Ingo Steinke
Posted on 21 Dec 2021 • Updated on 3 Jan

CSS :has(.parent-selectors)

#css #webdev #todayilearned #javascript

What's next in CSS? (3 Part Series)

1. **CSS :has(.parent-selectors)**
2. Aspect ratio: no need for container units!
3. Animated Gradient Text Color

I wonder why I have to follow "Tech Twitter" to find out the good news, so I'm the one to write a short post here on dev.to to celebrate a new CSS feature:

"Parent selectors", the second most awaited CSS feature according to State of CSS survey 2021, also known as the **has-selector**, have got browser support!

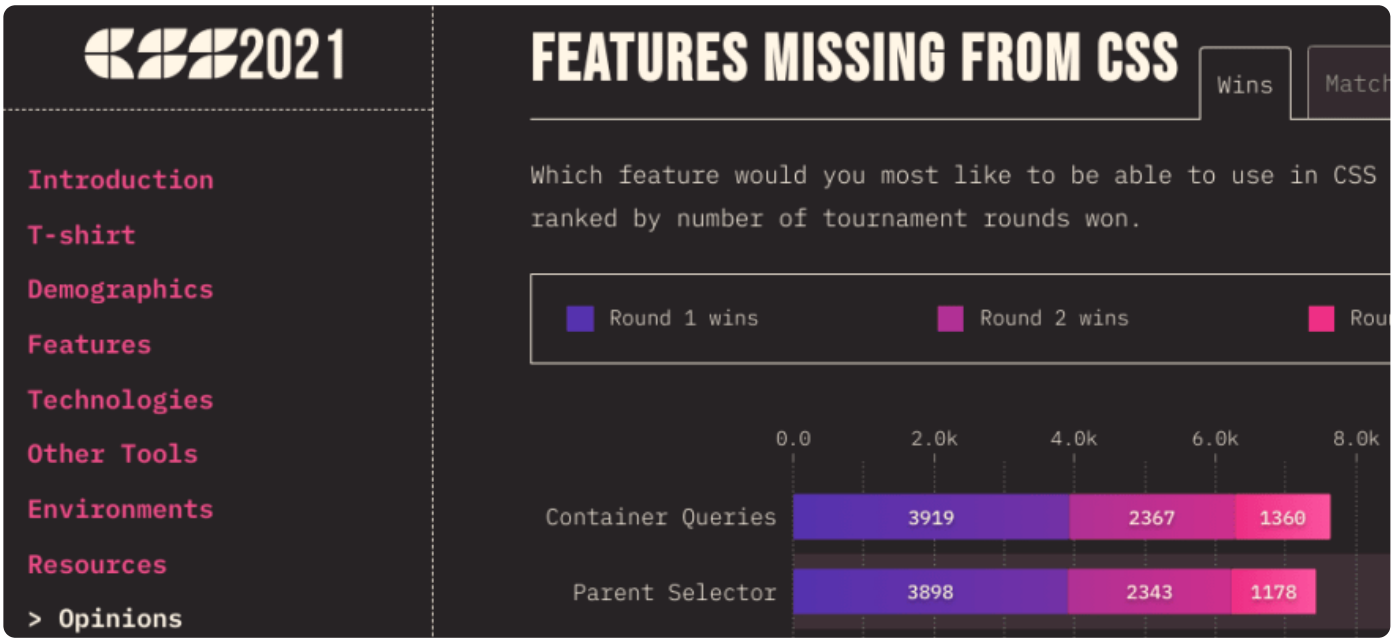
To quote Sara Soueidan quoting Jen Simmons on Twitter:

> :has() is essentially the long-awaited parent selector in CSS 🎊

> Don’t say Safari is always last. Sometimes we are first.

No longer "Missing from CSS"

Now parent selectors are no longer missing from CSS, let's hope that Firefox and Chromium follow quickly.



What does a "Parent Selector" select?

Parent selectors select parent elements, right? They actually select grandparents and any matching ancestors as well.

I haven't been the only one thinking of `:has()` as a "child selector", so should I call them "has-selectors" to avoid misunderstanding?

Timothy Huang called `:has()` "a CSS-selector that (selects) a parent with child which sounds like an appropriate description to me.

From caniuse.com/css-has:

> For example, `a:has(>img)` selects all `<a>` elements that contain an `<img>` child.

The `:has()` CSS pseudo-class is documented well on MDN.

Performance Considerations

The main reason that is took so long to implement is the fear of costly calculations. Parent selectors might be a feature that can hurt your site's speed and performance when applied to a document in real time.

Chris Coyier cited Jonathan Snook (back in 2010) "that when elements are dynamically added and removed from the page, it may result in the entire document needing to be re-rendered (major memory usage concerns)".

Maybe we should probably be extra careful to measure our performance when we actually start using parent selectors?

