## 重要术语

#### Grid Container

设置了 display: gird 的元素。 这是所有grid item的直接父项。 在下面的例子中，.container就是是 grid container。

<div **class**="container">

<div class="item item-1"></div>

<div **class**="item item-2"></div>

<div **class**="item item-3"></div>

</div>

#### Grid Item

Grid 容器的孩子（直接子元素）。下面的 .item 元素就是 grid item，但 .sub-item不是。

<**div** class="container">

<**div** class="item"></**div**>

<**div** class="item">

<p class="sub-item"></p>

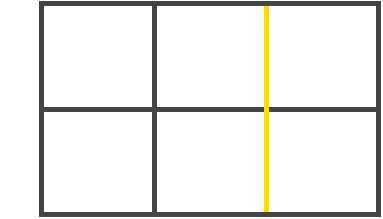
</**div**>

<**div** class="item"></**div**>

</**div**>

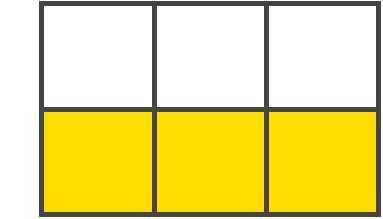
#### Grid Line

这个分界线组成网格结构。 它们既可以是垂直的（“column grid lines”），也可以是水平的（“row grid lines”），并位于行或列的任一侧。 下面例中的黄线就是一个列网格线。



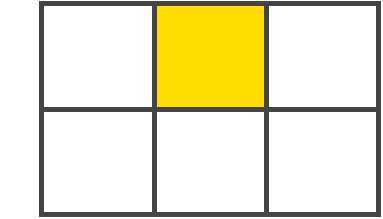
#### Grid Track

两个相邻网格线之间的空间。 你可以把它们想象成网格的列或行。 下面是第二行和第三行网格线之间的网格轨道。

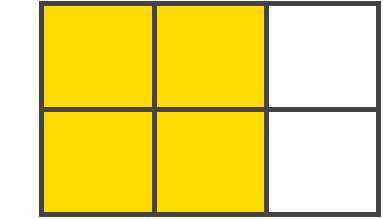


#### Grid Cell

两个相邻的行和两个相邻的列网格线之间的空间。它是网格的一个“单元”。 下面是行网格线1和2之间以及列网格线2和3的网格单元。



#### Grid Area

四个网格线包围的总空间。 网格区域可以由任意数量的网格单元组成。 下面是行网格线1和3以及列网格线1和3之间的网格区域。

#### Grid 属性列表

Grid Container 的全部属性

* display
* grid-template-columns
* grid-template-rows
* grid-template-areas
* grid-template
* grid-column-gap
* grid-row-gap
* grid-gap
* justify-items
* align-items
* justify-content
* align-content
* grid-auto-columns
* grid-auto-rows
* grid-auto-flow
* grid

Grid Items 的全部属性

* grid-column-start
* grid-column-end
* grid-row-start
* grid-row-end
* grid-column
* grid-row
* grid-area
* justify-self
* align-self

## 父容器(Grid Container)的属性

#### display

将元素定义为 grid contaienr，并为其内容建立新的网格格式化上下文(grid formatting context)。

值:

* grid - 生成一个块级(block-level)网格
* inline-grid - 生成一个行级(inline-level)网格

？grid和inline-grid有什么区别

* subgrid - 如果你的 grid container 本身就是一个 grid item（即,嵌套网格），你可以使用这个属性来表示你想从它的父节点获取它的行/列的大小，而不是指定它自己的大小。

.container {

display: grid | inline-grid | subgrid;

}

注意：columns, float, clear, 以及 vertical-align 对一个 grid container 没有影响。

columns：css3属性，规定列的宽度和列数：

**columns.html**

columns: *column-width* *column-count*;

eg:column*s: "100px 3"*

|  |  |
| --- | --- |
| **值** | **描述** |
| *column-width* | 列的宽度。 |
| *column-count* | 列数。 |

#### grid-template-columns / grid-template-rows

使用以空格分隔的多个值来定义网格的列和行。这些值表示轨道大小(track size)，它们之间的空格代表表格线(grid line)。

.**container** {

**grid**-template-columns: <**track**-**size**> ... | <line-name> <**track**-**size**> ...;

**grid**-template-rows: <**track**-**size**> ... | <line-name> <**track**-**size**> ...;

}

例子:

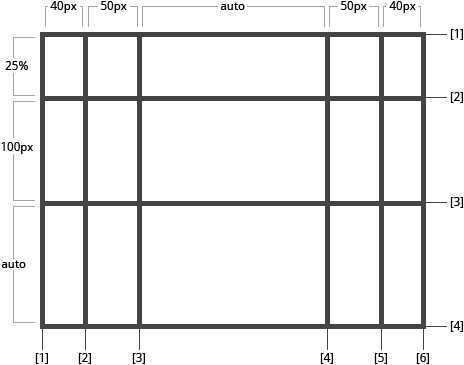
(如果未显示的给网格线命名)，轨道值之间仅仅有空格时，网格线会被自动分配数字名称：

.container {

grid-template-columns: 40px 50px auto 50px 40px;

grid-template-rows: 25% 100px auto;

}



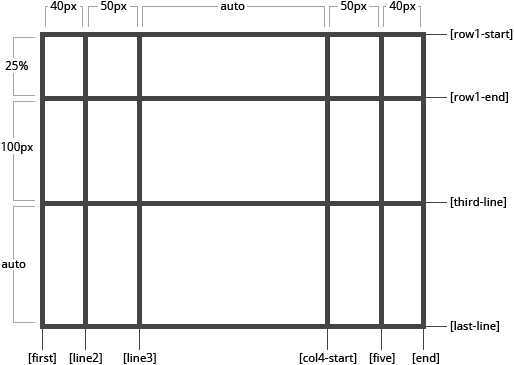
但你可以给网格线指定确切的命名。 注意中括号里的网格线命名语法：

.**container** {

grid-template-columns: [first] 40px [line2] 50px [line3] auto [col4-start] 50px [five] 40px [end];

grid-template-rows: [row1-start] 25% [row1-end] 100px [third-line] auto [last-line];

}



需要注意的是，一个网格线可以有不止一个名字。例如，这里第2条网格线有两个名字：row1-end 和 row2-start：

.container {

grid-template-rows: [row1-start] 25% [row1-end row2-start] 25% [row2-end];

}

如果你的定义中包含重复的部分，则可以使用repeat() 符号来简化写法：

.container {

grid-template-columns: repeat(3, 20px [col-start]) 5%;

