写给自己看的display: grid布局教程

这篇文章发布于 2018年11月6日，星期二，00:59，归类于 CSS相关。 阅读 10270 次, 今日 32 次 20 条评论

一、前言&索引

给 `<div>` 这类块状元素元素设置 `display:grid` 或者给 `<span>` 这类内联元素设置 `display:inline-grid`，Grid布局即创建！例如：

```
div {
    display: grid;
}
```

此时该div就是“grid容器”，其子元素称为“grid子项”。

//zxx: grid和inline-grid区别在于，inline-grid容器为inline特性，因此可以和图片文字一行显示；grid容器保持块状特性，宽度默认100%，不和内联元素一行显示。

在Grid布局中，所有相关CSS属性正好分为两拨，一拨作用在grid容器上，还有一拨作用在grid子项上。具体参见下表，点击可快速索引。

作用在grid容器上	作用在grid子项上
• grid-template-columns • grid-template-rows • grid-template-areas • grid-template • grid-column-gap • grid-row-gap • grid-gap • justify-items • align-items • place-items • justify-content • align-content • place-content • grid-auto-columns • grid-auto-rows • grid-auto-flow • grid	• grid-column-start • grid-column-end • grid-row-start • grid-row-end • grid-column • grid-row • grid-area • justify-self • align-self • place-self

Grid布局相关CSS属性虽然很多，但是实际上都不难理解，难的是这些属性不太容易记住，需要多多实战手写才能信手拈来。

Grid布局是一个二维的布局方法，纵横两个方向总是同时存在。其中的很多布局概念跟中国农田的布局是完全匹配的。

因此，在我看来，Grid布局就像是“分田种地”。故事是这样的，张老板是个程序员，省吃俭用攒了点小钱，然后老家因为城镇化建设，农村都没什么人，土地都荒废在那里，于是就承包了一块地，打算养养鱼，种种果树。承包的地方很挺大，如何划分土地就成了问题，于是张老板打算借助Grid布局来划分。

二、作用在grid容器上的CSS属性

1. grid-template-columns和grid-template-rows

这两个CSS属性用来对田地进行基本的划分，columns是列的意思，表示竖直方向的划分；rows是行的意思，表示水平方向的划分。现实世界中，农田的布局构造一般下面两种：

1. 田地A-田地B，下面是土地C-土地D，就是“田”这个字的构造，只不过4块地之间分隔就是个不能走路的小沟沟，宽度可以忽略不计。

2. 田地A-田垄-田地B，下面是土地C-田垄-土地D，也是“田”这个字的构造，只不过4块地之间分隔是个可以走路的田垄，有的地方也叫土埂。

这里的划分语法就和上面的农田划分一致，如下：

```
.container {
  grid-template-columns: <track-size> ... | <line-name> <track-size> ...;
  grid-template-rows: <track-size> ... | <line-name> <track-size> ...;
}
```

用中文表示就是：

```
.container {
  grid-template-columns: <田地> ... 或 <田垄> <田地> ...;
  grid-template-rows: <田地> ... 或 <田垄> <田地> ...;
}
```

也就是：

- **<track-size>**：划分田地的尺寸。可以是长度值，百分比值，以及fr单位（网格剩余空间比例单位）。
- **<line-name>**：中间用来走路的田垄的名字，可以任意命名。

看一个简单例子：

```
.container {
    grid-template-columns: 80px auto 100px;
    grid-template-rows: 25% 100px auto 60px;
}
```

`grid-template-columns`后面3个值，表示分为了3列，从左往右每列的尺寸分别是`80px`，auto（自动）和`100px`；
`grid-template-rows`属性值含4个值，表示分为了4行，从上往下每行的高度分别是`25%`，`100px`，auto（自动）和`60px`。

实时效果如下所示：

宽80px
高25%

宽auto
高25%

宽100px
高25%

宽80px
高100px

宽auto
高100px

宽100px
高100px

宽80px
高auto

宽auto
高auto

我们还可以给“田垄”，也就是网格分隔线进行命名，语法是使用`[]`包裹我们自定义的命名，可以是中文，例如：

```
.container {
    grid-template-columns: [第一根纵线] 80px [纵线2] auto [纵线3] 100px [最后的结束线];
    grid-template-rows: [第一行开始] 25% [第一行结束] 100px [行3] auto [行4] 60px [行末];
}
```

实时效果如下，选中对应网格线的名称可以高亮其位置：

为何要给网格线命名呢?

Grid布局的好比街道划分，田地划分，这中间分隔的线通常就是道路或者田垄，如果我们不给这些道路起个名字，回头想要描述某片区域的时候就不好描述。比方说：

南京东路有起外滩即中山东一路，西至西藏中路。

因为我们给道路命了名称，因此，我们在描述某个区域的时候，就好描述，别人也好辨认。但如果没有命名，而是下面这样描述：

南京东路东起靠近黄浦江第1条路，靠近黄浦江第8条路。

得，这个区域描述就有问题，万一哪天封路，或者新建了条路，岂不就混乱了?

也就是说，给Grid布局中的分隔线命名，为的就是可以更好地对区域进行描述。如果我们没有描述某片区域的的需求，自然也不需要命名了。

如何命名

由于网格中中间区域的网格线是两边格子公用的，就像道路有两边，因此，我们起名字的时候可以起两个名称（使用空格分隔），分别表示两侧。例如：

```
.container {
    grid-template-columns: [第一根纵线] 80px [第1根纵线结束 第2根纵线开始] 100px [最后的结束线];
}
```

repeat语法

有时候，我们网格的划分是很规律的，例如，基于`40px`创建栅格，要是我们布局宽度`960px`，岂不是要写24次`40px`，实在套啰嗦了，此时，就可以使用`repeat()`语法，如下示意：

```
.container {
    grid-template-columns: repeat(24, 40px [col-start]);
}
```

等同于：

```
.container {
    grid-template-columns: 40px [col-start], 40px [col-start], /* ...省略20个...*/, 40px [col-start], 40px [col-start];
}
```

fr单位是什么？

`fr` 是单词fraction的缩写，表示分数。

- 先从简单例子看起：

  ```
  .container {
      grid-template-columns: 1fr 1fr 1fr;
  }
  ```

