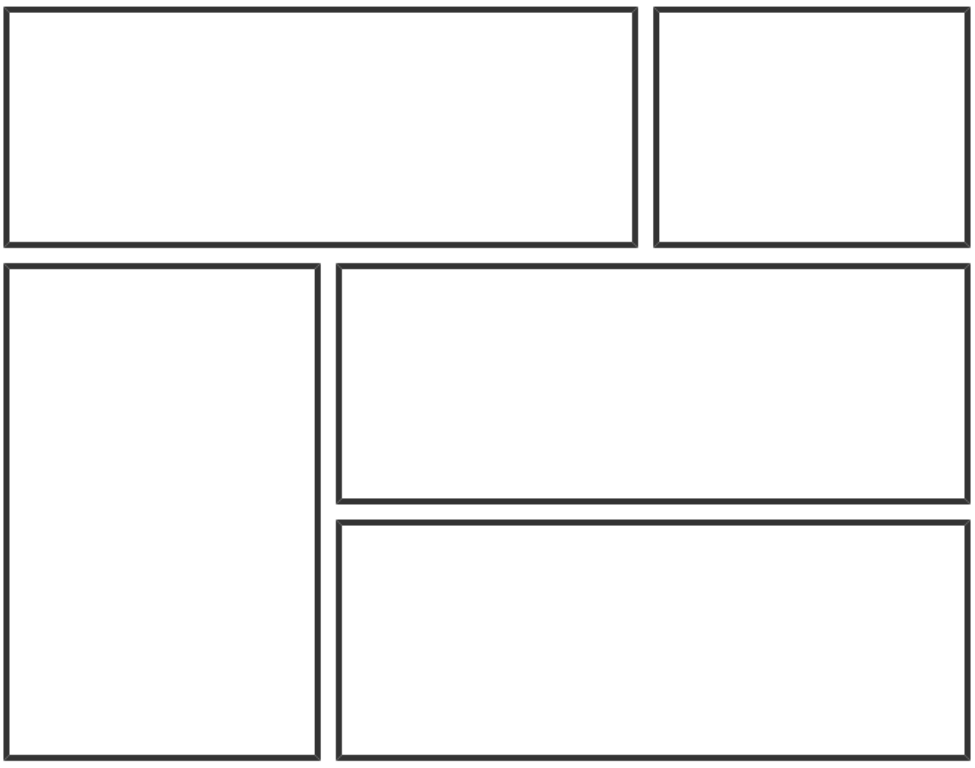
全栈迷你书系列

CSS Grid网格布局

迷你书

TG ◎ 著



CSS Grid网格布局迷你书

TG

Version 1.0, 2018-08-18

目录

前言

重要的事情说三遍…

此书是免费的！此书是免费的！此书是免费的！

《CSS Grid网格迷你书》到手，天下我有，希望此书不会浪费你宝贵的时间。

当读完后，如此书对你有所帮助，打个赏，让我有更大的动力去创作。

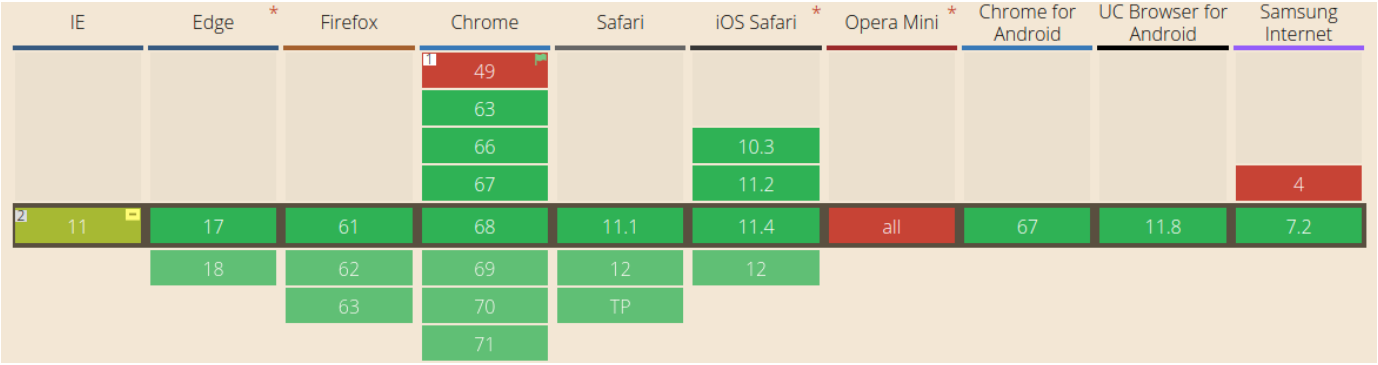
由于很多内容都是个人理解整理，难免会有不准确或者让大家产生疑问的地方，如果有意见或者问题的话，可以直接通过以下方式联系到我。