}

上面的写法和下面等价：

.container {

grid-template-columns: 20px [col-start] 20px [col-start] 20px [col-start] 5%;

}

“fr”单位允许您将轨道大小设置为网格容器自由空间的一部分。 例如，下面的代码会将每个 grid item 为 grid container 宽度的三分之一：

.container {

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

}

自由空间是在排除所有不可伸缩的 grid item 之后计算得到的。 在下面的示例中，fr单位可用的自由空间总量不包括50px:

.container {

grid-template-columns: 1fr 50px 1fr 1fr;

}

#### grid-template-areas

通过引用 grid-area属性指定的网格区域的名称来定义网格模板。 重复网格区域的名称导致内容扩展到这些单元格。 点号表示一个空单元格。 语法本身提供了网格结构的可视化。

值：

* <grid-area-name> - 使用 grid-area 属性设置的网格区域的名称
* . - 点号代表一个空网格单元
* none - 没有定义网格区域

举例：

.item-a {

grid-area: header;

}

.item-b {

grid-area: main;

}

.item-c {

grid-area: sidebar;

}

.item-d {

grid-area: footer;

}

.container {

grid-template-columns: 50px 50px 50px 50px;

grid-template-rows: auto;

grid-template-areas:

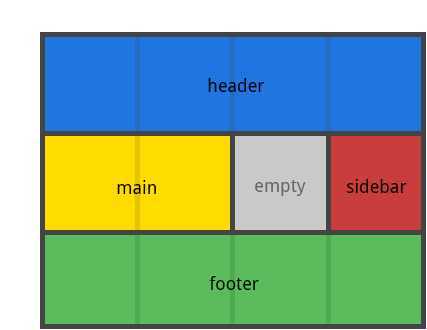
"header header header header"

"main main . sidebar"

"footer footer footer footer";

}

这将创建一个四列宽三行高的网格。 整个第一行将由 header 区域组成。 中间一行将由两个 main 区域、一个空单元格和一个 sidebar 区域组成。 最后一行是footer区域组成。



你的声明中的每一行都需要有相同数量的单元格。

您可以使用任意数量的相邻的.来声明单个空单元格。 只要这些点号之间没有空格，他们就代表了一个单一的单元格。

需要注意的是你不是在用这个语法命名网格线，而是在命名区域。 当你使用这种语法时，区域两端的网格线实际上是自动命名的。 比如，如果网格区域的名称是foo，那么区域的起始的行网格线和列网格线名称是 foo-start，并且区域终点的行网格线和列网格线名称是 foo-end。 这意味着某些网格线可能有多个名称，比如上面的例子中最左边的一条网格线有三个名字：header-start，main-start 和 footer-start。

#### grid-template

在单个声明中定义 grid-template-rows、grid-template-columns、grid-template-areas 的简写。

值：

* none - 将三个属性都设置为其初始值
* subgrid - 把 grid-template-rows 和 grid-template-columns 设置为 subgrid, 并且 grid-template-areas 设置为初始值
* grid-template-rows / <grid-template-columns - 把 grid-template-columns 和 grid-template-rows 设置为指定值, 与此同时, 设置 grid-template-areas 为 none

.container {

grid-template: none | subgrid | <grid-template-rows> / <grid-template-columns>;

}

它也可以使用一个更复杂但相当方便的语法来指定这三个值。 一个例子：

.container {

grid-**template**:

[row1-start] "header header header" 25px [row1-end]

[row2-start] "footer footer footer" 25px [row2-end]

/ **auto** 50px **auto**;

}

以上等价于：

.container {

grid-template-rows: [row1-start] 25px [row1-end row2-start] 25px [row2-end];

grid-template-columns: auto 50px auto;

grid-template-areas:

"header header header"

"footer footer footer";

}

由于 grid-template 不会重置隐式网格属性（grid-auto-columns，grid-auto-rows和grid-auto-flow），而这可能是大多数情况下你想要做的。因此建议使用grid属性来代替grid-template。

#### grid-column-gap / grid-row-gap

指定网格线的大小，你可以把它想象为设置列/行之间的间距的宽度。

值：

* line-size - 一个长度值

.container {

grid-column-gap: <line-size>;

grid-row-gap: <line-size>;

}

举例：

.container {

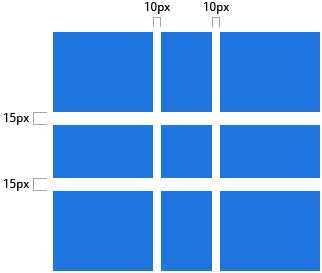
grid-template-columns: 100px 50px 100px;

grid-template-rows: 80px auto 80px;

grid-column-gap: 10px;

grid-row-gap: 15px;

}



只能在列/行之间创建缝隙，而不是在外部边缘创建。

#### grid-gap

grid-row-gap 和 grid-column-gap 的缩写

.container {

grid-gap: <grid-row-gap> <grid-column-gap>;

}

Example:

.container {

grid-template-columns: 100px 50px 100px;

grid-template-rows: 80px auto 80px;

grid-gap: 10px 15px;

}

如果没有指定 grid-row-gap，则会被设置为与 grid-column-gap 相同的值。

#### justify-items

沿着行轴对齐网格内的内容（与之对应的是 align-items, 即沿着列轴对齐），该值适用于容器内的所有的 grid items。

值：

* start: 内容与网格区域的左端对齐
* end: 内容与网格区域的右端对齐
* center: 内容位于网格区域的中间位置
* stretch: 内容宽度占据整个网格区域空间(这是默认值)

.container {

justify-items: start | end | center | stretch;

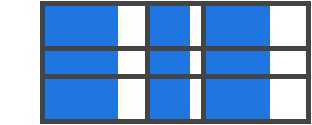
}

举例：

.container {

justify-items: start;

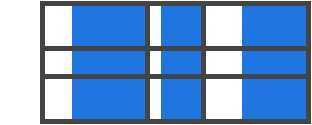
}



.container{

justify-items: end;

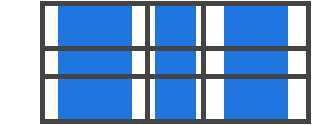
}



.container {

justify-items: center;

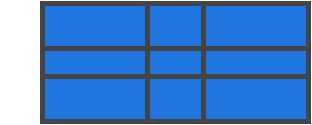
}



.container {

justify-items: stretch;

