CSS 灵活的流(Flow)布局

Sciter

摘要

本部分用于说明**流式(flow)属性**和**弹性(flex)单位**. 使用它们可以创建非常灵活地适用于各种各样的视图 `(view)` 和内容尺寸 `(content sizes)` 的布局。

流式 `(flow)` 属性和弹性 `(flex)` 单位旨在解决目前现下的 CSS 无法解决或实现的一些问题:

- 支持在容器 `(containers)` 和视口 `(viewport)` 中的元素的垂直和水平对齐;
- 支持当替代元素在同一行内且高度相同时，实现灵活的多栏布局（边栏-内容-边栏）。
- 支持定义复杂的位置：当元素的视图顺序 `(visual order)` 与DOM中的元素顺序不同时，可以使用静态布局。

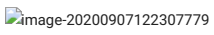
1. 概述

弹性布局 `(Flexible layouts)` 是指使用*弹性* `(flex)` 长度单位和*流* `(flow)` 属性。 弹性长度单位 `(flex units)` 允许使用在定义元素的尺寸 `(size)` 、外边距 `(margin)` 、 内边距 `(padding)` ，作为包含块 `(containing block)` 的剩余可用空间的一部分。弹性单位的值是一个十进制数字，数字后加上”*”(星号)作为单位识别符。

*流(flow)*属性用来定义包含块 `(contained blocks)` 在正常流 `(position:static)` 中的布局方法。或者可以说，*流(flow)* 定义了容器的布局管理器。此模块包含以下标准的布局管理器定义:

1. 垂直布局 `(vertical)`
2. 水平布局 `(horizontal)`
3. 水平布局-允许换行 `(horizontal-flow)`
4. 垂直布局-允许换行 `(vertical-flow)`
5. 模板化布局 `(templated layout)`

弹性(Flex)单元的值可以认为是有弹簧张力的。它根据自己本身的”弹性”来自动调整尺寸和位置。如定义值如下:



上面图片中块的布局的HTML代码如下:

```
<div class="container">
  <p>... some text ...</p>
</body>
```

p元素有以下样式:

```
p
{
  width: 40%;             /* 固定宽度 - 容器的40% */
  margin-left: 2*;        /* 左边的 "弹簧" 的力量为 2 */
  margin-right: 1*;       /* 右边的 "弹簧" 的力量为 1 */
  border:1px solid black; /* 固定宽度的边框 */
}
```

2. 弹性（Flex）长度单位

在弹性单位中，元素的尺寸是对剩余空间内的可用包含框的计算。

弹性单位值是在所有非弹性单位值计算完成后计算的——是布局算法的最后一步。在这一步中，它的值可能来于包含容器的剩余未分配空间。在容器的垂直和水平方向上，所有弹性值在这个剩余未分配空间通过竞争来得出自己的计算值。

弹性单位只适用于元素的CSS属性中的*内边距* `(padding)` 、*外边距* `(margin)` 、*宽度* `(width)` 、*高度* `(height)` 。 它也可以用在静态 `(static正常流)` 、绝对定位 `(absolute)` 的元素中。浮动元素 `(float)` 不支持弹性单位——若为浮动元素指定了弹性单位值时会被当做 `auto` 值。

在计算元素的最终尺寸时，弹性单元的值被解释为一个权重 `(weight)` 。如果剩余空间的弹性值总和小于1，*相应的剩余部分将保持未分配。若果弹性值总和大于或等于1*，所有的剩余空间将使用该弹性值作为比例分配权重来分配。

例如，下面这个示例:

```
#container { width:300px; }
#element { width:1*;
           margin-left:2*;
           margin-right:1*;
           border:2px solid;
           padding:0; }
```

#container元素中的#element元素将会应用下面的这个计算尺寸：

```
弹性值总和    = 1* + 2* + 1*                = 4*;
待分配宽度    = 300px - 2px - 2px           = 296px; // 容器宽度 - 固定边框宽度
宽度(width)  = 1/4 * 待分配宽度(296px)    = 74px;
右外边距      = 1/4 * 待分配宽度(296px)    = 74px;
左外边距      = 2/4 * 待分配宽度(296px)    = 148px;
```

弹性单位值的计算遵循所有的通常约束。 例如, 最小宽度(`min-width`)和最大宽度(`max-width`)定义可以"弹性"的宽度的边界。出于弹性单位计算的原因，初始（默认）的最小宽度属性值被解释为具有最小的内在价值(intrinsic value)。

