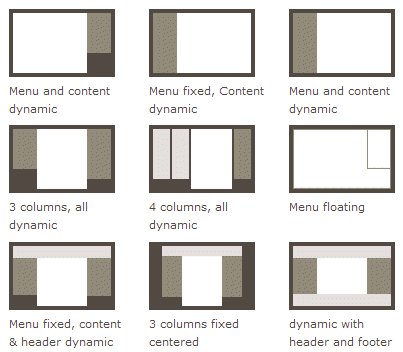
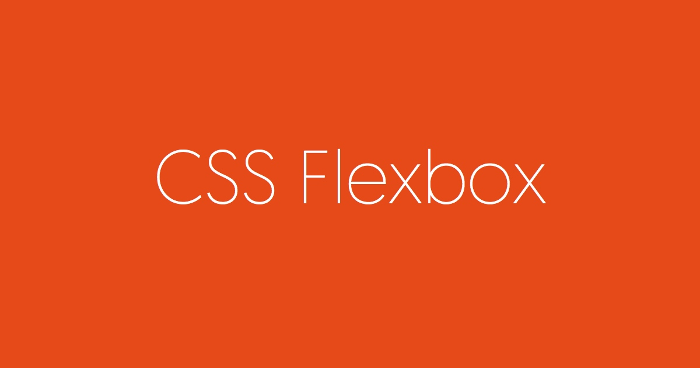
网页布局（layout）是CSS的一个重点应用。



布局的传统解决方案，基于[盒状模型](https://developer.mozilla.org/en-US/docs/Web/CSS/box_model)，依赖 [display](https://developer.mozilla.org/en-US/docs/Web/CSS/display)属性 + [position](https://developer.mozilla.org/en-US/docs/Web/CSS/position)属性 + [float](https://developer.mozilla.org/en-US/docs/Web/CSS/float)属性。它对于那些特殊布局非常不方便，比如，[垂直居中](https://css-tricks.com/centering-css-complete-guide/)就不容易实现。



2009年，W3C提出了一种新的方案----Flex布局，可以简便、完整、响应式地实现各种页面布局。目前，它已经得到了所有浏览器的支持，这意味着，现在就能很安全地使用这项功能。



Flex布局将成为未来布局的首选方案。本文介绍它的语法，[下一篇文章](http://www.ruanyifeng.com/blog/2015/07/flex-examples.html)给出常见布局的Flex写法。

以下内容主要参考了下面两篇文章：[A Complete Guide to Flexbox](https://css-tricks.com/snippets/css/a-guide-to-flexbox/) 和 [A Visual Guide to CSS3 Flexbox Properties](https://scotch.io/tutorials/a-visual-guide-to-css3-flexbox-properties)。

一、Flex布局是什么？

Flex是Flexible Box的缩写，意为"弹性布局"，用来为盒状模型提供最大的灵活性。

任何一个容器都可以指定为Flex布局。

.box{

display: flex;

}

行内元素也可以使用Flex布局。

.box{

display: inline-flex;

}

Webkit内核的浏览器，必须加上-webkit前缀。

.box{

display: -webkit-flex; /\* Safari \*/

display: flex;

}

注意，设为Flex布局以后，子元素的float、clear和vertical-align属性将失效。

二、基本概念

采用Flex布局的元素，称为Flex容器（flex container），简称"容器"。它的所有子元素自动成为容器成员，称为Flex项目（flex item），简称"项目"。



容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。主轴的开始位置（与边框的交叉点）叫做main start，结束位置叫做main end；交叉轴的开始位置叫做cross start，结束位置叫做cross end。

项目默认沿主轴排列。单个项目占据的主轴空间叫做main size，占据的交叉轴空间叫做cross size。

三、容器的属性

以下6个属性设置在容器上。

* flex-direction
* flex-wrap
* flex-flow
* justify-content
* align-items
* align-content

3.1 flex-direction属性

flex-direction属性决定主轴的方向（即项目的排列方向）。

.box {

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

}

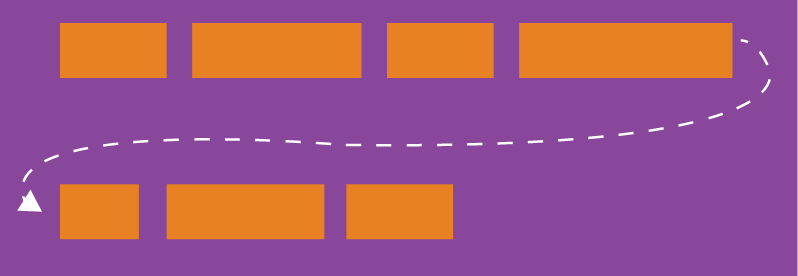


它可能有4个值。

* row（默认值）：主轴为水平方向，起点在左端。
* row-reverse：主轴为水平方向，起点在右端。
* column：主轴为垂直方向，起点在上沿。
* column-reverse：主轴为垂直方向，起点在下沿。