}



也可以通过给单个 grid item 设置justify-self属性来达到上述效果。

#### align-items

沿着列轴对齐grid item 里的内容（与之对应的是使用 justify-items 设置沿着行轴对齐），该值适用于容器内的所有 grid items。

值：

* start: 内容与网格区域的顶端对齐
* end: 内容与网格区域的底部对齐
* center: 内容位于网格区域的垂直中心位置
* stretch: 内容高度占据整个网格区域空间(这是默认值)

.container {

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

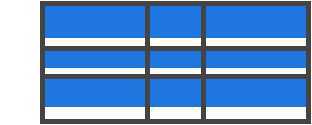
}

举例:

.container {

align-items: start;

}



.container {

align-items: end;

}



.container {

align-items: center;

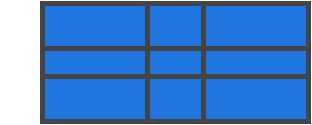
}



.container {

align-items: stretch;

}



也可以通过给单个 grid item 设置align-self属性来达到上述效果。

#### justify-content

有时，网格的总大小可能小于其网格容器的大小。如果你的所有 grid items 都使用像px这样的非弹性单位来设置大小，则可能发生这种情况。此时，你可以设置网格容器内的网格的对齐方式。 此属性沿着行轴对齐网格（与之对应的是 align-content, 沿着列轴对齐）。

值：

* start - 网格与网格容器的左边对齐
* end - 网格与网格容器的右边对齐
* center - 网格与网格容器的中间对齐
* stretch - 调整g rid item 的大小，让宽度填充整个网格容器
* space-around - 在 grid item 之间设置均等宽度的空白间隙，其外边缘间隙大小为中间空白间隙宽度的一半
* space-between - 在 grid item 之间设置均等宽度空白间隙，其外边缘无间隙
* space-evenly - 在每个 grid item 之间设置均等宽度的空白间隙，包括外边缘

.container {

justify-content: start | **end** | center | stretch | space-**around** | space-**between** | space-evenly;

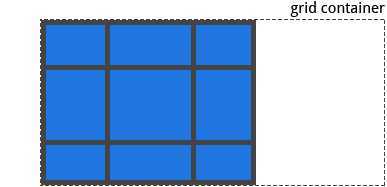
}

举例：

.container {

justify-content: start;

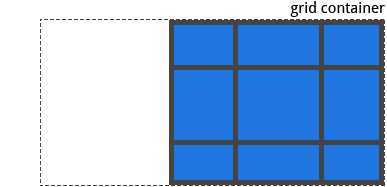
}



.container {

justify-content: end;

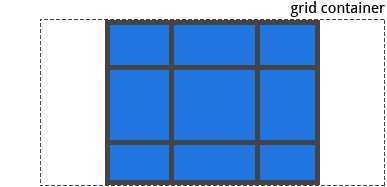
}



.container {

justify-content: center;

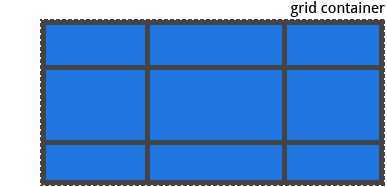
}



.container {

justify-content: stretch;

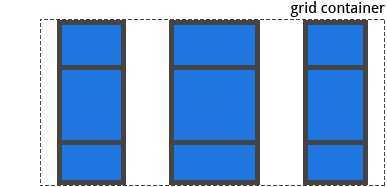
}



.container {

justify-content: space-around;

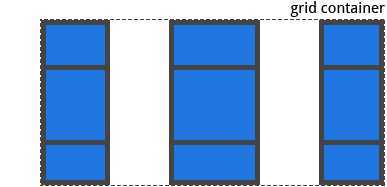
}



.container {

justify-content: space-between;

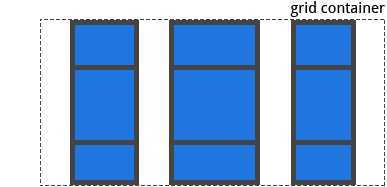
}



.container {

justify-content: space-evenly;

}



#### align-content

有时，网格的总大小可能小于其网格容器的大小。如果你的所有 grid items 都使用像px这样的非弹性单位来设置大小，则可能发生这种情况。此时，你可以设置网格容器内的网格的对齐方式。 此属性沿着列轴对齐网格（与之对应的是 justify-content, 即沿着行轴对齐）。

值：

* start - 网格与网格容器的顶部对齐
* end - 网格与网格容器的底部对齐
* center - 网格与网格容器的中间对齐
* stretch - 调整 grid item 的大小，让高度填充整个网格容器
* space-around - 在 grid item 之间设置均等宽度的空白间隙，其外边缘间隙大小为中间空白间隙宽度的一半
* space-between - 在 grid item 之间设置均等宽度空白间隙，其外边缘无间隙
* space-evenly - 在每个 grid item 之间设置均等宽度的空白间隙，包括外边缘

.container {

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

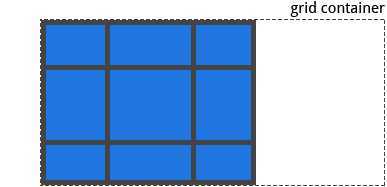
}

举例：

.container {

align-content: start;

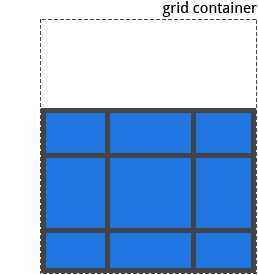
}



.container {

align-content: end;

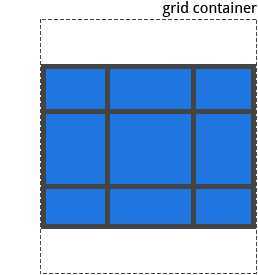
}



.container {

align-content: center;

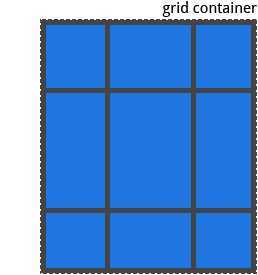
}



.container {

align-content: stretch;

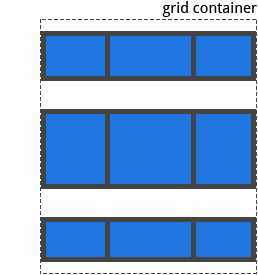
}



.container {

align-content: space-around;

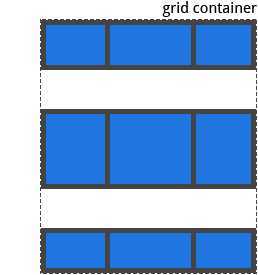
}



.container {

align-content: space-between;

