**CSS基础\_2 (1-15)**

**Flex 布局（弹性布局）**

当元素表现为 flex 框时，它们沿着两个轴来布局：

主轴（main axis）是沿着 flex 元素放置的方向延伸的轴（比如页面上的横向的行、纵向的列）。该轴的开始和结束被称为 **main start** 和 **main end**。

交叉轴（cross axis）是垂直于 flex 元素放置方向的轴。该轴的开始和结束被称为 **cross start** 和 **cross end**。

设置了 display: flex 的父元素被称之为 **flex 容器（flex container）**。

在 flex 容器中表现为柔性的盒子的元素被称之为 **flex 项（flex item）**。

* **列还是行？**

弹性盒子提供了 flex-direction 这样一个属性，它可以指定主轴的方向（弹性盒子子类放置的地方）— 它默认值是 row，这使得它们在按你浏览器的默认语言方向排成一排（在英语/中文浏览器中是从左到右）。

.wrapper {

display: flex;

flex-direction: row;

}

* **换行**

当你在布局中使用定宽或者定高的时候，可能会出现问题即处于容器中的 弹性盒子子元素会溢出，破坏了布局，我们可以设置 flex-wrap: wrap, 让溢出的元素将被移到下一行。

.wrapper {

display: flex;

width: 200px;

flex-wrap: wrap;

}

* **flex: 缩写与全写**

flex 是一个可以指定最多三个不同值的缩写属性：

1. 第一个是无单位比例。可以单独指定全写 flex-grow 属性的值。
2. 第二个无单位比例 — flex-shrink — 一般用于溢出容器的 flex 项。这指定了从每个 flex 项中取出多少溢出量，以阻止它们溢出它们的容器。这是一个相当高级的弹性盒子功能。
3. 第三个是最小值。可以单独指定全写 flex-basis 属性的值。

* **水平和垂直对齐**

**align-items**控制 flex 项在交叉轴上的位置。

1. 默认的值是 stretch，其会使所有 flex 项沿着交叉轴的方向拉伸以填充父容器。如果父容器在交叉轴方向上没有固定宽度（即高度），则所有 flex 项将变得与最长的 flex 项一样长（即高度保持一致）。
2. 你也可以设置诸如 flex-start 或 flex-end 这样使 flex 项在交叉轴的开始或结束处对齐所有的值。

**justify-content**控制 flex 项在主轴上的位置。

1. 默认值是 flex-start，这会使所有 flex 项都位于主轴的开始处。
2. 你也可以用 flex-end 来让 flex 项到结尾处。
3. center 在 justify-content 里也是可用的，可以让 flex 项在主轴居中。
4. 而我们上面用到的值 space-around 是很有用的——它会使所有 flex 项沿着主轴均匀地分布，在任意一端都会留有一点空间。
5. 还有一个值是 space-between，它和 space-around 非常相似，只是它不会在两端留下任何空间。

* **flex 项排序**

弹性盒子也有可以改变 flex 项的布局位置的功能，而不会影响到源顺序（即 dom 树里元素的顺序）。这也是传统布局方式很难做到的一点。我们可以设置 order 进行排序。

1. 所有 flex 项默认的 order 值是 0。
2. order 值大的 flex 项比 order 值小的在显示顺序中更靠后。
3. 相同 order 值的 flex 项按源顺序显示。所以假如你有四个元素，其 order 值分别是 2，1，1 和 0，那么它们的显示顺序就分别是第四，第二，第三，和第一。 可以给 order 设置负值使它们比值为 0 的元素排得更前面。

.wrapper {

display: flex;

}

.wrapper .box1 {

order: 1;

}

.wrapper .box2 {

order: 0;

}

.wrapper .box3 {

order: -1;

}

* **flex 嵌套**

弹性盒子也能创建一些颇为复杂的布局。设置一个元素为 flex 项目，那么他同样成为一个 flex 容器，它的孩子 (直接子节点) 也表现为 flexible box。

