你需要知道的所有 Flexbox 排列方式

<https://blog.csdn.net/weixin_34384915/article/details/87943067>

https://www.cnblogs.com/cydiacen/p/6322573.html

 FlexBox布局最适用于应用程序的组件和小规模布局，而网格布局更适用于大规模的布局。

**基本要素**

因为FlexBox是一整个模块并不是一个单独的属性，它涉及到很多东西包括它的所有设置属性。一些属性是需要被设置在容器（父级元素，称为『弹性容器』），而一些其他的属性需要被设置在子元素（称为『弹性项』）中。

如果常规布局是基于块布局与内联布局的方向流的，那么弹性布局就是基于“flex-flow流”。请看一下来自W3C规范的这张图，它解释了弹性布局的主要思想。

基本上，弹性项（flex item）会沿着主轴方向（main-start到main-end）或侧轴方向(cross-start到cross-end)排列。

主轴（main axis） - 弹性项主要是沿着弹性容器的主轴进行排列的。要注意一点，他不一定是水平的；这主要还是看flex-direction属性（见下文）。

main-start | main-end - 弹性项将由main-start到main-end方向放置在容器内。

main size - 弹性项在主轴方向的宽度或高度就是主轴的尺寸。弹性项主要的大小属性可以是宽度，也可以是高度属性，由哪一个对着主轴方向决定。

cross axis - 与主轴垂直相交的轴线就是侧轴。它的方向由主轴决定。

cross-start | cross-end - 弹性行是由弹性项填充起来的，它的配置是从容器的cross-start开始，往cross-end结束。

cross size - 弹性项在侧轴方向的宽度或高度就是cross size。cross size根据侧轴的方向来取决是宽度还是高度。

**父级（弹性容器）属性**

**display**

这个属性是定义在弹性容器上的；根据其值决定是内联还是块布局。这时它的直属下级将会变成flex文档流。

.container {

display: flex; /\* or inline-flex \*/

}

需要注意CSS的columns在flex容器里没有效果。

**flex-direction**

这个是建立在主轴上的，从而定义了弹性项放置在容器的方向。FlexBox是单方向的布局概念。可以将弹性项视为主要布置在水平行或垂直列中。

.container {

flex-direction: row | row-reverse | column | column-reverse;

}

row（默认）：在ltr排版方式下从左向右排列；在rtl排版方式下从右向左排列。

row-reverse：与row排列方向相反，在ltr排版方式下从右向左排列；在rtl排版方式下从左向右排列。

column：类似于row但是是顶部到底部

column-reverse：类似于row-reverse但是是底部到顶部

**flex-wrap**

弹性项默认会全部集中在一行。你可以使用这个属性来改变这种情况，让他们根据你的需要进行自动换行。文档方向在这里也起作用，决定了新的一行被堆叠的方向。

.container{

flex-wrap: nowrap | wrap | wrap-reverse;

}

nowrap（默认）：单行显示。在ltr排版下，项目自左向右；在rtl下，自右向左

wrap：多行显示。在ltr排版下，项目自左向右；在rtl下，自右向左

wrap-reverse：多行显示。与wrap相反

**flex-flow（定义在弹性容器中）**

这个是flex-direction和flex-wrap属性的缩写版，它同时定义了弹性容器的主轴和侧轴。默认是row nowrap。

flex-flow: <‘flex-direction’> || <‘flex-wrap’>

**justify-content**

这属性是用来定义主轴上的对齐方式的。当所有的弹性项在一行并且无法弹性伸展，或者可伸展但是达到了最大尺寸，它能帮助分配剩下的多余空间。并且当他们行内溢出时，这个属性也可以对项目对齐施加一些控制。

.container {

justify-content: flex-start | flex-end | center | space-between | space-around;