  1:1:1，网格宽度三等分，实时效果如下：

  宽1fr
  占据1/3

  宽1fr
  占据1/3

  宽1fr
  占据1/3

- 如果有固定尺寸值，则划分剩余空间大小，例如：

  ```
  .container {
      grid-template-columns: 200px 1fr 1fr 1fr;
  }
  ```

  4列，后面3列宽度是grid容器宽度减去200像素后的1/3大小，实时效果如下：

  宽200px

  宽1fr

  宽1fr

  宽1fr

- 如果和auto混用会如何呢？

  ```
  .container {
      grid-template-columns: auto 1fr 1fr 1fr;
  }
  ```

  宽auto

  宽1fr

  宽1fr

  宽1fr

从上面效果可以看出，当有设置 `fr` 尺寸的时候， `auto` 的尺寸表现为“包裹”，为内容宽度。如果没有设置 `fr` 尺寸的网格，则表现为拉伸。

- 如果 `fr` 数值之和小于1又当如何？

宽1fr

```
.container {
    grid-template-columns: auto 0.25fr .25fr .25fr;
}
```

宽auto

宽0.25fr

宽0.25fr

这里计算就相对复杂些，首先，由于第一个网格尺寸设置为 `auto`，因此 `fr` 计算需要的剩余空间尺寸是grid容器的宽度减去“宽auto”这几个字符的宽度。所以，后面3个 `0.25fr` 元素的宽度是：`(容器宽度 - “宽auto”字符宽度) * 0.25`。然后剩余尺寸就是第一个网格宽度。

2. grid-template-areas

area是区域的意思，`grid-template-areas` 就是给我们的网格划分区域的，就好像张老板承包的土地划分不同区域养殖不同的农作物或者水产品。

语法如下：

```
.container {
  grid-template-areas:
    "<grid-area-name> | . | none | ..."
    "...";
}
```

其中：

grid-area-name

对应网格区域的名称。

.

表示空的网格单元格。

none

没有定义网格区域。

我们还是通过案例了解这个CSS属性。张老板承包了一块地，然后划分成了3*4共12个小格子，然后张老板希望最上面3个格子种葡萄，最下面3个格子种西瓜，中间6个格子，左边2个养龙虾，右边4个养鱼。如下图示意：

则对应CSS代码如下：

```
.container {
    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 1fr 1fr 1fr 1fr;
    grid-template-areas:
        "葡萄 葡萄 葡萄"
        "龙虾 养鱼 养鱼"
        "龙虾 养鱼 养鱼"
        "西瓜 西瓜 西瓜";
}
```

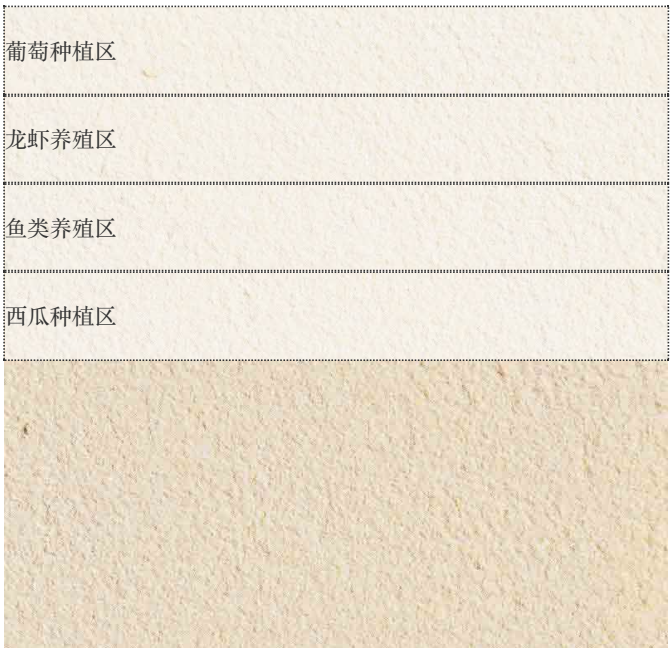
12个格子，四片区域，因此，我们grid子项只需要4个元素即可，HTML示意如下：

```
<div class="container">
    <div class="putao"></div>
    <div class="longxia"></div>
    <div class="yangyu"></div>
    <div class="xigua"></div>
</div>
```

此时grid子项只要使用 `grid-area` 属性指定其隶属于那个区域就可以了（支持中文区域名称）：

```
.putao { grid-area: 葡萄; }
.longxia { grid-area: 龙虾; }
.yangyu { grid-area: 养鱼; }
.xigua { grid-area: 西瓜; }
```

实时Grid布局效果如下：



注意：如果我们给网格区域命了名，但是没有给网格线命名，则会自动根据网格区域名称生成网格线名称，规则是区域名称后面加 `-start` 和 `-end` 。例如，某网格区域名称是“葡萄”，则左侧column线名称就是“葡萄-start”，左侧column线名称就是“葡萄-end”。

以及，我们的网格区域一定要形成规整的矩形区域，什么L形，凹的或凸的形状都是不支持的，会认为是无效的属性值。

//zxx: 实际开发的时候，葡萄种植区就是头部区域，龙虾就是侧边栏区域，鱼类养殖区就是主区域，西瓜种植区就是底部区域。

3. grid-template

`grid-template` 是 `grid-template-rows` ， `grid-template-columns` 和 `grid-template-areas` 属性的缩写。