<div class="wrapper">

<div class="box1"> </div>

<div class="inner">

<div class="box2"> </div>

</div>

</div>

**Grid 布局（网格布局）**

网格是由一系列水平及垂直的线构成的一种布局模式。根据网格，我们能够将设计元素进行排列，帮助我们设计一系列具有固定位置以及宽度的元素的页面，使我们的网站页面更加统一。一个网格通常具有许多的**列（column）与行（row）**，以及行与行、列与列之间的间隙，这个间隙一般被称为**沟槽（gutter）**。

**在 CSS 中创建自己的网格**

* **定义一个网格**

将容器的display属性设置为grid来定义一个网络。与弹性盒子一样，将父容器改为网格布局后，他的直接子项会变为网格项。

.wrapper {

display: grid;

grid-template-columns: 200px 200px 200px;

}

* **使用 fr 单位的灵活网格**

.wrapper {  
 display: grid;  
 grid-template-columns: 2fr 1fr 1fr;  
}

* **网格间隙**

使用 column-gap 属性来定义列间隙；使用 row-gap 来定义行间隙；使用 gap 可以同时设定两者。

.wrapper {

display: grid;

grid-template-columns: 2fr 1fr 1fr;

gap: 20px;

}

* **重复构建行/列**

你可以使用repeat来重复构建具有某些宽度配置的某些列。

.wrapper {

display: grid;

grid-template-columns: repeat(3, 1fr);

gap: 20px;

}

* **显式网格与隐式网格**

显式网格是我们用 grid-template-columns 或 grid-template-rows 属性创建的。而隐式网格则是当有内容被放到网格外时才会生成的。显式网格与隐式网格的关系与弹性盒子的 main 和 cross 轴的关系有些类似。

隐式网格中生成的行/列大小是参数默认是 auto，大小会根据放入的内容自动调整。当然，你也可以使用 grid-auto-rows 和 grid-auto-columns 属性手动设定隐式网格的大小

.wrapper {

display: grid;

grid-template-columns: repeat(3, 1fr);

grid-auto-rows: 100px;

gap: 20px;

}

* **方便的 minmax() 函数**

minmax 函数为一个行/列的尺寸设置了取值范围。比如设定为 minmax(100px, auto)，那么尺寸就至少为 100 像素，并且如果内容尺寸大于 100 像素则会根据内容自动调整。在这里试一下把 grid-auto-rows 属性设置为minmax 函数。

* **自动使用多列填充**

我们需要让网格自动创建很多列来填满整个容器，我们会用到 repeat 函数中的一个关键字 auto-fill 来替代确定的重复次数。而函数的第二个参数，我们使用 minmax 函数来设定一个行/列的最小值，以及最大值 1fr。

grid-template-columns: repeat(auto-fill, minmax(200px, 1fr))

**基于线的元素放置**

我们根据这些分隔线来放置元素，通过以下属性来指定从那条线开始到哪条线结束。

1. grid-column-start
2. grid-column-end
3. grid-row-start
4. grid-row-end

这些属性的值均为分隔线序号，你也可以用以下缩写形式来同时指定开始与结束的线。

1. grid-column
2. grid-row

开始与结束的线的序号要使用 / 符号分开

.wrapper .box1 {  
 grid-column: 1 / 3;  
 grid-row: 1;  
}

* **使用 grid-template-areas 属性放置元素**

另一种往网格放元素的方式是用 grid-template-areas 属性，并且你要命名一些元素并在属性中使用这些名字作为一个区域。

.wrapper {  
 display: grid;  
 grid-template-areas:"box1 box1""box3 box2""box4 box4";  
 grid-template-columns: 1fr 3fr;  
 gap: 20px;  
}