}

flex-start（默认）：子项会从一行的起始处开始放置

flex-end：子项会从一行的结尾处开始放置

center：子项会集中在一行的中央

space-between：子项会被均匀的分布在行内；首项放置在一行的开始，尾项放置在一行的结束

space-around：子项会均匀的按照同等距离分布在一行。需要注意的是，在视觉上会觉得并不等距，因为所有子项在两侧都需要加上同等的空间。首项会与容器开始边缘有一个单位空间的距离，但是与下一项会有两个单位空间的距离，因为下一项也有它自己的适配空间。

**align-items**

这用来定义弹性项目在弹性容器的当前侧轴上的默认行为。可以认为是侧轴版的justity-content。

.container {

align-items: flex-start | flex-end | center | baseline | stretch;

}

flex-start：弹性项在侧轴起点边的外边距紧靠住该行在侧轴起始的边。

flex-end：弹性项在侧轴起点边的外边距紧靠改行在侧轴结尾的边。

center：弹性项会被放置在侧轴的中央。

baseline：弹性项会根据他们的基线对齐。

stretch（默认）：在侧轴方向上拉伸弹性项以致填充满弹性容器。（任遵从min-width/max-width）

**align-content**

这个属性会根据在侧轴上的额外空间来排列容器的行，类似于justify-content在主轴在对齐单个弹性项的方式。

注意，这个属性对于只有单行的弹性项来说是没有效果的。

.container {

align-content: flex-start | flex-end | center | space-between | space-around | stretch;

}

flex-start：行会紧靠容器的起始位

flex-end：行紧靠容器的结束位

center：行紧靠容器的中间位

space-between：每行会均匀分布；首行在容器起始处而最后行在容器结束处

space-around：每行根据相同的距离均匀的分布

stretch（默认）：每行将会伸展以占用剩余的空间。

**子项（弹性项）的属性**

**order**

一般来说，弹性项会按照文档流的顺序进行布局。然而，order属性可以控制他们出现在弹性容器中的顺序。

.item {

order: <integer>;

}

**flex-grow**

这个属性给予弹性项在需要的时候可以伸展的能力。它接收一个不带单位的值作为比例。它规定了在容器内的弹性项可以占用多少的可用空间。

如果所有的子项都设置了flex-grow为1，那么容器内的剩余空间会被均匀的分配给所有自项。如果其中一项的值为2，那么这项的占用空间会是其他项的两倍。

.item {

flex-grow: <number>; /\* default 0 \*/

}

设置负数是无效的。

**flex-shrink**

这定义了弹性项在需要的时候具有伸展的能力。

.item {

flex-shrink: <number>; /\* default 1 \*/

}

负数是无效的。

**flex-basis**

这定义了，当一个元素在被分配到剩余空间之前的默认大小。它可以是一个长度（如：20%，5rem等）或者一个关键字。auto关键字的意思就是『按照我的宽度和高度属性调整尺寸』（他会暂时根据main-size来布局大小直到被弃用）。如果使用关键字content，意思就是『基于内容调整大小』——不过这个关键字并一定能很好的工作，因此很难去测试或者知晓它的兄弟们max-content，min-content和fit-content做了什么。

.item {

flex-basis: <length> | auto; /\* default auto \*/

}

如果设置为0，额外空间内容不会被分解开来。如果设置成auto，额外空间会基于它的flex-grow的值进行分布。

**flex**

这是flex-grow，flex-shrink和flex-basis组合缩写版。第二个和第三个参数（flex-shrink和flex-basis）是可选的。默认是0 1 auto。

推荐你使用的这种缩写属性，这比设置单独属性更好。可以智能的通过缩写形式设置值。

**align-self**

用来在单独的伸缩项目上覆写默认的对齐方式。

请看下align-items的解释，帮助你了解可用值。

.item {

align-self: auto | flex-start | flex-end | center | baseline | stretch;

