CSS (http://www.sitepoint.com/css/)

Easy Responsive CSS Grid Layouts: 4 Methods

(http://www.sitepoint.com/author/nsalloum/)

Nick Salloum (http://www.sitepoint.com/author/nsalloum/)

Published June 11, 2014

These days, there's a framework for everything, and it seems that before we can even digest one, another one is released. This holds particularly true when it comes to CSS grid layouts, and there's no shortage of frameworks deemed "the best, most lightweight to date". This overdose of information can baffle us, and leave us wondering if frameworks like HTML9 Responsive Boilerstrap JS (http://html9responsiveboilerstrapjs.com/) are really the way to go.

Let's step back and breathe a bit, and ask ourselves a question: Are we really going to use every one of those 24 variants and the million sub-variants that "That Great Framework" ships with? Often we need only a simple flexible solution with a few variants to work in our project, and with a strong command of the basics we can extend as we see fit. I'm going to present **four different techniques** for developing your own CSS grid, and each of them are easily extendable. Here are the four methods:

1. Responsive grid layout, v1 (using negative margins)

2. Responsive grid layout, v2 (using `box-sizing: border-box`)
3. Responsive grid layout using table display
4. Responsive grid layout using flexbox

I'm going to simplify these methods by using minimal amounts of easy, understandable CSS. A CodePen demo will be provided with each example, so you can fork and/or play with the CSS in the demos. Let's dive in.

Note: I've included embedded demos for each but in order to see the full responsive nature of each technique, it's best to view the CodePen demos at full screen by clicking on the "edit on CodePen" link at the top of each demo.

Common CSS

Before we dig in to each method, let's take a look at some common CSS we'll use. We'll be using the `box-sizing: border-box` declaration universally across our document, and we'll also implement a `.clearfix` class for clearing floats (http://www.sitepoint.com/clearing-floats-overview-different-clearfix-methods/). Here's our base CSS:

```
/* resets */
*,
*:before,
*:after {
  box-sizing: border-box;
}
.clearfix:after {
  content: "";
  display: table;
  clear: both;
}
```

In case any vendor prefixes are necessary, we'll be using CodePen's Autoprefixer (https://github.com/ai/autoprefixer) feature to fill those in for us. Let's now get into the first of our four methods.

Method 1: Using Negative Margins

This method makes use of negative margins to create CSS grid blocks with a fixed margin in between each block. The negative margin varies depending on the position of the grid block, and the margin in between grid blocks remains fixed. Let’s first look at the HTML:

```html
<div class="row-2 clearfix">
  <div class="col-1-2"></div>
  <div class="col-1-2"></div>
</div><!-- /.row -->

<div class="row-4 clearfix">
  <div class="col-1-4"></div>
  <div class="col-1-4"></div>
  <div class="col-1-4"></div>
  <div class="col-1-4"></div>
</div><!-- /.row -->

<div class="row-8 clearfix">
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
</div><!-- /.row -->
```

And our CSS:

```css
/* grid */
[class*="row-"] {
  margin-bottom: 20px;
}
[class*="row-"]:last-child {
  margin-bottom: 0;
}
```

```css
[class*="col-"] {
}

@media all and ( min-width: 768px ) {

  /* all cols margin */
  [class*="col-"] {
    margin-right: 20px;
  }
  [class*="col-"]:last-child {
    margin-right: 0;
  }

  /* make the columns responsive */
  .col-1-2 {
    float: left;
    width: 50%;
  }
  .col-1-4 {
    float: left;
    width: 25%;
  }
  .col-1-8 {
    float: left;
    width: 25%;
  }

  /* 2 span rows */
  .row-2 {
    padding-left: 20px;
  }
  .row-2 [class*="col-"]:first-child {
    margin-left: -20px;
  }

  /* 4 span rows */
  .row-4 {
    padding-left: 60px;
  }
  .row-4 [class*="col-"]:first-child {
    margin-left: -60px;
  }

  /* 8 span rows */
  .row-8 {
    padding-left: 60px;
  }
  .row-8 [class*="col-"]:nth-child(4n+1) {
    margin-left: -60px;
  }
  .row-8 [class*="col-"]:nth-child(5n-1) {
    margin-right: 0;
  }
  .row-8 [class*="col-"]:nth-child(6n-1) {
    clear: both;
  }

}

@media all and ( min-width: 1200px ) {

  /* adjust width */
  .col-1-8 {
    float: left;
    width: 12.5%;
  }

  /* 8 span rows */
  .row-8 {
    padding-left: 140px;
  }
  /* reset these... */
  .row-8 [class*="col-"]:nth-child(4n+1) {
    margin-left: 0;
  }
  .row-8 [class*="col-"]:nth-child(5n-1) {
    margin-right: 20px;
  }
  .row-8 [class*="col-"]:nth-child(6n-1) {
    clear: none;
  }
  /* and add this */
  .row-8 [class*="col-"]:nth-child(1) {
    margin-left: -140px;
  }

}
```