**定位**

position 属性用于指定一个元素在文档中的定位方式。top，right，bottom 和 left 属性则决定了该元素的最终位置。

**定位类型**

* 定位元素（positioned element）是其计算后位置属性为 relative, absolute, fixed 或 sticky 的一个元素（换句话说，除 static 以外的任何东西）。
* 相对定位元素（relatively positioned element）是计算后位置属性为 relative 的元素。
* 绝对定位元素（absolutely positioned element）是计算后位置属性为 absolute 或 fixed 的元素。
* 粘性定位元素（stickily positioned element）是计算后位置属性为 sticky 的元素。

除了刚刚描述的情况（绝对定位元素填充可用空间）：

* 如果 top 和 bottom 都被指定（严格来说，这里指定的值不能为 auto ），top 优先。
* 如果指定了 left 和 right ，当 direction 设置为 ltr（水平书写的中文、英语）时 left 优先，当 direction 设置为 rtl（阿拉伯语、希伯来语、波斯语由右向左书写）时 right 优先。

**语法**

* static 该关键字指定元素使用正常的布局行为，即元素在文档常规流中当前的布局位置。此时 top, right, bottom, left 和 z-index 属性无效。
* relative 该关键字下，元素先放置在未添加定位时的位置，再在不改变页面布局的前提下调整元素位置（因此会在此元素未添加定位时所在位置留下空白）。position:relative 对 table-\*-group, table-row, table-column, table-cell, table-caption 元素无效。
* absolute 元素会被移出正常文档流，并不为元素预留空间，通过指定元素相对于最近的非 static 定位祖先元素的偏移，来确定元素位置。绝对定位的元素可以设置外边距（margins），且不会与其他边距合并。
* fixed 元素会被移出正常文档流，并不为元素预留空间，而是通过指定元素相对于屏幕视口（viewport）的位置来指定元素位置。元素的位置在屏幕滚动时不会改变。打印时，元素会出现在的每页的固定位置。fixed 属性会创建新的层叠上下文。当元素祖先的 transform, perspective 或 filter 属性非 none 时，容器由视口改为该祖先。
* sticky 元素根据正常文档流进行定位，然后相对它的\*最近滚动祖先（nearest scrolling ancestor）\*和 containing block (最近块级祖先 nearest block-level ancestor)，包括 table-related 元素，基于 top, right, bottom, 和 left 的值进行偏移。偏移值不会影响任何其他元素的位置。 该值总是创建一个新的层叠上下文（stacking context）。注意，一个 sticky 元素会“固定”在离它最近的一个拥有“滚动机制”的祖先上（当该祖先的 overflow 是 hidden, scroll, auto, 或 overlay 时），即便这个祖先不是最近的真实可滚动祖先。这有效地抑制了任何“sticky”行为（详情见 Github issue on W3C CSSWG）。

**z-index**

z-index 属性设定了一个定位元素及其后代元素或 flex 项目的 z-order。当元素之间重叠的时候，z-index 较大的元素会覆盖较小的元素在上层进行显示。

对于一个已经定位的盒子（即其 position 属性值不是 static，这里要注意的是 CSS把元素看作盒子），z-index 属性指定：

* 盒子在当前堆叠上下文中的堆叠层级。
* 盒子是否创建一个本地堆叠上下文。

/\* 字符值 \*/

z-index: auto;

/\* 整数值 \*/

z-index: 0;

z-index: 3;

z-index: 289;

z-index: -1; /\* 使用负值降低优先级 \*/

/\* 全局值 \*/

z-index: inherit;

z-index: initial;

z-index: unset;

* auto 盒子不会创建一个新的本地堆叠上下文。在当前堆叠上下文中生成的盒子的堆叠层级和父级盒子相同。
* （整型数字）是生成的盒子在当前堆叠上下文中的堆叠层级。此盒子也会创建一个堆叠层级为 0 的本地堆叠上下文。这意味着后代（元素）的 z-indexes 不与此元素的外部元素的 z-indexes 进行对比。

*以上内容是总结/摘选自学习网站：https://github.com/WTFAcademy/WTF-CSS。*