- CSS vars -->
<link rel="import" href="../paper-styles/color.html"><!-- CSS vars -->
<link rel="import" href="../paper-styles/classes/global.html"><!-- normalize tags -->
<link rel="import" href="../paper-styles/classes/shadow.html"><!-- elevation classes -->
<link rel="import" href="../paper-styles/classes/typography.html"><!-- typography classes -->
<link rel="import" href="../paper-styles/classes/element-styles/paper-item-styles.html"><!-- CSS "component" item -->
<link rel="import" href="../paper-styles/classes/element-styles/paper-material-styles.html"><!-- CSS "component" card -->
<link rel="import" href="../iron-flex-layout/iron-flex-layout-classes.html"><!-- layout/grid classes -->

<dom-module id="my-component">
  <template>
    <style include="paper-item-styles paper-material-styles iron-flex">
      /* your component styles */
    </style>
    <!-- your component html -->
  </template>
</dom-module>
```

Style modules via HTML Imports in real life

```
<link rel="import" href="../font-roboto/roboto.html"><!-- @font-face from Google Fonts -->
<link rel="import" href="../paper-styles/default-theme.html"><!-- CSS vars -->
<link rel="import" href="../paper-styles/color.html"><!-- CSS vars -->
<link rel="import" href="../paper-styles/classes/global.html"><!-- normalize tags -->
<link rel="import" href="../paper-styles/classes/shadow.html"><!-- elevation classes -->
<link rel="import" href="../paper-styles/classes/typography.html"><!-- typography classes -->
<link rel="import" href="../paper-styles/classes/element-styles/paper-item-styles.html"><!-- CSS "component" item -->
<link rel="import" href="../paper-styles/classes/element-styles/paper-material-styles.html"><!-- CSS "component" card -->
<link rel="import" href="../iron-flex-layout/iron-flex-layout-classes.html"><!-- layout/grid classes -->

<dom-module id="my-component">
  <template>
    <style include="paper-item-styles paper-material-styles iron-flex">
      /* your component styles */
    </style>
    <!-- your component html -->
  </template>
</dom-module>
```

At some point it becomes 🙀

```
<link rel="import" href="../path/to/paper/import-all.html">

<dom-module id="my-component">
  <template>
    <style include="paper-all">
      /* your component styles */
    </style>
    <!-- your component html -->
  </template>
</dom-module>
```

I swear I saw smth like this...

At some point it becomes 🙀

```
<link rel="import" href="../path/to/paper/import-all.html">

<dom-module id="my-component">
  <template>
    <style include="paper-all">
      /* your component styles */
    </style>
    <!-- your component html -->
  </template>
</dom-module>
```

I swear I saw smth like this...

At some point it becomes 🙀

```
<link rel="import" href="../path/to/paper/import-all.html">

<dom-module id="my-component">
  <template>
    <style include="paper-all">
      /* your component styles */
    </style>
    <!-- your component html -->
  </template>
</dom-module>
```

I swear I saw smth like this...

How is it different to no Shadow DOM at all???

Modern JavaScript frameworks tended to have a much shorter lifespan than the products we were developing.

— Chad Killingsworth @ Polymer Summit 2017

What caused refactorings?

What caused refactorings?

— HTML Imports deprecation

What caused refactorings?

— HTML Imports deprecation

— Transition from Shadow DOM v0 👉 v1
(+ as well Custom Elements v0 👉 v1)

What caused refactorings?

— HTML Imports deprecation

— Transition from Shadow DOM v0 👉 v1
(+ as well Custom Elements v0 👉 v1)

— @apply deprecation

Episode IV.
A New Hope

Where we revive our hope for Shadow DOM and learn about lit-css.

These are some interesting points. I use both React & #WebComponents. Both separately & together. What intrigues me is that some many developers feel they need to choose a 'side'. Why not use Custom Elements for Simple UI leaf nodes & Shadow DOM for SEO content in HTML?...

— Erik Isaksen @eisaksen reply to @mjackson on October 12, 2018

The Reusability Dilemma

If it is **reusable**,
develop as a Web Component.

If it is **not reusable**,
develop in your app framework (React, Angular, Vue...).

If it is **unknown**,
then choose a Web Component since it will work anyway.

If it gets from **not reusable => reusable**,
then you need to rewrite it as a Web Component.

WEB COMPONENT IS THE DEFAULT

THERE IS AN OBVIOUS

NATURAL WAY

TO WORK WITH STYLES

IN MODERN WEB COMPONENTS

Let's combine

Let's combine

— **ES modules** as a single mechanism to distribute web components

Let's combine

— **ES modules** as a single mechanism to distribute web components

— styled-components idea to use **tagged template literals** for CSS

Let's combine

— **ES modules** as a single mechanism to distribute web components

— styled-components idea to use **tagged template literals** for CSS

— familiarity with **lit-html**

Let's combine

— **ES modules** as a single mechanism to distribute web components

— styled-components idea to use **tagged template literals** for CSS

— familiarity with **lit-html**

— idea of **Polymer's style modules**

... and recap this example