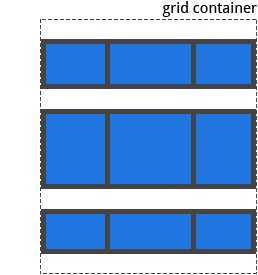
}



.container {

align-content: space-evenly;

}



#### grid-auto-columns / grid-auto-rows

指定自动生成的网格轨道（又名隐式网格轨道）的大小。 隐式网格轨道在你显式的定位超出指定网格范围的行或列（使用 grid-template-rows/grid-template-columns）时被创建。

值：

* <track-size> - 可以是一个长度值，一个百分比值，或者一个自由空间的一部分（使用 fr 单位）

.container {

grid-auto-columns: <track-size> ...;

grid-auto-rows: <track-size> ...;

}

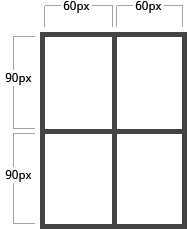
为了说明如何创建隐式网格轨道，思考如下代码：

.container {

grid-template-columns: 60px 60px;

grid-template-rows: 90px 90px

}



这里创建了一个 2x2的网格。

但是，现在想象一下，使用 grid-column 和 grid-row 来定位你的网格项目，如下所示：

.item-a {

grid-column: 1 / 2;

grid-row: 2 / 3;

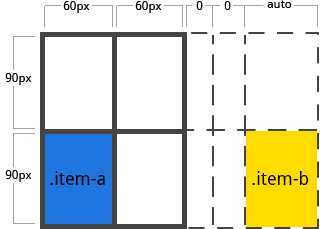
}

.item-b {

grid-column: 5 / 6;

grid-row: 2 / 3;

}

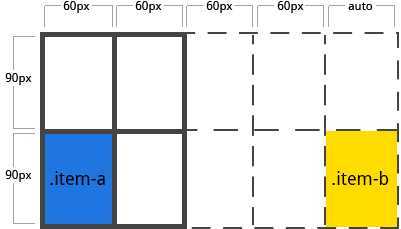


这里我们指定 .item-b开始于列网格线 5 并结束于在列网格线 6，但我们并未定义列网格线 5 或 6。因为我们引用不存在的网格线，宽度为0的隐式轨道的就会被创建用与填补间隙。我们可以使用 grid-auto-columns 和 grid-auto-rows属性来指定这些隐式轨道的宽度：

.container {

grid-auto-columns: 60px;

}



#### grid-auto-flow

如果你存在没有显示指明放置在网格上的 grid item，则自动放置算法会自动放置这些项目。 而该属性则用于控制自动布局算法的工作方式。

值：

* row - 告诉自动布局算法依次填充每行，根据需要添加新行
* column - 告诉自动布局算法依次填充每列，根据需要添加新列
* dense - 告诉自动布局算法，如果后面出现较小的 grid item，则尝试在网格中填充空洞

.container {

grid-auto-flow: row | column | row dense | column dense

}

需要注意的是，dense 可能导致您的 grid item 乱序。

举例， 考虑如下 HTML：

<section class="container">

<**div** class="item-a">item-a</**div**>

<**div** class="item-b">item-b</**div**>

<**div** class="item-c">item-c</**div**>

<**div** class="item-d">item-d</**div**>

<**div** class="item-e">item-e</**div**>

</section>

你定义一个有5列和2行的网格，并将 grid-auto-flow 设置为 row（这也是默认值）：

.container {

display: grid;

grid-template-columns: 60px 60px 60px 60px 60px;

grid-template-rows: 30px 30px;

grid-auto-flow: row;

}

当把 grid item 放在网格上时，你只把其中两个设置了固定的位置：

.item-a {

grid-column: 1;

grid-row: 1 / 3;

}

.item-e {

grid-column: 5;

grid-row: 1 / 3;

}

因为我们将 grid-auto-flow 设置为row，所以我们的grid就像这样。 注意观察我们没有做设置的三个项目（item-b，item-c和item-d）是如何在剩余的行水平摆放位置的：



如果我们将 grid-auto-flow 设置为 column，则 item-b，item-c 和 item-d 以列的顺序上下摆放：

.container {

display: grid;

grid-template-columns: 60px 60px 60px 60px 60px;

grid-template-rows: 30px 30px;

grid-auto-flow: column;

}



#### grid

在单个属性中设置所有以下属性的简写：grid-template-rows，grid-template-columns，grid-template-areas，grid-auto-rows，grid-auto-columns和grid-auto-flow。 它同时也将 sets grid-column-gap 和 grid-row-gap 设置为它们的初始值，即使它们不能被此属性显示设置。

值：

* none - 将所有子属性设置为其初始值
* <grid-template-rows> / <grid-template-columns> - 将 grid-template-rows 和 grid-template-columns 分别设置为指定值，将所有其他子属性设置为其初始值
* <grid-auto-flow> [<grid-auto-rows> [ / <grid-auto-columns>] ] - 接受所有与grid-auto-flow，grid-auto-rows和grid-auto-columns相同的值。 如果省略grid-auto-columns，则将其设置为为grid-auto-rows指定的值。 如果两者都被省略，则它们被设置为它们的初始值

.container {

grid: none | <grid-template-rows> / <grid-template-columns> | <grid-auto-flow> **[<grid-auto-rows> [/ <grid-auto-columns>]**];

}

举例：

以下代码写法等价

.container {

grid: 200px auto / 1fr auto 1fr;

}

.container {

grid-template-rows: 200px auto;

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

grid-template-areas: none;

}

以下代码写法等价

.container {

grid: column 1fr / auto;

}

.container {

grid-auto-flow: column;

grid-auto-rows: 1fr;

grid-auto-columns: auto;

}

它也可用使用一个更复杂但相当方便的语法来一次设置所有内容。 你可以指定 grid-template-areas、grid-template-rows 以及 grid-template-columns，并将所有其他子属性设置为其初始值。 你现在所做的是在其网格区域内，指定网格线名称和内联轨道大小。 可以看下面的例子：

.container {

grid: [row1-start] "header header header" 1fr [row1-end]

[row2-start] "footer footer footer" 25px [row2-end]

/ auto 50px auto;

}

上述代码等价于

.container {

grid-template-areas:

"header header header"

"footer footer footer";

grid-template-rows: [row1-start] 1fr [row1-end row2-start] 25px [row2-end];

grid-template-columns: auto 50px auto;