3.2 flex-wrap属性

默认情况下，项目都排在一条线（又称"轴线"）上。flex-wrap属性定义，如果一条轴线排不下，如何换行。



.box{

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

}

它可能取三个值。

（1）nowrap（默认）：不换行。



（2）wrap：换行，第一行在上方。



（3）wrap-reverse：换行，第一行在下方。



3.3 flex-flow

flex-flow属性是flex-direction属性和flex-wrap属性的简写形式，默认值为row nowrap。

.box {

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

}

3.4 justify-content属性

justify-content属性定义了项目在主轴上的对齐方式。

.box {

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

}



它可能取5个值，具体对齐方式与轴的方向有关。下面假设主轴为从左到右。

* flex-start（默认值）：左对齐
* flex-end：右对齐
* center： 居中
* space-between：两端对齐，项目之间的间隔都相等。
* space-around：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。

3.5 align-items属性

align-items属性定义项目在交叉轴上如何对齐。

.box {

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

}



它可能取5个值。具体的对齐方式与交叉轴的方向有关，下面假设交叉轴从上到下。

* flex-start：交叉轴的起点对齐。
* flex-end：交叉轴的终点对齐。
* center：交叉轴的中点对齐。
* baseline: 项目的第一行文字的基线对齐。
* stretch（默认值）：如果项目未设置高度或设为auto，将占满整个容器的高度。

3.6 align-content属性

align-content属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

.box {

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

}



该属性可能取6个值。

* flex-start：与交叉轴的起点对齐。
* flex-end：与交叉轴的终点对齐。
* center：与交叉轴的中点对齐。
* space-between：与交叉轴两端对齐，轴线之间的间隔平均分布。
* space-around：每根轴线两侧的间隔都相等。所以，轴线之间的间隔比轴线与边框的间隔大一倍。
* stretch（默认值）：轴线占满整个交叉轴。

四、项目的属性

以下6个属性设置在项目上。

* order
* flex-grow
* flex-shrink
* flex-basis
* flex
* align-self

4.1 order属性

order属性定义项目的排列顺序。数值越小，排列越靠前，默认为0。

.item {

order: <integer>;

}



4.2 flex-grow属性

flex-grow属性定义项目的放大比例，默认为0，即如果存在剩余空间，也不放大。

.item {

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

}



如果所有项目的flex-grow属性都为1，则它们将等分剩余空间（如果有的话）。如果一个项目的flex-grow属性为2，其他项目都为1，则前者占据的剩余空间将比其他项多一倍。

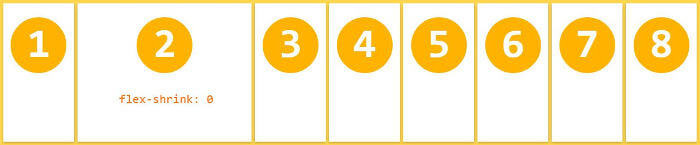
4.3 flex-shrink属性

flex-shrink属性定义了项目的缩小比例，默认为1，即如果空间不足，该项目将缩小。

.item {

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

}



如果所有项目的flex-shrink属性都为1，当空间不足时，都将等比例缩小。如果一个项目的flex-shrink属性为0，其他项目都为1，则空间不足时，前者不缩小。

负值对该属性无效。

4.4 flex-basis属性

flex-basis属性定义了在分配多余空间之前，项目占据的主轴空间（main size）。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为auto，即项目的本来大小。

.item {

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

}

它可以设为跟width或height属性一样的值（比如350px），则项目将占据固定空间。

假设flex盒子父级宽度固定为800px；

Flex-grow、Flex-shrink、Flex-basis 是Flex属性的分写模式；

如  
.box {  
    flex: 4 3 100px;  
}  
等于  
.box {  
    flex-grow: 4;  
    flex-shrink: 3;  
    flex-basis: 100px;  
}

看以下例子

<div class="flex-parent">  
    <div class="flex-son"></div>  
    <div class="flex-son"></div>  
    <div class="flex-son"></div>  
</div>  
<style type="text/css">  
    .flex-parent {  
        width: 800px;  
    }  
</style>