```
import { LitElement } from '@polymer/lit-element';
import { html } from 'lit-html';

customElements.define('badge-new', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```

... and recap this example

```
import { LitElement } from '@polymer/lit-element';
import { html } from 'lit-html';

customElements.define('badge-new', class extends LitElement {
  render() {
    return html`
      <style>
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      </style>
      NEW
    `;
  }
});
```

Extract into a static getter

```
import { LitElement } from '@polymer/lit-element';
import { html } from 'lit-html';


customElements.define('badge-new', class extends LitElement {
  static get style() {
    return `
      :host {
        display: inline-block;
        background-color: yellow;
        border: 1px solid red;
        color: red;
      }
    `;
  }
  render() {
    return html`
      <style>${this.constructor.style}</style>
      NEW
    `;
  }
});
```

Extract into a static getter

```
import { LitElement } from '@polymer/lit-element';
import { html } from 'lit-html';


customElements.define('badge-new', class extends LitElement {
  static get style() {
    return `
      :host {
        display: inline-block;
        background-color: yellow;
        border: 1px solid red;
        color: red;
      }
    `;
  }
  render() {
    return html`
      <style>${this.constructor.style}</style>
      NEW
    `;
  }
});
```

Extract into a static getter

```
import { LitElement } from '@polymer/lit-element';
import { html } from 'lit-html';


customElements.define('badge-new', class extends LitElement {
  static get style() {
    return `
      :host {
        display: inline-block;
        background-color: yellow;
        border: 1px solid red;
        color: red;
      }
    `;
  }
  render() {
    return html`
      <style>${this.constructor.style}</style>
      NEW
    `;
  }
});
```

Extract into a static getter

```
import { LitElement } from '@polymer/lit-element';
import { html } from 'lit-html';


customElements.define('badge-new', class extends LitElement {
  static get style() {
    return `
      :host {
        display: inline-block;
        background-color: yellow;
        border: 1px solid red;
        color: red;
      }
    `;
  }
  render() {
    return html`
      <style>${this.constructor.style}</style>
      NEW
    `;
  }
});
```

Extract into a static getter

```
import { LitElement } from '@polymer/lit-element';
import { html } from 'lit-html';


customElements.define('badge-new', class extends LitElement {
  static get style() {
    return `
      :host {
        display: inline-block;
        background-color: yellow;
        border: 1px solid red;
        color: red;
      }
    `;
  }
  render() {
    return html`
      <style>${this.constructor.style}</style>
      NEW
    `;
  }
});
```

lit-css 🚀

```
import { LitElement } from '@polymer/lit-element';
import { html } from 'lit-html';
import { css } from 'lit-css';

customElements.define('badge-new', class extends LitElement {
  static get style() {
    return css`
      :host {
        display: inline-block;
        background-color: yellow;
        border: 1px solid red;
        color: red;
      }
    `;
  }
  render() {
    return html`
      <style>${this.constructor.style}</style>
      NEW
    `;
  }
});
```