语法如下：

```
.container {
    grid-template: none;
}
```

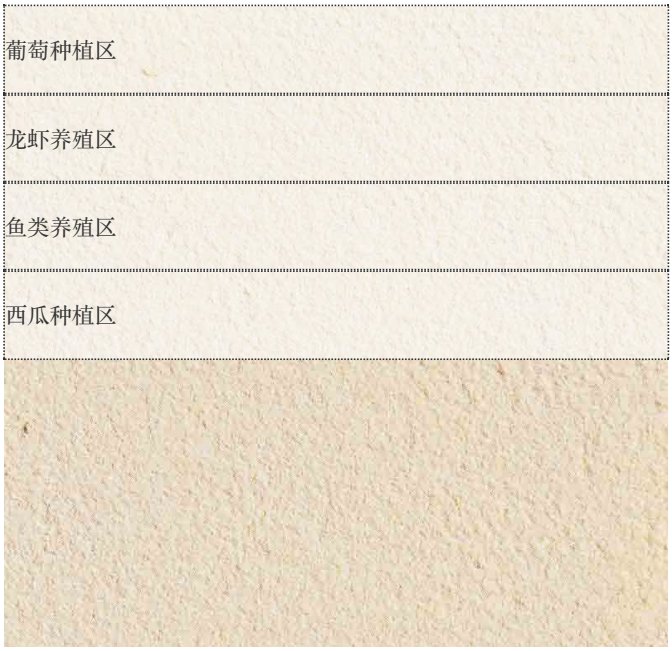
```
.container {
    grid-template: <grid-template-rows> / <grid-template-columns>;
}
```

其中 `none` 表示将3个CSS属性都设置为初始值。

举个例子，前面张老板养殖区划分，用 `grid-template` 缩写表示就是：

```
.container {
    grid-template:
        "葡萄 葡萄 葡萄" 1fr
        "龙虾 养鱼 养鱼" 1fr
        "龙虾 养鱼 养鱼" 1fr
        "西瓜 西瓜 西瓜" 1fr
        /1fr 1fr 1fr;
}
```

实时效果如下：



由于 `grid-template` 不会重置一些隐式的grid属性（如grid-auto-columns ， `grid-auto-rows` 和 `grid-auto-flow` ），因此，大多数时候，还是推荐使用 `grid` 代替 `grid-template` 。

4. grid-column-gap和grid-row-gap

`grid-column-gap` 和 `grid-row-gap` 属性用来定义网格中网格间隙的尺寸。你可以理解成田地之间走路的田垄宽度。

语法如下：

```
.container {
  grid-column-gap: <line-size>;
  grid-row-gap: <line-size>;
}
```

其中：

<line-size>

网格间的间隙尺寸。

实例说话，给定一个简单的2×2网格，设置水平网格间隙`10px`，垂直方向`15px`，如下：

```
.container {
    grid-template-columns: 2fr 1fr;
    grid-template-rows: 1fr 2fr;
    grid-column-gap: 10px;
    grid-row-gap: 15px;
}
```

浏览器实时布局渲染如下：

5. grid-gap

CSS `grid-gap`属性是`grid-column-gap`和`grid-row-gap`属性的缩写。语法如下：

```
.container {
    grid-gap: <grid-row-gap> <grid-column-gap>;
}
```

先横row后竖column，这个比较好记忆，古语有云：“横竖都是死”，先横后竖，网格的间隙就像是汉字“田”中间的那个“十”，按照汉字书写，先横后竖，就记住了。

例如，上面的2×2网格间隙案例也可以写作：

```
.container {
    grid-template-columns: 2fr 1fr;
    grid-template-rows: 1fr 2fr;
    grid-gap: 15px 10px;
}
```

效果一样的，这里就不占据篇幅重复示意了。

6. justify-items

`justify-items`指定了网格元素的水平呈现方式，是水平拉伸显示，还是左中右对齐，语法如下：

```
.container {
    justify-items: stretch | start | end | center;
}
```

其中：

stretch

默认值，拉伸。表现为水平填充。

start

表现为网格水平尺寸收缩为内容大小，同时沿着网格线左侧对齐显示（假设文档流方向没有变）。

end

表现为网格水平尺寸收缩为内容大小，同时沿着网格线右侧对齐显示（假设文档流方向没有变）。

center

表现为网格水平尺寸收缩为内容大小，同时在当前网格区域内部水平居中对齐显示（假设文档流方向没有变）。

各个属性值实时效果如下（点击单选框体验不同属性值布局效果）：

7. align-items

`align-items` 指定了网格元素的垂直呈现方式，是垂直拉伸显示，还是上中下对齐，语法如下：

```
.container {
    align-items: stretch | start | end | center;
}
```

其中（假设文档流方向为网页默认）：

stretch

默认值，拉伸。表现为垂直填充。

start

表现为网格垂直尺寸收缩为内容大小，同时沿着上网格线对齐显示。

end

表现为网格垂直尺寸收缩为内容大小，同时沿着下网格线对齐显示。

center

表现为网格垂直尺寸收缩为内容大小，同时在当前网格区域内部垂直居中对齐显示。

各个属性值实时效果如下（点击单选框体验不同属性值布局效果）：

8. place-items

`place-items` 可以让 `align-items` 和 `justify-items` 属性写在单个声明中。语法如下：

```
.container {
    place-items: <align-items> / <justify-items>;
}
```

这里顺序是 `align-items` 在前， `justify-items` 在后。首字母a,j，a,j，a,j，口中不断重复，有没有发现跟angelababy发音很像，没错，记住angelababy我们也就记住这里的顺序了。又或者有句古话，叫做“合纵连横”，这种网格对齐，就有“合纵连横”的意味在里面，纵在前，横在后，也可以方便我们记忆。

据说Edge15之前版本不支持 `place-items` 属性（自己未实测），因此，如果有兼容性顾虑，建议还是分开书写。

9. justify-content

`justify-content` 指定了网格元素的水平分布方式。此属性仅在网格总宽度小于grid容器宽度时候有效果。例如，我们网格设定的都是固定的宽度值，结果还有剩余空间。例如：

```
.container {
    display: grid;
    width: 300px;
    grid-template: 100px 100px/100px 100px;
}
```

此时，水平和垂直方向都有100px的剩余， `justify-content` 属性此时就有用武之地了。

语法如下：

```
justify-content: stretch | start | end | center | space-between | space-around | space-eve
nly;
```