}

## 孩子(Grid Items)的属性

#### grid-column-start / grid-column-end / grid-row-start /grid-row-end

使用特定的网格线确定 grid item 在网格内的位置。grid-column-start/grid-row-start 属性表示grid item的网格线的起始位置，grid-column-end/grid-row-end属性表示网格项的网格线的终止位置。

值：

* <line>: 可以是一个数字来指代相应编号的网格线，也可使用名称指代相应命名的网格线
* span <number>: 网格项将跨越指定数量的网格轨道
* span <name>: 网格项将跨越一些轨道，直到碰到指定命名的网格线
* auto: 自动布局， 或者自动跨越， 或者跨越一个默认的轨道

.item {

grid-column-**star**t: <number> | <name> | span <number> | span <name> | auto

grid-column-end: <number> | <name> | span <number> | span <name> | auto

grid-row-**star**t: <number> | <name> | span <number> | span <name> | auto

grid-row-end: <number> | <name> | span <number> | span <name> | auto

}

举例：

.item-a {

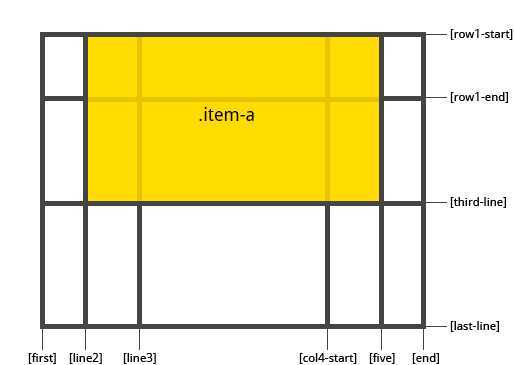
grid-column-start: 2;

grid-column-end: five;

grid-row-start: row1-start

grid-row-end: 3

}



.item-b {

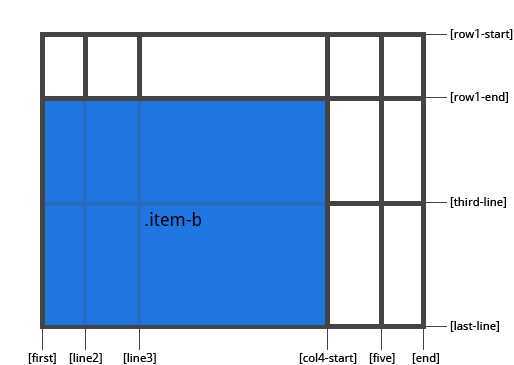
grid-column-start: 1;

grid-column-end: span col4-start;

grid-row-start: 2

grid-row-end: span 2

}



如果没有声明 grid-column-end / grid-row-end，默认情况下，该网格项将跨越1个轨道。

网格项可以相互重叠。 您可以使用z-index来控制它们的堆叠顺序。

#### grid-column / grid-row

grid-column-start + grid-column-end, 和 grid-row-start + grid-row-end 的简写形式。

值：

* <start-line> / <end-line> - 每个值的用法都和属性分开写时的用法一样

.item {

grid-column: <start-line> / <end-line> | <start-line> / span <value>;

grid-row: <start-line> / <end-line> | <start-line> / span <value>;

}

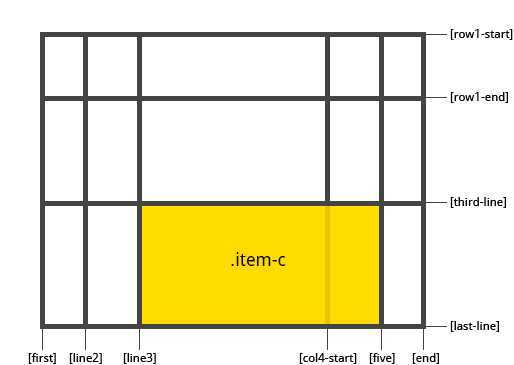
举例：

.item-c {

grid-column: 3 / span 2;

grid-row: third-line / 4;

}



如果没有指定结束行值，则该网格项默认跨越1个轨道。

#### grid-area

给 grid item 进行命名以便于使用 grid-template-areas 属性创建模板时来进行引用。另外也可以做为 grid-row-start + grid-column-start + grid-row-end + grid-column-end 的简写形式。

值：

* <name> - 你的命名
* <row-start> / <column-start> / <row-end> /<column-end> - 可以是数字，也可以是网格线的名字

.item {

grid-area: <name> | <row-start> / <column-start> / <row-end> / <column-end>;

}

举例:

给一个网格项命名

.item-d {

grid-area: header

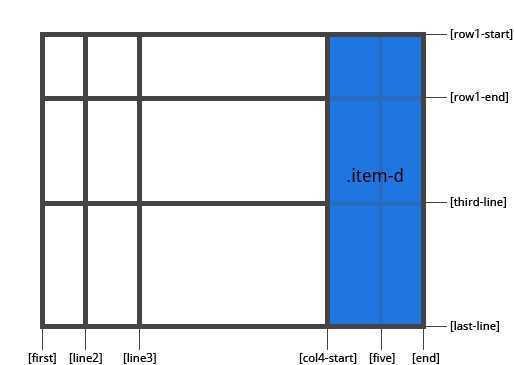
}

作为 grid-row-start + grid-column-start + grid-row-end + grid-column-end 的简写:

.item-d {

grid-area: 1 / col4-start / last-line / 6

}



#### justify-self

沿着行轴对齐grid item 里的内容（与之对应的是 align-self, 即沿列轴对齐）。 此属性对单个网格项内的内容生效。

值：

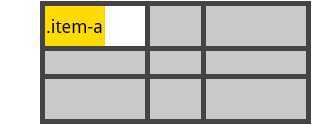
* start - 将内容对齐到网格区域的左端
* end - 将内容对齐到网格区域的右端
* center - 将内容对齐到网格区域的中间
* stretch - 填充网格区域的宽度 (这是默认值)

举例：

.item-a {

justify-self: start;

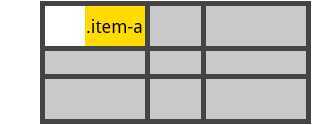
}



.item-a {

justify-self: end;

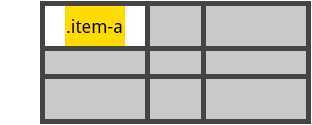
}



.item-a {

justify-self: center;

}



.item-a {

justify-self: stretch;