2.1. 最小内在价值(min-intrinsic), 最大内在价值(max-intrinsic) 长度值。

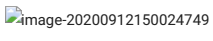
CSS 的最小宽度 `(min-width)` 、最大宽度 `(max-width)` 、宽度(width)和最小高度 `(min-height)` 、最大高度 `(max-height)` 、高度 `(height)` 属性值可以接受以下的指定值： 最小内在价值 `(min-intrinsic)'和最大内在价值` (max-intrinsic)'。

- 最小内在价值 `(min-intrinsic)` —— 在一些容器中，该值指在相应方向上无需溢出 `(overflow)` 渲染时的最小长度。例如：对于设置 `overflow:auto` 的元素，它的最小长度为显示时没有滚动条的最小长度。
- 最大内在价值 `(max-intrinsic)` —— 在一些容器中，该值指所有的子元素显示无需换行时的最小长度。或者说，当某元素的高度为 `max-intrinsic` 时，在水平布局(类似 `*rtl*/*ltr*)` 的系统,或者垂直布局的系统(类似 `*ttb*`)上，最大内在价值 `(max-intrinsic)` 指当该元素的宽度达到最小时的该元素最小高度。

> 若为段落 `<p>first second</p>` 设置 `width:min-intrinsic` 时，该段落的宽度为该段落中最宽的那个单词的宽度。
> 若为这个段落设置 `width:max-intrinsic` 时，则它的宽度为该段落中所有单词和空格长度的总和——该段落将呈现为单行文本。

3. 流(flow)属性

流(flow)属性定义了容器如何布局它的子元素们。或者说，它建立了容器元素的布局管理器。
'flow'



该属性定义了子元素在正常流中的布局方法：

文档中使用的术语：

- *流式容器* `(flow container)`

  流式容器 `(flow container)` 是指包含 `'flow'` 样式属性且值不为 `default` 的块元素。这种元素使用给定的布局管理器来布局它的块子元素。如果流式容器 `(flow container)` 包含 `display` 不为 `block` 、 `list-item` 、 `code>table` 的子元素，则这些子元素会包装在一个匿名块或表容器以便参与流式布

局。

- *流元素(* `flowed element)`

  流元素 `(flowed element)` 的直接父元素为流式容器 `(flow container)` 。这些元素被父流式容器的布局管理器用来替换。

3.1. 垂直布局(flow:vertical)

垂直流比较接近标准的自上而下的块元素的布局方式，例如 `div` ， `ul` 等等。和标准布局方式唯一的区别在于硫元素使用弹性单位时。

`flow:vertical` 容器中的所有静态子元素将会被替换为从上到下、一个接一个的根据容器宽度形成一个单一的列。其所包含的定义了弹性单位的子元素的宽度值将会使用容器的宽度值来计算。同样，被包含元素的垂直尺寸会使用容器的高度来计算。如果容器的高度未定义，或者高度定义为 `height:auto` ，则将没有剩余空间来分配给弹性值，这样的话，在这个方向上的弹性值可以被忽略掉。

如果容器定义了高度，且它的高度大于被包含元素的最小价值高度 `(min-intrinsic)` ，则存在剩余空间。在这样的容器里，这个空间被分配给定义了垂直弹性值的子元素。

例如，下面的样式:

```
#container { height:100%; border:1px dotted; }
#first { margin-bottom:1*; }
```

当定义下面的HTML标记时:

```
<div id="container">
  <h2>Alignment to top/bottom</h2>
  <div id="first" style="margin-bottom:1*">
    <code>margin-bottom:1*;</code>
    <p>Shifts rest to the bottom</p>
  </div>
  <div id="second">
    Normal div
  </div>
</div>
```

则会将#first元素放在 `#container` 的顶部，而将 `#second` 元素放在它的底部。

3.1.1. 在flow:vertical容器中外边距堆叠(collapsing)

在CSS中，被包含元素的垂直编辑通常会叠到一起。 这里唯一需要注意的时当堆叠到垂直边距包含弹性值，而对应的另一个元素是固定值时。这种情况下，这个固定值会作为两个元素间弹性计算值的”最小约束 `(min-constraint)` “。这样，边距是弹性的，但是它不能小于这个固定值。

3.2. 水平布局(flow:horizontal)

这是一个单行布局。 设置 `flow:horizontal` 的容器的所有静态子元素会水平地一个接一个的排列成一行。布局是相对于容器的方向 `direction` 属性进行。

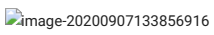
在水平方向上，若子元素的宽度 `(width)` 、左右外边距 `(margin)` 、边框 `(border)` 、内边距 `(padding)` 给定了一个弹性值，它们将参与剩余空间的分配。 `flow:horizontal` 容器的所有直接子元素将对容器内容区 `(content box)` 中左右边界间的剩余空间进行计算，使这个空间根据它们的弹性值进行分配。