其中：

stretch

默认值。拉伸，宽度填满grid容器，拉伸效果需要网格目标尺寸设为auto时候才有效，如果定死了宽度，则无法拉伸。

start

默认值。逻辑CSS属性值，与文档流方向相关。默认表现为左对齐。

end

逻辑CSS属性值，与文档流方向相关。默认表现为右对齐。

center

表现为居中对齐。

space-between

表现为两端对齐。between是中间的意思，意思是多余的空白间距只在元素中间区域分配。使用抽象图形示意如下：

space-around

around是环绕的意思，意思是每个flex子项两侧都环绕互不干扰的等宽的空白间距，最终视觉上边缘两侧的空白只有中间空白宽度一半。使用抽象图形示意如下：

space-evenly

evenly是匀称、平等的意思。也就是视觉上，每个flex子项两侧空白间距完全相等。使用抽象图形示意如下：

眼见为实，点击下面对应单复选框，可以看到实时的布局效果：

上面案例和下面案例中的grid布局相关CSS都是：

```
.container {
    grid-template: auto auto/auto auto;
}
```

10. align-content

`align-content`可以看成和`justify-content`是相似且对立的属性，`justify-content`指明水平方向grid子项的分布方式，而`align-content`则是指明垂直方向每一行grid元素的分布方式。如果所有grid子项只有一行，则`align-content`属性是没有任何效果的。

语法如下：

```
align-content: stretch | start | end | center | space-between | space-around | space-evenl
y;
```

其中：

stretch

默认值。每一行flex子元素都等比例拉伸。例如，如果共两行flex子元素，则每一行拉伸高度是50%。

start

逻辑CSS属性值，与文档流方向相关。默认表现为顶部堆砌。

end

逻辑CSS属性值，与文档流方向相关。默认表现为底部堆放。

center

表现为整体垂直居中对齐。

space-between

表现为上下两行两端对齐。剩下每一行元素等分剩余空间。

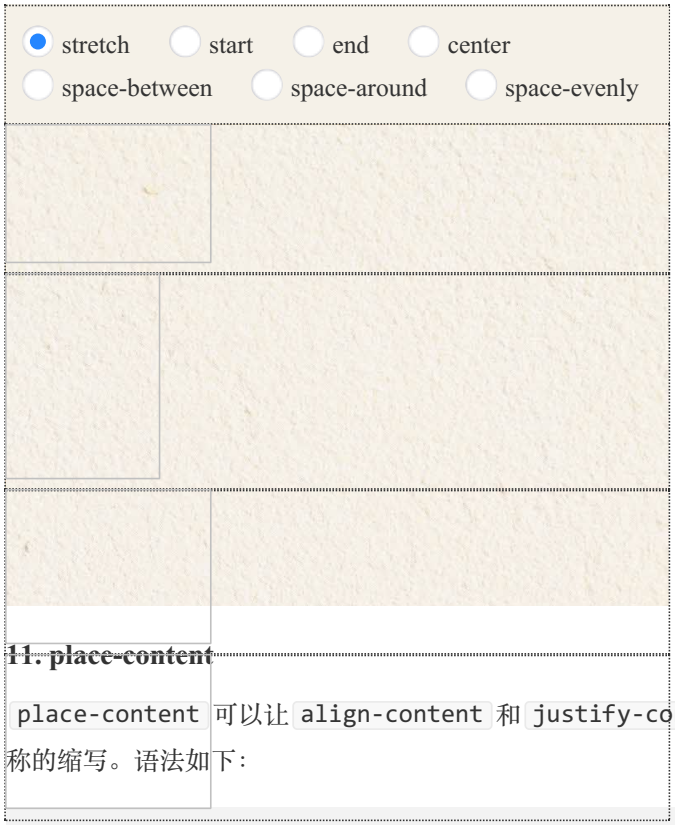
space-around

每一行元素上下都享有独立不重叠的空白空间。

space-evenly

每一行元素都完全上下等分。

眼见为实，我们给flex容器设置高度500像素，然后点击下面对应单选框，可以看到实时的布局效果：



11. place-content

`place-content` 可以让 `align-content` 和 `justify-content` 属性写在一个CSS声明中，也就是俗称的缩写。语法如下：

```
.container {
    place-items: <align-content> / <justify-content>;
}
```

这里顺序是 `align-content` 在前， `justify-content` 在后。首字母a,j，a,j，读个几遍，是不是和angelababy发音一致，记住angelababy就记住这里的顺序了。又或者有句古话，叫做“合纵连横”，这种网格分布，就有“合纵连横”的意味在里面，纵在前，横在后，也可以方便我们记忆。

据说Edge15及其之前版本尚不支持 `place-content` 属性（自己未实测），因此，如果有兼容性顾虑，建议还是分开书写。

12. grid-auto-columns和grid-auto-rows

指定任何自动生成的网格轨道（也称为隐式网格轨道）的大小。 当网格项目多于网格中的单元格或网格项目放置在显式网格之外时，将创建隐式轨道。

用张老板承包土地的案例解释就是：

1. 土地划分，计划分成16块区域搞农业，材料都买好了，结果发现承包的土地只能放下12块区域，多的4块怎么办呢？就在承包土地外面种点东西，不要浪费。
2. 土地划分，计划上面种葡萄，底部种西瓜。但是，种植的时候搞错了，西瓜种到了承包区域之外。

上面这两种情况都是因为各种原因在自己土地之外也种了东西。如果张老板想要对不在自己土地上的种植区域也进行尺寸规划，该怎么办？此时就需要用到`grid-auto-columns`和`grid-auto-rows`属性，就是应付这种场景的。

//zxx: 在Grid布局中，这些非正常网格称为“隐式网格”，在规定容器内显示的称之为“显式网格”。

语法如下：

```
.container {
    grid-auto-columns: <track-size> ...;
    grid-auto-rows: <track-size> ...;
}
```

其中：

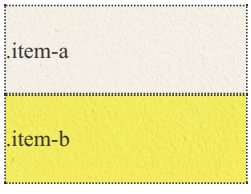
<track-size>

划分田地的尺寸。可以是长度值，百分比值，以及fr单位（网格剩余空间比例单位）。

我们通过一个实例来感受下`grid-auto-columns`和`grid-auto-rows`属性的样式表现。CSS如下：

```
.container {
    display: grid;
    width: 150px;
    grid-template-columns: 60px 60px;
    grid-template-rows: 30px 90px;
    grid-auto-columns: 60px;
}
.item-a {
    grid-column: 1 / 2;
    grid-row: 2 / 3;
}
.item-b {
    /* 容器水平只有2个格子，但这里设定的是第3个，隐式网格创建 */
    grid-column: 3 / 4;
    grid-row: 2 / 3;
    background-color: rgba(255,255,0, .5);
}
```