}

注意，float，clear和vertical-align在弹性项中会失效。

## 例子

让我们从一个非常非常简单的例子开始，解决一个非常日常的问题：完美居中。如果你使用flexBox布局，这将会变得非常简单。

.parent {

display: flex;

height: 300px; /\* Or whatever \*/

}

.child {

width: 100px; /\* Or whatever \*/

height: 100px; /\* Or whatever \*/

margin: auto; /\* Magic! \*/

}

这个依赖于设置margin值为auto值，会自动获取弹性容器的额外空间。因此设置垂直方向margin为auto可以让弹性项完美的居中在两个轴。

现在让我们使用下一些其他属性。

考虑使用一个包含六个项的列表，并且为了视觉审美给他设置了一个固定大小尺寸，但他们也有可能可以自动获取尺寸大小。我们想要他们可以均匀的，并且完美分布在水平轴上，并且当我改变浏览器的大小，他们还是可以很好的展示（不需要引用媒体查询！）。

.flex-container {

/\* 我们先创建一个弹性布局环境\*/

display: flex;

/\* 然后如果我们允许子项换行可以定义flow-direction

\* 记住这相等于:

\* flex-direction: row;

\* flex-wrap: wrap;

\*/

flex-flow: row wrap;

/\* 然后我们再定义怎么样分布剩余的空间 \*/

justify-content: space-around;

}

完工。其他的一切不过是美化样式。下面提供一些html，css代码，可以在codePen调试并且改变下浏览器的大小看看会发生什么。  
[codePen](http://codepen.io/team/css-tricks/pen/YqaKYR" \t "https://www.cnblogs.com/cydiacen/p/_blank)

<ul class="flex-container">

<li class="flex-item">1</li>

<li class="flex-item">2</li>

<li class="flex-item">3</li>

<li class="flex-item">4</li>

<li class="flex-item">5</li>

<li class="flex-item">6</li></ul>

SASS：

@import "compass/css3";

.flex-container {

padding: 0;

margin: 0;

list-style: none;

display: -webkit-box;

display: -moz-box;

display: -ms-flexbox;

display: -webkit-flex;

display: flex;

-webkit-flex-flow: row wrap;

justify-content: space-around;

}

.flex-item {

background: tomato;

padding: 5px;

width: 200px;

height: 150px;

margin-top: 10px;

line-height: 150px;

color: white;

font-weight: bold;

font-size: 3em;

text-align: center;

}

让我们再尝试下其他东西。想象下，我们的网站所有页面头部都有个右对齐布局的导航，但是我们想让他在中等大小的屏幕中居中显示并且在小屏幕设备中单行显示。非常简单。

/\* Large \*/.navigation {

display: flex;

flex-flow: row wrap;

/\* This aligns items to the end line on main-axis \*/

justify-content: flex-end;

}

/\* Medium screens \*/@media all and (max-width: 800px) {

.navigation {

/\* When on medium sized screens, we center it by evenly distributing empty space around items \*/

justify-content: space-around;

}

}

/\* Small screens \*/@media all and (max-width: 500px) {

.navigation {

/\* On small screens, we are no longer using row direction but column \*/

flex-direction: column;

}

}

[codePen调试链接](http://codepen.io/team/css-tricks/pen/YqaKYR" \t "https://www.cnblogs.com/cydiacen/p/_blank)

让我们再来试试一些更加灵活性的弹性项！关于移动先行，3列布局与页眉页脚全屏。和独立的文档顺序。

.wrapper {

display: flex;

flex-flow: row wrap;

}

/\* We tell all items to be 100% width \*/.header, .main, .nav, .aside, .footer {

flex: 1 100%;

}

/\* We rely on source order for mobile-first approach

\* in this case:

\* 1. header

\* 2. nav

\* 3. main

\* 4. aside

\* 5. footer

\*/

/\* Medium screens \*/@media all and (min-width: 600px) {

/\* We tell both sidebars to share a row \*/

.aside { flex: 1 auto; }

}

/\* Large screens \*/@media all and (min-width: 800px) {

/\* We invert order of first sidebar and main

\* And tell the main element to take twice as much width as the other two sidebars

\*/

.main { flex: 2 0px; }

.aside-1 { order: 1; }

.main { order: 2; }

.aside-2 { order: 3; }

.footer { order: 4; }

}