And here’s the CodePen demo:

- HTML
- CSS
-
- Result

Edit on

```
<body>
  <div class="wrapper">
    <h1>Easy Responsive CSS Grid Layout Version 1: Negative Margins</h1>
      <p class="p">Demo by Nick Salloum. <a href="http://www.sitepoint.com/easy-responsive
    <div class="row-2 clearfix">
      <div class="col-1-2">
        <h2>1/2</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
      <div class="col-1-2">
        <h2>1/2</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
    </div>
    <div class="row-4 clearfix">
      <div class="col-1-4">
        <h2>1/4</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
      <div class="col-1-4">
        <h2>1/4</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
      <div class="col-1-4">
```

As you can see, when we hit the responsive breakpoint, a fixed margin value (let’s call it x) multiplied by the number of columns minus 1 (n-1) is added to the left of the row. Each column has a right margin of x, except the last child. And the first child has a negative margin of (n-1)*(x).

Shortcomings and Bugs

Some math is required for this method, and becomes impractical as the number of grid items increases. Also, when we extend it on multiple column steps (example going from 1 per row, to 4, to 8), we have to reset the CSS and a lot of `nth-child` math gets involved.

Another interesting bug occurs when there are multiple floated elements. The combined sum of margins at some point clash, bumping columns at the end onto a new line. This can be seen in the 8-column scenario. If we change the final media query minimum width to less than 1200px, we can see the bug in action. Be mindful of this. There are benefits to this technique though.

Benefits and Real World Uses

The real beauty of this technique though lies in creating fixed/fluid grid combinations. As an example, let's imagine a fluid width primary content area and a secondary fixed-width aside area. Our HTML might look like this:

```html
<div class="container clearfix">
  <div class="primary">
    <h2>Primary</h2>
    Lorem ipsum dolor...
  </div>
  <div class="secondary">
    <h2>Secondary</h2>
    Lorem ipsum dolor...
  </div>
</div><!-- /.container -->
```

And our CSS like this:

```css
/* layout */
.primary {
  margin-bottom: 20px;
}

@media all and ( min-width: 600px ) {

  .container {
    padding-right: 300px;
  }
  .primary {
    float: left;
    padding-right: 60px;
    width: 100%;
  }
  .secondary {
    float: right;
    margin-right: -300px;
    width: 300px;
  }

}
```

Here’s a CodePen demo to see it in action:

- HTML
- CSS
-
- Result

Edit on

```
<body>
  <div class="wrapper">
    <h1>Easy Responsive CSS Grid Layout, Version 1b: Fluid Content Fixed Sidebar</h1>
    <p>Demo by Nick Salloum. <a href="http://www.sitepoint.com/easy-responsive-css-grid-la

    <div class="container clearfix">
      <div class="primary">
        <h2>Primary</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
      <div class="secondary">
        <h2>Secondary</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
    </div><!-- /.container -->
  </div><!-- /.wrapper -->
```

Method 2: Using `box-sizing: border-box`

This method leverages the full power of the `box-sizing: border-box` declaration. Because this feature allows us to pad an element without adding to the element's overall width, we can still achieve a flexible grid with fixed "margins". This time, instead of using the `margin` property, the inner padding values will act as the margins for our grid items. Let's look at some HTML:

```
<div class="row clearfix">
  <div class="col-1-2"></div>
  <div class="col-1-2"></div>
</div><!-- /.row -->

<div class="row clearfix">
  <div class="col-1-4"></div>
  <div class="col-1-4"></div>
  <div class="col-1-4"></div>
  <div class="col-1-4"></div>
</div><!-- /.row -->

<div class="row clearfix">
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
</div><!-- /.row -->
```

We won't require any crazy math here either, so our CSS becomes really simple. Here it is, for up to eight columns:

```
/* grid */
.row {
  margin: 0 -10px;
  margin-bottom: 20px;
}
.row:last-child {
  margin-bottom: 0;
}
[class*="col-"] {
  padding: 10px;
}

@media all and ( min-width: 600px ) {

  .col-2-3 {
    float: left;
    width: 66.66%;
  }
  .col-1-2 {
    float: left;
    width: 50%;
  }
  .col-1-3 {
    float: left;
    width: 33.33%;
  }
  .col-1-4 {
    float: left;
    width: 25%;
  }
  .col-1-8 {
    float: left;
    width: 12.5%;
  }

}
```

The negative left and right margins on each row are to compensate for the padding on the columns. At a specified media query minimum width, our grid items take on their own width and become floated inside their containers. You may choose to alter this width as you please, and specifying a different media query breakpoint for different groups of grid items is a breeze too.