实时效果如下，`.item-b`宽度强制表现为了`60px`，否则，则表现为`auto`，在这里，则是可怜巴巴填满剩余的`30px`：



13. grid-auto-flow

`grid-auto-flow`属性控制没有明确指定位置的grid子项的放置方式。比方说定义了一个5*2的10格子，共有5个元素，其中2个元素指定了放在哪个格子里，还有3个则自生自灭排列。此时，这3个元素如何排列就是由`grid-auto-flow`属性控制的。

语法如下：

```
.container {
  grid-auto-flow: row | column | row dense | column dense
}
```

其中：

row

默认值。没有指定位置的网格依次水平排列优先。

column

没有指定位置的网格依次垂直排列优先。

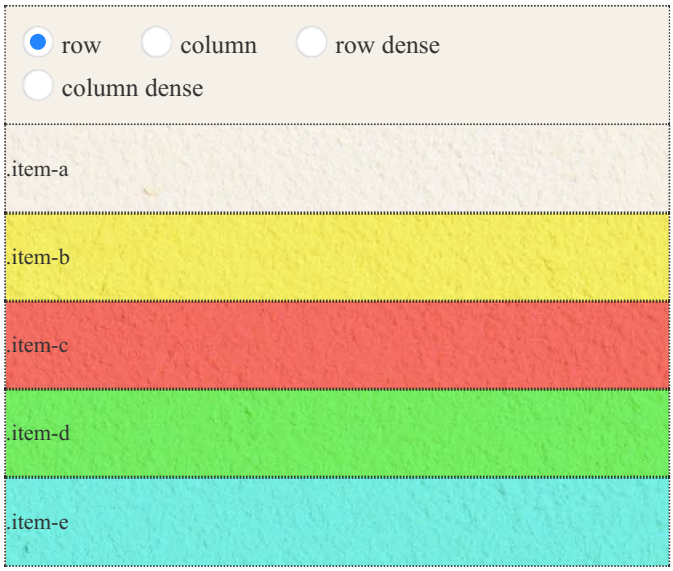
dense

dense这个英文是稠密的意思。如果有设置，则表示自动排列启用“密集”打包算法。如果稍后出现的网格比较小，则尝试看看前面有没有合适的地方放置，使网格尽可能稠密紧凑。此属性值仅仅改变视觉顺序，会导致DOM属性和实际呈现顺序不符合，这对于可访问性是不友好的，建议谨慎使用。

实例说话，已知CSS如下：

```
.container{grid-template: 1fr 1fr 1fr/1fr 2fr 2fr 1fr 2fr;}
.item-a { grid-column: 1; grid-row: 2 / 4; }
.item-b { grid-row: 1 / 3; }
.item-c {}
.item-d {}
.item-e {}
```

也就是`.item-a.item-b`水平位置固定，点击下面单选项，体验布局变化。



- 选中`row`，水平排列，此时`.item-c`高度足够放在左上角那个网格中，因此，视觉顺序是c, b, d, e。
- 选中`column`，垂直排列，此时`.item-c`宽度不足够放在左上角那个网格中，因此，视觉顺序（

先上下后左右）是b, c, d, e。

- 选中`row dense`，水平排列，同时前面有空就钻。视觉顺序同`row`属性。
- 选中`column dense`，垂直排列，此时`.item-c`放置在左上角那个网格中，因此，视觉顺序（先上下后左右）是c, b, d, e，b和d垂直排列。

14. grid

是下面所有这些CSS属性的缩写集合，`grid-template-rows`，`grid-template-columns`，`grid-template-areas`，`grid-auto-rows`，`grid-auto-columns`和`grid-auto-flow`。

语法如下：

- ```
  grid: none
  ```

  `none`表示设置所有的子属性为初始值。

- ```
  grid: <grid-template>
  ```

  和`grid-template`用法一致。例如这样：

  ```
  .container {
      grid: 100px 300px / 3fr 1fr;
  }
  ```

  等同于下面：

  ```
  .container {
      grid-template-rows: 100px 300px;
      grid-template-columns: 3fr 1fr;
  }
  ```

- ```
  grid: <grid-template-rows> / [ auto-flow && dense? ] <grid-auto-columns>?
  ```

  问号`?`表示0或1，可有可无的意思。也就是`dense`关键字和`grid-auto-columns`值都可以省略。

  具体说明：

  - `auto-flow && dense?`其实就是`grid-auto-flow`属性的值，等同于`row`或`column`或`row dense`或`column dense`。

    但这里`row`和`column`这两个关键字却使用了`auto-flow`这一个关键字代替了。那岂不有问题：什么时候解析成`row`，什么时候解析成`column`呢？

    原来，是根据`auto-flow`关键字是在斜杠的左侧还是右侧决定的。如果`auto-flow`关键字在斜杠左侧，则解析为`row`，如果是在右侧，则解析为`column`。这里的语法是在斜杠的右侧，因此，会将`grid-auto-flow`解析为`column`。

  - `<grid-auto-columns>`后面有个问号`?`，因此是可以省略的，如果省略，则将`grid-auto-columns`解析为`auto`。

  我们通过几个案例学习这里的语法：
  ```

```
.container {
    grid: 100px 300px / auto-flow 200px;
}
```

上面CSS代码省略了`dense`关键字，启用了`<grid-auto-columns>`，因此，等同于下面CSS：

```
.container {
    grid-template-rows: 100px 300px;
    grid-auto-flow: column;
    grid-auto-columns: 200px;
}
```

记住，在Grid布局中，斜杠前面都是rows相关属性，斜杠后面都是columns相关属性（下同）。

- `grid: [ auto-flow && dense? ] <grid-auto-rows>? / <grid-template-columns>`

此语法和上面一个语法类似，只是这个斜杠前面是隐式网格，后面是显示。在这里，由于`auto-flow`在斜杠左侧，因此解析为`row`。所以：