- 1973178583@qq.com
- TG博客
- 掘金

要获取最新，请在这里下载：

《CSS Grid网格布局迷你书》

1. 兼容性一览



除了万恶的IE，可见浏览器的支持已经很好了

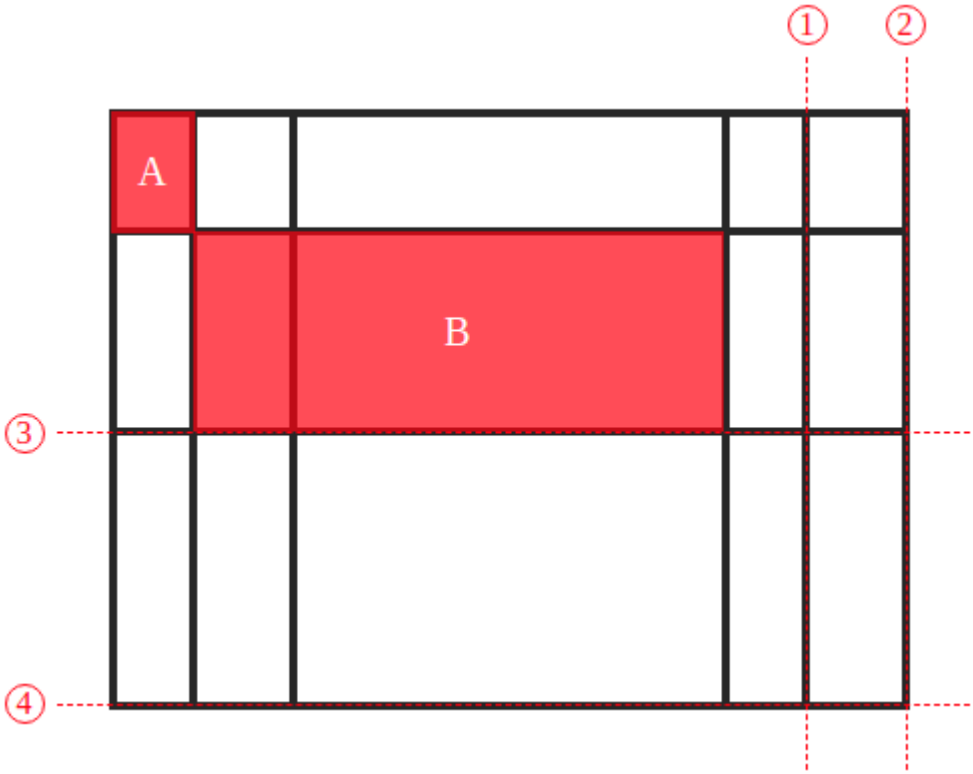
2. Grid 简介

CSS Grid(网格) 布局（又称为 “Grid(网格)” ），是一个二维的基于网格的布局系统，旨在改变我们基于网格的用户界面的布局方式。

CSS Grid兼容性： Grid

2.1. 重要术语

在使用之前，先来了解一些重要术语。



- 网格容器（Grid Container）：应用 display: grid | inline-grid 的元素，也就是所有网格项（Grid Items）的直接父级元素，也就是上面最大的矩形。
- 网格项（Grid Items）： 网格容器的直接子元素
- 网格线（Grid Lines）：如上图①、②、③、④
- 网格单元格（Grid cell）：类似表格的单元格，如上图A区域
- 网格区域（Grid area）：由一个或者多个网格单元格组成的一个矩形区域，如上图B区域。
- 网格轨道（Grid tracks）：指两条网格线之间的空间。如上图①和②之间、③和④之间
- 网格列（Grid column）：指两个垂直网格线之间的空间。如上图①和②之间
- 网格行（Grid row）：指两个水平网格线之间的空间。如上图③和④之间

- 网格间隙（Gutters）：指网格轨道之间的间距
- 网格轴（Grid Axis）：类似坐标轴，每个网格单元都有一个横轴（即行轴，内联）和纵轴（即列轴，块）

2.2. 简单入门

定义块级网格容器：

```
.container {
  display: grid;
}
```

定义内联级网格容器：

```
.container {
  display: inline-grid;
}
```

3. 网格容器的属性

这一章来介绍网格容器的各个属性。

- grid-template-columns
- grid-template-rows
- grid-template-areas
- grid-template
- grid-column-cap
- grid-row-cap
- grid-cap
- grid-auto-flow
- grid-auto-columns
- grid-auto-rows
- justify-items
- align-items
- justify-content
- align-content
- place-content
- grid

3.1. 网格行和网格列

使用以空格分隔的值列表定义网格的列和行：

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

属性值：

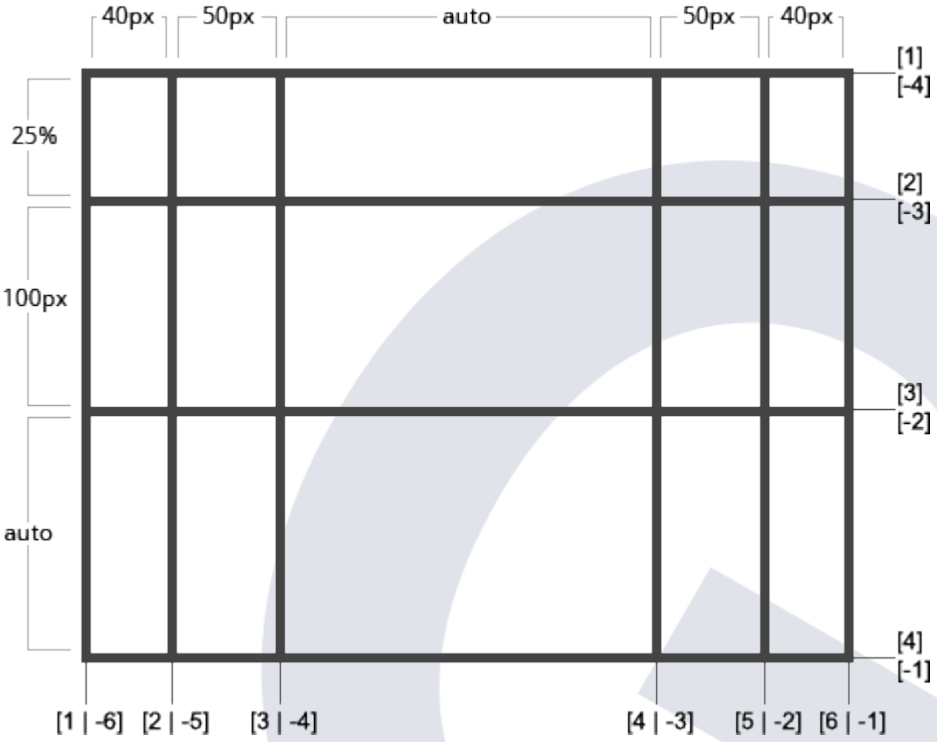
- `<track-size>` - 可以是网格中可用空间的长度，百分比或等分数（使用 `fr` 单位）
- `<line-name>` - 定义网格线的任意名称

注意：当要自定义网格线的名称时，别漏了中括号 `[line-name]`

3.1.1. 基本行列

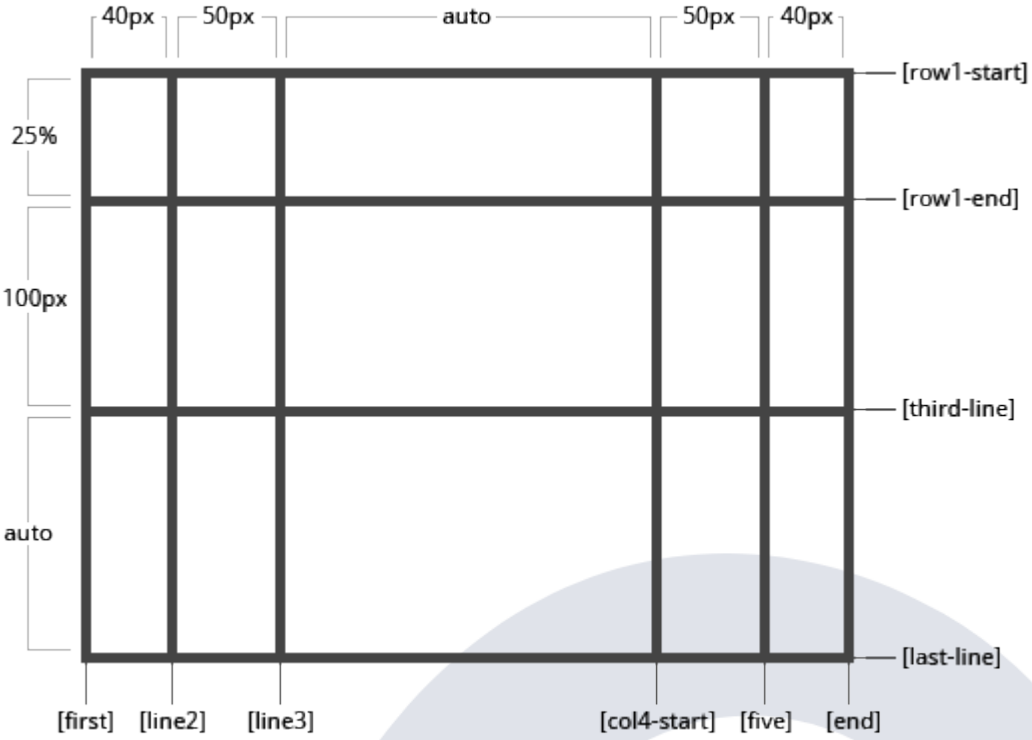
当我们不提供网格线名时，网格会自动分配名称（用正数和负数表示）：

```
.container {
  grid-template-columns: 40px 50px auto 50px 40px;
  grid-template-rows: 25% 100px auto;
}
```



