

CSS is Awesome



Brandon Smith

Jul 3, 2017

I bought this mug recently for use at work. Being a professional web developer, I decided it would establish me as the office's king of irony. The joke on it isn't unique, of course. I've seen it everywhere from t-shirts to conference presentations.



Most of you reading this have probably encountered this image at least once. It's a joke we can all relate to, right? You try and do something simple with CSS, and the arcane ways in which even basic properties interact inevitably borks it up.

If this joke epitomizes the collective frustration that developers have with CSS, then at the risk of ruining the fun, I thought it would be interesting to dissect the bug at its heart, as a case study in why people get frustrated with CSS.

↳ (#the-problem) The problem

Embedded Pen Here

There are three conditions that have to be met for this problem to occur:

- ⦿ The content can't shrink to fit the container
- ⦿ The container can't expand to fit the content
- ⦿ The container doesn't handle overflow gracefully

In real-world scenarios, the second condition is most likely the thing that needs to be fixed, but we'll explore all three.

↳ (#fixing-the-content-size) Fixing the content size

This is little bit unfair to the box's content because the word **AWESOME** can't fit on one line at the given font size and container width. By default, text wraps at white space and doesn't break up words. But let's assume for a moment that we absolutely cannot afford to change the container's size. Perhaps, for instance, the text is a blown-up header on a site that's being viewed on an especially small phone.

↳ (#breaking-up-words) Breaking up words

To get a continuous word to wrap, we have to use the CSS property `word-break`. Setting it to `break-all` will instruct the browser to break up words if necessary to wrap text content within its container.

Embedded Pen Here

↳ (#other-content) Other content

In this case, the only way to make the content more responsive was to enable word breaking. But there are other kinds of content that might be overflowing. If `word-wrap` were set to `nowrap`, the text wouldn't even wrap in-between words. Or, the content could be a block-level element, whose `width` or `min-width` is set to be greater than the container's width.

↳ [\(#fixing-the-container-size\)](#) **Fixing the container size**

There are many possible ways the container element might have been forced to not grow. For example: `width`, `max-width`, and `flex`. But the thing they all have in common, is that *the width is being determined by something other than its content*. This isn't inherently bad, especially since there is no fixed height, which in most cases would cause the content to simply expand downwards. But if you run into a variation on this situation, it's worth considering whether you really *need* to be controlling the width, or whether it can be left up to the page to determine.

↳ [\(#alternatives-to-setting-width\)](#) **Alternatives to setting width**

More often than not, if you set an element's `width`, and you set it in pixels, you really meant to set either `min-width` or `max-width`. Ask yourself what you really care about. Was this element disappearing entirely when it lacked content because it shrunk to a width of 0? Set `min-width`, so that it has dimension but still has room to grow. Was it getting so wide that a whole paragraph fit on one line and was hard to read? Set `max-width`, so it won't go beyond a certain limit, but also won't extend beyond the edge of the screen on small devices. CSS is like an assistant: you want to guide it, not dictate its every move.

↳ [\(#overflow-caused-by-flexbox\)](#) **Overflow caused by flexbox**

If one of your flex items has overflowing content, things get a little more complicated. The first thing you can do is check if you're specifying its width, as in the previous section. If you aren't, probably what's happening is the element is "flex-shrinking". Flex items first get sized following the normal rules; `width`, `content`, etc. The resulting size is called their `flex-basis` (which can also be set explicitly with a property of the same name). After establishing the flex basis for each item, `flex-grow` and `flex-shrink` are applied (or `flex`, which specifies both at once). The items grow and shrink in a weighted way, based on these two values and the container's size.

Setting `flex-shrink: 0` will instruct the browser that this item should never get smaller than its flex basis. If the flex basis is determined by content (the default), this should solve your problem. be careful with this, though. You could end up running into the same problem again in the element's parent. If this flex item refuses to shrink, even when the flex container is smaller than it, it'll overflow and you're back to square one.

▷ (#handling-overflow) Handling overflow

Sometimes there's just no way around it. Maybe the container width is limited by the screen size itself. Maybe the content is a table of data, with rows that can't be wrapped and columns that can't be collapsed any further. We can still handle the overflow more gracefully than just having it spill out wherever.

▷ (#overflow-hidden) `overflow: hidden;`

The most straightforward solution is to hide the content that's overflowing. Setting `overflow: hidden;` will simply cut things off where they reach the border of the container element. If the content is of a more aesthetic nature and doesn't include critical info, this might be acceptable.

Embedded Pen Here

If the content is text, we can make this a little more visually appealing by adding `text-overflow: ellipsis;`, which automatically adds a nice little "..." to text that gets cut off. It is worth noting, though, that you'll see slightly less of the actual content to make room for the ellipsis. Also note that this requires `overflow: hidden;` to be set.

Embedded Pen Here

▷ (#overflow-auto) `overflow: auto;`

The preferable remedy is usually going to be setting `overflow-x: auto;`. This gives the browser the go-ahead to add a scroll bar if the content overflows, allowing the user to scroll the container in that direction.

Embedded Pen Here

This is a particularly graceful fallback, because it means that *no matter what*, the user will be able to access all of the content. Plus, the scrollbar will only appear if it's needed, which means it's not a bad idea to add this property in key places, even if you don't expect it to come into play.

▷ (#why-does-this-conundrum-resonate-so-universally-with-people-who-have-used-css) Why does this conundrum resonate so universally with people who have used CSS?

CSS is hard because its properties interact, often in unexpected ways. Because when you set one of them, you're never just setting that one thing. That one thing combines and bounces off of and contradicts with a dozen other things, including default things that you never actually set yourself.

One rule of thumb for mitigating this is, **never be more explicit than you need to be**. Web pages are responsive by default. Writing good CSS means leveraging that fact instead of overriding it. Use percentages or viewport units instead of a media query if possible. Use `min-width` instead of `width` where you can. Think in terms of rules, in terms of *what you really mean to say*, instead of just adding properties until things look right. Try to get a feel for how the browser resolves layout and sizing, and make your changes and additions on top of that judiciously. Work with CSS, instead of against it.

Another rule of thumb is to **let either width or height be determined by content**. In this case, that wasn't enough, but in most cases, it will be. Give things an avenue for expansion. When you're setting rules for how your elements get sized, especially if those elements will contain text content, think through the edge cases. "What if this content

was pared down to a single character? What if this content expanded to be three paragraphs? It might not look great, but would my layout be totally broken?”

CSS is weird. It’s unlike any other code, and that makes a lot of programmers uncomfortable. But used wisely it can, in fact, be awesome.