lit-css 🚀

```
import { LitElement } from '@polymer/lit-element';
import { html } from 'lit-html';
import { css } from 'lit-css';

customElements.define('badge-new', class extends LitElement {
  static get style() {
    return css`
      :host {
        display: inline-block;
        background-color: yellow;
        border: 1px solid red;
        color: red;
      }
    `;
  }
  render() {
    return html`
      <style>${this.constructor.style}</style>
      NEW
    `;
  }
});
```

```
import { LitElement } from '@polymer/lit-element';
import { html } from 'lit-html';

customElements.define('badge-new',
  class extends LitElement {
    static get style() {
      return `
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      `;
    }
    render() {
      return html`
        <style>${this.constructor.style}</style>
        NEW
      `;
    }
  }
);
```

```
import { LitElement } from '@polymer/lit-element';
import { html } from 'lit-html';
import { css } from 'lit-css';

customElements.define('badge-new',
  class extends LitElement {
    static get style() {
      return css`
        :host {
          display: inline-block;
          background-color: yellow;
          border: 1px solid red;
          color: red;
        }
      `;
    }
    render() {
      return html`
        <style>${this.constructor.style}</style>
        NEW
      `;
    }
  }
);
```

styled-lit-element 🚀

```
import { StyledLitElement } from 'styled-lit-element';
import { html } from 'lit-html';
import { css } from 'lit-css';

customElements.define('badge-new', class extends StyledLitElement {
  static get style() {
    return css`
      :host {
        display: inline-block;
        background-color: yellow;
        border: 1px solid red;
        color: red;
      }
    `;
  }
  render() {
    return html`
      <!-- <style> will be injected and cached when needed -->
      NEW
    `;
  }
});
```

styled-lit-element 🚀

```
import { StyledLitElement } from 'styled-lit-element';
import { html } from 'lit-html';
import { css } from 'lit-css';

customElements.define('badge-new', class extends StyledLitElement {
  static get style() {
    return css`
      :host {
        display: inline-block;
        background-color: yellow;
        border: 1px solid red;
        color: red;
      }
    `;
  }
  render() {
    return html`
      <!-- <style> will be injected and cached when needed -->
      NEW
    `;
  }
});
```

styled-lit-element 🚀

```
import { StyledLitElement } from 'styled-lit-element';
import { html } from 'lit-html';
import { css } from 'lit-css';

customElements.define('badge-new', class extends StyledLitElement {
  static get style() {
    return css`
      :host {
        display: inline-block;
        background-color: yellow;
        border: 1px solid red;
        color: red;
      }
    `;
  }
  render() {
    return html`
      <!-- <style> will be injected and cached when needed -->
      NEW
    `;
  }
});
```

styled-lit-element 🚀

```
import { StyledLitElement } from 'styled-lit-element';
import { html } from 'lit-html';
import { css } from 'lit-css';

customElements.define('badge-new', class extends StyledLitElement {
  static get style() {
    return css`
      :host {
        display: inline-block;
        background-color: yellow;
        border: 1px solid red;
        color: red;
      }
    `;
  }
  render() {
    return html`
      <!-- <style> will be injected and cached when needed -->
      NEW
    `;
  }
});
```

Use Case 1: White-label and branded components

```
class WhiteLabelComponent extends StyledLitElement {
  static get style() {
    return css`/* minimalistic styles */`;
  }
  render() {
    return html`<!-- shared template -->`;
  }
  /* shared logic */
}

class BrandedComponent extends WhiteLabelComponent {
  static get style() {
    return css`
      ${super.style}
      /* design specific styles */
    `;
  }
}
```

Use Case 1: White-label and branded components

```
class WhiteLabelComponent extends StyledLitElement {
  static get style() {
    return css`/* minimalistic styles */`;
  }
  render() {
    return html`<!-- shared template -->`;
  }
  /* shared logic */
}

class BrandedComponent extends WhiteLabelComponent {
  static get style() {
    return css`
      ${super.style}
      /* design specific styles */
    `;
  }
}
```

Use Case 1: White-label and branded components

```
class WhiteLabelComponent extends StyledLitElement {
  static get style() {
    return css`/* minimalistic styles */`;
  }
  render() {
    return html`<!-- shared template -->`;
  }
  /* shared logic */
}

class BrandedComponent extends WhiteLabelComponent {
  static get style() {
    return css`
      ${super.style}
      /* design specific styles */
    `;
  }
}
```