我们也可以定义名称，且可以定义多个：

```
.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];
}
```



一行可以有多个名称：

```
.container {
  grid-template-rows: [row1-start] 25% [row1-end row2-start] 25% [row2-end];
}
```

网格的第二行将有两个名称：row1-end和row2-start

在上面提到一个单位 `fr`，它的作用是平分网格的可用空间。

```
.container {
  grid-template-columns: 1fr 1fr 1fr 1fr;
  /* 相当于 grid-template-columns: 25% 25% 25% 25%; */
}
```

上面代码的意思是将每个网格项设置为网格容器宽度的四分之一，也就是平分网格的可用空间。

何为可用空间呢？其实就是减去用精确长度或百分比定义的剩余长度。

```
.container {
  grid-template-columns: 1fr 50px 1fr 25%;
}
```

可用空间 = 网格的宽度 - 50px - 25% * 网格宽

3.1.2. 重复

如果需要定义包含重复部分，我们可以这样：

```
.container {
  grid-template-columns: [col-start] 20px [col-start] 20px [col-start]
20px;
}
```

除了像上面那样重复定义外，grid还提供了一个 `repeat()` 来简化代码：

```
.container {
  grid-template-columns: repeat(3, 20px [col-start]);
}
```

这样定义和上面重复写三个的效果是一样的。

auto-fit vs auto-fill

使用关键字 `auto-fit` 和 `auto-fill`。

- `auto-fill` 用尽可能多的列填充行。
- `auto-fit` 通过扩展使当前可用的列进入空间，以便它们占用任何可用空间。

```
.container {
    grid-gap:10px;
    grid-template-columns: repeat(auto-fill, minmax(50px,1fr));
    grid-auto-rows:40px;
}

.container:nth-child(2) {
    grid-template-columns: repeat(auto-fit, minmax(50px,1fr));
}
```

3.1.3. 最大值与最小值

我们还可以通过 `minmax()` 定义一个网格单元格的最小值和最大值，此函数包含两个参数，最小值 和 最大值

```
minmax( [ <track-size> | min-content | max-content | auto ] , [ <track-size> | min-content | max-content | auto ] )
```

属性值：

- `<track-size>` - 可以是网格中可用空间的长度，百分比或等分数（使用 `fr` 单位）
- `max-content` - 表示网格的轨道长度自适应内容最大的那个单元格
- `min-content` - 表示网格的轨道长度自适应内容最小的那个单元格
- `auto` - 作为最大值时，等价于 `max-content` 。作为最小值时，它表示轨道中单元格最小长宽 (由 `min-width/min-height`)的最大值。

实例

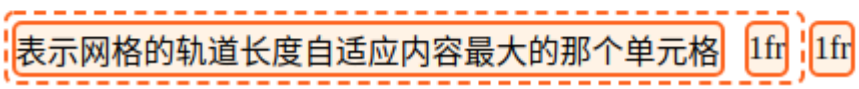
精确长度

```
.container {
  grid-template-columns: minmax(100px, 200px) 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
}
```

max-content

`max-content` 表示网格的轨道长度自适应内容最大的那个单元格。这句话有点抽象，先看看下面的实例。

```
.container {
  grid-template-columns: minmax(max-content, max-content) 1fr 1fr;
}
```

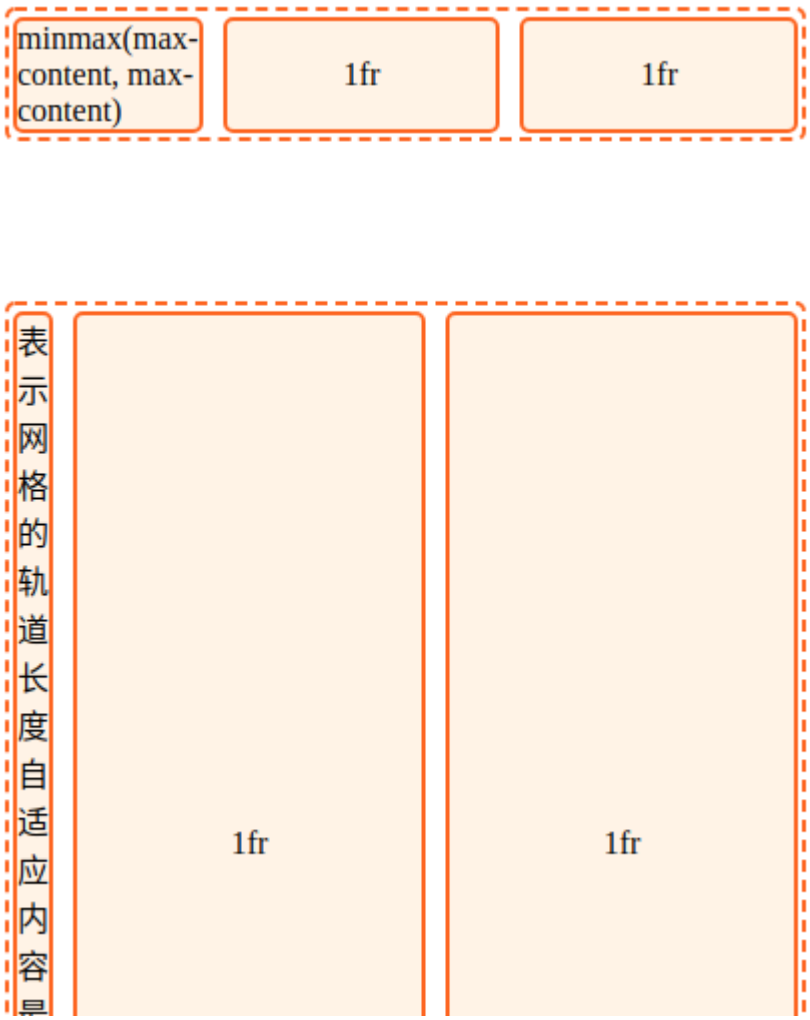


可见，当设置 `minmax(max-content, max-content)` ，不管单元格内容的长度是多少，都不会换行，且单元格的长度刚好围绕住内容。也可以理解为： 设置 `min-width` 为内容一行展开的最大宽度。

min-content

`min-content` 表示网格的轨道长度自适应内容最小的那个单元格

```
.container {
  grid-template-columns: minmax(min-content, min-content) 1fr 1fr;
}
```



