--

[布局新时代 1](#_Toc16303)

[关于多列布局[--column] 1](#_Toc23798)

[基础概念 1](#_Toc25368)

[说明： 1](#_Toc25856)

[属性列表 1](#_Toc22476)

[列宽 2](#_Toc13708)

[列个数 列宽度 2](#_Toc23250)

[列间距 列边 2](#_Toc19616)

[列之间的缝隙间隔宽度 2](#_Toc15048)

[列间距column-gap 2](#_Toc11338)

[列边框column-rule 2](#_Toc20420)

[demo 2](#_Toc14723)

[demo-columns 2](#_Toc2864)

[demo 3](#_Toc1814)

[关于伸缩盒Flexbox布局 3](#_Toc10125)

[flex 3](#_Toc5341)

[基础概念 3](#_Toc32576)

[flexbox术语 3](#_Toc3540)

[伸缩容器的属性 5](#_Toc2771)

[主轴对齐justify-content 5](#_Toc9167)

[侧轴对齐align-items(align-self) 5](#_Toc19038)

[堆栈伸缩行 align-content 5](#_Toc24025)

[flex-flow 伸缩方向与换行 6](#_Toc15075)

[伸缩项目item的属性 6](#_Toc5761)

[flex 伸缩性 6](#_Toc16359)

[order 显示顺序 8](#_Toc4515)

[align-self 侧轴对齐 8](#_Toc27287)

[visibility 叠加项目 8](#_Toc3441)

[其他 8](#_Toc4910)

[margin 外边距的应用 8](#_Toc4295)

[grid布局 11](#_Toc7805)

布局新时代

|  |  |
| --- | --- |
|  | CSS布局的未来  <http://www.w3cplus.com/css3/future-css-layouts.html>  CSS多列布局Multi-column、伸缩布局Flexbox、网格布局Grid详解  <http://www.5icool.org/a/201308/a2080.html>  FLEXBOX  <http://www.w3cplus.com/blog/tags/157.html>  COLUMNS  <http://www.w3cplus.com/blog/tags/86.html> |
|  | 如何理解CSS的display属性  <http://www.w3cplus.com/css/how-well-do-you-know-display.html>  CSS Grid布局指南  <http://www.w3cplus.com/css3/a-complete-guide-css-grid-layout.html>  一个完整的Flexbox指南  <http://www.w3cplus.com/css3/a-guide-to-flexbox-new.html>  Web布局新系统：CSS Grid,Flexbox和Box Alignment  https://www.w3cplus.com/css/css-grids-flexbox-and-box-alignment-our-new-system-for-web-layout.html |

关于多列布局[--column]

基础概念

|  |  |
| --- | --- |
|  | [css3中的多列布局columns详解](https://segmentfault.com/a/1190000003748570)  <https://segmentfault.com/a/1190000003748570>  CSS3多列布局(columns)的用法  <http://www.webhek.com/post/css3-multi-columns.html>  [CSS3布局样式相关：Columns、column-width、column-count、column-gap、column-rule、column-span、盒子模型、伸缩布局](http://blog.csdn.net/xxssyyyyssxx/article/details/53127860)  <http://blog.csdn.net/xxssyyyyssxx/article/details/53127860> |
| 说明： | 网页布局是指在网页中如何对导航栏、标题、主要内容、脚注、表单、、各种构成元素进行一个合理的编排。  在css3之前，组要使用float与position进行网页布局，但是使用这些属性也存在一些缺点，多栏中如果元素的内容高度不一致则底部很难对齐。  在css3中增加了一些新的布局方式，使用这些新的布局方式，除了可以修改之前存在的问题之外，还可以更快捷、更复杂的页面布局。  Multiple Columns多列自动布局功能可数排列，这种特性特别适合杂志和报纸类网页布局。使用多列自动布局功能能够自动将内容按指定的列数排列。（适合较长纯文档版式设计，不适合做网页结构布局，可以实现在多列中显示文字和图片）  css的Multi-column布局模块。这个模块给予了我们脱离position和float这些属性，就能在网页栏目上布局的能力。  Multi-column好就好在能够自动为你安排好流体内容，你用不着计算确定栏目的数量，让他们排排站好就行了！  然而，在使用Multi-column的时候，你还是有东西要注意下的，你要注意设置栏目的高度，否则别人在访问网站的时候，就因为高高的栏目，看个栏目内容还要上拉下拉滚动条那么费劲，这用户体验那该有多不好。  大部分的桌面浏览器和移动浏览器都支持Multicolumn（对于IE来说，你现在只能在IE10的平台上来预览一下）。一个快速查询一个属性是否被浏览器支持的方法就是到caniuse.com查找主流的桌面和移动浏览器是否都支持这些属性。我们要记住的是，也许一个属性是被支持的，但是也是添加浏览器供应商前缀，有些浏览器也不完全对属性进行支持，总之，我们一定要多测试，这才能更好地解决问题，避免不必要的麻烦。  浏览器完美兼容  对于一些不支持多列布局特征的浏览器，比如IE9/IE8，会把这些属性全部忽略，这样布局就呈现出传统的单块布局。  为了保证浏览器最大的兼容性，我们在使用多列布局属性时，最好添加[浏览器引擎前缀](http://www.webhek.com/vendor-prefixes)，最基本的要加上三种：谷歌浏览器的-webkit-，火狐浏览器的-moz-，IE浏览器的-ms-，最后，别忘了不带前缀的写法。  列高度的平衡  CSS3规范里描述的是，各列的高度是均衡的，浏览器会自动调整每列里填充多少文本、均分文本，来使各列的高度保持均衡一致。  然而，有时候，我们需要设定列的最大高度，这个时候，文本内容会从第一列开始填充，然后第二列，第三列，也许以后的列会填不满，也许会溢出。所以，当对多列布局设定了height或max-height属性值后，列会伸长到指定高度——无论内容有多少，够不够或超不超。  总结  CSS3的多列布局(columns)是一种方便WEB开发者高效利用宽屏显示器的非常有用的功能特征。你会发现在很多地方都需要用到它们，特别是需要自动平衡列高度的地方。 |
| 属性列表 | 以下列出column运用的相关属性  1.column-width：<length> | auto 设置或检索对象每列的宽度  如：-moz-column-width:200px;   1. column-count：<integer> | auto 设置或检索对象的列数 如：-webkit-column-count:3;   3.column-gap：<length> | normal 设置或检索对象的列与列之间的间隙 如：column-gap:normal;column-gap:40px;  4.column-rule：[ column-rule-width ] || [ column-rule-style ] || [ column-rule-color ]设置或检索对象的列与列之间的边框。复合属性。相当于border属性 如：column-rule:10px solid #090;  5.column-span：none | all 设置或检索对象元素是否横跨所有列。 如：column-span:all;  6.column-fill：auto | balance 设置或检索对象所有列的高度是否统一。 auto：列高度自适应内容 balance：所有列的高度以其中最高的一列统一 如：column-fill:balance;  7.column-break-before：auto | always | avoid | left | right | page | column | avoid-page | avoid-column 设置或检索对象之前是否断行。  auto：既不强迫也不禁止在元素之前断行并产生新列 always：总是在元素之前断行并产生新列 avoid：避免在元素之前断行并产生新列  8.column-break-after：auto | always | avoid | left | right | page | column | avoid-page | avoid-column 设置或检索对象之后是否断行。  auto：既不强迫也不禁止在元素之后断行并产生新列 always：总是在元素之后断行并产生新列 avoid：避免在元素之后断行并产生新列  9.column-break-inside：auto | avoid | avoid-page | avoid-column 设置或检索对象内部是否断行。  auto：既不强迫也不禁止在元素内部断行并产生新列avoid：避免在元素内部断行并产生新列  其实常用的也就前面5条。 |
| column-width:20px;[规定每列的长度]  column-count:3;[分隔的列数]  column-gap:10px;[规定列之间的距离](gap:缺口、间隔)  column-rule:1px solid red;[-webkit--moz-][列之间的填充方式](rule: 规则)  column-span: 1|all;[默认值为1，all:跨列显示] |

列宽

|  |  |
| --- | --- |
| 列个数 列宽度 | 写法：  columns:250px 3;[一列最少250px,最多3列][简写：column-width:250px;column-count:3;]  columns:250px;[一列最少250px]  columns:3;[一定是3列]  同样，根据容器的大小，就可以控制创建栏目的数量，这是非常了不起的一个特点。举个例子，如果你需要一个栏目宽度为13em（用px作为长度单位也是可以的）的容器，并且根据这个容器的大小，创建数量适合的栏目数：  #mcexample {  column-width: 13ems;  }  还可以自己定义栏目的数量，在这个时候，这些栏目就会自动平均分配栏目宽度，尽量充满容器的宽度。  #mcexample {  column-count: 3;  }  同时声明了 column-width:8em 和 column-count:12 的情况可以用以下简写替换：columns:12 8em;  columns：[ column-width ] || [ column-count ]  设置或检索对象的列数和每列的宽度，其中：[ column-width ]：设置或检索对象每列的宽度；[ column-count ]：设置或检索对象的列数。 |