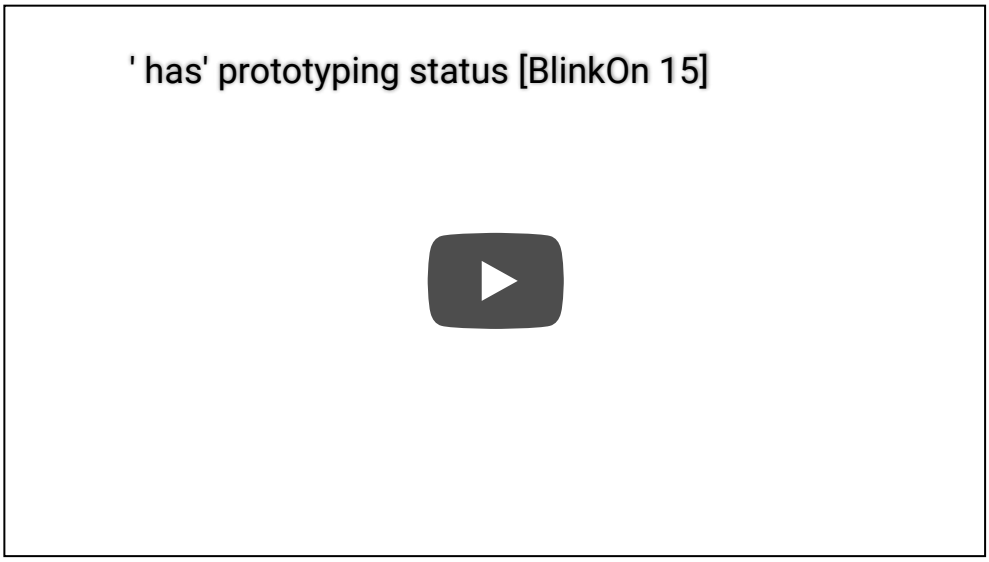
Implementation that sidesteps Performance Issues:

Update: the performance problems have probably been solved. Eric Meyer mentioned a talk about nerdy details of how the implementation sidesteps performance issues.

After watching Byungwoo Lee on YouTube, I would say that the Blink engine's strategy is somehow similar to the efficiency of a chess engine that must figure out how to ignore irrelevant moves quickly instead of predicting every possible outcome of every possible combination of moves.

In the case of CSS, the Blink engine will prevent walk up and invalidation on irrelevant elements. To reduce the irrelevant recalculations after applying the style, the engine can mark if a style is affected by a `:has()` state change during the recalculation.

But let Byungwoo Lee explain the details of the problems and solutions implementing parent selectors.



His explanation includes complex use cases like

```
.a:has(.b ~ .c)
```

or

```
.a:is(:has(b), :has(c))
```

Wow! I didn't even know that could be valid CSS.

Never stop learning! But still don't show code like that to me in a code review. I will probably request you to refactor that to something which is more easy to understand and maintain for the human brain!

Actual Use Case

As I see many people struggle to contrive examples how to make use of has selectors: lucky you!

Here is a real world example: I had to hotfix a Shopware theme that had already been hotfixed before, and it was urgent and `!important`, so no clean code at that part of the roadmap at least.

I wish I had been able to use `:has()` here, to lower the risk of accidentally targeting the wrong elements in the CMS-generated markup. The selector is so long that you have to scroll to see all of it!

```
/* override template heading style */
body.is-act-index .cms-sections .col-12 .cms-element-alignment.align-self-start
```

Using `has` makes the fix at least a little bit clearer:

```
/* override template heading style */
body.is-act-index .cms-element-alignment:has(> h1) {
```

We might want to write that code just for the sake of readability. But we have to ensure browser support.

How to Polyfill :has() Selectors?

As there is no way to workaround parent selectors in recent CSS syntax, they can't be transpiled. Don't hope for PostCSS or SASS! This time you will need JavaScript to polyfill older browsers.

"jQuery has had the :has selector in its arsenal since basically forever", Eric Ponto wrote in 2015 already showing a polyfill based on jQuery:

```
Polyfill({
    selectors: [":has"]
}).doMatched(rules => {
    rules.each(rule => {
        // just pass it into jQuery since it supports `:has`
        $(rule.getSelectors()).css(rule.getDeclaration())
    });
});
```

Quiz: How to polyfill without using jQuery?

Take the quiz and submit your Vanilla JS solution!

```
// TODO: add a parent selector polyfill without using jQuery
```

If you know the solution, you can also post it as an answer to the StackOverflow question if there is a vanilla JS equivalent of jQuery .has().

querySelectorAllWithHas

Josh Larson's polyfill-css-has provides a `querySelectorAllWithHas` (thanks to @lukeshiru for the link!)

But we have managed to live without parent selectors for so long, maybe we don't want to worry anymore, and rather move on to there next upcoming upgrades to the CSS language:

What's next in CSS?

What to expect from CSS in 2022?

There are more useful features in the pipeline, like CSS Container Queries which we can already use in Chrome and Edge by enabling them using feature flags.

This article is part of a small series about new (and underrated) CSS features that you can start to learn and use in 2022.

What's next in CSS? (3 Part Series)

1. **CSS :has(.parent-selectors)**
2. Aspect ratio: no need for container units!
3. Animated Gradient Text Color

Discussion (10)

Iain Simmons • Dec 28 '21

Can you please share more of the code surrounding your use case? Because it seems like the only thing you add to the parent is `width: 100%`, and the rest you are nesting in a `h1` selector anyways.