Use Case 1: White-label and branded components

```
class WhiteLabelComponent extends StyledLitElement {
  static get style() {
    return css`/* minimalistic styles */`;
  }
  render() {
    return html`<!-- shared template -->`;
  }
  /* shared logic */
}

class BrandedComponent extends WhiteLabelComponent {
  static get style() {
    return css`
      ${super.style}
      /* design specific styles */
    `;
  }
}
```

Use Case 1: White-label and branded components

```
class WhiteLabelComponent extends StyledLitElement {
  static get style() {
    return css`/* minimalistic styles */`;
  }
  render() {
    return html`<!-- shared template -->`;
  }
  /* shared logic */
}

class BrandedComponent extends WhiteLabelComponent {
  static get style() {
    return css`
      ${super.style}
      /* design specific styles */
    `;
  }
}
```

Use Case 1: White-label and branded components

```
class WhiteLabelComponent extends StyledLitElement {
  static get style() {
    return css`/* minimalistic styles */`;
  }
  render() {
    return html`<!-- shared template -->`;
  }
  /* shared logic */
}

class BrandedComponent extends WhiteLabelComponent {
  static get style() {
    return css`
      ${super.style}
      /* design specific styles */
    `;
  }
}
```

Use Case 1: White-label and branded components

```
class WhiteLabelComponent extends StyledLitElement {
  static get style() {
    return css`/* minimalistic styles */`;
  }
  render() {
    return html`<!-- shared template -->`;
  }
  /* shared logic */
}

class BrandedComponent extends WhiteLabelComponent {
  static get style() {
    return css`
      ${super.style}
      /* design specific styles */
    `;
  }
}
```

Use Case 1: White-label and branded components

```
class WhiteLabelComponent extends StyledLitElement {
  static get style() {
    return css`/* minimalistic styles */`;
  }
  render() {
    return html`<!-- shared template -->`;
  }
  /* shared logic */
}

class BrandedComponent extends WhiteLabelComponent {
  static get style() {
    return css`
      ${super.style}
      /* design specific styles */
    `;
  }
}
```

Use Case 1: White-label and branded components

```
class WhiteLabelComponent extends StyledLitElement {
  static get style() {
    return css`/* minimalistic styles */`;
  }
  render() {
    return html`<!-- shared template -->`;
  }
  /* shared logic */
}

class BrandedComponent extends WhiteLabelComponent {
  static get style() {
    return css`
      ${super.style}
      /* design specific styles */
    `;
  }
}
```

Use Case 1: White-label and branded components

```
class WhiteLabelComponent extends StyledLitElement {
  static get style() {
    return css`/* minimalistic styles */`;
  }
  render() {
    return html`<!-- shared template -->`;
  }
  /* shared logic */
}

class BrandedComponent extends WhiteLabelComponent {
  static get style() {
    return css`
      ${super.style}
      /* design specific styles */
    `;
  }
}
```

Use Case 1: White-label and branded components

Prototypes for the **"next generation" Vaadin components**
by Serhii Kulykov recently switched to StyledLitElement in PR#11:

```
class BaseButton extends StyledLitElement {
  // logic and basic styles
}

class LumoButton extends BaseButton {
  // Vaadin's Lumo styles
}

class MaterialButton extends BaseButton {
  // Material Design styles
}
```

Use Case 1: White-label and branded components

Prototypes for the **"next generation" Vaadin components**
by Serhii Kulykov recently switched to StyledLitElement in PR#11:

```
class BaseButton extends StyledLitElement {
  // logic and basic styles
}

class LumoButton extends BaseButton {
  // Vaadin's Lumo styles
}

class MaterialButton extends BaseButton {
  // Material Design styles
}
```

Use Case 1: White-label and branded components

Prototypes for the **"next generation" Vaadin components**
by Serhii Kulykov recently switched to StyledLitElement in PR#11:

```
class BaseButton extends StyledLitElement {
  // logic and basic styles
}

class LumoButton extends BaseButton {
  // Vaadin's Lumo styles
}

class MaterialButton extends BaseButton {
  // Material Design styles
}
```