列间距 列边

|  |  |  |  |
| --- | --- | --- | --- |
| 列之间的缝隙间隔宽度 | 更具体地控制栏目的数目，栏目的自身宽度，栏目与栏目之间的间隔等样式：  在默认的情况下，每个栏目之间的间隔长度是1em，但是你可以通过column-gap这个属性来进行改动。同样，你也可以在column-rule对column进行样式上的修改，我们可以回想到border这个属性也有着同样的写法规则。下面这个代码语句生成一个宽度为2px的虚线，而且column的间隔为28px-为大家展示了另一种的规则。  <style>  div {  columns: auto 13em; /\* column-count,column-width \*/  column-gap: 28px;  column-rule: 2px dotted #ccc; /\* width, style, color \*/  } </style>  [column-width和column-count](http://www.w3.org/TR/css3-multicol/" \l "the-number-and-width-of-columns" \t "http://www.w3cplus.com/_blank)可让让一个元素进行多列布局，那么[column-gap和column-rule](http://www.w3.org/TR/css3-multicol/" \l "column-gaps-and-rules" \t "http://www.w3cplus.com/_blank)就处在相邻两列之间，用我们印刷排版来说，[column-gap](http://www.w3.org/TR/css3-multicol/" \l "column-gap0" \t "http://www.w3cplus.com/_blank)就相当于两列之间的空白处，而[column-rule](http://www.w3.org/TR/css3-multicol/" \l "column-rule0" \t "http://www.w3cplus.com/_blank)就相当于一条分隔线，换句话说呢？[column-gap](http://www.w3.org/TR/css3-multicol/" \l "column-gap0" \t "http://www.w3cplus.com/_blank)就像我们web页面中的margin一样，而column-rule就类似于border，不过他们只是存在相邻两列之间。另外[column-gap](http://www.w3.org/TR/css3-multicol/" \l "column-gap0" \t "http://www.w3cplus.com/_blank)和[column-rule](http://www.w3.org/TR/css3-multicol/" \l "column-rule0" \t "http://www.w3cplus.com/_blank)是有高度的，其高度和列等高，最大区别是，[column-gap](http://www.w3.org/TR/css3-multicol/" \l "column-gap0" \t "http://www.w3cplus.com/_blank)没有任何样式，而且他在列与列之间占有一定的空间，而[column-rule](http://www.w3.org/TR/css3-multicol/" \l "column-rule0" \t "http://www.w3cplus.com/_blank)是有一定的样式，类似于border一样（只是能用的相当的少），其在列与列之间不占有任何空间位置。 | | |
| 列间距column-gap | 语法： column-gap: normal || <length>  取值说明：  1、normal为默认值，默值为1em（如果你的字号是px伫值，其默认值为你的font-size值）；  2、<length>:此值用来设置列与列之间的距离，其可以使用px,em单位的任何整数值，但不能了负值。  兼容的浏览器：  ie10+ firefox3.5+ chrome1.0+ ... | | |
| .columnGap {  column-count: 2;  column-gap: 20px;  } | IMG_257 | |
| .columnGapWidth {  column-width: 193px;  column-count: 2;  column-gap: 20px;  } | IMG_258 | |
| > 例子1是在column-width为auto下，我们把元素分成两列，并把列与列的间距定为20px，从效果图中，明显可以看出，column-gap将相邻两列以20px的宽度隔开了。  > 例子2在例子1基础上加上一个列的宽度值，从效果中很容易得到答案，就是容器无法显示两列，大家都知道，元素总宽度是400px,现在每列定在193px,总共分成了2列，在前面的例子大家都知道，column-gap为normal时刚好正常以2列并且每列为193px的宽度显示，可现在加上一个列间距20px。  > 这是一个很简单的数学题，多列各糁数值之各超过多列元素总宽度，以至于元素无法按参数值进行布局。这里说这么多，无非想告诉大家column-gap可以用来改变相邻列之间距离，但在多列元素同时设置了column-width时，column-gap与column-width之和大于多列元素总宽度时，会导至列被撑破，并以第一列显示，此时的列宽自动调节到元素的总宽度。  > 其实这一点很像我们平时中用float布局一样，大家可以慢慢体会一下。 | | |
| 列边框column-rule | 前面就说到过，column-rule就是用来定义列与列之间的边框宽度，样式和颜色的，简单点说，就有点类似于常用border属性。但column-rule是不占用任何倥间位置的，在列与列之间改变其宽度并不会改变任何元素的位置。这样的话，当column-rule的宽度大于column-gap时，column-rule将会和相邻的列重叠，甚至有可能延长超出了multi元素框，从而形成了元素的背景色；但有一点需要注意column-rule只存在两边都有内容的列之间。  为了更能形像点的理解，在你的脑海中你可以把column-rule当作元素中的border来理解，因为column-rule同样就具有border类似的属性：宽度column-rule-width，样式column-rule-style，颜色column-rule-color，不同的是border占有一定的空间位置，而column-rule不占有任何窨控间的位置，下面也和column-gap一样，从其语法着手，看看column-rule如何应用在多列元素中，以将会如何改变多列元素？  语法：  column-rule: <column-rule-width> || <column-rule-style> || <column-rule-color>  取值说明：  1、column-rule-width：此值是用来定义column-rule的宽度，默认值为“medium”，不允许取负值。类似于border-width属性，详细可以参考[column-rule-width](http://www.w3.org/TR/css3-multicol/" \l "column-rule-width" \t "http://www.w3cplus.com/_blank)；  2、column-rule-style：此值是用来定义column-rule的样式，其默认值为“none”，如果取值为默认值时，column-rule-width值将等于“0”，column-rule-style样式种类和border-style一样，详细可以参考[column-rule-style](http://www.w3.org/TR/css3-multicol/" \l "column-rule-style" \t "http://www.w3cplus.com/_blank)；  3、column-rule-color：此值用来定义column-rule的颜色，其默认值为前景色color的值，使用相当于border-color，如果不希望显示颜色，也可以将其设置为transparent(透明色)，详细可以参考[column-rule-color](http://www.w3.org/TR/css3-multicol/" \l "column-rule-color" \t "http://www.w3cplus.com/_blank)。  前面也说过column-rule的使用方法大家就可以把他当作border的使用，所以我们也可以把column-rule像border属性一样拆分成：  column-rule-width: <border-width>;  column-rule-style: <border-style>;  column-rule-color: <border-color>;  兼容的浏览器：  ie10+ firefox 3.5+ chrome1.0+ | | |
| .columnRule {  -moz-column-width: 190px;  -webkit-column-width: 190px;  column-width: 190px;  -webkit-column-count: 2;  -moz-column-count: 2;  column-count: 2;  -webkit-column-gap: 20px;  -moz-column-gap: 20px;  column-gap: 20px;  -moz-column-rule: 5px solid red;  -webkit-column-rule: 5px solid red;  column-rule: 5px solid red;  } | | IMG_260 |
| .columnRule {  background: none repeat scroll 0 0 orange;  color: blue;  -moz-column-rule: 150px solid red;  -webkit-column-rule: 150px solid red;  column-rule: 150px solid red;  } | | IMG_261 |
| 上面是一个非常简单的实例，下面我想在上面的例子基础上，稍作一下改变，给元素的column-rule-width变大，并同时改变一下元素的color和background值  上图的效果告诉我们两点信息，其一，column-rule-width增大并不会影响列的布局，只会将其往元素两边扩展，直到元素边缘为止，同时也再一次说明column-rule是不占有任何空间位置的；其二，column-rule在z轴上是介于background和content之间的，有时也可以利用其这点优势制作一些特殊的效果，大家可以发挥一下自己的想像空间。  column-rule好是好，只是不能使用图片，如果能像border-image一样增加一个column-rule-image多好，期待以后会有这么一个属性。  到这里有关于Multi-columns中的column-gap和column-rule就介绍完了，最后规纳一点：column-gap就类似于元素中的margin和padding，具有一定的空间位置，当其值过大时也会撑破列布局，但和margin,padding不同的是，其只存在列与列之间，并与列高度相等；而column-rule就类似于元素的border，可以设置宽度，边框样式，边框颜色，并且column-rule不具有任何空间位置，同时在z轴介于元素的background和content之间，其同样具有与列一样的高度。 | | |

demo

|  |  |
| --- | --- |
| demo-columns | <**style**>  **body** {  font: 14px/1.5 georgia, serif, sans-serif;  }  **p** {  margin: 0;  padding: 5px 10px;  background: #eee;  }  **h1** {  margin: 10px 0;  font-size: 16px;  }  .test {  width: 628px;  -moz-columns: 200px 3;  -webkit-columns: 200px 3;  columns: 200px 3;  }  .test2 {  -moz-columns: 200px;  -webkit-columns: 200px;  columns: 200px;  } </**style**>  <**h1**>列数及列宽固定:</**h1**> <**div** class="test">  <**p**>This module describes multi-column layout in CSS. By using functionality described in this document, style sheets  can declare that the content of an element is to be laid out in multiple columns. </**p**>   <**p**>On the Web, tables have also been used to describe multi-column layouts. The main benefit of using CSS-based  columns is flexibility; content can flow from one column to another, and the number of columns can vary  depending on the size of the viewport. Removing presentation table markup from documents allows them to more  easily be presented on various output devices including speech synthesizers and small mobile devices.</**p**> </**div**> <**h1**>列宽固定，根据容器宽度液态分布列数:</**h1**> <**div** class="test2">  <**p**>This module describes multi-column layout in CSS. By using functionality described in this document, style sheets  can declare that the content of an element is to be laid out in multiple columns. </**p**>   <**p**>On the Web, tables have also been used to describe multi-column layouts. The main benefit of using CSS-based  columns is flexibility; content can flow from one column to another, and the number of columns can vary  depending on the size of the viewport. Removing presentation table markup from documents allows them to more  easily be presented on various output devices including speech synthesizers and small mobile devices.</**p**> </**div**> |
|  |
| demo | 将column-width值缩小，column-count值放大；  结合inline-block应该有所应用； |
|  |  |