在垂直方向上：被包含元素的高度、上下外边距、边框、内边距的弹性值使用容器的高度值来计算。这使 `flow:horizontal` 容器的子元素排序不仅是水平地，也可以是垂直的。

下面的样式中，所有的子元素拥有相同的高度：

```
#container       { flow:horizontal; border-spacing:4px; padding:4px; }
#container > div { margin:0; height:1*; }
```

渲染结果如下:



下面，所有的子元素都设置为最小内在价值高度。而将上外边距设置为 `1*` ：

```
#container       { flow:horizontal; border-spacing:4px; padding:4px; }
#container > div { margin-top:1*; height:auto; }
                              /* height:auto意思是&quot时(<&quot高度 */
```

3.2.1. flow:horizontal容器中的外边距堆叠

`flow:horizontal` 容器中包含块的水平外边距堆叠处理方式和 `flow:vertical` 容器相同。而对于 `in-flow` 子元素，它们不与 `flow:horizontal` 容器的外边距堆叠。

3.2.2. 内在尺寸(Intrinsic dimensions)

`flow:horizontal` 容器的内在高度(`Intrinsic height`)是在它内部的一行中最高的元素的外边距框高度。 `flow:horizontal` 容器的内在宽度是所有子元素在水平边距堆叠的情况下的最小内在宽度之和。

3.3. 水平可换行布局(flow:horizontal-flow)

`flow:horizontal-flow` 布局是 `flow:horizontal` 布局的一种变种。该布局允许容器的子元素在水平方向上没有足够空间时换行。

`clear:left|right|both` 属性可以明确的中断元素布局流，使其成为多行。

在满足以下基本条件之一时，允许换行:

1. 某元素上使用了 `clear:left|right|both` 属性;
2. 该行没有足够的水平工具来放置该元素。

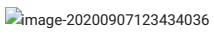
在垂直方向上，设置弹性值的包含块的高度、上下外边距、边框、内边距的值是使用当前*行*的高度。该行的高度等于在不影响弹性值计算的情况下该行中最高元素的高度。

在水平方向上，一行中的元素的弹性值计算和 `flow:horizontal` 布局相同。

例如, 下面的HTML标记语言:

```
<div style="flow:horizontal-flow" >
      <div style="width:100px" >
            width:100px
          </div>
      <div style="width:1*"    >
            flexible width:1*
        flexible width:1*
          </div>
      <div style="width:1*">
        flexible width:1*
        flexible width:1*
        flexible width:1*
        </div>
   <div style="width:150px" >
     width:150px
   </div>
</div>
```

渲染结果如下:



3.4. 垂直可换列布局(flow:vertical-flow)

`flow:vertical-flow` 布局类是一个多列布局，似于 `flow:horizontal-flow` 。在垂直方向上，元素会从上到下的排列放置。如果容器没有足够的垂直空间，元素会换列，变成多列布局。

`clear:left|right|both` 属性允许中断列，使其明确地成为多列布局。

在满足一下条件之一时，会进行换列：

1. 在垂直方向上，该列上所有元素的弹性值之和大于1*；
2. `clear:left|right|both` 属性在某个元素中明确使用；
3. 没有足够的垂直空间来放置下一个元素。

在水平方向上，容器内的子元素的宽度、左右外边距、内边距的弹性值使用当前列的宽度来计算。该行的宽度等于在不影响弹性值计算的情况下该行中最宽元素的宽度。

在垂直方向上，某列中元素的弹性值计算方法与 `flow:vertical` 布局相同。

例如，下面的HTML标记：

```
<ul style="flow:vertical-flow">
  <li style="height:150px" >
    1. height:150px
  </li>
  <li style="height:100px" >
    2. height:100px
  </li>
  <li style="height:0.3*" >
    3. flexible height:0.3*
       flexible height:0.3*
  </li>
  <li style="height:0.7*" >
    4. flexible height:0.7*
       flexible height:0.7*
       flexible height:0.7*
  </li>
  <li style="height:150px" >
    5. height:150px
  </li>
  <li style="height:150px" >
```

```
      6. height:150px
    </li>
    <li style="height:150px" >
      7. height:150px
    </li>
  </ul>
```

每个列表项设置了 `width:150px` ，这会生成下面的布局，列表项会处理成3列：

3.5. 模板布局(flow:”template” / flow: grid(…))