A trick I use for CMS generated HTML (i.e. everything without classes) is to use a `:not([class])` selector. Then anything else, e.g. an unordered list or hyperlinks in a nav menu, just need any class added (or manually override the styles).

Ingo Steinke • Dec 28 '21

Here is an example (similar markup from a demo shop, with h2 headings. The top-level `cms-section.cms-section-default` only differ by their `pos-n` position classes which have no benefit compared to `nth-child`.

Any other distinction, even the content id's, can only be made on a descendent level.

The structure is as follows:

```
<div class="cms-sections">
    <div class="cms-section pos-0 cms-section-default">
        <div class="cms-block-container" style="/* some inline style */">
            <div class="cms-block-container-row row cms-row">
                <div class="col-12" data-cms-element-id="somecrypticidstring">
                    <div class="cms-element-text">
                        <h2>Headline (not every CMS content type has a headlin
                        <p>Lorem ipsum...</p>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

Here is a screenshot as well

Iain Simmons • Dec 29 '21

Thanks, but sorry I still don't see the value here.

Your CSS only applied `width: 100%` in the `:has` selector, which a `div` would have by default as a block level element.

Otherwise, if targeting the `h2` or `p`, you could do that with descendent selectors or the `:not([class])` as I suggested in my previous comment.

Perhaps a better example could be for forms or something, where you could combine with the `:invalid` pseudo-class or similar to change the parent `div` styles:

```
.form-control:has(:invalid) {
  background-color: rgb(220 53 69 / 0.25);
}
```

Ingo Steinke • Dec 29 '21

Thanks for the form example!

Still don't see how you would have done it with `not` when all the `div` elements are undistinguishable by their class names. Of course, a div should be a block level element by default. You are lucky you didn't have to deal with the existing CSS. Using descendant selectors was what I did in the end, still didn't find it elegant or robust, probably it will break after the next framework update.

Better way would be if the customers content editors would choose different types of block templates which could then be given distinct class names.
Complex code should always arise suspicion that something might be conceptually wrong by design and could be solved at the problem's root cause. That's what I meant by "hotfixing a hotfix" in CSS (oh, and don't even start to count the number of `!important` in the existing style sheets).

Iain Simmons • Dec 30 '21

Sorry, I didn't make that clear. The `:not([class])` would be for targeting the paragraphs or headings that come from the CMS or whatever, where you don't have the ability to add or target classes.

e.g.

```
body {
  color: #000;
}

p:not([class]) {
  font-size: 1rem;
  color: #222;
}

.lead {
  font-size: 1.25rem;
}


<main>
  <p class="lead">Lorem ipsum...</p>
  <!-- other content not generated by the CMS -->

<div class="cms-sections">
    <div class="cms-section pos-0 cms-section-default">
        <div class="cms-block-container" style="/* some inline style */">
            <div class="cms-block-container-row row cms-row">
                <div class="col-12" data-cms-element-id="somecrypticidstrir
                    <div class="cms-element-text">
                        <h2>Headline</h2>
```

```
                        <p>CMS Content. Lorem ipsum...</p>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

The lead paragraph would be black, inheriting the color from the body.

So the not selector makes them act like global element styles that you opt out of by simply adding a class to the HTML element (even a blank string).

The assumption is that anything you want custom styling for that isn't regular body copy content (from a CMS or similar), you probably have control over the HTML and would normally be adding classes to style them anyways.

But anyways, this is obviously a whole other conversation! 🙂

Yutamago • Dec 22 '21

Last time I checked, this was still a working draft. It doesn't make sense to implement this for any browser as long as the details are still not set in stone.
I'm waiting for this feature too. Had a use case for it yesterday and had to reside to a workaround instead. :(

Ingo Steinke • Dec 21 '21 • Edited on Dec 21

Updated my article quoting Byungwoo Lee, Eric Meyer, Chris Coyier and Jonathan Snook on performance issues of parent selectors and how the Blink team finally solved them. Included jQuery code as it's still the only working polyfill I found.
Anyone know a vanilla JS solution?

LUKESHIRU • Dec 21 '21

github.com/jplhomer/polyfill-css-h...

Ingo Steinke • Dec 21 '21

Thanks @lukeshiru ! That looks very useful. So we can use `has` in css and add `querySelectorAllWithHas` as a progressive enhancement for any browser without has support.

Ingo Steinke • Dec 21 '21

Another update: added an actual use case I found in my git history, as many people seemed to have a hard time imagining any useful scenario for `:has()` apart from the simple MDN example.

Ingo Steinke

Web Development, Sustainability, Art and Music, Nature and Travel, Sustainability

LOCATION
Germany

WORK
Creative Web Developer at Ingo Steinke

JOINED
21 Sep 2019

More from Ingo Steinke

Animated Gradient Text Color
#webdev #showdev #css #tutorial

Aspect ratio: no need for container units!
#css #html #webdev #todayilearned

Printable Lazy Loading
#webdev #html #performance #a11y