```
.container {
    grid: auto-flow dense 100px / 1fr 2fr;
}
```

就等同于下面CSS：

```
.container {
    grid-auto-flow: row dense;
    grid-auto-rows: 100px;
    grid-template-columns: 1fr 2fr;
}
```

grid属性缩写语法第一次学习会有些懵逼，乍一看，头都大了，各种非字母以外的符号，天书一样。其实`&`，`[`，`]`，`?`这些符号实际书写都不参与的，仅仅用来表示逻辑。

好吧，送佛送到西，我最后再给大家梳理下：

- `grid:none`好简单好舒爽，没什么好说的。
- 如果网格布局中规中矩，没有哪个格子跑到grid容器外面，就是`grid-template`属性。
- 最后两个语法完全是出现了grid容器外面格子时候才使用的，要么`grid-template/auto-flow`，要么`auto-flow/grid-template`，就这么简单。

说穿了，其实也没啥，一开始使用不熟练是很正常的，你多实践几次，多手写几次，很快就能掌握，成为Grid布局小能手了。

三、作用在grid子项上的CSS属性

1. grid-column-start, grid-column-end, grid-row-start和grid-row-end

表示grid子项所占据的区域的起始和终止位置，包括水平方向和垂直方向。

就好比张老板养鱼，这鱼塘东边从什么地方开始，往西到什么地方，南边是哪里，北面又到何处，都要说清楚，这样，这鱼池的面积和位置也就出来了。

语法如下：

```
.item {
    grid-column-start: <number> | <name> | span <number> | span <name> | auto
    grid-column-end: <number> | <name> | span <number> | span <name> | auto
    grid-row-start: <number> | <name> | span <number> | span <name> | auto
    grid-row-end: <number> | <name> | span <number> | span <name> | auto
}
```

语法中的管道分隔符`|`表示“或者”的意思，所以别看上面好长，实际上就一个属性值，具体来讲：

<number>

起止与第几条网格线。

<name>

自定义的网格线的名称。

span <number>

表示当前网格会自动跨越指定的网格数量。

span <name>

表示当前网格会自动扩展，直到命中指定的网格线名称。

auto

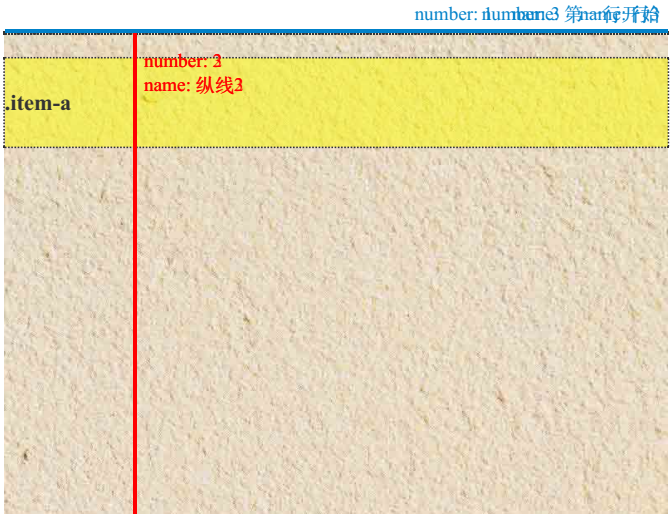
全自动，包括定位，跨度等。

看例子说话，CSS和HTML如下：

```
.container {
    grid-template-columns: [第一根纵线] 80px [纵线2] auto [纵线3] 100px [最后的结束线];
    grid-template-rows: [第一行开始] 25% [第一行结束] 100px [行3] auto [行末];
}
.item-a {
    grid-column-start: 2;
    grid-column-end: 纵线3;
    grid-row-start: 第一行开始;
    grid-row-end: 3;
}
```

```
<div class="container">
    <div class="item-a"></div>
</div>
```

实时效果如下：



每根网格线都有内置的`<number>`，从1开始计数，上面Grid布局为3×3的九宫格，因此，水平和垂直都

是4条网格线（含边缘），从左往右4条线 `<number>` 值依次是 `1-4` ，垂直方向从上往下也是类似。

再本例中，所有网格线都命名了中文名称，例如“第一根纵线”，就是最左边竖直网格线。因此，最终效果也就不难理解了——

`grid-column-start:2` 表示 `.item-a` 网格左侧起始于 `<number>` 为 `2` 的线；
`grid-column-end:纵线3` 表示 `.item-a` 网格右侧结束于 `<name>` 为 `纵线3` 的线；
`grid-row-start:第一行开始` 表示 `.item-a` 网格上方开始于 `<name>` 为 `第一行开始` 的线；
`grid-row-end:3` 表示 `.item-a` 网格下方结束于 `<number>` 为 `3` 的线。

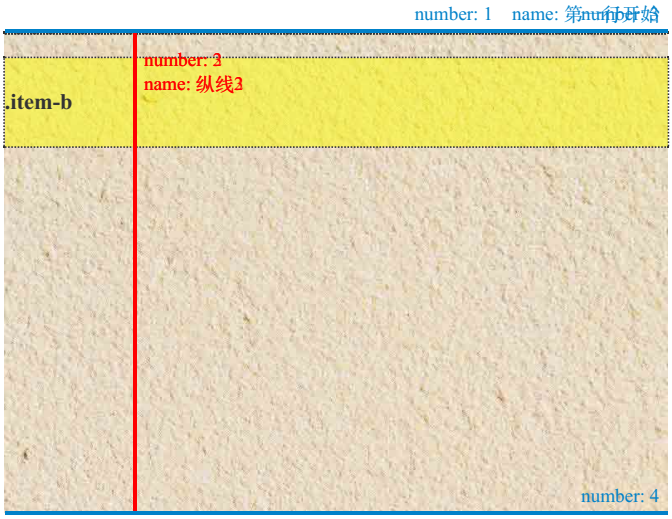
span作用表现

下面我们再来看下 `span` 关键字的作用。如下CSS和HTML：

```
.item-b {
    grid-column-start: 2;
    grid-column-end: span 纵线3;
    grid-row-start: 第一行开始;
    grid-row-end: span 3;
}
```