关于伸缩盒Flexbox布局

|  |  |
| --- | --- |
| flex | Css3-flexbox/zh-hans 完全指南！  <https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans#.E6.8E.92.E5.BA.8F.E4.B8.8E.E6.96.B9.E5.90.91>  深入了解 Flexbox 伸缩盒模型  <http://www.w3cplus.com/blog/666.html>  Flexbox——快速布局神器  <http://www.w3cplus.com/css3/flexbox-basics.html>  <https://dev.opera.com/articles/flexbox-basics/>  一个完整的Flexbox指南 - 小总结 实例 兼容@mixin  <http://www.w3cplus.com/css3/a-guide-to-flexbox.html> |

基础概念

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ===  规范状态和浏览器支持情况  Flexbox 规范的相关工作已经进展了n年。不同的浏览器也实现了不同的实验版本。在2012年9月，[Flexbox 语法的第三个主要修订版本进入到候选推荐阶段](http://www.w3.org/TR/css3-flexbox/)。这意味着 W3C 认为当前的语法是稳定的，并鼓励浏览器开发商去实现它。  Flexbox 规范时间表：  2009年7月 工作草案 (display: box;)  2011年3月 工作草案 (display: flexbox;)  2011年11月 工作草案 (display: flexbox;)  2012年3月 工作草案 (display: flexbox;)  2012年6月 工作草案 (display: flex;)  2012年9月 候选推荐 (display: flex;)  ===  CSS 2.1 定义了四种布局模式 ― 由一个盒与其兄弟、祖先盒的关系决定其尺寸与位置的算法：  块布局 ― 为了呈现文档而设计出来的布局模式  行内布局 ― 为了呈现文本而设计出来的布局模式  表格布局 ― 为了用格子呈现 2D 数据而设计出来的布局模式  定位布局 ― 为了非常直接地定位元素而设计出来的布局模式，定位元素基本与其他元素毫无关系  本模块引入了新的布局模式：伸缩布局，是为了呈现复杂的应用与页面而设计出来的。  ===  伸缩布局只与块布局有非常微薄的相似之处。伸缩布局不具有[浮动](http://www.w3.org/html/ig/zh/wiki/CSS21/visuren" \l "floats)或是[多栏](http://www.w3.org/csswg/css3-multicol/)等等在块布局中可以用的复杂、面向文本／文档的属性，换来的是简单、强大的分配空间与对齐内容的工具，是 Web 应用与复杂的页面所需要的。  一个伸缩容器的内容：  1.可以以任何[伸缩方向](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-direction)（向左、向右、向下，甚至是向上！）配置  2.可以在样式层（也就是视觉顺序可以跟源码、语音顺序无关）[调换](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-flow-row-reverse)、[重排](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "order)显示顺序  3.可以沿着单一（[主](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "main-axis)）轴线性配置或是沿着第二（[侧](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "cross-axis)）轴[折行](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-wrap)  4.可以因为可用空间的存在[扩展内容的尺寸](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flexibility)  5.可以沿着容器或彼此[对齐](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "alignment)  6.可以在保持[侧轴长度](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "cross-size)不变之下动态[折叠](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "visibility-collapse)或反折叠  ===  Flexbox通常能让我们更好的操作他的子元素布局，例如：  1.如果元素容器没有足够的空间，我们无需计算每个元素的宽度，就可以设置他们在同一行；  2.可以快速让他们布局在一列；  3.可以方便让他们对齐容器的左、右、中间等；  4.无需修改结构就可以改变他们的显示顺序；  5.如果元素容器设置百分比和视窗大小改变，不用提心未指定元素的确切宽度而破坏布局，因为容器中的每个子元素都可以自动分配容器的宽度或高度的比例。 | | | |
| flexbox术语 | ===  一个设有「display:flex」或「display:inline-flex」的元素是一个[伸缩容器](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-container)，伸缩容器的子元素被称为[伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)，这些子元素使用伸缩布局模型来排版。  与布局计算偏向使用书写模式方向的块布局与行内布局不同，伸缩布局偏向使用[伸缩流](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-flow-property)的方向。为了让描述伸缩布局变得更容易，本章节定义一系列相对于伸缩流的术语。「[flex-flow](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-flow)」的值决定了这些术语如何对应到物理方向（上／右／下／左）、物理轴（垂直／水平）、物理大小（宽度／高度）。  · CSS的columns在伸缩容器上没有效果。  · float、clear和vertical-align在伸缩项目上没有效果。  ===  Flexbox 由 伸缩容器 和 伸缩项目 组成。通过设置元素的 [display](http://learnlayout.com/display.html) 属性为 flex 或inline-flex 可以得到一个伸缩容器。设置为 flex 的容器被渲染为一个块级元素，而设置为 inline-flex 的容器则渲染为一个行内元素。  伸缩容器中的每一个子元素都是一个伸缩项目。伸缩项目可以是任意数量的。伸缩容器外和伸缩项目内的一切元素都不受影响。简单地说，Flexbox 定义了伸缩容器内伸缩项目该如何布局。  ===  伸缩容器：一个设有“display:flex”的块级[伸缩容器](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-container)盒或“display:inline-flex”的行内级[伸缩容器](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-container)盒。  伸缩容器会为其内容建立新的伸缩格式化上下文(flex formatting context)。除了使用伸缩排版而不块排版以外，伸缩格式化上下文与块级格式化上下文(block formatting context)根元素相同 ― 浮动不会闯入伸缩容器，且伸缩容器的边界不与其内容的边界叠加。  伸缩容器不是块容器，因此有些设计用来控制块布局的属性，在伸缩布局中不适用。特别是：  1.多栏(column)模块的所有「column-\*」属性在伸缩容器上没有作用。  2.「float」与「clear」在[伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)上没有作用。(However, the ‘float’ property still affects the computed value of ‘display’ on children of a flex container, as this occurs before flex items are determined.)  3.「vertical-align」在[伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)上没有作用。  若元素「display」的指定值是「[inline-flex](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "display-inline-flex)」且元素是一个浮动或绝对定位元素，则「display」的计算值是「[flex](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "display-flex)」。也就是 [CSS 2.1 章节 9.7](https://www.w3.org/html/ig/zh/wiki/CSS2" \l "dis-pos-flo" \o "CSS2) 的表格增修「指定值」是「[inline-flex](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "display-inline-flex)」、「计算值」是「[flex](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "display-flex)」的一行。  ===  伸缩项目：伸缩容器的子元素。  一个[伸缩容器](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-container)的内容具有零个以上的伸缩项目 ― 伸缩容器的每一个子元素（除了需要盒修复的元素之外）都会成为一个[伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)，且用户代理会将任何直接在伸缩容器里的连续文字块包起来成为匿名[伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)。然而，若匿名[伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)仅包含空白，则用户代理不会渲染该伸缩项目，如同将之视为「display: none」。  [伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)为其内容建立一个新的伸缩格式化上下文，这个格式化上下文的种类由项目「display」的值决定。用户代理使用 [CSS 2.1 章节 9.7](https://www.w3.org/html/ig/zh/wiki/CSS2" \l "dis-pos-flo" \o "CSS2) 的表格决定[伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)「display」的计算值。CSS 工作组预计会在未来的规范里引入伸缩容器的新的「display」值，以取代这条叙述。  伸缩项目还需关注：绝对定位的伸缩子元素、伸缩项目的外边界、伸缩项目的绘制、流外项目、叠加项目、伸缩项目的默认最小长度... | | | |
|  | ===  主轴、主轴方向：用户代理沿着一个伸缩容器的主轴配置伸缩项目，主轴是主轴方向的延伸。  主轴起点、主轴终点：伸缩项目的配置从容器的主轴起点边开始，往主轴终点边结束。  主轴长度、主轴长度属性：伸缩项目的在主轴方向的宽度或高度就是项目的主轴长度，伸缩项目的主轴长度属性是width或height属性，由哪一个对着主轴方向决定。  侧轴、侧轴方向：与主轴垂直的轴称作侧轴，是侧轴方向的延伸。  侧轴起点、侧轴终点：填满项目的伸缩行的配置从容器的侧轴起点边开始，往侧轴终点边结束。  侧轴长度、侧轴长度属性：伸缩项目的在侧轴方向的宽度或高度就是项目的侧轴长度，伸缩项目的侧轴长度属性是「width」或「height」属性，由哪一个对着侧轴方向决定。  ===  主轴和侧轴  如果你要想flexbox工作正常，你有些概念必须得清楚，比如说让flexbox正常工作的主轴和侧轴，他们看上去有点像X轴和Y轴，但还是有所差别的。主轴的方向主要是用来确定flex的主方向，所以你子元素要么放置在一行，要么放置在一列。侧轴主要垂直于主轴运行，如下图所示：  IMG_256  【根据伸缩项目排列方式不同，主轴和侧轴方向也有所变化】  ===  Flex Lines 伸缩行  伸缩项目沿着伸缩容器内的一个 伸缩行 定位。通常每个伸缩容器只有一个伸缩行。  ===  The Main Axis and the Cross Axis 主轴和侧轴  为了描述抽象的书写模式，Flexbox 使用 主轴 和 侧轴的概念。伸缩行跟随主轴。侧轴则垂直于主轴。  在继续了解之前明白主轴和侧轴是至关重要的。Flexbox 中的一切都和这些轴有关。  在我们所有的例子中，书写模式都是从左至右，从上到下，但是你需要记住并不是所有的 Flexbox 都是这样的。  ===  Writing Modes 书写模式  在你设计 Flexbox 时的有一个重要的部分是更改伸缩行的方向。默认情况下，伸缩行和文本方向一致：从左至右，从上往下。  这是 [W3C 关于一个名为书写模式的新特性工作草稿](http://www.w3.org/TR/css3-writing-modes/)。书写模式是一个新的方法，让你可以从右往左写，甚至竖着写，就像你知道的某些语言一样。  书写模式是一个正在进行的计划，但是 Chrome 已经率先支持了 direction CSS 属性。如果我们在上一个例子中设置方向为 rtl (从右往左) 那么不仅仅文字会从右往左书写，而且 伸缩行也改变了方向，并更改了页面的布局。  这也许就是 Flexbox 为什么如此抽象难懂的地方。当你正在制作一个语言不确定的页面时你不能简单的只是说上下左右。  flex container  flex items  flex line  主轴 起点  侧轴 起点  主轴 终点  侧轴 终点 | | | |
| 属性值 | 属性 | 属性值 | 初始值 | 适用于 |
| [justify-content](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "justify-content) | flex-start | flex-end |  center |  space-between | space-around | flex-start | flex containers |
| [align-items](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "align-items) | flex-start | flex-end |  center |  baseline |  stretch | stretch |
| [flex-flow](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-flow) | <flex-direction> || <flex-wrap> |  |
| flex-wrap | nowrap |  wrap | wrap-reverse | nowrap |
| [flex-direction](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-direction) | row | row-reverse |  column | column-reverse | row |
|  | | | |
| [align-self](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "align-self) | auto |  flex-start | flex-end |  center |  baseline | stretch | auto | flex items |
| [flex](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex) | none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ] |  |
| [flex-grow](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-grow) | <number> | 0 |
| [flex-shrink](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-shrink) | <number> | 1 |
| [flex-basis](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-basis-propdef) | <'width'> | auto |
| [order](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "order) | <整数> | 0 | flex items and absolutely-positioned flex container children |
| [align-content](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "align-content) | flex-start | flex-end |  center |  space-between | space-around |  stretch | stretch | multi-line flex containers |
| display |  |  |  |
| min-width  min-height |  |  |  |