第一种情况  
flex-parent 是父级，而且他的宽度是固定为800px，不会改变；  
开始设置子级flex属性；  
<style type="text/css">  
    .flex-son:nth-child(1){  
        flex: 3 1 200px;  
    }  
    .flex-son:nth-child(2){  
        flex: 2 2 300px;  
    }  
    .flex-son:nth-child(3){  
        flex: 1 3 500px;  
    }  
</style>  
flex-basis总和加起来为1000px； 那么 1000px > 800px (父级的宽度)；子元素势必要压缩；溢出了200px；

son1 = (flex-shrink) \* flex-basis；  
son2 = (flex-shrink) \* flex-basis；  
…..  
sonN = (flex-shrink) \* flex-basis；

如果flex-basis的总和加起来大于父级宽度，子级被压缩，最后的选择是flex-shrink来进行压缩计算  
加权值 = son1 + son2 + …. + sonN；

那么压缩后的计算公式就是

压缩后宽度 w = (子元素flex-basis值 \* (flex-shrink)/加权值) \* 溢出值

所以最后的加权值是  
1\*200 + 2\*300 + 3\*500 = 2300px

son1的扩展量：(200 \* 1/ 2300) \* 200，即约等于17px；  
son2的扩展量：(300 \* 2/ 2300) \* 200，即约等于52px；  
son3的扩展量：(500 \* 3/ 2300) \* 200，即约等于130px；

最后son1、son2、son3，的实际宽度为：  
200 – 16  = 184px；  
300 – 52  = 248px；  
500 – 230 = 370px；

第二种情况

上面的例子已经说明，继续看第二个例子，同样上面的例子，我们改下父级宽度为1200px;  
flex-basis的总和为 1000px，小于父级宽度，将有200px的剩余宽度；  
既然有剩余，我们就不要加权计算，剩余的宽度将根据flex-grow，值得总和进行百分比，那么200px就会根据份数比来分配剩余的空间；

剩余后宽度 w = (子元素flex-grow值 /所有子元素flex-grow的总和) \* 剩余值

总分数为 total = 1 + 2 + 3；

son1的扩展量：(3/total) \* 200，即约等于100px；  
son2的扩展量：(2/total) \* 200，即约等于67px；  
son3的扩展量：(1/total) \* 200，即约等于33px；

最后son1、son2、son3，的实际宽度为：  
200 + 100 = 300px；  
300 + 67 = 367px；  
500 + 33 = 533px；

总结

所以以上两种情况下，第二种flex-basis和flex-shrink是不列入计算公式的；第一种flex-grow是不列入计算公式的

ok，上面的两种情况总结完毕，但是很多时候我们的父级是不固定的，那么怎么办，其实很简单了，对照上面的公式，前提是已经设置了flex-basis值得元素，如果宽度的随机值小于flex-basis的时候就按第一种计算，反之第二种；明白了吧。

但是在实际中，我们有些子元素不想进行比例分配，永远是固定的，那么flex就必须设置为none；  
否则设置的宽度（width）将无效；

flex: 1,    则其计算值为 flex: 1 1 0%；  
flex: auto, 则其计算值为 flex: 1 1 auto；  
flex: none, 则其计算值为 flex: 0 0 auto；

根据上面的公式  
flex：1的时候第一种方式其实是无效的，因为加权值是0，所以只能是第二种方式计算；  
flex：none的时候，两种都失效，自己元素不参与父级剩余还是溢出的分配

4.5 flex属性

flex属性是flex-grow, flex-shrink 和 flex-basis的简写，默认值为0 1 auto。后两个属性可选。

.item {

flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]

}

该属性有两个快捷值：auto (1 1 auto) 和 none (0 0 auto)。

建议优先使用这个属性，而不是单独写三个分离的属性，因为浏览器会推算相关值。

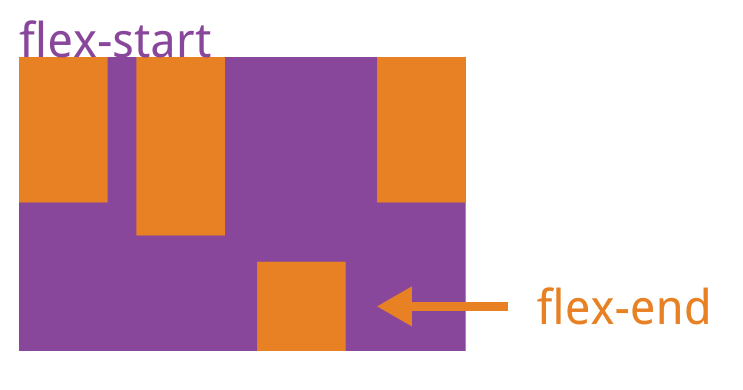
4.6 align-self属性

align-self属性允许单个项目有与其他项目不一样的对齐方式，可覆盖align-items属性。默认值为auto，表示继承父元素的align-items属性，如果没有父元素，则等同于stretch。

.item {

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

}



该属性可能取6个值，除了auto，其他都与align-items属性完全一致。