Here’s a demo:

- HTML
- CSS
-
- Result

Edit on

```
<body>
  <div class="wrapper">
    <h1>Easy Responsive CSS Grid Layout, Version 2: Using box-sizing</h1>
    <p>Demo by Nick Salloum. <a href="http://www.sitepoint.com/easy-responsive-css-grid-la
    <div class="row clearfix">
      <div class="col-1-2">
        <h2>1/2</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
      <div class="col-1-2">
        <h2>1/2</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
    </div>
    <div class="row clearfix">
      <div class="col-2-3">
        <h2>2/3</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
```

Extending This Method

Let’s say you wanted the `.col-8` items to jump to 4 per row, then 8 per row. With a little thought, it’s very easy. Assuming the same markup as above, our CSS would look like this:

```
@media all and ( min-width: 600px ) {

  .col-1-8 {
    float: left;
    width: 25%;
  }
  .col-1-8:nth-child(4n+1) {
    clear: both;
  }

}

```

```
@media all and ( min-width: 960px ) {

  .col-1-8 {
    width: 12.5%;
  }
  .col-1-8:nth-child(4n+1) {
    clear: none;
  }

}
```

Here’s the Demo:

- HTML
- CSS
-
- Result

Edit on

```
<body>
  <div class="wrapper">
    <h1>Easy Responsive CSS Grid Layouts, Version 2b: 1 to 4 to 8</h1>
    <p>Demo by Nick Salloum. <a href="http://www.sitepoint.com/easy-responsive-css-grid-l
    <div class="row clearfix">
      <div class="col-1-8">
        <h2>1/8</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Expedita itaque ipsa si
      </div>
      <div class="col-1-8">
        <h2>1/8</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Expedita itaque ipsa si
      </div>
      <div class="col-1-8">
        <h2>1/8</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Expedita itaque ipsa si
      </div>
      <div class="col-1-8">
```

Method 3: Using Table Display

This method implements the age-old `table` functionality, but without breaking our semantics or structure. In this method, the visible elements are block-level by default. But at a certain breakpoint, grid rows become tables and columns become table cells. Let's look at the markup — it's similar to method 2, but no clearfix is required:

```html
<div class="row">
  <div class="col-1-2"></div>
  <div class="col-1-2"></div>
</div><!-- /.row -->

<div class="row">
  <div class="col-1-4"></div>
  <div class="col-1-4"></div>
  <div class="col-1-4"></div>
  <div class="col-1-4"></div>
</div><!-- /.row -->

<div class="row">
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
</div><!-- /.row -->
```

And here's the CSS:

```css
/* grid */
.row {
  margin: 0 -10px;
  margin-bottom: 10px;
}
.row:last-child {
  margin-bottom: 0;
}
```

```css
[class*="col-"] {
  padding: 10px;
}

@media all and ( min-width: 600px ) {

  .row {
    display: table;
    table-layout: fixed;
    width: 100%;
  }
  [class*="col-"] {
    display: table-cell;
  }

  /* set col widths */
  .col-2-3 {
    width: 66.66%;
  }
  .col-1-2 {
    width: 50%;
  }
  .col-1-3 {
    width: 33.33%;
  }
  .col-1-4 {
    width: 25%;
  }
  .col-1-8 {
    width: 12.5%;
  }

}
```

And our demo:

- HTML
- CSS
-
- Result

Edit on

```
<body>
  <div class="wrapper">
    <h1>Easy Responsive CSS Grid Layout, Version 3: Table Display</h1>
    <p>Demo by Nick Salloum. <a href="http://www.sitepoint.com/easy-responsive-css-grid-l
    <div class="row">
      <div class="col-1-2">
        <h2>1/2</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
      <div class="col-1-2">
        <h2>1/2</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
    </div><!-- /.row -->

    <div class="row">
      <div class="col-2-3">
        <h2>2/3</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
      <div class="col-1-3">
        <h2>1/3</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
    </div><!-- /.row -->

    <div class="row">
      <div class="col-1-4">
        <h2>1/4</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
      <div class="col-1-4">
```

This method may seem convoluted, but there are benefits. For starters, we’re not breaking semantics by using traditional table layouts, and we don’t have to clear floats. Equal height columns are also a breeze. Flexible and fluid-width combinations? No problem. Table display brings issues of its own though, and is my least favourite out of these four methods — even though it’s sometimes a good option.