简要说明

|  |  |  |
| --- | --- | --- |
|  | [Flexible Box Layout module](http://www.w3.org/TR/2009/WD-css3-flexbox-20090723/" \t "http://www.w3cplus.com/css3/_blank)（flexbox）为大家提供了一个在父级容器中不用考虑具体height和width数值的自适应方法。  举个简单的例子来说吧，想象你有两个子级元素，你希望他们都能够以相同的宽度一起来填满他们的父级宽度。你也许能够使用百分比来实现，但是一旦要把border，padding，margins考虑进来的时候，情况就复杂多了。Flexbox的出现便让这个问题不再成为问题：  <**style**>  .parent {  display: flex;  }  .child-one, .child-two {  flex: 1;  } </**style**> <**div** class="parent">  <**div** class="child-one">1</**div**>  <**div** class="child-two">1</**div**> </**div**>  简单地放置两个子级元素水平排列在父级容器当中，同时使其宽度一致。  > 当child-one设置宽度时，效果将不一样；  flex的数值实际上就是比例的意思，在这些子级元素当中按比例地重新分配空出来的空间在他们的宽度：  .child-one { flex: 1; }.child-two { flex: 2; }  当这两个子级元素的宽度都在父级容器中都得到分配的时候，  .child-two的宽度将会以剩余空间的三分之二得到补充，  .child-one的宽度将会以剩余空间的三分之一得到补充； | |
| demo-direction | <**style**>  **body** {  direction: rtl;**/\*\*/** }  .flex-container {  display: flex;  width: 300px;  height: 240px;  }  .flex-item {  width: 100px;  height: 100px;  margin: 5px;  } </**style**> <**div** class="flex-container">  <**div** class="flex-item">flex item 1</**div**>  <**div** class="flex-item">flex item 2</**div**> </**div**> | 【direction:rtl】 |