Use Case 1: White-label and branded components

Prototypes for the **"next generation" Vaadin components**
by Serhii Kulykov recently switched to StyledLitElement in PR#11:

```
class BaseButton extends StyledLitElement {
  // logic and basic styles
}

class LumoButton extends BaseButton {
  // Vaadin's Lumo styles
}

class MaterialButton extends BaseButton {
  // Material Design styles
}
```

Use Case 1: White-label and branded components

Prototypes for the **"next generation" Vaadin components**
by Serhii Kulykov recently switched to StyledLitElement in PR#11:

```
class BaseButton extends StyledLitElement {
  // logic and basic styles
}

class LumoButton extends BaseButton {
  // Vaadin's Lumo styles
}

class MaterialButton extends BaseButton {
  // Material Design styles
}
```

Use Case 2: Old-school CSS components

```
// my-styles/grid.js
import { css } from 'lit-css';
export default css`
  .o-grid { ... }
  .o-grid__1/2 { ... }
  /* grid fractions */
`;
```

Long live BEMIT.

Use Case 3: More powerful composition

```
const myUtilityClasses = css`.u-myutil { property: value; }`;
const myValue = 'value';
const myMixin = () => css`property: value;`;

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      ${myUtilityClasses}
      .using-value {
        property: ${myValue};
      }
      .using-mixin {
        ${myMixin()}
      }
    `;
  }
  render() { /* template using classes above */ }
}
```

Use Case 3: More powerful composition

```
const myUtilityClasses = css`.u-myutil { property: value; }`;
const myValue = 'value';
const myMixin = () => css`property: value;`;

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      ${myUtilityClasses}
      .using-value {
        property: ${myValue};
      }
      .using-mixin {
        ${myMixin()}
      }
    `;
  }
  render() { /* template using classes above */ }
}
```

Use Case 3: More powerful composition

```
const myUtilityClasses = css`.u-myutil { property: value; }`;
const myValue = 'value';
const myMixin = () => css`property: value;`;

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      ${myUtilityClasses}
      .using-value {
        property: ${myValue};
      }
      .using-mixin {
        ${myMixin()}
      }
    `;
  }
  render() { /* template using classes above */ }
}
```

Use Case 3: More powerful composition

```
const myUtilityClasses = css`.u-myutil { property: value; }`;
const myValue = 'value';
const myMixin = () => css`property: value;`;

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      ${myUtilityClasses}
      .using-value {
        property: ${myValue};
      }
      .using-mixin {
        ${myMixin()}
      }
    `;
  }
  render() { /* template using classes above */ }
}
```

Use Case 3: More powerful composition

```
const myUtilityClasses = css`.u-myutil { property: value; }`;
const myValue = 'value';
const myMixin = () => css`property: value;`;

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      ${myUtilityClasses}
      .using-value {
        property: ${myValue};
      }
      .using-mixin {
        ${myMixin()}
      }
    `;
  }
  render() { /* template using classes above */ }
}
```

Use Case 3: More powerful composition

Use JavaScript constants for:

- static CSS
- mixins (static 🤷)

Use CSS Custom Properties for:

- dynamical CSS
- theming

Available today 👩‍💻👩‍💻

Try and share feedback 💬

- npmjs.com/package/lit-css
- npmjs.com/package/styled-lit-element

Support and contribute ⭐

- GitHub monorepository lit-styles/lit-styles
  - package lit-css
  - package styled-lit-element

lit-styles / lit-styles

Unwatch 5 | Star 20 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

Monorepository with lit-css and integration with other tools and frameworks.

Edit

Manage topics

28 commits | 1 branch | 5 releases | 1 contributor | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

bashmish chore(release): publish ... Latest commit 29a43d1 6 days ago

packages	chore(release): publish	6 days ago
.editorconfig	feat: css template literal with deduping and auto coercion to a string	a month ago
.eslintignore	chore(lit-css): setup monorepo for lit-css and other packages	8 days ago
.eslintrc	feat(lit-css): validate foreign object type to improve DX	6 days ago
.gitignore	chore(lit-css): setup monorepo for lit-css and other packages	8 days ago
.npmrc	chore(lit-css): setup monorepo for lit-css and other packages	8 days ago
LICENSE	feat: css template literal with deduping and auto coercion to a string	a month ago
README.md	docs: create initial monorepo docs	8 days ago
lerna.json	chore(lit-css): setup monorepo for lit-css and other packages	8 days ago
package.json	chore(lit-css): setup monorepo for lit-css and other packages	8 days ago

COMMUNITY IS KEY

Episode V.
The Reality Strikes Back

Where we learn about new challenges.

