When to use float, inline-block, table or flex?

8-10 minutes The Blog of Karen Menezes - 13 April, 2014

I avoid using CSS frameworks for my own projects, because I personally find most of them too dictatorial for custom design or final production code. You may beg to differ, of course.

However, it's always a good idea to see how different frameworks handle their grids. You'll generally see attribute selectors like selector[class*="col-"], which uses a substring match and may interfere with other classes, !importants and plenty of decimal points. For smaller projects, grids are often as simple as floating a 75% and 25% column and adding some padding.

Secondly, most ready-to-ship grids have some custom breakpoints (often called mobile/tablet/desktop or small/medium/large), which is not ideal, because your breakpoints depend on where the content breaks and not at some random idea of a device breakpoint. Not that you can't change these using a mixin via a preprocessor that many of these frameworks depend on, but different projects have different needs. The web has always been fluid, so it's best to think of your content and usability and then design around that.

Using box-sizing border-box universally makes grids so much simpler. I'm a big fan of using padding instead of margins, where possible, since border-box includes padding

in the total width/height of the element.

I also love calc. It has great browser support. With calc, you could effectively have 7 boxes of equal width by declaring calc: width(100% / 7). How cool is that!

Most grids and front-end frameworks I've seen insist on floats for all layouting. But sometimes, you don't want floats. You may occasionally be better off with inline-block, rarely display-table, and if your browser support is ideal, flexbox. In fact, I've started using flexbox on real world projects for layout enhancements like vertical centering (when it's alright for older browsers to have content aligned at the top).

Let's look at some advantages and disadvantages of popular layouting methods for responsive sites.

1.Floats

Advantages

- Most popular way to lay things out; most grid frameworks follow this.
- Everyone's aware of float bugs due to the popularity of floating, and there are well documented ways to overcome them.

Disadvantages

- Need to be cleared. Can be quite painful if you're changing widths at 2-3 media query breakpoints, because the floats will need to be cleared that many times.No vertical centering
- No equal heights
- No source order independence

Use for:

- Large layout blocks that don't need equal heights and vertical centering

2.Inline Block

Advantages

- Vertical centering is most useful for some layouts
- Don't need to be cleared. Useful for complex layouts when breakpoints need to be changed at more than a couple of screen widths.

Disadvantages

- Suffers from inherent whitespace, which will disturb your grid calculations. You can't make a two column grid with the left being 30% and the right being 70%, with display: inline-block, because the whitespace will break the grid and push the second column to the next line. There are ways around this. The easiest is to use lists with the HTML5 doctype, because you don't need to add closing list tags and this removes the whitespace bug.
- No equal heights
- No source order independence

Use for:

- When you require vertical centering, but not equal heights with CSS.
- When you want to avoid clearing your floats at different breakpoints.
- When you can get around the whitespace issue, without driving the rest of your dev team crazy.
- When you can get around the whitespace issue, whilst ensuring that your templating language does not try to add closing li items. That's not cool, bro.
- When you're using list items for layout and don't need a closing li tag, inline-block truly shines.

3.Display Table

Sometimes, you want equal heights + vertical centering without using Javascript. If you can't use flexbox, use display table. But it's a little unpredictable.

Advantages

- Provides you with vertical centering and equal heights

Disadvantages

- Extra wrapper divs (although the spec says you don't need to declare all of them, I've seen weird bugs in iOS when you don't have them).
- If you require spacing between table-cells, you'll have to use border-collapse: separate on the parent element with display: table (since table-cells can't have margins), but this will add padding to the leftmost edge of the first cell and the rightmost edge of the last cell. That could break your grid.
- Content can overflow its cells and it's a bit hard to deal with.
- How the hell do you go from 4 cells to 2 to 1 in a responsive layout? Going from 4 to 1 is easy, because you can replace the table display with a block display in your stylesheet.

Use for:

- Those delicate moments when you need vertical centering + equal heights for a responsive layout with only 1 breakpoint to move from table to block

4.Flexbox

Flexbox is awesome and beyond your wildest, most perverted CSS layout dreams.

Advantages

- Source order independence. Could be of tremendous value for responsive layouts and
- Eliminates the need for any JS code for layout.
- Vertical centering
- Equal heights
- Flex boxes move along interchangeably the X and Y axis, with ease: can change things with a couple of properties.
- Boxes grow and shrink, can be columns or rows, fit to the available space however you wish to declare this.
- Multiple ways to do the same thing with flexbox.

Disadvantages

- Syntax is initially unintuitive.
- A deep understanding of Flexbox takes a while. Once the layout gets more complex, or you add a couple of divs, things can get confusing.
- Doesn't work on very old browsers.

Use for:

- You can already start using it for vertical centering.
- Perfect for source order independent layouts, equal heights.
- App layouts where things need to stretch and squish. Flexbox really shines here.

Modern Browsers

For modern browsers, the best approach is this:

- **Floats**: when an image or media needs to be “floated” in a flow of text (you know... how it was meant to be used)
- **inline-block**: usually for buttons or inputs; things that need some padding and margin, but not to be full block level
- **flex**: content and component-level layouts. I try to avoid it for page layouts, though. That doesn’t mean I won’t... but I find that flex isn’t really ideal in a page layout scenario... by nature of being flexible, that makes it hard to ask for rigid containers.
- **table**: tables. Only tables. Never anything other than tables.

Additional references

[When should you use float vs inline-block vs table vs flex? - Quora]