CSS Containment

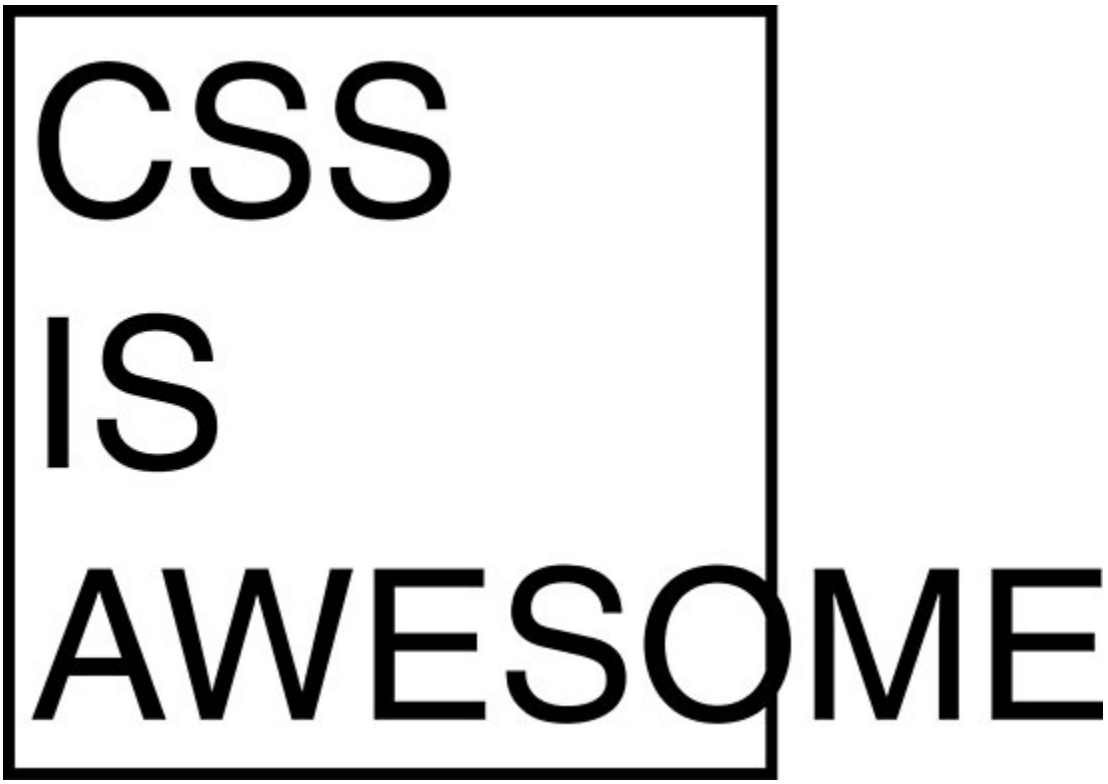

Figure 1: Containment may help prevent this and make CSS even more awesome :)

Whenever we insert HTML elements after the document loads by inserting new CCSS rules or new elements via Javascript, we may be slowing down the rendering of the page because every change means the browser has to navigate all the elements in scope and re-render them as needed, they may have been moved or changed their dimensions when our target element grew smaller or larger;

Layout is almost always scoped to the entire document meaning that the browser will navigate all the way to the begnning of the document to calculate sizes and layout for the document. If you have a lot of elements, it's going to take a long time to figure out their locations and dimensions.

The `contain` CSS property allows an author to indicate that an element and its contents are, **as much as possible**, independent of the rest of the document tree. This allows the browser to recalculate layout, style, paint, size, or any combination of them for a limited area of the DOM and not the entire page.

It can take one or more of the following values:

`size`

The size of the element can be computed without checking its children, the

element dimensions are independent of its contents.

`layout`
: The internal layout of the element is totally isolated from the rest of the page, it's not affected by anything outside and its contents cannot have any effect on the ancestors.

`style`
: Indicates that, for properties that can have effects on more than just an element and its descendants, those effects don't escape the containing element.

  The style values has been marked `at risk` and, as such, it may not make it to the final recomendation. Mozilla has already dropped it from Firefox.

`paint`
: Descendants of the element cannot be displayed outside its bounds, nothing will overflow this element (or if it does it won't be visible).

In addition there are two grouping values that shorten what you type as the value of the attribute:

strict
: This value turns on all forms of containment except style contain for the element. It behaves the same as `contain: size layout paint`

content
: This value turns on all forms of containment ***except*** size containment and style containment for the element. It behaves the same as `contain: layout paint;`.

When we add the `newly-added-element` element to the page, it will trigger styles, layout and paint but, one thing we need to consider, is that the DOM for the whole document is in scope. The browser will have to consider all the elements irrespective of whether or not they were changed when it comes to styles layouts and paint.

The bigger the DOM, the more computation work the browser has to do, meaning that your app may become unresponsive to user input in lager documents.

In addition to what the browser already does to help with scoping of your CCSS, you can use the scope property of CSS as an additional indicator of how the

browser should handle layout, size and paint containment.

In the example below adding the `new-element` div will cause styles, layout and paint redraw of the whole document tree. For illustration we haven't added content to the HTML but you can imagine how large it can become, particularly in a single page application.

```
<section class="view">
  Home
</section>

<section class="view container">
  About
  <div class="new-element">Check me out!</div>
</section>

<section class="view">
  Contact
</section>
```

In CSS we can use containmnet to help the browser out with optimizations. It would be temmpting to use `strict` for all items that we want to use containment for but we need to know the dimensions ahead of time and include them in our CSS otherwise the element might be rendered as a 0px by 0px box. Test everything thoroughly both in browsers that support containment and those that don't support it.

Content containment (`contains: content`) offers significant scope improvements, without having to specify the dimensions of the element ahead of time.

You should look at `contain: content` as your default and treat `contain: strict` as an escape hatch when `contain: content` doesn't quite cut the mustard.

To make sure that the layout and paint for our `new-element` div don't affect the rest of the document, we can use a rule like this:

```
.new-element {
  contain: content;
  /* the rest of the rules for the class */
}
```

Links and resources

- An introduction to CSS Containment
- CSS Containment Module Level 1
- Can I use: CSS containment
- CSS Triggers — What gets triggered by mutating a given property
- CSS Containment in Chrome 52
- Avoid Large, Complex Layouts and Layout Thrashing
- CSS Contain