当设置 `minmax(min-content, min-content)` ，单元格会采取内容的最小宽度，将可用空间尽可能多的转移给其他单元格，而且内容也不会引起任何的溢出。也可理解为：设置 `max-width` 为刚好可以围绕内容的最小宽度。

auto

`auto` 作为最大值时，等价于 max-content。作为最小值时，它表示轨道中单元格最小长宽 (由min-width/min-

height)的最大值

`minmax()` 小结：

- 如果 最大值 < 最小值，则最大值被忽略并且minmax(最小值,最大值)被看成最小值
- `fr` 值作为最大值时设置网格轨道的弹性系数；作为最小值时无效
- 网格项的长度永远不会缩小到比最小值小的值。
- 优先级由高到低： 精确的长度|百分比 → 最小值 → 最大值

3.2. 网格区域

由一个或者多个网格单元格组成的一个矩形区域。

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

属性值:

- `<grid-area-name>` - 使用指定的网格区域的名称grid-area
- `.` - 英文句点表示空网格单元格
- `none` - 没有定义网格区域

实例

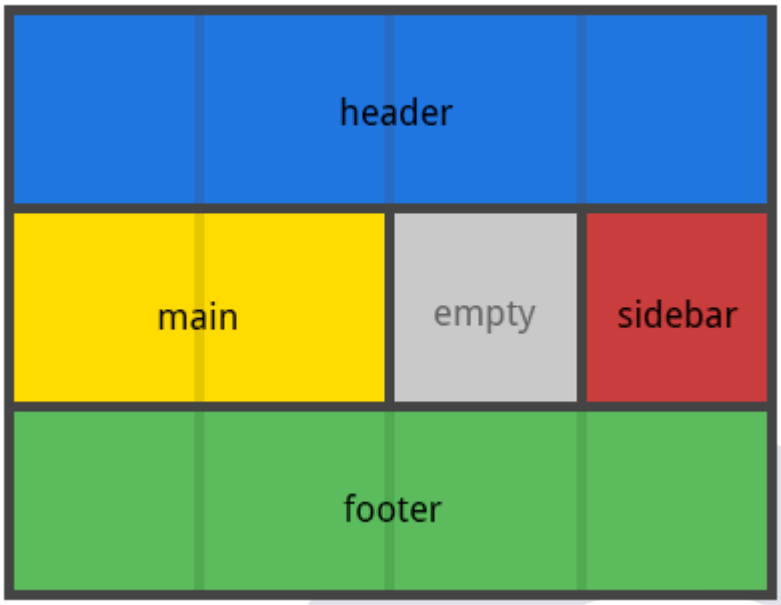
使用 `grid-template-areas`， 需要结合网格单元格的 `grid-area`，后面会详细讲解。

```
.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";
}
```

在上面的代码中，将创建一个三行四列的网格。其中第一行的四列组成header网格区域，第二行由三部分组成，第一个单元格和第二个单元格组成main区域，第三个单元格为空单元格，第四个单元格为sidebar;最后，

第三行的四个单元格组成footer网格区域。



注意：

- `grid-template-areas` 属性声明中的每一行都需要具有相同数量的单元格，否则将会出现乱格。
- 当声明网格区域时，区域两端的线条实际上会自动命名。比如上面的网格区域header，则区域的起始行和起始列行的名称将为header-start，其最后一行和最后一行的名称将为header-end。

3.2.1. 简写

`grid-template` 是一个CSS属性缩写，用于定义 grid columns, rows 和areas

`grid-template` 有三种用法。

```
grid-template: none;  ①

grid-template: <'grid-template-rows'> / <'grid-template-columns'>; ②

grid-template: [ <line-names>? <string> <track-size>? <line-names>? ]+ [
/ <explicit-track-list> ]? ③
```

① 将三个属性（`grid-template-columns`、`grid-template-rows`、`grid-template-areas`）设置为初始值

② 设置 `grid-template-columns` 和 `grid-template-rows` 属性，且将 `grid-template-areas` 设置为 `none`

③ `<line-name>` 设置网格线名，`<string>` 设置网格区域，`<track-size>` 设置行轨道，`<explicit-track-list>` 设置列轨道

下面两个例子是等价的。

```
.container {
  grid-template:
      [row1-start] "header header header" 25px [row1-end]
      [row2-start] "footer footer footer" 25px [row2-end]
      / auto 50px auto;
}
```

```
.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";
}
```

3.3. 网格间隙

使用 `grid-column-gap` 和 `grid-row-gap` 定义列间隙或行间隙。

语法：

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

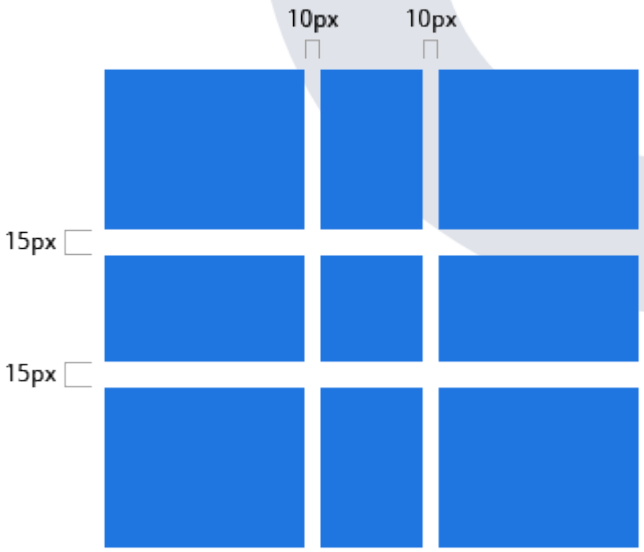
属性值：

- `normal` - 浏览器默认的间隔
- `<line-size>` - 非负长度

注： grid-column-gap和grid-row-gap被重命名为column-gap和row-gap，但支持度还不高，推荐使用前者

实例

```
.container {
  grid-template-columns: 100px 50px 100px;
  grid-template-rows: 80px auto 80px;
  grid-column-gap: 10px;
  grid-row-gap: 15px;
}
```



网格间隙简写

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

注： 如果 `grid-column-gap` 未指定，则将其设置为与 `grid-row-gap` 相同的值

3.4. 网格项对齐

3.4.1. 沿着行轴对齐（水平对齐）

沿着内联（行）轴对齐网格项：

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

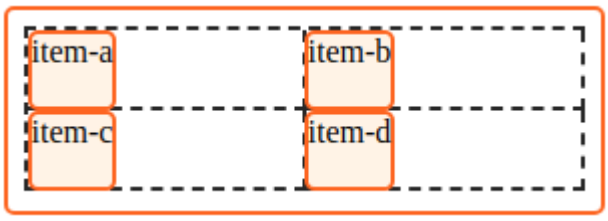
属性值：

- `start` - 将网格项与单元格的起始边缘对齐
- `end` - 将网格项与单元格的结束边缘对齐
- `center` - 将网格项与单元格中心的对齐
- `stretch` - 填充单元格的整个宽度（默认值）

实例：

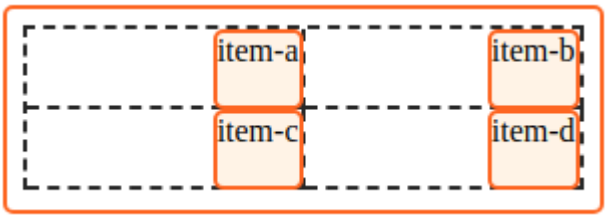
左对齐

```
.container {
  justify-items: start;
}
```