Why static instead of instance getter?

```
class MyAlert extends StyledLitElement {
  static get properties() {
    return { type: String };
  }
  get style() { /* instance getter => has access to properties */
    return css`
      :host {
        /* ShadyCSS performace killer (👋 IE11) */
        background-color: ${alertBgColors[this.type]};
        border: 1px solid ${alertBorderColors[this.type]};
        color: ${alertColors[this.type]};
        padding: 10px;
      }
    `;
  }
  render() {
    return html`<slot></slot>`;
  }
}
```

Why static instead of instance getter?

```
class MyAlert extends StyledLitElement {
  static get properties() {
    return { type: String };
  }
  get style() { /* instance getter => has access to properties */
    return css`
      :host {
        /* ShadyCSS performace killer (👋 IE11) */
        background-color: ${alertBgColors[this.type]};
        border: 1px solid ${alertBorderColors[this.type]};
        color: ${alertColors[this.type]};
        padding: 10px;
      }
    `;
  }
  render() {
    return html`<slot></slot>`;
  }
}
```

Why static instead of instance getter?

```
class MyAlert extends StyledLitElement {
  static get properties() {
    return { type: String };
  }
  get style() { /* instance getter => has access to properties */
    return css`
      :host {
        /* ShadyCSS performace killer (👋 IE11) */
        background-color: ${alertBgColors[this.type]};
        border: 1px solid ${alertBorderColors[this.type]};
        color: ${alertColors[this.type]};
        padding: 10px;
      }
    `;
  }
  render() {
    return html`<slot></slot>`;
  }
}
```

Why static instead of instance getter?

```
class MyAlert extends StyledLitElement {
  static get properties() {
    return { type: String };
  }
  get style() { /* instance getter => has access to properties */
    return css`
      :host {
        /* ShadyCSS performace killer (👋 IE11) */
        background-color: ${alertBgColors[this.type]};
        border: 1px solid ${alertBorderColors[this.type]};
        color: ${alertColors[this.type]};
        padding: 10px;
      }
    `;
  }
  render() {
    return html`<slot></slot>`;
  }
}
```

Why static instead of instance getter?

```
class MyAlert extends StyledLitElement {
  static get properties() {
    return { type: String };
  }
  get style() { /* instance getter => has access to properties */
    return css`
      :host {
        /* ShadyCSS performace killer (👋 IE11) */
        background-color: ${alertBgColors[this.type]};
        border: 1px solid ${alertBorderColors[this.type]};
        color: ${alertColors[this.type]};
        padding: 10px;
      }
    `;
  }
  render() {
    return html`<slot></slot>`;
  }
}
```

Why static instead of instance getter?

```
class MyAlert extends StyledLitElement {
  static get properties() {
    return { type: String };
  }
  get style() { /* instance getter => has access to properties */
    return css`
      :host {
        /* ShadyCSS performace killer (👋 IE11) */
        background-color: ${alertBgColors[this.type]};
        border: 1px solid ${alertBorderColors[this.type]};
        color: ${alertColors[this.type]};
        padding: 10px;
      }
    `;
  }
  render() {
    return html`<slot></slot>`;
  }
}
```

Why static instead of instance getter?

```
class MyAlert extends StyledLitElement {
  static get properties() {
    return { type: String };
  }
  get style() { /* instance getter => has access to properties */
    return css`
      :host {
        /* ShadyCSS performace killer (👋 IE11) */
        background-color: ${alertBgColors[this.type]};
        border: 1px solid ${alertBorderColors[this.type]};
        color: ${alertColors[this.type]};
        padding: 10px;
      }
    `;
  }
  render() {
    return html`<slot></slot>`;
  }
}
```

Code editors support

For now use **styled-components** plugins which have:

- css`` literal
- syntax highlighting
- autocomplete

In VSCode, WebStorm (JetBrains), etc...