```
<div class="container">
    <div class="item-b"></div>
</div>
```

效果为：



对于命名的网格线，有 `span` 和没有 `span` 没有区别（包括多个同名网格线），但是，对于数值网格线，则可以看出差异，有 `span` 则表示跨越的个数，而非网格线的序号。例如这里 `grid-row-end:span 3` 表示当前网格需要覆盖3个格子。于是，我们可以看到 `.item-b` 高度贯穿整个grid容器。

2. grid-column和grid-row

`grid-column` 和 `grid-row` 都是缩写啦，前者是 `grid-column-start` + `grid-column-end` 的缩写，后者是 `grid-row-start` + `grid-row-end` 的缩写。

语法上是使用斜杠分隔，如下：

```
.item {
    grid-column: <start-line> / <end-line> | <start-line> / span <value>;
    grid-row: <start-line> / <end-line> | <start-line> / span <value>;
}
```

语法中的管道分隔符 `|` 表示“或者”的意思。然后 `<start-line>` 就是 `grid-*-start` 属性值，`<end-line>` 就是 `grid-*-end` 属性值。例如：

```
.item-b {
    grid-column: 2 / span 纵线3;
    grid-row: 第一行开始 / span 3;
}
```

等同于：

```
.item-b {
    grid-column-start: 2;
    grid-column-end: span 纵线3;
    grid-row-start: 第一行开始;
    grid-row-end: span 3;
}
```

3. grid-area

`grid-area` 表示当前网格所占用的区域。在介绍 `grid-template-areas` 属性的时候就演示过该属性，我们使用 `grid-template-areas` 属性自定义一些网格区域，然后使用 `grid-area` 属性让grid子项指定使用这些区域，就自动进行了区域分布。

`grid-area` 和 `grid-column/grid-row` 作用都是grid子项的分布，但 `grid-area` 语义要更好，识别度更佳，非常适合具有功能属性的布局区域（如头部，底部），同时，还支持非规则区域。

语法如下：

```
.item {
    grid-area: <name> | <row-start> / <column-start> / <row-end> / <column-end>;
}
```

其中：

<name>

区域名称。由 `grid-template-areas` 属性创建。

<row-start> / <column-start> / <row-end> / <column-end>

占据网格区域的纵横起始位置。

`<name>` 属性值使用参见上面张老板划分鱼塘案例。我们这里演示下后面的基于位置的区域划分，如下代码：

```
.container {
    grid-template: 1fr 1fr 1fr/1fr 1fr 1fr 1fr;
}
.item-c {
    grid-area: 1 / 2 / 3 / 4;
}
```

表示水平网格线位置起止分别是1,3，垂直起止网格线位置是2,4。于是得到一个2×2大小的区域。

4. justify-self

`justify-self` 表示单个网格元素的水平对齐方式。语法如下：

```
.item {
    justify-self: stretch | start | end | center;
}
```

其中（假设文档流方向没有变）：

stretch

默认值，拉伸。表现为水平填充。

start

表现为网格水平尺寸收缩为内容大小，同时沿着网格线左侧对齐显示。

end

表现为网格水平尺寸收缩为内容大小，同时沿着网格线右侧对齐显示。

center

表现为网格水平尺寸收缩为内容大小，同时在当前网格区域内部水平居中对齐显示。

各个属性值实时效果如下（点击单选框体验不同属性值布局效果）：

5. align-self

`align-self` 指定了网格元素的垂直呈现方式，是垂直拉伸显示，还是上中下对齐，语法如下：

```
.container {
    align-self: stretch | start | end | center;
}
```

其中（假设文档流方向为网页默认）：

stretch

默认值，拉伸。表现为垂直填充。

start

表现为网格垂直尺寸收缩为内容大小，同时沿着上网格线对齐显示。

end

表现为网格垂直尺寸收缩为内容大小，同时沿着下网格线对齐显示。

center

表现为网格垂直尺寸收缩为内容大小，同时在当前网格区域内部垂直居中对齐显示。

各个属性值实时效果如下（点击单选框体验不同属性值布局效果）：

6. place-self

`place-items` 可以让 `align-self` 和 `justify-self` 属性写在单个声明中。语法如下：

