Flexbox

Flexbox

Up to this point, we have covered quite a few different CSS layout concepts. Inline vs. block level display, different position values, various positioning properties, six different sizing properties, plus countless details and interactions. So by this time, we should know enough to make a two-column page design, right? Sadly, we cannot. The intrepid among us might be able to cobble something together by creatively using inline-block, or maybe absolute or fixed positioning, but ultimately, any design built with just the topics we've covered so far will likely be brittle or unwieldy. Why is that?

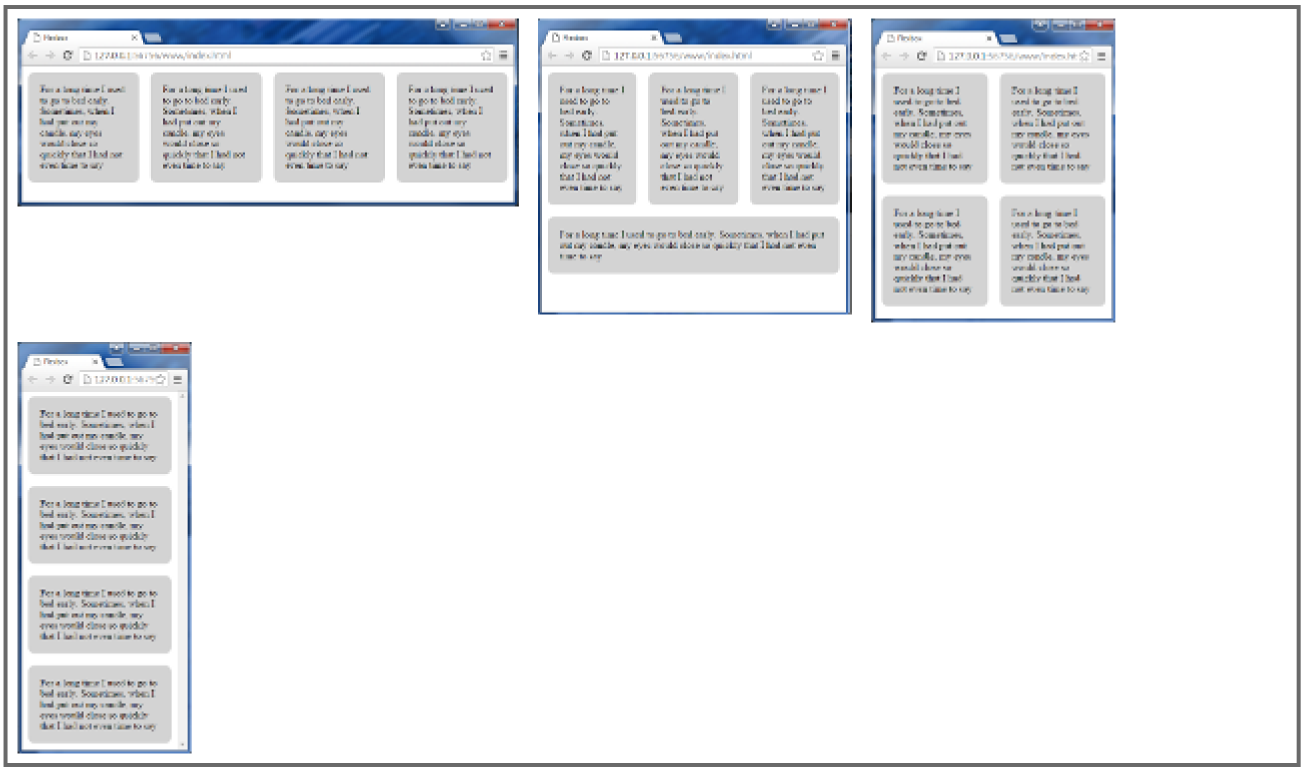
All the layout properties we've looked at have all applied to an *individual* element. But performing layout tasks like columnar layout or anything responsive requires coordinating *multiple* elements. This is where the flexbox comes in. When working with flexbox layout, there are some CSS properties that are applied to a parent element (also known as the flex container) and other CSS properties that are applied to the direct children of that parent (also known as the flex items). The flex container will handle laying out of its children. And, best of all, the flex container will lay out its children smartly, making the best use of the screen size available to it, while still following the general guidelines you laid down for it.   As a general rule, layout with flexbox is pretty easy and the results are great. So let's get started.

The minimum

The minimum scenario for using flexbox is to make use of two CSS rules, and better results are achieved with a third.

1. `display:flex;` on the flex container
2. `flex:1;` on the flex items (the children of the flex container)
3. (better) `flex-flow: row wrap;` on the flex container.

Here is a series of screen captures showing these minimum options applied to a parent `<div>` and four identical paragraphs at various browser sizes, with no other properties applied except some small margin and padding on the paragraph, and a background color and a border radius to help visualize.



flex container

```
div  { display: flex; }
span { display: inline-flex; }
```

To designate an element as a flex container, we simply set the `display` property to be `flex` or `inline-flex`. A flex element will itself be a block level element, and an inline-flex element will itself be an inline element. However, in both cases the element is now a flex container and will be handling the layout of its children.

flex-flow

```
.fc {
  display: flex;  /* this is now a flex container */
  flex-flow: row wrap;
 }
```

Flexbox containers can lay out their children both horizontally, as in a row, and vertically, as in a column, and *both at the same time*. This means that a single flex container not only can help you lay out a three column design, but also handle the header and footer above and below. (Did we mention that flexbox rocks?)