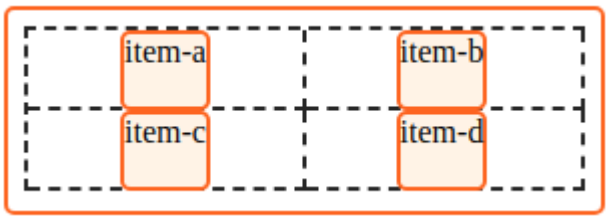
右对齐

```
.container {
  justify-items: end;
}
```



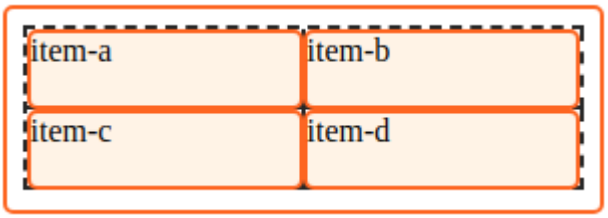
中间对齐

```
.container {
  justify-items: center;
}
```



填充

```
.container {
  justify-items: stretch;
}
```



3.4.2. 沿着列轴对齐（垂直对齐）

沿着内联（行）轴对齐网格项：

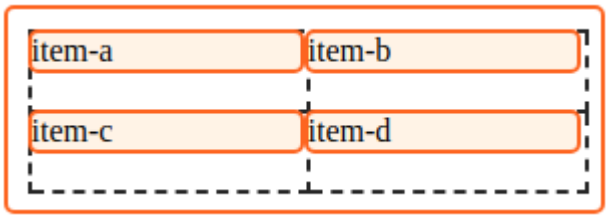
```
.container {
  align-items: start | end | center | stretch;
}
```

属性值：

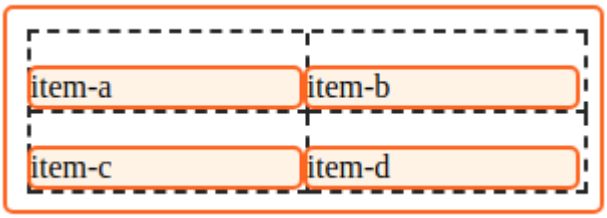
- `start` - 将网格项与单元格的起始边缘对齐
- `end` - 将网格项与单元格的结束边缘对齐
- `center` - 将网格项与单元格中心的对齐
- `stretch` - 填充单元格的整个宽度（默认值）

实例：

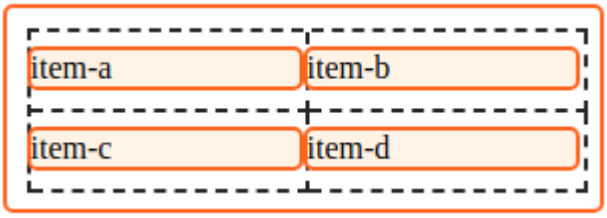
```
.container {
  align-items: start;
}
```



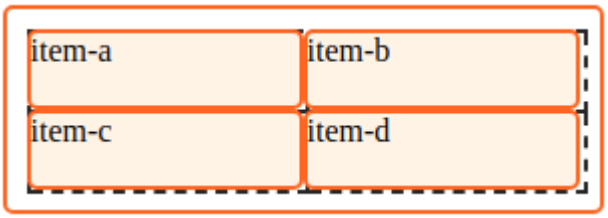

```
.container {
  align-items: end;
}
```




```
.container {
  align-items: center;
}
```



```
.container {
  align-items: stretch;
}
```



对齐简写

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

注： 如果省略第二个值，则将第一个值分配给这两个属性

3.5. 网格对齐

有些时候，网格的总大小可能小于网格容器的大小，这个时候，我们可能需要一种对齐方式。

3.5.1. 沿着内联（行）轴对齐（水平对齐）

```
.container {
  justify-content: start | end | center | stretch | space-around |
space-between | space-evenly;
}
```

属性值：

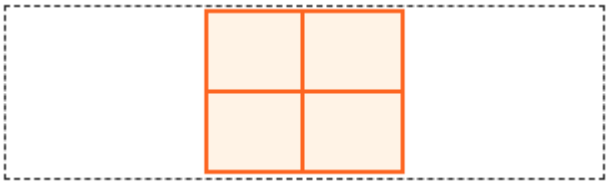
- start - 将网格项与单元格的起始边缘对齐
- end - 将网格项与单元格的结束边缘对齐
- center - 将网格项与单元格中心的对齐
- stretch - 填充单元格的整个宽度（默认值）
- space-around - 在每个网格项之间放置一个均匀大小的空间，在左右两端放置半个大小的空间
- space-between - 在每个网格项之间放置一个均匀大小的空间，在左右两端没有空间
- space-evenly - 在每个网格项之间放置一个均匀v的空间，包括左右两端

实例

```
.container {
  justify-content: start;
}
```

```
.container {
  justify-content: end;
}
```

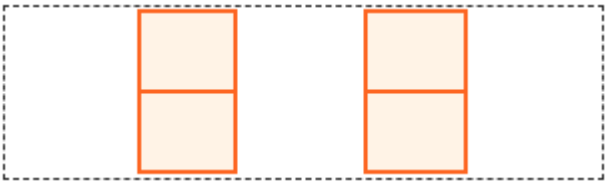
```
.container {
  justify-content: center;
}
```



```
.container {
  justify-content: space-around;
}
```

```
.container {
  justify-content: space-between;
}
```

```
.container {
  justify-content: space-evenly;
}
```



3.5.2. 沿着内联（列）轴对齐（垂直对齐）

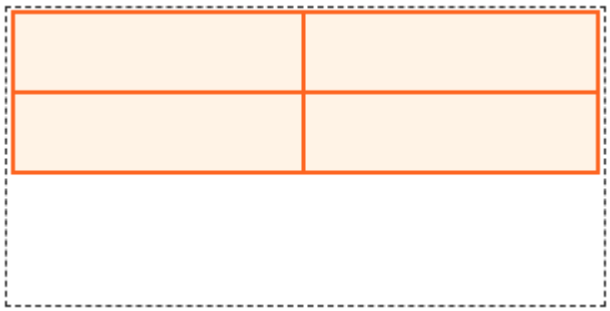
```
1 .container {
2   align-content: start | end | center | stretch | space-around | space-
between | space-evenly;
3 }
```

属性值：

- start - 将网格项与单元格的起始边缘对齐
- end - 将网格项与单元格的结束边缘对齐
- center - 将网格项与单元格中心的对齐
- stretch - 填充单元格的整个宽度（默认值）
- space-around - 在每个网格项之间放置一个均匀大小的空间，在上下两端放置半个大小的空间
- space-between - 在每个网格项之间放置一个均匀大小的空间，在上下两端没有空间
- space-evenly - 在每个网格项之间放置一个均匀大小的空间，包括上下两端

