CSS Modules and Constructable Stylesheets

If you work with web components, one of the pain points is how to add styles to multiple copies of the same component.

None of the existing solutions work well and all have one or more rough edges. See the CSS Modules Scripts Chrome Status Entry for an explanation of the rough edges as seen by the team implementing the feature.

CSS Module Scripts (CSS Modules for short) allows you to import and external stylesheet and attach it to multiple documents and components.

```
import sheet from './index.css' assert { type: 'css' };

document.adoptedStyle'css's = [...document.adoptedStyleSheets, sheet];

// Attach to a shadow root
shadowRoot.adoptedStyleSheets = [...document.adoptedStyleSheets, sheet];
```

The first step is to import the stylesheet. The `assert { type: 'css' }` statement is important. It tells the Javascript engine that the module you're importing is a CSS module. Otherwise, the module will be interpreted as a Javascript module and the import will fail (modules imports use strict mime-types).

To add the imported module we use `adoptedStyleSheets` on the document or shadow root and append the imported stylesheet to the existing array of stylesheets.

The `adoptedStyleSheets` method is part of the constructable stylesheets proposal.

We can also use this technique with dynamic imports.

Here the main difference is use `cssModule.default` rather than `cssModule` since dynamic imports return a module namespace object. The CSSStyleSheet is the default export of the module, so it's accessed at `cssModule.default`.

```
<!-- Dynamic Import -->
const cssModule = await import('./index.css', {
  assert: { type: 'css' }
});
document.adoptedStyleSheets = [...adoptedStylesheets, cssModule.default];
```

Working with shadow DOM

When working with Shadow DOM, the process gets a little more complicated (at least for me).

We define the element in its own script and then we import it using a `script` tag in the host page.

The second script will import the stylesheet and attach it to the custom element's shadow root.

I always prefer to wait on user action so we create a button and capture a reference in the `myButton` constant.

We attach a click event handler to the button so when the user clicks the button, the browser will create a new `demo-element` element, append it to the body of the host page (making it appear) and attaching the imported stylesheet using the `shadowRoot.adoptedStyelSheets` method. Note that we also add a spread of all existing adopted stylesheets to the array we assign to the custom element. This way, if the element has already defined styles, we won't lose them.

```
<!-- Imports the demo-element element -->
<script src="./demo.js"></script>

<script src="./demo.js"><!-- add elements and import stylesheet-->
<script type="module">
import sheet from './index2.css' assert { type: 'css' };

const myButton = docume'css'tElementById('newwin');

'newwin'// button event listener
```

```
myButton.addEventListener('click', () => {
  const win = document.createElement('demo-element');
  document.body.appendChild(win);
  win.shadowRoot.adoptedStyleSheets = [
    ...document.adoptedStyleSheets,
    sheet
  ];
});
</script>
```

SO that’s it! with CSS modules and constructable stylesheets you can create custom elements that can be styled without having to create style elements to go with them