To Do 👷👷

To Do 👷👷

— autoprefixer / PostCSS

To Do 👷👷

— autoprefixer / PostCSS

— minification

To Do 👷👷

— autoprefixer / PostCSS

— minification

— linting
(last minute change - stylelint should work)

Episode VI.
Return of the CSS-in-JS

Where we bring closer lit-css and CSS-in-JS again.

DISCLAIMER:

PURELY EXPERIMENTAL

AND DANGEROUS

DON'T DO THIS AT HOME

Mixin deduping issue

```
// good
const myMixinFunction = () => css`
  property: value;
  property2: value2;
`;

// bad: will be deduped, because it creates a module
const myPlainMixin = css`
  property: value;
  property2: value2;
`;
```

Mixin deduping issue

```
// good
const myMixinFunction = () => css`
  property: value;
  property2: value2;
`;

// bad: will be deduped, because it creates a module
const myPlainMixin = css`
  property: value;
  property2: value2;
`;
```

Mixin deduping issue

```
// good
const myMixinFunction = () => css`
  property: value;
  property2: value2;
`;

// bad: will be deduped, because it creates a module
const myPlainMixin = css`
  property: value;
  property2: value2;
`;
```

Mixin deduping issue

```
// good
const myMixinFunction = () => css`
  property: value;
  property2: value2;
`;

// bad: will be deduped, because it creates a module
const myPlainMixin = css`
  property: value;
  property2: value2;
`;
```

Mixin deduping issue

```
// good
const myMixinFunction = () => css`
  property: value;
  property2: value2;
`;

// bad: will be deduped, because it creates a module
const myPlainMixin = css`
  property: value;
  property2: value2;
`;
```

Mixin deduping issue

```
// good
const myMixinFunction = () => css`
  property: value;
  property2: value2;
`;

// bad: will be deduped, because it creates a module
const myPlainMixin = css`
  property: value;
  property2: value2;
`;
```

IDEA: cssdecl

Allows to define mixins (or **decl**aration blocks):

```
// will never be deduped
const myMixin = cssdecl`
  property: value;
  property2: value2;
`;
```

IDEA: cssdecl

Allows to define mixins (or **decl**aration blocks):

```
// will never be deduped
const myMixin = cssdecl`
  property: value;
  property2: value2;
`;
```

IDEA: cssdecl

Allows to define mixins (or **decl**aration blocks):

```
// will never be deduped
const myMixin = cssdecl`
  property: value;
  property2: value2;
`;
```

Dead code and bloated Shadow DOM styles

```
import myUtilityClasses from 'my-styles/utility-classes.js';

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      ${myUtilityClasses} /* what's in here? */
    `;
  }
}
```

Dead code and bloated Shadow DOM styles

```
import myUtilityClasses from 'my-styles/utility-classes.js';

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      ${myUtilityClasses} /* what's in here? */
    `;
  }
}
```

IDEA: styledHtml

```
import { StyledLitElement, styledHtml } from 'styled-lit-element';
import { myUtilityMixin1, myUtilityMixin2 } from 'my-styles/utility-mixins.js';

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      /* no need to inject mixins here */
    `;
  }
  render() {
    return styledHtml`
      <div class="${myUtilityMixin1} ${myUtilityMixin2}"></div>
    `;
  }
}
```

IDEA: styledHtml

```
import { StyledLitElement, styledHtml } from 'styled-lit-element';
import { myUtilityMixin1, myUtilityMixin2 } from 'my-styles/utility-mixins.js';

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      /* no need to inject mixins here */
    `;
  }
  render() {
    return styledHtml`
      <div class="${myUtilityMixin1} ${myUtilityMixin2}"></div>
    `;
  }
}
```

IDEA: styledHtml

```
import { StyledLitElement, styledHtml } from 'styled-lit-element';
import { myUtilityMixin1, myUtilityMixin2 } from 'my-styles/utility-mixins.js';

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      /* no need to inject mixins here */
    `;
  }
  render() {
    return styledHtml`
      <div class="${myUtilityMixin1} ${myUtilityMixin2}"></div>
    `;
  }
}
```