伸缩容器的属性

|  |  |  |
| --- | --- | --- |
|  | Flexbox有一系列的方法来帮助你调整伸缩项目（flexbox子元素）沿着主轴和侧轴的对齐。  主轴对齐伸缩项目：justify-content  侧轴对齐伸缩项目：align-items align-self  伸缩项目堆栈伸缩行：align-content  您还可以调准伸缩行在伸缩容器里的对齐方式：flex-flow | |
| 主轴对齐justify-content | 伸缩容器的 justify-content 属性主要用来设置伸缩项目沿主轴的对齐方式，从而调整伸缩项目之间的间距。  flex-start：（默认值）伸缩项目向一行的起始位置靠齐。该行的第一个伸缩项目在主轴起点边的外边距与该行在主轴起点的边对齐，同时所有后续的伸缩项目与其前一个项目对齐。  flex-end：伸缩项目向一行的结束位置靠齐。该行的最后一个伸缩项目在主轴终点边的外边距与该行在主轴终点的边对齐，同时所有前面的伸缩项目与其后一个项目对齐。  center：伸缩项目向一行的中间位置靠齐。该行的伸缩项目将相互对齐并在行中居中对齐，同时第一个项目与该行的在主轴起点的边的距离等同与最后一个项目与该行在主轴终点的边的距离（如果剩余空间是负数，则保持两端溢出的长度相等）。  space-between：伸缩项目会平均地分布在一行里。如果剩余空间是负数，或该行只有一个伸缩项目，则此值等效于「flex-start」。在其它情况下，第一个项目在主轴起点边的外边距会与该行在主轴起点的边对齐，同时最后一个项目在主轴终点边的外边距与该行在主轴终点的边对齐，而剩下的伸缩项目在确保两两之间的空白空间相等下平均分布。  space-around：伸缩项目会平均地分布在行里，两端保留一半的空间；空白会平均分；  IMG_256  flex-start, flex-end, 和 center 一看就懂。  space-between 和 space-around 则是分配伸缩项目之间空白空间的不同方法。 | |
| ？？？设置了这个属性，在主轴方向上设置的任何margin都不会起作用。  为伸缩项目设置了一个百分比宽度，然后在伸缩容器设置了一个值，证明伸缩项目在主轴方向的margin不起作用：  <**style**>  .box {  display: flex;  flex-flow: row nowrap;  align-items: stretch; **/\*垂直等高\*/** justify-content: space-around; **/\*内容与空白平均分\*/** }  .box > **div** {  background: limegreen;  margin: 100px;  }  #first {  width: 25%;  height: 100px;  }  #second {  width: 40%;  }  #third {  width: 25%;  } </**style**> <**div** class="box">  <**div** id="first">1</**div**>  <**div** id="second">2</**div**>  <**div** id="third">3</**div**> </**div**> | |
| 侧轴对齐[align-items](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "align-items)([align-self](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "align-self)) | align-items 是一个和 justify-content 相呼应的属性，用于调整伸缩项目在侧轴上的定位方式，它充许您调整伸缩项目在侧轴的对齐方式，可以用来设置伸缩容器中包括匿名伸缩项目的所有项目的对齐方式。  flex-start：(默认值)伸缩项目在侧轴起点边的外边距紧靠住该行在侧轴起点的边。  flex-end：伸缩项目在侧轴终点边的外边距靠住该行在侧轴终点的边。  center：伸缩项目的外边距盒在该行的侧轴上居中放置。（如果伸缩行的尺寸小于伸缩项目，则伸缩项目会向两个方向溢出相同的量）。  stretch：伸缩项目拉伸，填满整个侧轴；它会将伸缩项目从侧轴起点拉伸到侧轴终点。  baseline：让伸缩项目与它们的基线对齐。基线根据伸缩项目的内容计算得到。  为了更形像的理解"align-items"各个属性值对应在侧轴上的效果，可以参考下图：  IMG_256  这些都是术语解释，只有动手去尝试，调整不同的值，才能知道各个属性值所代表的运行效果，大家可以看看下图所运行的效果。  往往所有列表项的内容无法填满父元素的整个高度，特别是在不知道宽度和高度的视窗变化之下。很多情况之下为了让这些列实现等高效果，是一件多么可怕的事情，而且很多时间都浪费在一些呆反的解决方案和处理兼容上，比如说设置一个等高的效果，使用假的列？  如果你执着于等高的解决方案，你不仿移步看看这篇教程《[八种创建等高列布局](http://www.w3cplus.com/css/creaet-equal-height-columns" \t "http://www.w3cplus.com/css3/_blank)》。——大漠 | |
| 堆栈伸缩行 [align-content](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "align-content) | 当伸缩容器的侧轴还有多余空间时，「align-content」属性可以用来调准伸缩行在伸缩容器里的对齐方式，各属性值的意义如下：  flex-start：各行向伸缩容器的起点位置堆叠。伸缩容器中第一行在侧轴起点的边会紧靠住伸缩容器在侧轴起点的边，之后的每一行都紧靠住前面一行。  flex-end：各行向伸缩容器的结束位置堆叠。伸缩容器中最后一行在侧轴终点的边会紧靠住该伸缩容器在侧轴终点的边，之前的每一行都紧靠住后面一行。  center：各行向伸缩容器的中间位置堆叠。各行两两紧靠住同时在伸缩容器中居中对齐，保持伸缩容器在侧轴起点边的内容边和第一行之间的距离与该容器在侧轴终点边的内容边与第最后一行之间的距离相等。（如果剩下的空间是负数，则行的堆叠会向两个方向溢出的相等距离。）  space-between：各行在伸缩容器中平均分布。如果剩余的空间是负数或伸缩容器中只有一行，该值等效于「flex-start」。在其它情况下，第一行在侧轴起点的边会紧靠住伸缩容器在侧轴起点边的内容边，最后一行在侧轴终点的边会紧靠住伸缩容器在侧轴终点的内容边，剩余的行在保持两两之间的空间相等的状况下排列。  space-around：各行在伸缩容器中平均分布，在两边各有一半的空间。如果剩余的空间是负数或伸缩容器中只有一行，该值等效于「center」。在其它情况下，各行会在保持两两之间的空间相等，同时第一行前面及最后一行后面的空间是其他空间的一半的状况下排列。  stretch：各行将会伸展以占用剩余的空间。如果剩余的空间是负数，该值等效于「flex-start」。在其它情况下，剩余空间被所有行平分，扩大各行的侧轴尺寸。  IMG_256  align-content和align-items、justify-content相似，但是不是对齐伸缩项目，它对齐的是伸缩行；  注：只有多行的伸缩容器才会在侧轴上有多余的空间以供对齐，因为仅包含一行的伸缩容器中，唯一的一行会自动伸展填充全部的空间。 | |
| flex-flow 伸缩方向与换行 | flex-flow 是 flex-direction 和 flex-wrap 的缩写。  > flex-direction : row column row-reverse column-reverse  > flex-wrap : no-wrap wrap wrap-reverse  可以定义关键词row、column来指定子元素布局是在一行还是一列显示。  可以定义关键词wrap来指定内容容器在新的一行（当父元素容器太小，flexbox元素想在同一行显示）。 | |
| 有关于flex-flow  flex-flow是用来伸缩行换行，flex-flow属性是同时设定“flex-direction(伸缩流的方向)”和“flex-wrap（伸缩行换行）”属性的缩写，两个属性决定了伸缩容器的主轴与侧轴。此属性主要适用于伸缩容器。在这篇文章的例子中，主要是“footer”元素。  flex-direction  flex-direction属性可以用来设定伸缩容器的主轴的方向，这也决定了用户代理配置伸缩项目的方向。主要适用于伸缩容器，主要包括以下几个值：  > row:flex-direction的默认值，表示伸缩容器的主轴与当前书写模式的行内轴（文字布局的主要主向）。主轴起点与主轴终点方向分别等同于当前书写模式的始与结方向。  > row-reverse：表示的是除了主轴起点与主轴终点方向交换以外同row属性值的作用。  > column：表示的是伸缩容器的主轴与当前书写模式的块轴（块布局的主要方向）同向。主轴起点与主轴终点方向分别等同于当前书写模式的前与后方向。简单的可以理解为列布局。  > column-reverse：表示的是除了主轴起点与主轴终点方向交换以外同“column”的属性值作用。  flex-wrap  flex-wrap属性主要用来控制伸缩容器是单行还是多行，也决定了侧轴方向一新的一行的堆放方向。主要适用于伸缩容器，主要包括以下几个值：  > nowrap:flex-wrap的默认值，表示的是伸缩容器为单行。侧轴起点方向等同于当前书写模式的起点或前/头在侧轴的那一边，而侧轴终点方向是侧轴起点的相反方向。  > wrap：表示的是伸缩容器为多行。侧轴起点方向等同于当前书写模式的起眯或前/头在侧轴的那一边，而侧轴终点方向是侧轴起点的相反方向。  > wrap-reverse：除了侧轴起点与侧轴终点方向交换以外同wrap所起作用相同。  注：以上内容来自于[Css3-flexbox/zh-hans](http://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l ".E6.8E.92.E5.BA.8F.E4.B8.8E.E6.96.B9.E5.90.91" \t "http://www.w3cplus.com/css3/_blank) ——大漠 | |
| flex-direction 伸缩流的方向 | flex-direction 允许你更改伸缩容器的主轴方向。  row：（默认值）该值表示伸缩项目根据书写模式的方向布局。（默认是从左至右，从上到下）。  row-reverse: 主轴起点和主轴终点交换。如果书写模式是从左至右，伸缩项目则是从右往左显示。  column: 主轴和侧轴交换。如果书写系统是垂直的，那么伸缩项目也是垂直显示的。  column-reverse: 和 column 一样，但是方向相反。  让我们把前一个示例中的 flex-direction 改为 column。 | |
| <**style**>  .flex-container {  display: flex;  flex-direction: column;  width: 300px;  height: 240px;  }  .flex-item {  width: 100px;  height: 100px;  margin: 5px;  } </**style**> <**div** class="flex-container">  <**div** class="flex-item">flex item 1</**div**>  <**div** class="flex-item">flex item 2</**div**> </**div**> |  |
| flex-wrap 伸缩行换行 | 伸缩容器可以是[单行](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "single-line)的，也可以是[多行](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "multi-line)的，这由「[flex-wrap](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-wrap)」属性决定：  > 单行的伸缩容器会将其所有子元素在单独的一行上进行布局，即便这可能导致内容溢出。  > 多行的伸缩窗口会将其[伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)配置在多个伸缩行上，这类似于文本的排列：当文本过宽导致一行无法容纳时，内容会断开并移至新的一行。当用户代理创建新的伸缩行时，这些伸缩行会根据「[flex-wrap](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-wrap)」属性沿着[侧轴](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "cross-axis)进行堆叠。除非伸缩容器本身是空的，每一个伸缩行至少包含一个[伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)。  使用 flex-wrap 你可以为伸缩容器创建多个伸缩行。这个属性接受以下值：  nowrap： (默认值)，伸缩项目的width不起绝对作用  wrap：当一个伸缩行中的伸缩项目容不下时（设置width），会换行到一条新增的伸缩行上，新增的伸缩行根据侧轴的方向添加。  wrap-reverse：wrap-reverse 和 wrap 一样，只是新的伸缩行会被添加到侧轴的反方向上。 | |
| <**style**>  .flex-container {  display: flex;  flex-wrap: wrap;  width: 300px;  height: 240px;  }  .flex-item {  width: 100px;  height: 100px;  margin: 5px;  } </**style**> <**div** class="flex-container">  <**div** class="flex-item">flex item 1</**div**>  <**div** class="flex-item">flex item 2</**div**>  <**div** class="flex-item">flex item 3</**div**> </**div**> | > 由于容器的宽度是 300px，同一行上仅能放得下两个项目，由于「[flex-wrap](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-wrap)」属性让伸缩容器变为多行伸缩容器（因为使用了「[wrap](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-wrap-wrap)」关键字），伸缩容器会多创建一个伸缩行来放置最后一个项目。    【nowrap（默认值）】    【wrap】 |
| <**style**>  .flex-container {  display: flex;  flex-wrap: wrap;  width: 300px;  height: 240px;  }  .flex-item {  flex: auto;**/\*\*/** width: 100px;  height: 100px;  margin: 5px;  } </**style**> <**div** class="flex-container">  <**div** class="flex-item">flex item 1</**div**>  <**div** class="flex-item">flex item 2</**div**>  <**div** class="flex-item">flex item 3</**div**> </**div**> | > 用户代理将内容分布到不同的行之后，每一行单独进行布局：可伸缩长度的计算以及「[justify-content](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "justify-content)」、「[align-self](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "align-self)」的效果计算每次仅考虑单独一个伸缩行上的项目。  > 当一个伸缩容器有多个伸缩行时，每一行的[侧轴长度](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "cross-size)为可容纳该行中的所有[伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)（依据「[align-self](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "align-self)」属性进行对齐之后）的最小值，这些伸缩行依据「[align-content](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "align-content)」属性在伸缩容器中进行对齐。当伸缩容器（[单行](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "single-line)或[多行](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "multi-line)）只有单独一行伸缩行时，这一行的[侧轴长度](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "cross-size)为伸缩容器的[侧轴长度](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "cross-size)，在这种情形下「[align-content](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "align-content)」属性没有效果。一行的[主轴长度](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "main-size)始终和伸缩容器内容盒的[主轴长度](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "main-size)相同。    【flex:auto】 |