}



要为网格中的所有grid items 设置对齐方式，也可以通过 justify-items 属性在网格容器上设置此行为。

#### align-self

沿着列轴对齐grid item 里的内容（与之对应的是 justify-self, 即沿行轴对齐）。 此属性对单个网格项内的内容生效。

值：

* start - 将内容对齐到网格区域的顶部
* end - 将内容对齐到网格区域的底部
* center - 将内容对齐到网格区域的中间
* stretch - 填充网格区域的高度 (这是默认值)

.item {

align-self: start | end | center | stretch;

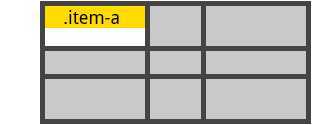
}

举例:

.item-a {

align-self: start;

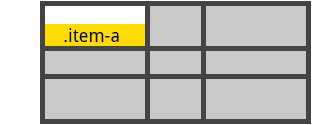
}



.item-a {

align-self: end;

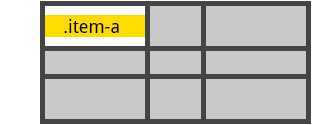
}



.item-a {

align-self: center;

}



.item-a {

align-self: stretch;

}



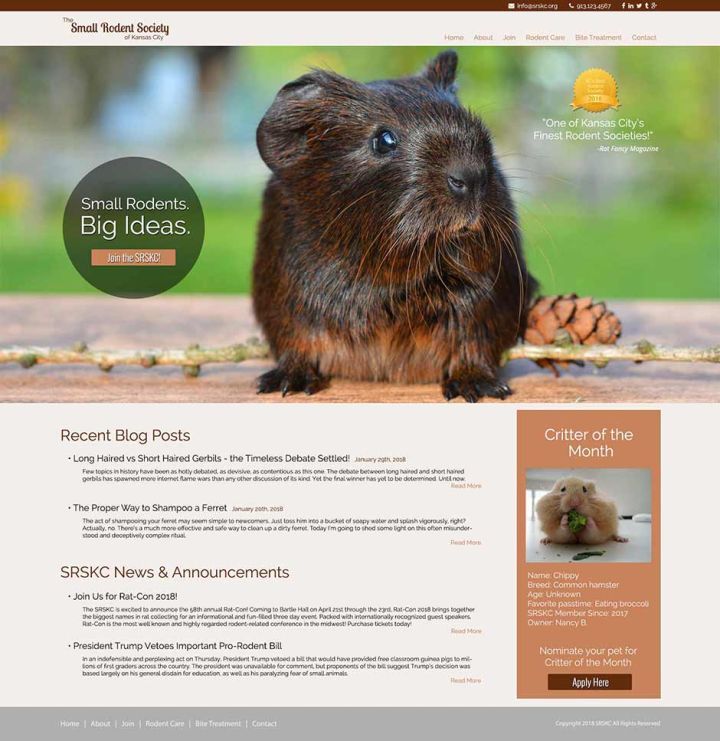
要为网格中的所有grid items 统一设置对齐方式，也可以通过 align-items 属性在网格容器上设置此行为。

## 实例

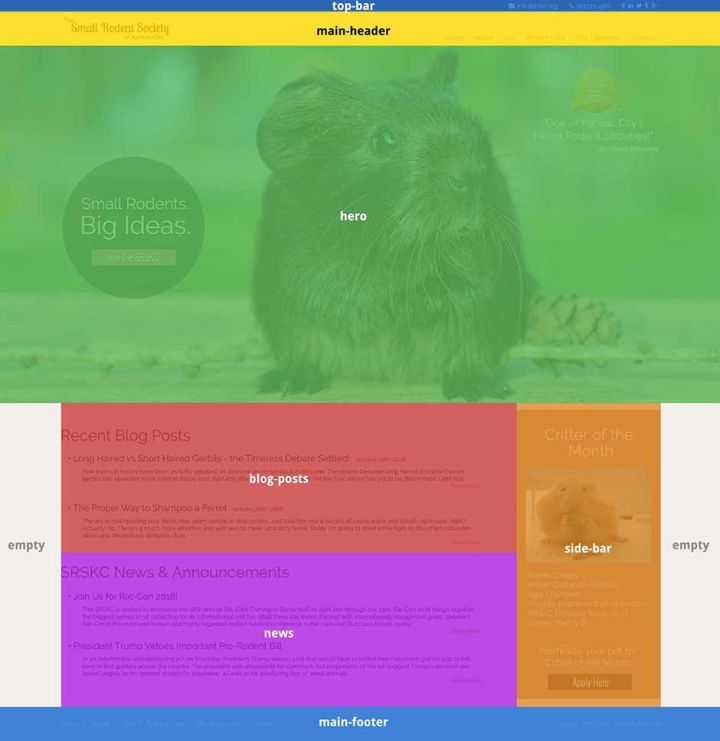
当我开始一个项目，并开始计划如何布局主页时，我的大脑复现出浮动和定位。有些人可能会使用 Bootstrap 或其他框架。 那是因为这是2016年，我们一直在用这些方法来做布局。 但假设我们乘坐时光机来到2018年，所有主流浏览器都支持CSS Grid 布局模块。此时我们的页面布局模式已经完全改变，CSS的功能最终强大到能轻松实现我们的设计目标，这是一个web开发人员最美好的时代。现在，让我们使用超赞的工具——Grid布局来创建一个主页。

## 设计

下面是我们将要实现的页面



在我们开始编码之前，我们需要进入网格的思维模式。 第一步是观察我们的设计稿，并将其划分为主要的网格组件。 以下是我为此设计做的划分：



你会发现整个页面分为7个顶级网格区域。 我之所以说“顶级”是因为我们可以在其内部继续嵌套网格，这正是我们将要对hero部分所做的事：



## HTML

这是HTML的基本结构。 稍后我会显示整个完成的文件，但现在我已经省去了大部分的细节。 这里要注意的重要部分是作为 body 的直接后代的7个元素：top-bar、main-header、hero、 blog-posts、 news、 side-bar 以及 main-footer。 body将成为我们的网格容器(grid container)，它的孩子将成为网格项(grid items)。