But the flexbox container does need a starting rule to follow. Do you want it to primarily line things up horizontally like a row? Or vertically like a column? And will you be wanting that row or column to wrap? The `flex-flow` property lets you specify both of those things.

Strictly speaking, the `flex-flow` property is actually an abbreviation that replaces two other flexbox container properties: `flex-direction` and `flex-wrap`. But the `row wrap` value is so useful that it will likely be the standard.

```
flex-flow: <flex-direction> <flex-wrap>;
```

The possible values for the flex-direction are: `row`, `row-reverse`, `column`, and `column-reverse`.

The values for the flex-wrap part are: `wrap`, `wrap-reverse`, and `nowrap`.

There are more properties that we can apply to a flex container and we'll look at them in the upcoming sections. But these two will take care of most of what you might want.

flex items

The direct children of a flex container are automatically converted into flex items, with the exception of children that are position-fixed or position-absolute, which are taken out of the "flow" of the flex container. So there is no property needed to designate a child as a flex item, as it happens automatically.

One other automatic behavior to be aware of is that empty flex items are automatically removed from the flex container. Keep that in mind if you were planning on using an empty `<div></div>` construct as a placeholder for a CSS background image.

There is an array of flex item properties that can be applied to the children of a flex container, but there are three that we are not supposed to use in isolation. These three are: `flex-grow`, `flex-shrink`, and `flex-basis`. These three properties interrelate, so rather than using them in isolation the CSS3 specification encourages us to use the `flex` property, which can act as an abbreviation for all the three.

flex property

Earlier, we saw that `display:flex;` can be used to designate a parent element as a flex container. In that case, the symbol "`flex`" is used as a value of the `display` property.

But `flex` is also the name of a property. It is a property that is applied to flex items, the children of a flex container.

```
span { flex: <flex-grow> <flex-shrink> <flex-basis>; }
```

The `flex` property provides a convenient way to abbreviate the three interrelated properties of `flex-grow`, `flex-shrink`, and `flex-basis`. The `flex` property *also* gives a flex item nice defaults for the optional properties. Therefore, `flex:1;` is **better** than `flex-grow:1;` .

flex-grow

```
p { flex: 1; /* rather than use flex-grow, use flex: <flex-grow>; */ }
```

The `flex-grow` property is set simply to a positive number. In isolation that number means nothing. However, when the flex container is laying out its children, for any row (or column) it is processing it may end up with a little extra space. The `flex-grow` property determines how much extra space this flex item should get relative to its siblings. If one sibling has a `flex-grow` value of 2 and another - `flex-grow` value of 1, the former will receive twice as much of the extra space that is divided among the children.

A larger `flex-grow` value does not necessarily mean that an element is larger than its siblings that have smaller `flex-grow` values. The content of each sibling is first accounted for by the flex container when creating any row or column and only after that has been settled is any extra space distributed among the children.

Setting the `flex-grow` to 0 will prevent the flex item from growing. But remember, that will cause the item to shed its "flexible size" super-power.

flex-shrink

```
p { flex: 1 1; /* rather than use flex-shrink directly, use flex:
<flex-grow> <flex-shrink> */ }
```

The `flex-shrink` is the opposite of `flex-grow`. When laying out any row or column, if the flex container needs to take away some space from the children, then those with the highest `flex-shrink` values contribute more of the needed space. Again, the `flex-shrink` value is just a number and it only has meaning when compared to its sibling `flex-shrink` values. And, again, this only occurs in the situation where the flex-container might need some space from its children.

Note: Like `flex-grow`, setting the `flex-shrink` to 0 will prevent the flex item from shrinking. However, this may *not* be as desirable as it first seems. Remember the box model from the previous section? If an item `flex-shrink` value is 0, then its border or padding may end up off-screen or pushed out of the parent, because there is a difference between "fitting" and "fitting nicely", and without the ability to be shrunk an item might fit but not "fit nicely". If you must set `flex-shrink` to 0, then it is recommended that you also set the `box-sizing` to `border-box`.

flex-basis

```
p { flex: 1 1 87px;  /* use flex: <flex-grow> <flex-shrink> <flex-basis> */}
```

The `flex-basis` can be used instead of the sizing properties on a flex item. If the `flex-direction` of the parent flex container is `row` or `row-reverse`, then the `flex-basis` will govern the width of the flex item. If the `flex-direction` is `column` or `column-reverse`, it governs the height.

The `flex-basis` provides the starting dimension (width or height) for the flex-item. It may be grown or shrunk from that. If you do not want it to change at all, then set the `flex-grow` and `flex-shrink` to 0, and the `box-sizing` to `border-box`. However, this is not advisable. Read the `flex-shrink` discussion above.