伸缩项目item的属性

|  |  |  |  |
| --- | --- | --- | --- |
|  | 一个伸缩项目是一个伸缩容器的子元素。伸缩容器中的文本也被视为一个伸缩项目。  伸缩项目中内容与普通流一样。举例来说，当一个伸缩项目被设置为浮动，你依然可以在这个伸缩项目中放置一个浮动元素。  伸缩项目都有一个 主轴长度(Main Size) 和一个 侧轴长度(Cross Size)。主轴长度是伸缩项目在主轴上的尺寸。侧轴长度是伸缩项目在侧轴上的尺寸。或者说，一个伸缩项目的宽或高取决于伸缩容器的轴，可能就是它的主轴长度或侧轴长度。  下面的属性可以调整伸缩项目的行为； | | |
| flex 伸缩性 | flex: [flex-grow] [flex-shrink] [flex-basis]; 扩展比率flex-grow，收缩比率flex-shrink以及伸缩基准值flex-basis。  flex用来决定伸缩项目的伸缩性。一个伸缩容器会等比地按照各伸缩项目的扩展比率分配剩余空间，也会按照收缩比率缩小各项目以避免溢出。  当一个元素是[伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)时，「[flex](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex)」属性将代替[主轴长度属性](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "main-size-property)决定元素的[主轴长度](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "main-size)。若元素不是[伸缩项目](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex-item)，则「[flex](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "flex)」属性没有效果。  具体介绍见：<https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans#flex>  1.flex-grow：此属性值为正数值，用来设置扩展比率，也就是剩余空间是正值的时候此伸缩项目相对于伸缩容器里其他伸缩项目能分配到空间比例。若省略则会被设置为“0”。  2.flex-shrink：此属性值为正数值，用来设置收缩比率，也就是剩余空间是负值的时候此伸缩项目相对于伸缩容器里其他伸缩项目能收缩的空间比例。若省略则会被设置为“1”，在收缩的时候收缩比率会以伸缩基准值加权。  3.flex-basis：与width属性使用相同的值，可以用来设置flex-basis长写并指定伸缩基准值，也就是根据可伸缩比率计算 出剩余空间的分布之前，伸缩项目主轴长度的起始数值。若在flex缩写省略了此属性设置，则flex-basis的指定值是“0”，若flex-basis的指定值是“auto”，则伸缩基准值的指定值是元素主轴长度属性的值。  IMG_256  [ 一个显示「绝对」伸缩（以零为基准值开始）与「相对」伸缩（以项目的内容大小为基准值开始）差异的图解。这三个项目的伸缩比例分别是「1」、「1」、「2」。] | | |
| 大多数情况下没必要使用这种语法，可以把 flex-grow, flex-shrink, 和 flex-basis 这3个缩写为1个声明；默认值为 0 1 auto ;  当然你也可以将 flex-grow, flex-shrink, 和 flex-basis 作为单个属性分开来设置，如果你觉得自己挺厉害的，[到规范里看一下吧](http://www.w3.org/TR/css3-flexbox/" \l "flex)，但我强烈反对这种方式：当使用 flex 缩写时，即使没有某些值没有设置也能获得更合理的默认值； | | |
| > flex: initial; 设置该属性为它的默认值，即为 0 1 auto。请参阅 initial。一个 flex 属性值被设为 initial 的伸缩项目，在有剩余空间的情况下不会有任何变化，但是在必要的情况下会被收缩。  > flex: auto; 与 1 1 auto 相同。一个 flex 属性值被设为 auto 的伸缩项目，会根据主轴自动伸缩以占用所有剩余空间。 auto 目前仅在 Opera 12.11 尚有效，在 Chrome 23.0.1271.95 上无效。你可以通过使用 flex: 1; 来达到一样的效果。  > flex: none; 与 0 0 auto 相同。一个 flex 属性值被设为 none 的伸缩项目，在任何情况都不会发生伸缩。  ===  flex还有其他的值，[Common values of flex](http://dev.w3.org/csswg/css-flexbox/" \l "flex-common" \t "http://www.w3cplus.com/css3/_blank)做了详细的介绍。其中“auto”和“initial”非常有用。  设置“flex:auto”时，伸缩容器中的伸缩项目（相当于flex: 1 1 auto）将使用其大小根据任何width/height或者min-width/min-height设定，它将扩展占用一个比例的任何自由空间可用，但在没有额外的自由空间将缩小以适应其内容。结合min-width将可能会产生一些有趣的效果，可以看看[flex auto](http://devfiles.myopera.com/articles/10532/flex-auto-test.html" \t "http://www.w3cplus.com/css3/_blank)的实例。在这个实例中，父容器设置了“flex-flow:row”，三个子元素设置了“flex:auto”并设置了一个“min-width”。因此不管是多行还是单行，任何多余的空间都扩展到伸缩项目上。然后会收缩整齐，因为他变小后，充许子元素重新自适应。  试着将“flex:auto”修改成“flex:initial”(相当于flex:0 1 auto)，你会看到，当有多余的空间时，第三个子元素安容器大小不再增加，但仍然需要收缩。 | | |
| flex-grow | > flex: [number]  这个语法指定了一个数字，代表了这个伸缩项目该占用的剩余空间比例。  如果把每个伸缩项目都设置为 1 的话，那么剩余空间就会被平均分配了。  如下写法代表：第一个伸缩项目占用了 2/4 的剩余空间，而另外两个各占用了 1/4 的剩余空间。  <**style**>  .item1 { flex: 2; }  .item2 { flex: 1; }  .item3 { flex: 1; } </**style**> | | |
| flex-basis | 首先flex-basis的值主要取决于伸缩项目的width或者高，同时取决于流动方向。然后，剩下的空间根据flex-grow给伸缩项目最后宽度来划分。  所以伸缩项目会沿着主轴线大小为200px、300px和500px,总共1000px。如果伸缩容器沿主轴方向是1200px,这样就会多出一个200px空间，那么这多出的200px空间将分配给伸缩项目。  第一个和第三个伸缩项目将得到50px的空间，因为他的flex-grow值是“1”,他们最终的空间是250px和550px。第二个伸缩项目将获得100px空间，因为他的flex-grow值为“2”,他的最后空间大小为400px。 | | |
| <**style**>  .box {  display: flex;  width: 1200px;  }  #item1 {  flex: 1px;  }  #item2 {  flex: 2px;  }  #item3 {  flex: 1px;  } </**style**> <**div** class="box">  <**div** id="item1">1</**div**>  <**div** id="item2">2</**div**>  <**div** id="item3">3</**div**> </**div**>  [300\*1][300\*2][300\*1]    分配所有空间：flex-basics:0 | <**style**>  .box {  display: flex;  width: 1200px;  }  #item1 {  flex: 1 200px;  }  #item2 {  flex: 2 300px;  }  #item3 {  flex: 1 500px;  } </**style**> <**div** class="box">  <**div** id="item1">1</**div**>  <**div** id="item2">2</**div**>  <**div** id="item3">3</**div**> </**div**>  1200-(200+300+500)=200;  200/(1+2+1)=50;  [200+1\*50][300+2\*50][500+1\*50]  [250][400][550]    分配多余空间：flex-basics:auto | |
| flex-shrink | <**style**>  #first {  flex: 1 1 400px;  }  #second {  flex: 2 3 600px;  }  #third {  flex: 1 2 400px;  } </**style**>  flex-shrink称为收缩比率。这个值只有伸缩项目在没主轴方向溢出伸缩容器才会发挥作用。他们充当比例值，但这回指的是溢出量，将这个溢出量按比例分配给每个伸缩项目，用于防止伸缩容器溢出。  比如说，我们伸缩容器沿主轴方向宽度是1100px，按照上面的示例代码计算，我们的伸缩项目会超出300px(伸缩项目沿主轴方向总值为1400px)，这个时候通过flex-shrink收缩他们：  第一个伸缩项目将得到溢出量的六分之一，也就是50px，这个时候他的值为350px(在当初的基础上减少50px)。  第二个伸缩项目将得到溢出量的六分之三，也就是150px，这个时候他的值为450px(在当初的基础上减少150px)。  第三个伸缩项目将得到溢出的六分之二，就就是100px，这个时候他的值为300px(在当初的基础 上减少100px)。  这样flex-shrink使用伸缩项目得到一个较小的宽度。 | | |
| flex与响应式 | <**body**> <**footer** id="fixed">  <**section** id="first">1</**section**>  <**section** id="second">2</**section**>  <**section** id="third">3</**section**> </**footer**> </**body**>  <**style**>  **footer** {  display: flex;  flex-flow: row wrap;  align-items: stretch;  }  #first {  flex: 1 0 7rem;  }  #second {  order: 1;  flex: 2 0 8rem;  }  #third {  flex: 1.5 0 7rem;  }  **@media** screen and (max-width: 1000px) {  **body** {  width: 100%;  }  #fixed {  left: 0;  right: 0;  }  }  **@media** screen and (max-width: 520px) {  #fixed {  position: static;  }  **section** {  padding: 1rem 2rem;  }  **footer** {  padding: 0 1rem;  }  #first {  flex: 1 0 10rem;  }  #second {  flex: 1 0 10rem;  }  #third {  flex: 1 0 10rem;  }  }  **@media** screen and (max-width: 380px) {  **section** {  padding: 1rem 1rem;  }  **footer div** {  right: 1rem;  }  } </**style**> | <style>  #first {  flex: 1 0 7rem;  }  #second {  order: 1;  flex: 2 0 8rem;  }  #third {  flex: 1.5 0 7rem;  } </style>  结合一个多行flxbox(flex-flow:row wrap)和伸缩长度flex（如:flex:1 0 7rem）以及媒体查询实现了一个完美的效果。在不同视窗宽度下，伸缩项目在伸缩容器中可以平滑的进行变化。如图所示：    【Some simple Flexbox application has given us a useful responsive layout】 | |
|  | https://css-tricks.com/snippets/css/a-guide-to-flexbox/ | | |
| order 显示顺序 | order 是最简单明了的属性。设置伸缩项目的 order 可以调整它们渲染时的顺序。如果需要文档顺序和显示顺序不同时，这就是个很有用的功能了。  传统上不改变元素的结构要改变元素的布局顺序一直是一个痛苦的事情。不过在Flexbox中，你可以通过“order”属性来修改伸缩项目的布局顺序（在不调整结构前提之下）。这个属性一直接受的整数值——称为系数集——也称为排序组，会出现在伸缩项目中。  默认状态下，用户代理会用伸缩项目出现在源文档的次序配置这些伸缩项目。order属性透过将元素分到有序号的组以控制元素出现的顺序。在伸缩布局中，order属性控制伸缩项目在伸缩容器里的顺序。  默认情况之下，所有的伸缩项目的顺序组都是“0”。我们可以很容易的给每个伸缩项目设置不同的顺序值。更高的值会排在后面，而原来的HTML结构并不会有任何变化。order取值越大，越排在后面。并且order可以取负值。我们设置其中一个伸缩项目的order 为 -1，于是它被提前到了其他伸缩项目的最前面。 | | |
| <**style**>  .flex-container {  display: flex;  width: 300px;  height: 240px;  }  .flex-item {  width: 100px;  height: 100px;  margin: 5px;  }  .order{  order: -1;  } </**style**> <**div** class="flex-container">  <**div** class="flex-item">flex item 1</**div**>  <**div** class="flex-item order">flex item 2</**div**>  <**div** class="flex-item">flex item 3</**div**> </**div**> | |  |
| align-self 侧轴对齐 | 伸缩项目的 align-self 属性会覆盖该项目的伸缩容器的 align-items 属性。它的值和 align-items 一样：  stretch (默认)  flex-start  flex-end  center  baseline | | |
| <**style**>  .flex-container {  display: flex;  width: 300px;  height: 240px;  background-color: Silver;  }  .flex-item {  background-color: DeepSkyBlue;  width: 44px;  min-height: 100px;  margin: 3px;  }  .item1 {  align-self: flex-start;  }  .item2 {  align-self: flex-end;  }  .item3 {  align-self: center;  }  .item4 {  align-self: baseline;  }  .item5 {  align-self: baseline;  padding: 2em 0;  }  .item6 {  align-self: stretch;  } </**style**> <**div** class="flex-container">  <**div** class="flex-item item1">flex-start</**div**>  <**div** class="flex-item item2">flex-end</**div**>  <**div** class="flex-item item3">center</**div**>  <**div** class="flex-item item4">baseline</**div**>  <**div** class="flex-item item5">baseline</**div**>  <**div** class="flex-item item6">stretch</**div**> </**div**> | > 为每个伸缩项目应用不同的 align-self值 | |
| visibility 叠加项目 | 当该值生效时，应用 visibility: collapse; 和 visibility: hidden; 与display: none; 的效果是不一样的。如果是 collapse，该元素会影响伸缩容器的侧轴长度，但不会被现实或占用主轴的空间。如果你想动态添加或移除伸缩项目又不会影响伸缩容器的侧轴长度，这将会非常有用。  目前为止，visibility: collapse; 还没有被让任何浏览器正确的实现。现在visibility: collapse; 还和 visibility: hidden; 实现着一样的效果。我希望能尽快得到改观。  你可以在 [这里](http://www.w3.org/TR/css3-flexbox/" \l "visibility-collapse) 看到 collapse 应该是如何工作的。 | | |
|  |  | | |