Method 4: Flexbox

The last method that I'll present uses the flexbox module. According to MDN (https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Flexible_boxes):

> **The CSS3 Flexible Box, or flexbox, is a layout mode providing for the arrangement of elements on a page such that the elements behave predictably when the page layout must accommodate different screen sizes and different display devices.**

Flexbox presents us with a host of different features that give us a powerful arsenal of layout options. Making a flexbox module responsive is an absolute breeze. As before, our markup looks like this:

```html
<div class="row">
  <div class="col-1-2"></div>
  <div class="col-1-2"></div>
</div><!-- /.row -->

<div class="row">
  <div class="col-2-3"></div>
  <div class="col-1-3"></div>
</div><!-- /.row -->

<div class="row">
  <div class="col-1-4"></div>
  <div class="col-1-4"></div>
  <div class="col-1-4"></div>
  <div class="col-1-4"></div>
</div><!-- /.row -->

<div class="row">
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
  <div class="col-1-8"></div>
</div><!-- /.row -->
```

Here’s a look now at our CSS:

```css
/* grid */
.row {
  display: flex;
  flex-flow: row wrap;
  margin: 0 -10px;
  margin-bottom: 10px;
}
.row:last-child {
  margin-bottom: 0;
}
[class*="col-"] {
  padding: 10px;
  width: 100%;
}

@media all and ( min-width: 600px ) {

  /* set col widths */
  .col-2-3 {
    width: 66.66%;
  }
  .col-1-2 {
    width: 50%;
  }
  .col-1-3 {
    width: 33.33%;
  }
  .col-1-4 {
    width: 25%;
  }
  .col-1-8 {
    width: 12.5%;
  }

}
```

And the CodePen demo:

- HTML
- CSS
-
- Result

Edit on

```
<body>
  <div class="wrapper">
    <h1>Easy Responsive CSS Grid Layout, Version 4: Flexbox</h1>
    <p>Demo by Nick Salloum. <a href="http://www.sitepoint.com/easy-responsive-css-grid-la
    <div class="row">
      <div class="col-1-2">
        <h2>1/2</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
      <div class="col-1-2">
        <h2>1/2</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
    </div><!-- /.row -->

    <div class="row">
      <div class="col-2-3">
        <h2>2/3</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
      <div class="col-1-3">
        <h2>1/3</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
    </div><!-- /.row -->

    <div class="row">
      <div class="col-1-4">
        <h2>1/4</h2>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid cum quasi nulla
      </div>
```

This time, we need to set the `display` property to `flex` on the rows, and also specify the `flex-flow` property. Full definitions and descriptions of these properties are available in MDN's flexbox documentation (https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Flexible_boxes). At our responsive breakpoint, we switch our column widths, and flexbox handles the rest. Neat!

Wrap Up

We've looked at 4 unique ways to create a responsive CSS grid system, each with its own benefits. There's no "best" way to do anything, and I often find myself in situations where one is more efficient than the other, or I need to combine both. Methods 1 and 2 are my preferred ones, and I use them together a lot in the same projects (defining main layouts with method 1, and responsive grids with method 2).

Method 3 has its benefits too as mentioned above, but I personally tend to resort to table-based layouts only when absolutely necessary. Method 4 is simply beautiful, and I long for the day when I can freely implement it across all projects. Flexbox is on the move, but is supported only by IE10 and up. There are polyfills available, but I prefer to shy away from polyfills in general. Know your market though; there are scenarios today where flexbox does indeed provide the perfect solution.

Each of these methods is easily scalable and expandable. With the thought processes presented above, you should have no problem at all setting up a grid tailor-made to your project with minimal amounts of CSS. The CSS Grid Module (http://dev.w3.org/csswg/css-grid/) is also on the way, but it'll be a while before we consider implementing it. I hope you enjoyed reading this, and are now less intimidated by CSS grids!

(http://www.sitepoint.com/author/nsalloum/)

Nick Salloum (http://www.sitepoint.com/author/nsalloum/)

I'm a web designer & developer from Trinidad & Tobago, with a degree in Mechanical Engineering. I love the logical side of the web, and I'm an artist/painter at heart. I endorse progressive web techniques, and try to learn something every day. I try to impart my knowledge as much as possible on my personal blog, callmenick.com (http://www.callmenick.com/). I love food, I surf every weekend, and I have an amazing creative partnership with fellow mischief maker Elena (http://www.elenamolchanova.com/). Together, we run SAYSM (http://saysm.com/).

(https://twitter.com/nicksalloum_)
(https://www.facebook.com/callmenick1)