What's the CSS :scope pseudo-class for?

By Eric Bidelman

Engineer @ Google working on web tooling: Headless Chrome, Puppeteer, Lighthouse

`:scope` is defined in CSS Selectors 4 as:

> A pseudo-class which represents any element that is in the contextual reference element set. This is is a (potentially empty) explicitly-specified set of elements, such as that specified by the `querySelector()`, or the parent element of a `<style scoped>` element, which is used to "scope" a selector so that it only matches within a subtree.

An example of using this guy is within a `<style scoped>` (more info):

```
<style>
  li {
    color: blue;
  }
</style>

<ul>
  <style scoped>
    li {
      color: red;
    }
    :scope {
      border: 1px solid red;
    }
  </style>
  <li>abc</li>
  <li>def</li>
  <li>efg</li>
</ul>

<ul>
  <li>hij</li>
  <li>klm</li>
  <li>nop</li>
</ul>
```

Note: `<style scoped>` can be enabled in Chrome using the "Enable experimental WebKit features" flag in about:flags.

This colors the `li` elements in the first `ul` red and, because of the `:scope` rule, puts a border around the `ul`. That's because in the context of this `<style scoped>`, the `ul` matches `:scope`. It's the local context. If we were to add a `:scope` rule in the outer `<style>` it would match the entire document. Essentially, equivalent to `:root`.

Contextual elements

You're probably aware of the `Element` version of `querySelector()` and `querySelectorAll()`. Instead of querying the entire document, you can restrict the result set to a contextual element:

```
<ul>
 <li id="scope"><a>abc</a></li>
 <li>def</li>
 <li><a>efg</a></li>
</ul>
<script>
  document.querySelectorAll('ul a').length; // 2

  var scope = document.querySelector('#scope');
  scope.querySelectorAll('a').length; // 1
</script>
```

When these are called, the browser returns a `NodeList` that's filtered to only include the set of nodes that a.) match the selector and b.) which are also descendants of the context element. So in the the second example, the browser finds all `a` elements, then filters out the ones not in the `scope` element. This works, but it can lead to some bizarre behavior if you're not careful. Read on.

When querySelector goes wrong

There's a *really* important point in the Selectors spec that people often overlook. Even when `querySelector[All]()` is invoked on an element, **selectors still evaluate in the context of the entire document**. This means unanticipated things can happen:

```
scope.querySelectorAll('ul a').length); // 1
scope.querySelectorAll('body ul a').length); // 1
```

WTF! In the first example, **`ul`** *is* my element, yet I'm still able to use it and matches nodes. In the second, **body** isn't even a descendant of my element, but "**`body ul a`**" still matches. Both of these are confusing and not what you'd expect.

It's worth making the comparison to jQuery here, which takes the right approach and does what you'd expect:

```
$(scope).find('ul a').length // 0
$(scope).find('body ul a').length // 0
```

...enter **`:scope`** to solve these semantic shenanigans.

Fixing querySelector with :scope

WebKit recently landed support for using the **`:scope`** pseudo-class in **`querySelector[All]()`**. You can test it in Chrome Canary 27.

You can use it **restrict selectors to a context element**. Let's see an example. In the following, **`:scope`** is used to "scope" the selector to the scope element's subtree. That's right, I said scope three times!

```
scope.querySelectorAll(':scope ul a').length); // 0
scope.querySelectorAll(':scope body ul a').length); // 0
scope.querySelectorAll(':scope a').length); // 1
```

Using **`:scope`** makes the semantics of the **`querySelector()`** methods a little more predictable and inline with what others like jQuery are already doing.

Performance win?

Not yet :(

I was curious if using **`:scope`** in qS/qSA gives a performance boost. So...like a good engineer I threw together a test. My rationale: less surface area for the browser to do selector matching means speedier lookups.

In my experiment, WebKit currently takes ~1.5-2x longer than not using `:scope`. Drats! When crbug.com/222028 gets fixed, **using it should theoretically give you a slight performance boost** over not using it.

Last updated July 2, 2018.