实例

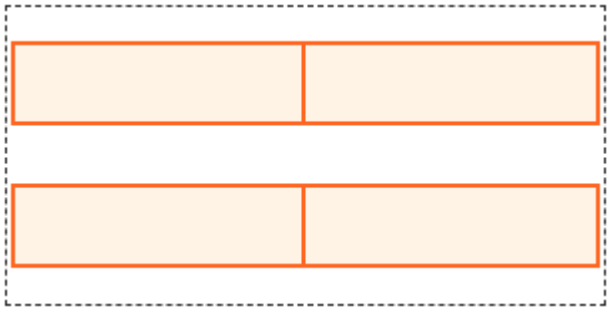
```
1 .container {
2   align-content: start;
3 }
```



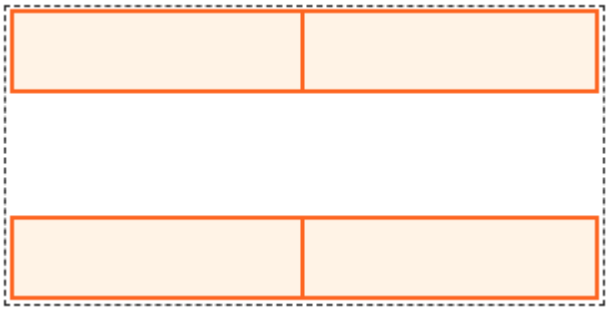
```
.container {
  align-content: end;
}
```

```
.container {
  align-content: center;
}
```

```
.container {
  align-content: space-around;
}
```



```
.container {
  align-content: space-between;
}
```



```
.container {
  align-content: space-evenly;
}
```

3.5.3. 简写

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

3.6. 隐式网格

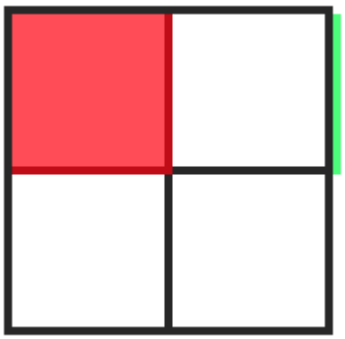
隐式网格，也可以说不在我们最初创建网格范围内(使用 `grid-template-rows` 和 `grid-template-columns` 声明)的网格。

实例

```
.container {
  grid-template-columns: 80px 80px;
  grid-template-rows: 80px 80px;
}

.item-a {
    grid-column:1 / 2;
    grid-row:1 / 2;
}
.item-b {
    grid-column:4 / 5;
    grid-row:1 / 2;
}
```

在上面的代码中，我们定义了一个2 * 2的网格，且.item-b从第4行开始到第5行结束，但我们从未定义过第4行或第5行。这就是前面提到的隐式网格，也就是不显式存在的行，此时，Gird会自动创建宽度为0的隐式轨道以填充间隙。

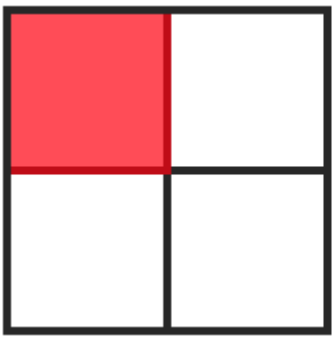


那如何定义隐式网格的大小呢？可以使用 `grid-auto-columns` 和 `grid-auto-rows` 来定义。

实例

```
.container {
  grid-auto-columns: 80px;
  grid-template-columns: 80px 80px;
  grid-template-rows: 80px 80px;
}

.item-a {
    grid-column:1 / 2;
    grid-row:1 / 2;
}
.item-b {
    grid-column:4 / 5;
    grid-row:1 / 2;
}
```



更多关于《显式网格和隐式网格的区别》

3.7. 自动布局

当没有显式放置网格中的网格项时，Grid会使用自动布局算法来精确指定在网格中被自动布局的网格项怎样排列。

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

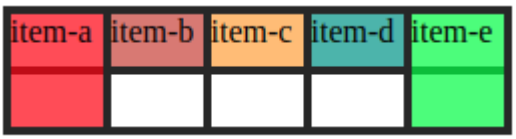
属性值：

- row - 默认值，指定自动布局算法按照通过逐行填充来排列元素，在必要时增加新行。
- column - 指定自动布局算法通过逐列填充来排列元素，在必要时增加新列。
- dense - 指定自动布局算法使用一种「稠密」堆积算法，如果后面出现了稍小的元素，则会试图去填充网格中前面留下的空白。这样做会填上稍大元素留下的空白，可能导致原来出现的次序被打乱。

实例

```
.container {
    display:grid;
    grid-template-columns:repeat(5,50px);
    grid-template-rows:30px 30px;
    grid-auto-flow:row;
}
item-a {
    grid-column: 1;
    grid-row: 1 / 3;
}
.item-e {
    grid-column: 5;
    grid-row: 1 / 3;
}
```

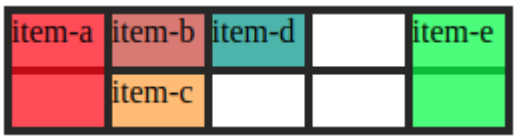
效果图如下： 注意我们没有显式放置的三个网格项（item-b，item-c和item-d）如何在可用行中流动



上面可以看不出有什么作用，但将 `grid-auto-flow` 设为 `column` ，你会发现区别。

```
.container {
    display:grid;
    grid-template-columns: repeat(5,50px);
    grid-template-rows: 30px 30px;
    grid-auto-flow: column;
}
item-a {
    grid-column: 1;
    grid-row: 1 / 3;
}
.item-e {
    grid-column: 5;
    grid-row: 1 / 3;
}
```

效果图如下：



区别：

- 当 grid-auto-flow 设为 row 时，没有显式设置的网格项会先排满一行，当有多余项时，再换行
- 当 grid-auto-flow 设为 column 时，没有显式设置的网格项会先排满一列，当有多余项时，再换列

3.8. grid

grid 是一个CSS简写属性，可以用来设置以下属性： 显式网格属性 grid-template-rows 、 grid-template-columns 和 grid-template-areas ， 隐式网格属性 grid-auto-rows 、 grid-auto-columns 和 grid-auto-flow ， 间距属性 grid-column-gap 和 grid-row-gap 。

grid属性有四种用法：

```
grid: none ①

grid: <grid-template> ②

grid: <grid-template-rows> / [ auto-flow && dense? ] <grid-auto-columns>? ③

grid: [ auto-flow && dense? ] <grid-auto-rows>? / <grid-template-columns> ④
```

① 将所有子属性设置为其初始值

② 与 grid-template 属性定义相同

③ 显式使用 grid-template-rows 设置行轨道（ grid-template-columns 属性自动设为 none ），且将 grid-auto-flow 设为 column;可选的设置 grid-auto-columns 来确定如何重复列轨道，同时 grid-auto-rows 属性设为 auto