其他

|  |  |  |  |
| --- | --- | --- | --- |
| margin 外边距的应用 | 你应该对 margin: auto; 的[这种用法](http://learnlayout.com/margin-auto.html)很熟悉。在伸缩盒中，它也能做同样的事情，但是更加强大。一个 "auto" 的 margin 会合并剩余的空间。它可以用来把伸缩项目挤到其他位置。 | | |
|  | 某个子元素内部内容过宽会撑开父层,通过  Overflow:hidden  或者  Width:0; flex:1;  修改控制 | | |
| demo-auto | <**style**>  .flex-container {  display: flex;  width: 300px;  height: 240px;  }  .flex-item {  width: 20px;  height: 100px;  margin: 5px;  }  .larger{  **/\*margin-right: auto;\*/**  margin: auto; } </**style**> <**div** class="flex-container">  <**div** class="flex-item">flex item 1</**div**>  <**div** class="flex-item larger">flex item 2</**div**>  <**div** class="flex-item">flex item 3</**div**> </**div**> | | 【margin-right:auto;】    【margin:auto;】  > margin-right: auto;导致了所有的剩余空间被合并到那个元素的右边去了；【1】  > margin: auto;实现水平垂直居中；【2】 |
| demo-挤开 | <**style**>  **ul**{  list-style: none;  padding: 0;  }  **ul** {  display: flex;  }  **ul**>**li**{  padding: 5px;  }  **ul** > #login {  margin-left: auto;  } </**style**> <**ul**>  <**li**><**a** href="">关于</**a**>  <**li**><**a** href="">项目</**a**>  <**li**><**a** href="">交互</**a**>  <**li** id='login'><**a** href="">登入</**a**> </**ul**> | | > 应用场景较多 |
| demo-居中 | 使用flex实现绝对居中，设置“margin”值为“auto”值，自动获取伸缩容器中剩余的空间。所以设置垂直方向margin值为“auto”，可以使伸缩项目在伸缩容器的两上轴方向都完全集中。 | | |
| <**style**>  .parent {  display: flex;  height: 300px; **/\* Or whatever \*/** }  .child {  width: 100px; **/\* Or whatever \*/** height: 100px; **/\* Or whatever \*/** margin: auto; **/\* Magic! \*/** } </**style**> <**div** class="parent">  <**div** class="child">  完美居中  </**div**> </**div**> |  | |