> 请注意，该布局是http://www.w3.org/TR/css3-layout/. 的一个简化版本，该想法的所有版权属于该文档的作者。

`flow: <模板表达式>` 允许根据模板表达式来替换放置元素。

在这里，模板表达式是一个字符串标识序列。每个字符串标识是一个使用空格分隔的名称标识列表中的一项，其中每个标识指定一个网格中的单元格。多列允许有相同的名称。在这种情况下，该标识相当于定义了一个横跨多个单元网格的占位符。

例如, 下面的模板定义了从”a”到”f”的一个3x4的共6个占位网格的表格。某些占位网格跨越了多个单元格：

```
flow: "a a a"
      "b c e"
      "d c e"
      "d c f";
```

容器中的每个子元素使用 `float:"占位标识名称"` 属性绑定到模板定义的特定的占位网格对应的位置：

```
li:nth-child(1) { float:"a"; }
li:nth-child(2) { float:"b";
                  width:150px;
                  height:max-intrinsic; }
li:nth-child(3) { float:"c";
                  width:*; height:*; } /* flexes, a.k.a. shrink-to-fit */
li:nth-child(4) { float:"d";
                  width:150px;
                  height:*; }
li:nth-child(5) { float:"e";
                  width:150px;
                  height:*; }
li:nth-child(6) { float:"f";
                  width:150px;
                  height:150px; }
```

> 注意，在使用该流式(flow)布局的容器内的直接子元素使用 `float:left|right` 将不起作用。或者说，被流式布局的元素只能使用 `flow:"template"` 定义的单元格来布局。

模板容器中的所有没有绑定到网格的子元素将会作为单独的一行追加到最后。如果有多个子元素有相同的占位符名称，只有第一个(DOM中的顺序)将被绑定到占位网格上，剩余的元素会变成未绑定的。

每个占位符名称在模板中必须是唯一的、矩形的。否则，该模板将无效，流式布局将采用默认的 `flow:default` .

例如，下面的HTML标记：

```
<ul>
    <li> "a", width:auto (that is 1*), height:auto(that is max-intrinsic) </li>
```

```
      <li> "b", width:150px, height:max-intrinsic </li>
      <li> "c", width:*, height:* (a.k.a. shrink-to-fit) </li>
      <li> "d", width:150px, height:* </li>
      <li> "e", width:150px, height:* </li>
      <li> "f", width:150, height:150px</li>
   </ul>
```

设置样式后渲染结果如下：

除了用字符定义名称外，还可以使用子元素在模板容器中的顺序数字，所以下面的这个模板：

```
  flow: grid(1 1,

             2 3);
```

会导致容器的前三个子元素布局到两行，且第一个元素放置在第一行，而第二、第三个元素放置在第二行。

3.6. row函数布局(flow: row(tag1, tag2, ...))

`flow:row()` 函数用来实现类似table的布局。 `row()` 函数的参数为元素标签的列表，该列表定义的元素将会放置在表格中的单独一行。

考虑下面的HTML标记:

```
  <dl><dt>第一项</dt>
      <dd>第一项的描述</dd>    <dt>第二项</dt>
      <dd>第二项的描述</dd></dl>
```

并且设置它的样式如下:

```
  dl { flow: row(dt,dd); }
```

则它们将会渲染成如下：

第一项	第一项的描述
第二项	第二项的描述

如果 `flow:row(...)` 布局的元素内存在不匹配row中的模板的元素，则该元素将会被放置在单独的一行并跨越所有列。所有考虑下面的HTML标记：

```
  <dl>
    <header>组别1</header>
      <dt>第一项</dt>
      <dd>第一项的描述</dd>    <dt>第二项</dt>
      <dd>第二项的描述</dd>
    <header>组别2</header>
      <dt>第三项</dt>
      <dd>第三项的描述</dd></dl>
```

依然使用上面的样式，则它们会渲染成：

组别1	
第一项	第一项的描述
第二项	第二项的描述
组别2	

组别1	
第三项	第三项的描述

`flow:row` 的声明可以在一列中接受一个元素列表。如:

```
flow: row(label, input select textarea);
```

定义了两列，第一列放置了 `<label>` 元素，而所有其他的 `<input>,<select>` 和 `<textarea>` 元素被放置在第二列。

3.7. flow: stack

`flow:stack` 布局用于在容器中的任意位置放置元素。渲染的顺序取决于元素的DOM位置或 `z-index` 属性定义的顺序。

在水平和垂直方向上, 被包含元素的 `width、height、margin、padding` 的弹性值使用容器元素的宽度和高度来计算。在弹性计算中，没有子元素都被当做容器元素的唯一子元素来对待 - 子元素的位置不会影响其他子元素的位置。