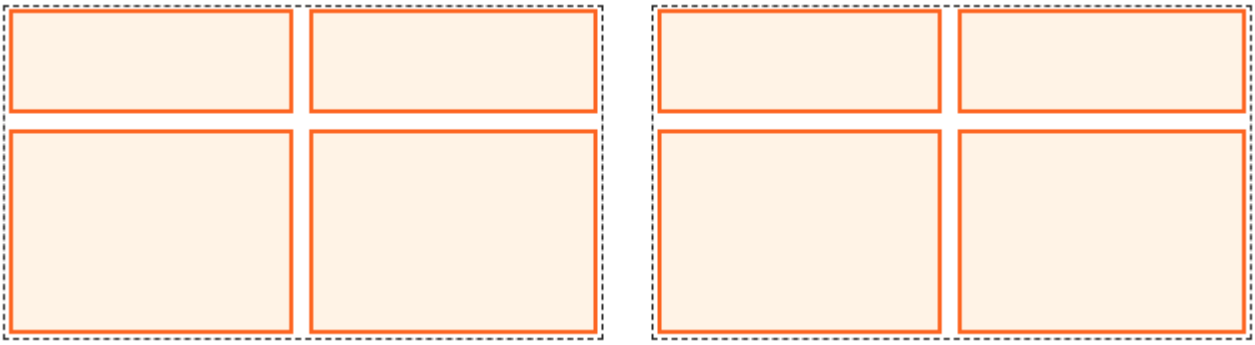
④ 显式使用 grid-template-columns 设置行轨道（ grid-template-rows 属性自动设为 none ），且将 grid-auto-flow 设为 row;可选的设置 grid-auto-rows 来确定如何重复行轨道，同时 grid-auto-columns 属性设为 auto

实例

下面两个例子是等价的。

```
.container {
  grid: 50px 100px / 1fr 1fr;
}
```

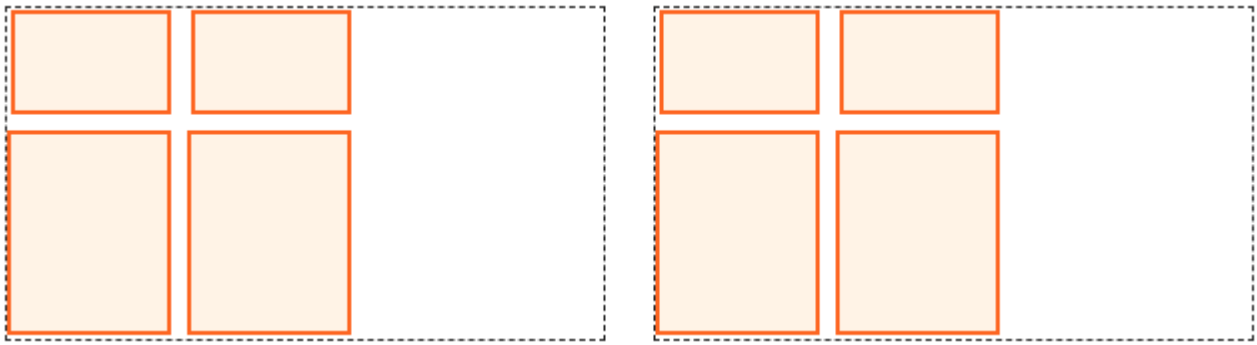
```
.container {
  grid-template-rows: 50px 100px;
  grid-template-columns: 1fr 1fr;
}
```



下面两个例子是等价的。

```
.container {
  grid: 50px 100px / auto-flow 80px;
}
```

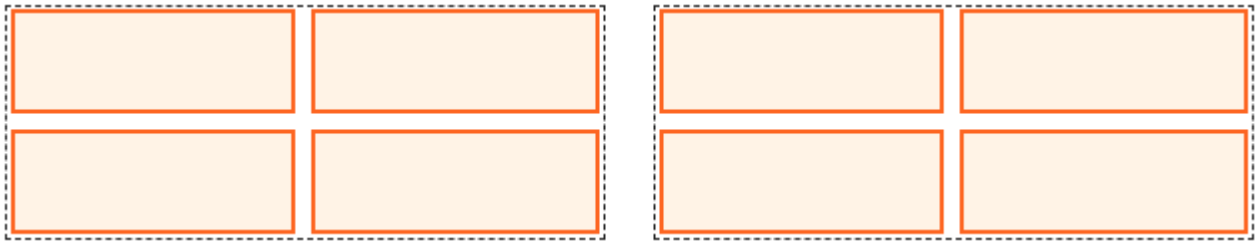
```
.container {
  grid-template-rows: 50px 100px;
  grid-auto-flow: column;
  grid-auto-columns: 80px;
}
```



下面两个例子是等价的。

```
.container {
  grid: auto-flow 50px / 1fr 1fr;
}
```

```
.container {
  grid-template-columns: 1fr 1fr;
  grid-auto-flow: row;
  grid-auto-rows: 50px;
}
```



4. 网格项的属性

这一章来介绍网格项的各个属性。

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

4.1. 网格单元格的起始位置

通过引用特定网格线来改变网格项在网格中的位置。

`grid-column-start/ grid-row-start` 是网格项开始的列/行，`grid-column-end/ grid-row-end` 是网格项结束的列/行。

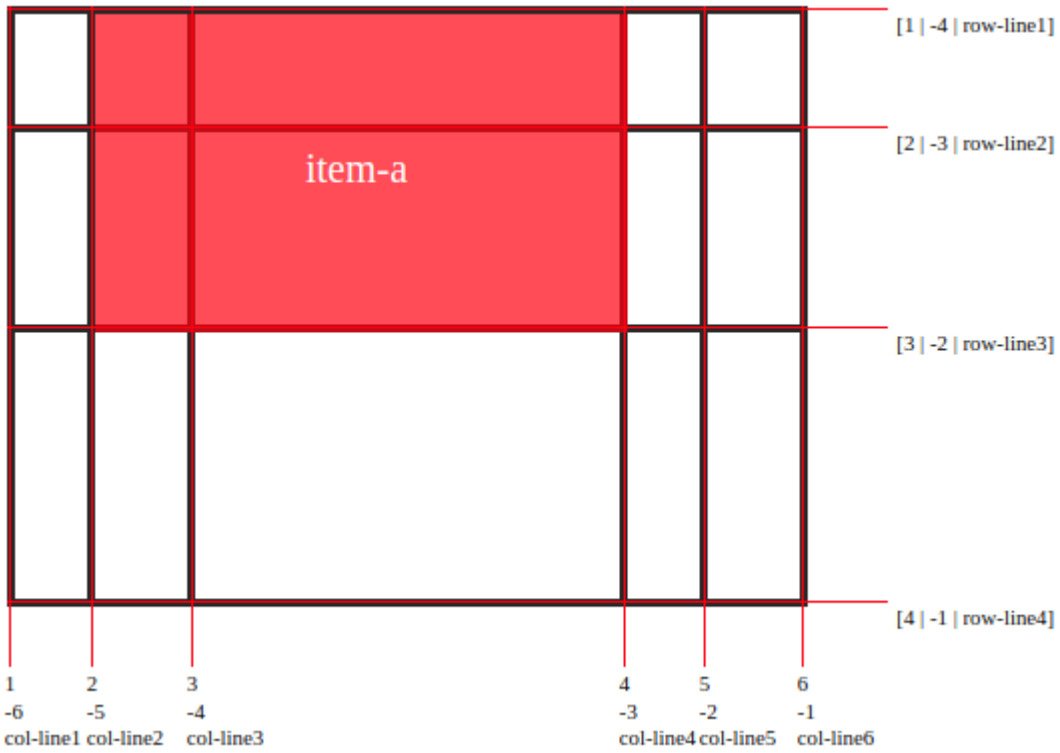
```
.item {
  grid-column-start: <line> | span <number> | span <name> | auto
  grid-column-end: <line> | span <number> | span <name> | auto
  grid-row-start: <line> | span <number> | span <name> | auto
  grid-row-end: <line> | span <number> | span <name> | auto
}
```