```
.container {
    place-items: <align-self> / <justify-self>;
}
```

这里顺序是 `align-self` 在前， `justify-self` 在后。首字母a,j，a,j，a,j，口中不断重复，有没有发现跟angelababy发音很像，没错，记住angelababy我们也就记住这里的顺序了。

据说Edge15及之前版本尚不支持 `place-self` 属性（自己未实测），因此，如果有兼容性顾虑，建议还是分开书写。

四、其他Grid知识点

- 在Grid布局中， `float` ， `display:inline-block` ， `display:table-cell` ， `vertical-align` 以及 `column-*` 这些属性和声明对grid子项是没有任何作用的。这个可以说是Grid布局中的常识，面试经常会问的，一定要记得。
- Grid布局则适用于更大规模的布局（二维布局），而Flexbox布局最适合应用程序的组件和小规模布局（一维布局），关Flex布局请参见“写给自己看的display: flex布局教程”一文。
- 命名虽然支持中文，但由于CSS文件中文存在乱码的风险，所以……创新还是保守就看大家自己的抉择了。
- IE10-IE15虽然名义上支持Grid布局，但支持的是老版本语法（本文是介绍的全是2.0全新语法），还需要加 `-ms-` 私有前缀，精力原因，IE下的使用并未深究，以后有机会再补充。

另外：

本文所有水平和垂直，左侧与右侧这类方位的描述均是在网页的水平和垂直流都是默认方向前提下的表述。

本教程优点在于交互效果可以实时体验，更直观。如果是转载文章，必定没有效果，访问原文即可。

Grid布局自己之前并未在实际项目中使用过，本文内容也是边学边写，文中若有表述不准确的地方欢迎指正。

感谢阅读！

参考文章：A Complete Guide to Grid

> 《CSS世界》签名版独家发售，包邮，可指定寄语，点击显示购买码

（本篇完） // 想要打赏？点击这里。有话要说？点击这里。

« 写给自己看的display: flex布局教程

小tips: 如何HTML标签和JS中设置CSS3 var变量 »

猜你喜欢

- 写给自己看的display: flex布局教程
- 粉丝群第1期CSS小测点评与答疑
- 我熟知的三种三栏网页宽度自适应布局方法
- css行高line-height的一些深入理解及应用
- 中国古代道家思想与网页重构的思考
- CSS+JavaScript实现页面不同布局的切换
- 纯CSS实现侧边栏/分栏高度自动相等
- CSS样式分离之再分离
- 准确理解CSS clear:left/right的含义及实际用途
- CSS3 Media Queries的些野史外传
- 页面可用性之浏览器默认字体与CSS中文字体

标签： align-content, align-items, align-self, css3, css相关, display:grid, grid, grid-area, grid-auto-columns, grid-auto-flow, grid-auto-rows, grid-gap, grid-template, grid-template-areas, justify-content, justify-items, justify-self, 布局, 网格布局

发表评论（目前20条评论）

名称 (必须)

邮件地址(不会被公开) (必须)

网站

提交评论

1. **郝冰说道：**
2019年01月17日 11:22

grid-gap的疑问 如果每个网格之间的间隙不一样 那怎设置?

回复

2. **seasonley说道：**
2018年12月31日 00:37

有个可视化的，不用记那么多属性的方案
A PostCSS plugin to keep CSS grids stupidly simple
https://github.com/sylvainpolletvillard/postcss-grid-kiss

回复

3. **emmhhh说道：**
2018年12月18日 16:14

place-items的a,j a,j，直接记成品牌AJ就好啦哈哈哈

回复

4. **lzh说道：**
2018年12月14日 14:27

css就服你

回复

5. **心满说道：**
2018年12月3日 15:11

期待已久

回复

6. **夜晚硬邦邦说道：**
2018年11月14日 17:23

发现个Grid布局的在线游戏http://cssgridgarden.com/

回复

> **M说道：**
> 2018年11月19日 15:21
>
> 挺欢乐的小游戏
>
> 回复

7. **DemonQ说道：**
2018年11月13日 18:22

写了一个动态的 grid 代码生成， https://qishaoxuan.github.io/css_tricks/grid/#column-row-gap

回复

8. **chenzesam说道：**
2018年11月12日 14:58

我觉得可以

回复

9. **cshenger说道：**
2018年11月10日 12:22

有一回看过一个python的图形库，它用的布局就和grid很像，我想css的grid布局大概也是借鉴了其他软件的思想吧

回复

10. **孙国强说道：**
2018年11月9日 13:40

感觉 grid 布局 也许可以取代table布局

回复

11. **杭州吴彦祖说道：**
2018年11月8日 11:01

学了N次了，就这次看懂了。 百度有些人写的grid教程简直就是增加学习难度的。

回复

12. **fox说道：**
2018年11月7日 14:59

除了grid的框架感，其他感觉和flex大同小异

回复

> **郝冰说道：**
> 2019年01月10日 14:19
>
> 同感
>
> 回复

13. **CC说道：**
2018年11月6日 17:50

哇!!!!總算等到大神寫了~~~~!!
趕緊筆記下來
最近真的常看到更新

回复

14. **sd说道：**
2018年11月6日 13:52

板凳

回复

sd说道：
2018年11月6日 15:50

例如，某网格区域名称是“葡萄”，则左侧column线名称就是“葡萄-start”，左侧column线名称就是“葡萄-end”。
大哥是不是写错字了。

回复

15. **pliybird说道：**
2018年11月6日 09:41

板凳

回复

16. **XboxYan说道：**
2018年11月6日 09:15

沙发

回复

17. **曲双如说道：**
2018年11月6日 08:44

看一遍觉得好清楚,早上好，这大半夜的写文章。

回复

最新文章

» 常见的CSS图形绘制合集
» 粉丝群第1期CSS小测点评与答疑
» 分享三个纯CSS实现26个英文字母的案例
» 小tips: 纯CSS实现打字动画效果
» CSS/CSS3 box-decoration-break属性简介
» CSS :placeholder-shown伪类实现Material Design占位符交互效果
» 从天猫某活动视频不必要的3次请求说起
» CSS vector-effect与SVG stroke描边缩放
» CSS ::backdrop伪元素是干嘛用的?
» 周知：CSS -webkit-伪元素选择器不再导致整行无效

今日热门

» 常见的CSS图形绘制合集 (178)
» 粉丝群第1期CSS小测点评与答疑 (113)
» 未来必热：SVG Sprite技术介绍 (111)
» HTML5终极备忘大全（图片版+文字版） (85)
» 让所有浏览器支持HTML5 video视频标签 (83)
» Selectivizr-让IE6~8支持CSS3伪类和属性选择器 (80)
» CSS3下的147个颜色名称及对应颜色值 (78)
» 小tips: 纯CSS实现打字动画效果 (72)
» 写给自己看的display: flex布局教程 (69)
» 分享三个纯CSS实现26个英文字母的案例 (69)

今年热议

» 《CSS世界》女主角诚寻靠谱一起奋斗之人 (76)
» 不借助Echarts等图形框架原生JS快速实现折线图效果 (64)
» 看，for..in和for..of在那里吵架！ (60)
» 是时候好好安利下LuLu UI框架了！ (47)
» 原来浏览器原生支持JS Base64编码解码 (35)
» 妙法攻略：渐变虚框及边框滚动动画的纯CSS实现 (33)
» 炫酷H5中序列图片视频化播放的高性能实现 (31)
» CSS scroll-behavior和JS scrollIntoView让页面滚动平滑 (30)

» windows系统下批量删除OS X系统.DS_Store文件 (26)
» 写给自己看的display: flex布局教程 (26)

猜你喜欢

- 写给自己看的display: flex布局教程
- 粉丝群第1期CSS小测点评与答疑
- 我熟知的三种三栏网页宽度自适应布局方法
- css行高line-height的一些深入理解及应用
- 中国古代道家思想与网页重构的思考
- CSS+JavaScript实现页面不同布局的切换
- 纯CSS实现侧边栏/分栏高度自动相等
- CSS样式分离之再分离
- 准确理解CSS clear:left/right的含义及实际用途
- CSS3 Media Queries的些野史外传
- 页面可用性之浏览器默认字体与CSS中文字体