`flow:stack` 容器元素的内在尺寸等同于容器中子元素外边距盒的最宽的和最高值。

考虑下面的HTML代码:

```
<section tab="标签页一">
   <div>标签页二</div>
   <div>标签页三</div>
</section>
```

它的样式为:

```
section { flow: stack; width: max-content; }
section > div { size:*; /* 跨越整个容器 */
                visibility:hidden; }
section[tab=first] > div:nth-child(1) { visibility:visible; }section[tab=second] > div:nth-child(2) { visibility:visible; }
```

通过修改 `section` 元素的 `tab` 属性值，我们可以切换标签页的显隐。

> 原则上， `flow:stack` 布局在某些方面类似于在 `position:relative` 容器中包含 `position:absolute` 子元素。不过 `flow:stack` 的内在尺寸计算规则是其他CSS属性无法模拟的。

4. *flow*属性、*float*属性 和块格式上下文(block formatting context)

那些 `flow` 属性设置了非默认值的元素的直接子元素，将会建立一个新的块格式上下文，这个上下文类似于表格中的单元格。

5. *flow*属性 与*position*属性

流元素(`flowed element`)是指在流容器中的 `position` 为静态(默认)的子元素们。这意味着那些包含 `position: absolute | fixed` 的子元素会被当做 `position:static` 来处理。

在流元素中 `position:relative` 是被允许的。因此，这些元素可以使用 `left` 、 `right` 、 `bottom` 和 `top` 属性来定义它们相对于静态位置的偏移。

5.1 *position: absolute | fixed* 与 弹性(*Flex*)单位值

弹性单位可以用在定义了 `position:absolute` 或 `position:fixed` 的元素的 `left` 、 `top` 、 `right` 和 `bottom` 属性中，这些元素的内边距(`padding`), 外边距(`margin`), 宽度(`width`)和高度(`height`)中的弹性单位值的计算将会参考包含它们的父块。

例如：下面的样式将会将 `#light-box-dialog` 元素放置在视口 `(viewport)` 的中央：

```
#light-box-dialog
{
  position: fixed;
  left:1*; top:1*; right:1*; bottom:1*;
  width: 400px;
  height: auto;
}
```

`#light-box-dialo` g元素的宽度为400像素，高度为自动(即最小内在高度 `height:min-intrinsic`)，且该元素将会放置在视口中央。

6. *flow*属性与*vertical-align*属性

流元素建立了一个块格式上下文。因此它们的 `vertical-align` 属性定义了它们内部垂直方向上的对齐方式，而不是它们本身在垂直方向上的对齐方式。或者说 `vertical-align` 类似于表格中的单元格。

7. *flow*属性与*border-spacing*属性

`border-spacing` 属性定义了两个流元素在水平和垂直方向上外边距的最小值。如果一个流元素定义了自己的外边距，则外边距使用的值是*border-spacing*定义值和该外边距定义中比较大的那个值。 如果这个外边距值使用了弹性单位值，则该弹性值的计算会将*border-spacing*值作为最小约束值。在这种情况下，计算出的弹性值不能小于 `border-spacing` 属性值。

8. *flow*属性 与 外边距堆叠

不同的布局管理器的对流元素的外边距堆叠处理是不一样的。流容器外边距不会与它内部的硫元素堆叠。

9. *flow*属性和 行内块(*inline-block*)元素

行内块放置在对应的行框中。原则上，行框是可以”弹性”的。

所以, 像 `<img>、<input>、<span style="display:inline-block">` 这些*inline-block*元素可以使用弹性单位来定义它们的尺寸、外边距、内边距。行框上下文的弹性单位计算依据于行框的水平、垂直尺寸，而它们的内容将不是可弹性的。

在水平方向上，行框在分配完所有非弹性内容后可能存在剩余的空间(例如单词框)。这些空间将会在所有指定了弹性值的元素的宽度、左右外边距(或内边距)间分配。

在垂直方向上, 行内块 `(inline-block)` 元素的高度、上下外边距、内边距中的弹性值依据于行框的高度。例如，可以定义多个和行框同高度的子元素。

如果这样定义了， `text-align:justify` 属性的计算将会放置弹性值计算之后。

例如下面的HTML标记:

```
<style>
  p
  {
    padding:4px; border:2px solid black; line-height:1.8em;
  }
  span
  {
    display:inline-block; border: 2px solid salmon;
    background:seashell;
  }
</style>
<p>
   First span:<span style="width:2*">width:2*</span>
   and second one:<span style="width:1*">width:1*</span>
</p>
```

这回到这 `p` 元素各种各样的宽度值：

> 注意，上面的最后一个图像上，span已经达到了它的最小内在宽度(min-intrinsic)，所以它们已经排除掉了相关的弹性值计算。