IDEA: styledHtml

```
import { StyledLitElement, styledHtml } from 'styled-lit-element';
import { myUtilityMixin1, myUtilityMixin2 } from 'my-styles/utility-mixins.js';

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      /* no need to inject mixins here */
    `;
  }
  render() {
    return styledHtml`
      <div class="${myUtilityMixin1} ${myUtilityMixin2}"></div>
    `;
  }
}
```

IDEA: styledHtml

```
import { StyledLitElement, styledHtml } from 'styled-lit-element';
import { myUtilityMixin1, myUtilityMixin2 } from 'my-styles/utility-mixins.js';

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      /* no need to inject mixins here */
    `;
  }
  render() {
    return styledHtml`
      <div class="${myUtilityMixin1} ${myUtilityMixin2}"></div>
    `;
  }
}
```

IDEA: styledHtml

```
import { StyledLitElement, styledHtml } from 'styled-lit-element';
import { myUtilityMixin1, myUtilityMixin2 } from 'my-styles/utility-mixins.js';

class MyComponent extends StyledLitElement {
  static get style() {
    return css`
      /* no need to inject mixins here */
    `;
  }
  render() {
    return styledHtml`
      <div class="${myUtilityMixin1} ${myUtilityMixin2}"></div>
    `;
  }
}
```

IDEA: styledHtml

Bootstrap-like utilities for screen readers:

```
html`
  <a class="sr-only sr-only-focusable" href="#content">Skip to main content</a>
`

const srOnly = cssdecl`/* CSS properties */`;
const srOnlyFocusable = cssdecl`/* CSS properties */`;

styledhtml`
  <a class="${srOnly srOnlyFocusable}" href="#content">Skip to main content</a>
`
```

IDEA: styledHtml

Bootstrap-like utilities for screen readers:

```
html`
  <a class="sr-only sr-only-focusable" href="#content">Skip to main content</a>
`

const srOnly = cssdecl`/* CSS properties */`;
const srOnlyFocusable = cssdecl`/* CSS properties */`;

styledhtml`
  <a class="${srOnly srOnlyFocusable}" href="#content">Skip to main content</a>
`
```

IDEA: styledHtml

Bootstrap-like utilities for screen readers:

```
html`
  <a class="sr-only sr-only-focusable" href="#content">Skip to main content</a>
`

const srOnly = cssdecl`/* CSS properties */`;
const srOnlyFocusable = cssdecl`/* CSS properties */`;

styledhtml`
  <a class="${srOnly srOnlyFocusable}" href="#content">Skip to main content</a>
`
```

IDEA: styledHtml

Bootstrap-like utilities for screen readers:

```
html`
  <a class="sr-only sr-only-focusable" href="#content">Skip to main content</a>
`

const srOnly = cssdecl`/* CSS properties */`;
const srOnlyFocusable = cssdecl`/* CSS properties */`;

styledhtml`
  <a class="${srOnly srOnlyFocusable}" href="#content">Skip to main content</a>
`
```

IDEA: styledHtml

Bootstrap-like utilities for screen readers:

```
html`
  <a class="sr-only sr-only-focusable" href="#content">Skip to main content</a>
`

const srOnly = cssdecl`/* CSS properties */`;
const srOnlyFocusable = cssdecl`/* CSS properties */`;

styledhtml`
  <a class="${srOnly srOnlyFocusable}" href="#content">Skip to main content</a>
`
```

IDEA: styledHtml

Bootstrap-like utilities for screen readers:

```
html`
  <a class="sr-only sr-only-focusable" href="#content">Skip to main content</a>
`

const srOnly = cssdecl`/* CSS properties */`;
const srOnlyFocusable = cssdecl`/* CSS properties */`;

styledhtml`
  <a class="${srOnly srOnlyFocusable}" href="#content">Skip to main content</a>
`
```

Should I #usetheplatform?

Thanks!

Web Components and Styles Scoping

by Mikhail Bashkirov

for FrontMania 2018

Slides: bit.ly/wc-fm2018

Twitter: @bashmish
GitHub: @bashmish
GitHub: lit-styles

Q&A