正如刚刚提到的，我们也将设置 hero 作为网格容器。 它有两个孩子，将作为网格项：message 和 award。

<body>

<header class="top-bar">

<!-- social links and contact info -->

</header>

<header class="main-header">

<!-- logo and main navigation -->

</header>

<section class="hero">

<div class="message">

<!-- circular element -->

</div>

<div class="award">

<!-- award image and quote -->

</div>

</section>

<section class="blog-posts">

<!-- blog posts and excerpts -->

</section>

<section class="news">

<!-- news headlines and excerpts -->

</section>

<aside class="side-bar">

<!-- critter of the month info -->

</aside>

<footer class="main-footer">

<!-- footer menu and copyright -->

</footer>

</body>

## CSS

Okey，我们按照这种方式讲解，教程中我们不会展示所有使用到的CSS，在文章的最后我会展示最终完整的文件。现在我们只关注吸引我们的网格部分以及任何与它直接相关的样式即可。

我们首先在body上定义主网格容器：

**body**{

display: grid;

grid-template-columns: 12% auto 400px 12%;

grid-template-rows: auto auto 950px auto auto auto;

}

我们刚刚创建了一个4列6行的网格，第一列和最后一列将作为主内容两侧的填充。 我把第三列设置为400px，因为这是我们将要放置side-bar元素的地方，我们希望这是一个固定的宽度。 hero 元素（第三行）的固定高度为950px。

现在我们使用grid-template-areas来定义某个网格区域会跑到哪里。 这是非常有趣的部分：

**body**{

display: grid;

grid-template-columns: 12% auto 400px 12%;

grid-template-rows: auto auto 950px auto auto auto;

grid-template-areas: "top-bar top-bar top-bar top-bar"

"main-header main-header main-header main-header"

"hero hero hero hero"

". blog-posts side-bar ."

". news side-bar ."

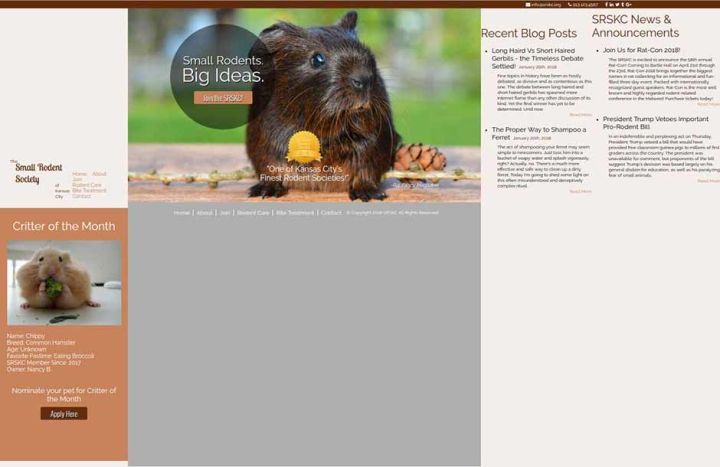
"main-footer main-footer main-footer main-footer";

}

grid-template-areas让我们能把元素放在任何想要放置的地方，并且对于元素的布局该属性给我们提供一个不错的可视化。 值得注意的是，这里使用的值（top-bar，main-header，hero等）不是指那些元素的类名，而是指我们用grid-area属性给它们起的名字，下一步我们将对它们命名。

当网格区域名称重复时，该元素将跨越这些列/行。 例如，top-bar 横跨四列，side-bar横跨四行和五行。 .号代表空单元格。如果你回头看看上面的完整设计，您会看到这个定义如何与我们的网格模式相匹配。

假设我们已经应用了我们所有的样式，但还没有为网格项分配网格区域名称，到目前为止我们的页面看起来还不太好：



在将网格区域名称分配给网格项之前，网格将根据它们的源顺序自动将我们的元素放置在网格中。 显然这不是我们想要的。 为了使我们的布局按预期工作，我们需要定义我们的网格区域。所以我们继续往下走：

.top-bar{

grid-area: top-bar;

}

.main-header{

grid-area: main-header;

}

.hero{

grid-area: hero;

}

.blog-posts{

grid-area: blog-posts;

}

.news{

grid-area: news;

}

.side-bar{

grid-area: side-bar;

}

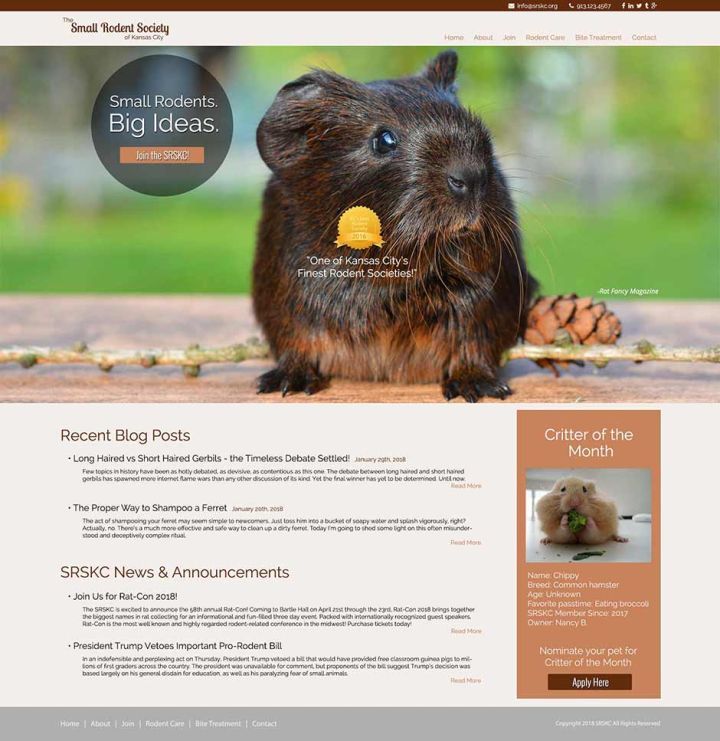
.main-footer{

grid-area: main-footer;

}

需要注意的是这些名称可以随意设置。 为了方便，我选择了让它们与类名相匹配。

现在，我们已经为网格项分配了网格区域名称，它们将在被放置在网格中合适的位置。 这一步带来的变化很大：



除了 hero 部分中的网格项外，所有内容都完全按照需要正确放置，我们差不多要完成了。

但是在我们修复 hero 部分之前，我想解释一下一些难以理解的地方：主要内容两边的填充区域的设置。 作为提醒，我们再次把刚刚的设置搬过来，用如下方式调整列：

**body**{

grid-template-columns: 12% auto 400px 12%;

}

设置为12％的两列用于填充主要内容两边的空白，但是它们仅用于第四行和第五行。 回想一下，我们告诉我们的top-bar、main-header、hero和main-footer元素跨越所有列，包括这两个“填充”列。 我们为什么这样做？ 因为我们希望这些元素的背景色横跨越整个视窗宽度，且任何一侧都没有空白。 我们只想在 blog-post/news和sidebar元素周围留出空白（第四行和第五行）。