|  |  |  |
| --- | --- | --- |
|  |  | |
| demo-居中 | 正如能够动态改变一个容器的尺寸大小，FlexBox对父级容器提供属性支持控制多余的空间分配，也可以对该子级元素进行定位。  justify-content:center; 让伸缩项目在主轴上居中对齐，作用在宽度上；  align-items:center; 让伸缩项目在侧轴上居中对齐，作用在高度上；  通过这两个属性可以实现水平垂直居中效果： | |
| <**style**>  .parent {  display: flex;  height: 200px;  width: 100px;  justify-content: center; **/\*水平居中\*/** align-items: center; **/\*垂直居中\*/** }  .child {  flex: 0;  height: 100px;  } </**style**> <**div** class="parent">  <**div** class="child">1</**div**>  <**div** class="child">2</**div**> </**div**> | .child的flex数值为0所以它不会有什么动态的尺寸变化，他的宽度高度都是父级容器的一半，因此，该子级元素将被定位在水平，垂直方向上的中心位置。 |
|  |  | |
|  |  |  |
| demo-列表均匀展示 | 使用6个列表项，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。我们希望他们能均匀的、很好的分布在水平轴上，就算当我们调整浏览器，他们也依然显示得很好（不使用媒体查询）。 | |
|  | <**style**>  .list {  **/\* 我们第一步要创建一个flex文档流（创建伸缩容器） \*/** display: flex;   **/\* 然后我们定义伸缩流方向，并且可以换行  \* 记得我们要这样设置:  \* flex-direction: row;  \* flex-wrap: wrap;  \*/** flex-flow: row wrap;   **/\* 然后我们定义了如何分配伸缩容器的剩余空间 \*/** justify-content: space-around;  }  .item{  height: 100px;  width: 300px;  background: limegreen;  } </**style**> <**div** class="list">  <**div** class="item">1</**div**>  <**div** class="item">1</**div**>  <**div** class="item">1</**div**>  <**div** class="item">1</**div**>  <**div** class="item">1</**div**> </**div**> | > 不管浏览器宽度如何变化，完美平铺 |
| demo-响应式 | 假设我们网站顶部有一个右对齐的导航，但是我们希望它在小屏幕和小型设备中能单列居中显示。非常简单。 | |
| <**style**>  **/\* Large \*/** .navigation {  display: flex;  flex-flow: row wrap;  **/\* 所有列面向主轴终点位置靠齐 \*/** justify-content: flex-end;  }  **/\* Medium screens \*/  @media** all and (max-width: 800px) {  .navigation {  **/\* 当在中等屏幕中, 导航项目居中显示，并且剩余空间平均分布在列表之间 \*/** justify-content: space-around;  }  }  **/\* Small screens \*/  @media** all and (max-width: 500px) {  .navigation {  **/\* 在小屏幕下, 我们没有足够空间行排列，但我们可以换成列排列 \*/** flex-direction: column;  }  } </**style**> <**div** class="navigation">  <**span**><**a** href="#">Home</**a**></**span**>  <**span**><**a** href="#">About</**a**></**span**>  <**span**><**a** href="#">Products</**a**></**span**>  <**span**><**a** href="#">Contact</**a**></**span**> </**div**> |  |
| demo-响应式 | 更灵活性的伸缩项目：移动先行，3列布局与页眉页脚全屏。和独立的文档顺序。 | |
| <**style**>  .wrapper {  display: flex;  flex-flow: row wrap;  }  **/\* 设置所有标签宽度为100% \*/** .header,  .main,  .nav,  .aside,  .footer {  flex: 1 100%;  }  **/\* 我们利用文档流顺序，考虑移动端先行  \* 在这个例子中的顺序:  \* 1. header  \* 2. nav  \* 3. main  \* 4. aside  \* 5. footer  \*/   /\* Medium screens \*/  @media** all and (min-width: 600px) {  **/\* 两个边栏在同一行 \*/** .aside {  flex: 1 auto;  }  }  **/\* Large screens \*/  @media** all and (min-width: 800px) {  **/\* 设置左边栏在主内容左边  \* 设置主内容区域宽度是其他两个侧边栏宽度的两倍  \*/** .main {  flex: 2 0px;  }  .aside-1 {  order: 1;  }  .main {  order: 2;  }  .aside-2 {  order: 3;  }  .footer {  order: 4;  }  } </**style**> <**div** class="wrapper">  <**header** class="header">Header</**header**>  <**article** class="main">  <**p**>Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum  tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas  semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.</**p**>  </**article**>  <**aside** class="aside aside-1">Aside 1</**aside**>  <**aside** class="aside aside-2">Aside 2</**aside**>  <**footer** class="footer">Footer</**footer**> </**div**> |  |
| demo-等高列表 | 这时一个产品型录的例子，每个商品有品名、相片、描叙、购买按钮。设计师希望每个产品栏有相同的尺寸，相片要在文字叙述之上，而且不管商品的描叙有多长购买按钮要沿着底部对齐。伸缩布局让这个设计的很多部分的实现起来很简单：  > 产品型录可以用伸缩布局，让商品栏水平排列成一行一行，并让每一行的商品栏有相同的高度。每一个商品栏本生是一个竖排的伸缩容器，里面内容垂直排列。  > 在每一个商品栏里，源码里的内容以逻辑顺序排列：品名、叙述、相片，这为语音、非 CSS 浏览器提供了一个恰当的顺序。在一个视觉浏览器上，可以用「[order](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "order)」属性来把相片从内容的后面挪到顶部，并用「[align-self](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "align-self)」将相片水平置中。  > 不管每一个商品的叙述有多高，购买按钮上的[「auto」外边距](https://www.w3.org/html/ig/zh/wiki/Css3-flexbox/zh-hans" \l "auto-margins)让按钮总是在商品栏里置底。 | |
| <**style**>  **/\*外层flex容器: row wrap\*/** #deals {  display: flex; **/\* 应为是伸缩布局每一个商品有同样高度 \*/** flex-flow: row wrap; **/\* 让商品折行 \*/** }  **/\*内层flex容器: column ；外层flex项目 ；\*/** .sale-item {  display: flex; **/\* 用伸缩布局配置每一个商品的描述 \*/** flex-flow: column; **/\* 垂直配置商品的内容 \*/** flex: 2;  }  .sale-item > **img** {  order: -1; **/\* 将图像挪到其他内容之前（视觉顺序） \*/** align-self: center; **/\* 在侧轴（水平向）置中图像 \*/** }  .sale-item > **button** {  margin-top: auto; **/\* 自动顶端外边距让按钮置底 \*/** } </**style**> <**section** id='deals'>  <**section** class='sale-item'>  <**h1**>标题</**h1**>  <**p**>说明  <**ul**>  <**li**>电脑  <**li**>屏幕  <**li**>键盘  <**li**>鼠标  </**ul**>  <**img** src='images/1.jpg' alt=''>  <**button**>下单！</**button**>  </**section**>  <**section** class='sale-item'>  <**h1**>标题 标题标题 标题标题 标题标题 标题</**h1**>  <**p**>说明  <**ul**>  <**li**>电脑  </**ul**>  <**img** src='images/1.jpg' alt=''>  <**button**>下单！</**button**>  </**section**>  <**section** class='sale-item'>  <**h1**>标题 标题标题 标题标题 标题标题 标题</**h1**>  <**p**>说明  <**ul**>  <**li**>电脑  </**ul**>  <**img** src='images/1.jpg' alt=''>  <**button**>下单！</**button**>  </**section**> </**section**> | |
| > 按钮处于底部  > flex嵌套使用 | |
| 其他 | Flexbox Tester 【Understand how to calculate the width of flex items.】  <https://www.madebymike.com.au/demos/flexbox-tester/>  口诀：  <http://weibo.com/1712131295/CoRnElNkZ?ref=collection&type=comment#_rnd1504706629430>  IMG_256IMG_256  IMG_256IMG_256  IMG_256 | |
|  | 关于flex的练习  <http://flexboxfroggy.com/> | |
| 经验-子元素被撑开控制 | 有一个main容器是flex布局，左边一个logo固定宽高，右边content动态宽度。  Content中内容可能会非常长，一些设备下需要隐藏显示，即不换行，并留下省略符…作标记。  .content { flex: 1; width: 0; }  .content { flex: 1; overflow: hidden; } | |

grid布局

|  |  |  |
| --- | --- | --- |
|  | /\*设定columns（列）和rows（行）的大小规格\*/ #grid-example {  display: grid;  grid-rows: 30px 5em auto;  grid-columns: auto 70px 2fr 1fr; } 上面的代码会向我们呈现四行三列的网格。 第一行的高度将会是30px。 第二行的高度将会是5em。 第三行的高度根据剩下的空间大小自动调整 第一列的宽度根据剩下的空间大小自动调整 第二列的宽度是70px 第三列的宽度将会是剩余宽度的三分之二 第四列的宽度讲会是剩余宽度的三分之一  现在我们也可以在我们的网格中为子级定制特别的位置。如果我们希望我们的子级元素位于第一行，第二列，我们就可以这么干，看代码： #griditem1 {   -ms-grid-row: 1;   -ms-grid-column: 2; }  记住，这个跟Flexbox差不多，我们的子级元素需要被定义为网格元素的子级。还有，我们如果希望元素跨行显示，那么我们就可以使用grid-row-span，看代码！ #griditem1 {   -ms-grid-row: 1;   -ms-grid-column: 2;   -ms-grid-row-span: 2; }  而且我们还可以通过使用grid-row-align和grid-colum-align来排列元素。这些属性能够是你定义内容在行列当中居中显示，你还可以在水平或者垂直对其排列元素。举个例子吧，如果要使元素在栏目中居中显示，我们可以这样干： #griditem1 {   -ms-grid-row: 1;   -ms-grid-column: 2;   -ms-grid-column-align: center; } | |
|  | <**style**>  .container {  grid-template-columns:  100px 50px 100px;  grid-template-rows:  80px auto 80px;  grid-column-gap:  10px;  grid-row-gap:  15px;  } </**style**> | IMG_256 |