A New Experimental Feature: scoped stylesheets

By Alex Danilo

Alex is a contributor to Web**Fundamentals**

Chromium recently implemented a new feature from HTML5: scoped stylesheets, aka. **<style scoped>**. A web author can limit style rules to only apply to a part of a page by setting the 'scoped' attribute on a **<style>** element that is the direct child of the root element of the subtree you want the styles to be applied to. This limits the styles to affect just the element that is the parent of the **<style>** element and all of its descendants.

Example

Here's a simple document that uses standard styling:

```
<html>
<body>
  <div>a div! <span>a span!</span></div>
    <div>
      <style>
        div { color: red; }
        span { color: green; }
      </style>
      a div! <span>a span!</span></div>
  <div>a div! <span>a span!</span></div>
</body>
</html>
```

The style rules specified will color text within any ** ** red, and within any **** green:

a div! a span!
a div! a span!
a div! a span!

However, if we set **scoped** on the **<style>** element:

```
<html>
<body>
  <div>a div! <span>a span!</span></div>
```

```
    <div>
      <style scoped>
        div { color: red; }
        span { color: green; }
      </style>
      a div! <span>a span!</span></div>
  <div>a div! <span>a span!</span></div>
</body>
</html>
```

then it restricts the style rules so that they're applied to the enclosing `<div>` that is the parent of the `<style scoped>` element and anything inside just that `<div>`. We call this 'scoped' and the result looks like:

a div! a span!
a div! a span!
a div! a span!

This of course can be done anywhere in the markup. So if you're adventurous, you could nest scoped styles within other scoped parts of the markup as much as you like to get fine-grained control over where styles get applied.

Use cases

Now what is this good for?

A common use case is syndicated content: when you as a web author would like to incorporate content from a third party, including all its styles, but do not want to risk those styles "polluting" other, unrelated parts of the page. A great advantage here is the ability to combine content from other sites like yelp, twitter, ebay, etc. into a single page without needing to isolate them using an `<iframe>` or on-the-fly editing the external content.

If you're using a content management system (CMS) that sends you snippets of markup that are all mashed together into a final page display then this is a great feature to make sure each snippet gets styled in isolation from anything else on the page. This can be just as useful for a wiki as well.

When you want to author some nice demo code on a page, it's easy to limit the styles to just the demo content. That lets you go wild with the CSS on the demo, yet nothing else on the page will get affected.

Another use case is simply encapsulation: for example, if your web page has a side menu, it makes sense to put styles that are specific to that menu into a `<style scoped>` section in

that part of the markup. Those style rules won't have any effect when rendering other parts of the page, which keeps them nicely separated from the main content!

Possibly one of the most compelling use cases is for the web component model. Web components are going to be a great way to build things like sliders, menus, date pickers, tab widgets, etc. By providing the scoped styles, a designer can build a widget and package it with their styles as a self-contained unit that others can grab and combine into a rich web application. We plan to use **<style scoped>** heavily with web components and the shadow DOM (that can already be enabled by setting the experimental "shadow DOM" flag in chrome://flags). Right now there's no really good way to make sure that styles are limited to web components without resorting to bad practices like inline styling, so scoped styles are a perfect fit for this.

Why include the parent element?

The most natural way is to include the parent element so that the **<style scoped>** rules could, for example, set a common background color for the entire scope. It also allows scoped style sheets to be written "defensively" for browsers that don't yet support **<style scoped>**, by prefixing rules with an ID or class selector as a fallback:

```
<div id="menu">
  <style scoped>
    #menu .main { … }
    #menu .sub { … }
  …
```

This mimics the effect of using styles when 'scoped' is implemented but with some run-time performance penalty due to the more complex selector. The nice thing about this approach is that it allows for a graceful fallback approach until the day when **<style scoped>** is widely supported and the ID selectors could simply be dropped.

Status

Given that the implementation of scoped style sheets is still new, they are currently hidden behind a run-time flag in Chrome. To enable them you need to get a version of Chrome that has a version number of 19 or higher (Chrome Canary right now), then locate the 'Enable **<style scoped>**' entry in chrome://flags (towards the end), click 'Enable' and then restart the browser.

There are currently no known bugs, but **@global** and scoped versions of **@keyframes** and **@-webkit-region** and are stil in the process of being implemented. Also, **@font-face** is ignored

for the time being since there is a good chance that the spec will change.

We would like to encourage everyone interested in the feature to try it out and let us know about your experiences: the good, the bad and (maybe) the buggy.

Last updated July 2, 2018.