为了让元素水平覆盖整个宽度，同时让元素里面的内容保存一定的padding，我们需要显示地在这些元素上设置padding：

.top-bar{

padding: 4px 12%;

}

.main-header{

padding: 12px 12%;

}

.hero{

padding: 55px 12% 0 12%;

}

.main-footer{

padding: 25px 12%;

}

我们给元素设置左右 padding 为12％，这和grid-template-areas定义中的第一列和最后一列的宽度是一样的。 现在，需要填充整个宽度的元素最终呈现的结果是，背景横跨水平宽度，但其内容在两侧都预留出12％的空白。 很赞！

好了，让我们来修复 hero 部分。 这也将是一个网格容器，因此我们把它定义为一个网格，就像刚刚做过的那样：

.hero{

display: grid;

grid-template-columns: auto 1fr auto;

grid-template-rows: auto auto auto;

grid-template-areas: ". . award"

"message . . "

". . . ";

}

这是一个3×3的网格，除了中间的列，其它都设置为 auto。 我们给中间一列大小设为1fr，因为我们希望在第一列和最后一列用东西填充后，剩下的空间完全需要完全填满。

hero中只有两个元素：message和award。 我们要message占据第二行的第一列，我们要award占据第一行的第三列。所以我们的完整网格定义应该如下所示：

.hero{

display: grid;

grid-template-columns: auto 1fr auto;

grid-template-rows: auto auto auto;

grid-template-areas: ". . award"

"message . . "

". . . ";

}

下面我们所要做的就是命名我们的元素：

.message{

grid-area: message;

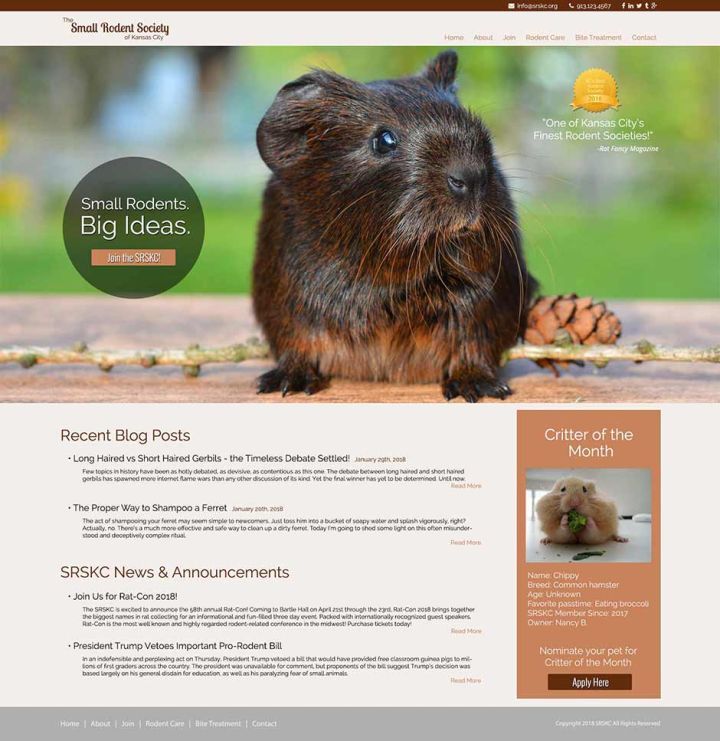
}

.award{

grid-area: award;

}

就这样，message和award卡入到位，我们的页面完成：



## 引入响应式

CSS Grid 使用媒体查询让重新排列整个布局变得非常简单。你所做的就是重新放置你的网格项。现在回到我们的设计，简单起见，我们只对两个宽度临界值做响应式处理，1600px 和 1050px。我们需要对一些元素（padding、margin等）进行一些小的样式调整，但是我不会把所有的样式调整都全部展示在这里。后面我会放出完整的代码，现在我们只需要关注关注网格相关的东西即可。

1600px 这个临界点的处理比较简单，当浏览器宽度到底1600px时我们将减少网站外部填充的地方。 之所以选择1600px，是到了这个宽度后12％填充看起来不太合适。为了解决这个问题，我们需要做的是在body上改变grid-template-columns的值，将第一列和最后一列减少到2％。 我们还需要调整其他元素的填充以匹配：

@**media** (max-width: 1600px) {

**body**{

grid-template-columns: 2% auto 400px 2%;

}

.top-bar{

padding: 4px 2%;

}

.main-header{

padding: 12px 2%;

}

.hero{

padding: 55px 2% 0 2%;

}

.main-footer{

padding: 25px 2%;

}

}

对于下一个临界值，我们对网格项重新排列，使它们排列在一个列中。 再次回头看看我们原来的代码是如何对body进行设置的：

**body**{

display: grid;

grid-template-columns: 12% auto 400px 12%;

grid-template-rows: auto auto 950px auto auto auto;

grid-template-areas: "top-bar top-bar top-bar top-bar"

"main-header main-header main-header main-header"

"hero hero hero hero"

". blog-posts side-bar ."

". news side-bar ."

"main-footer main-footer main-footer main-footer";

}

下面是重新设置的媒体查询：

@media (max-width: 1050px) {

body{

grid-**template**-columns: 3% **auto** 3%;

grid-**template**-rows: **auto** **auto** **auto** **auto** **auto** **auto** **auto**;

grid-**template**-areas: "top-bar top-bar top-bar"

"main-header main-header main-header"

"hero hero hero"

". blog-posts ."

". news ."

". side-bar ."

"main-footer main-footer main-footer";

}

}

我们在这里做了一些重要的改变：将列数从四个减少到三个，将第一列和最后一列的值改为3％（3％在较窄的宽度上优于2％），添加了 附加行，将所有行的长度改为auto，并将side-bar移动到自己的行。 现在我们的页面元素很适合在较窄的宽度下展示：



## The Live Code

下面是我们的主页，以及完整的HTML和CSS文件。 你需要一个支持grid的浏览器来查看预览。 我建议启用Experimental Web Platform Features标志的Chrome 49+（地址栏输入 chrome：// flags ，并向下滚动到“Experimental Web Platform Features”）。

下面的嵌入式页面默认会以移动视图展示，可以点击“Edit on Codepen”在页面全宽下展示不同的效果：

在 CodePen 查看效果 Building a Home Page with Grid by Chris House (@chrishouse) .

补充：[基本布局代码](http://js.jirengu.com/riyeneduce/edit?html,css,output)