属性值：

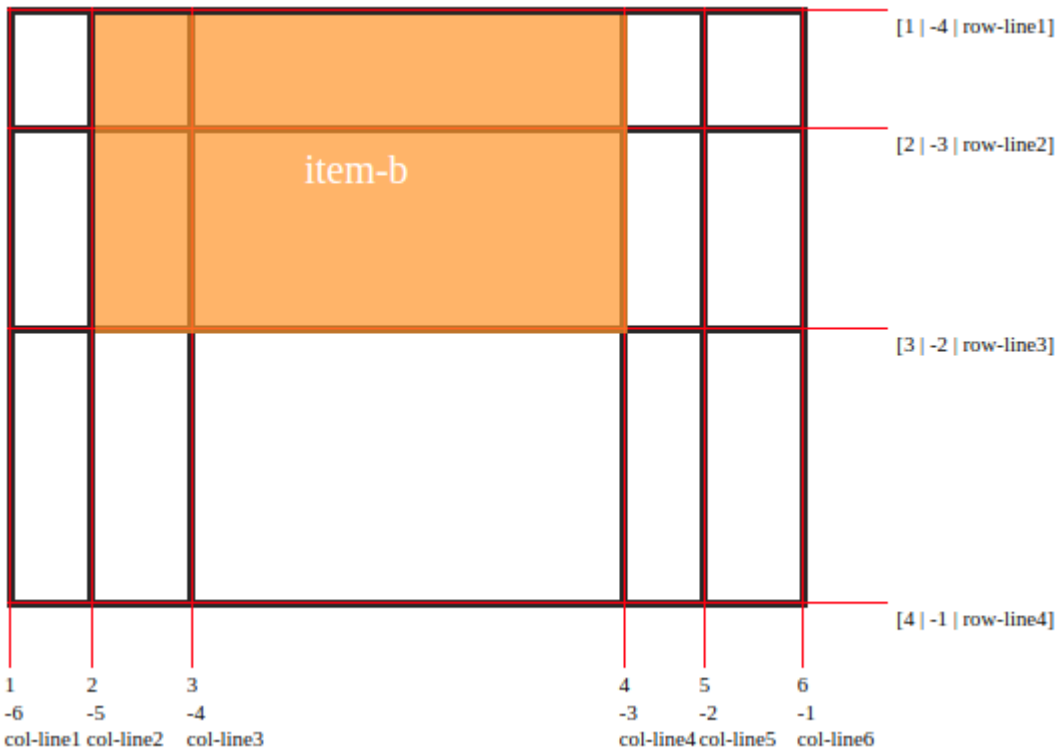
- `<line>` - 可以是一个引用编号网格线的数字，也可以是一个引用命名网格线的名称
- `span <number>` - 该网格项将跨越指定的网格轨道数量
- `span <name>` - 该网格项将跨越，直到它使用提供的名称命中下一行
- `auto` - 表示自动放置，自动跨度或默认跨度

实例

```
.item-a {
    grid-column-start:2;
    grid-column-end:col-line4;
    grid-row-start:-4;
    grid-row-end:3;
}
```




```
.item-b {
    grid-column-start: 2;
    grid-column-end: span col-line4;
    grid-row-start: 1;
    grid-row-end: span 2;
}
```



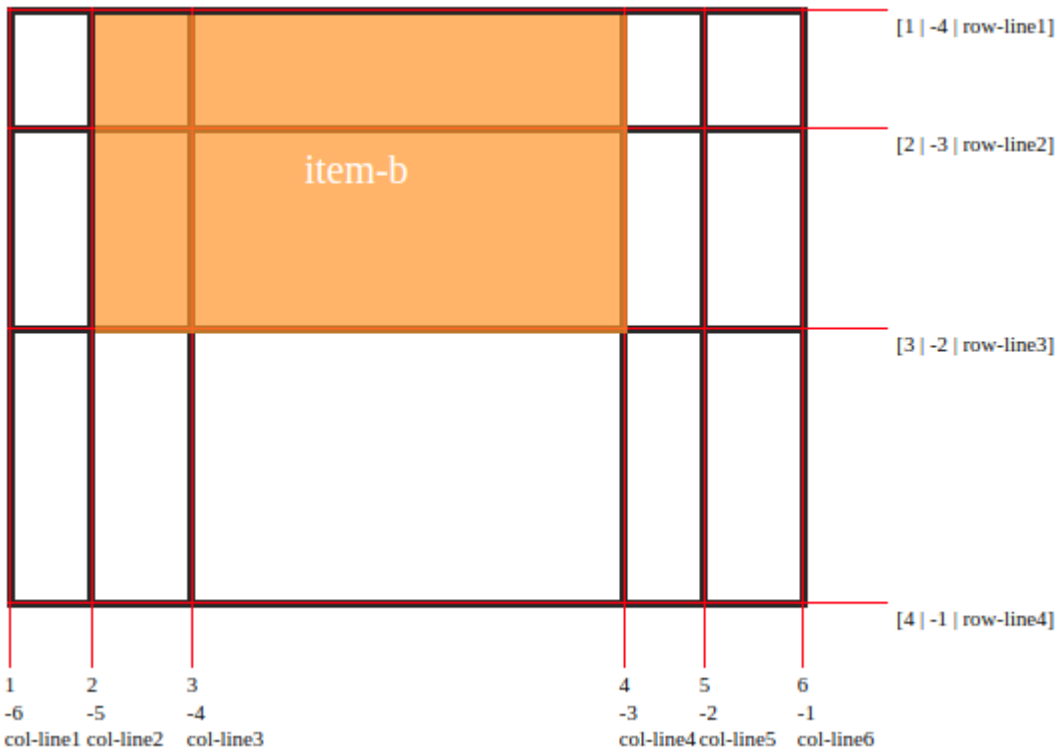
4.1.1. 简写

`grid-column` 为grid-column-start+ grid-column-end的简写; `grid-row` 为grid-row-start+ grid-row-end的简写

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

实例

```
.item-b {
    grid-column: 2 / span col-line4;
    grid-row: 1 / span 2;
}
```


item-b
[1 | -4 | row-line1]
[2 | -3 | row-line2]
[3 | -2 | row-line3]
[4 | -1 | row-line4]
1
-6
col-line1
2
-5
col-line2
3
-4
col-line3
4
-3
col-line4
5
-2
col-line5
6
-1
col-line6

4.2. 网格单元格的区域

在讲解 网格区域 一节中，我们也用到了 grid-area，其用于为网格项指定名称，以便 grid-template-areas 引用。

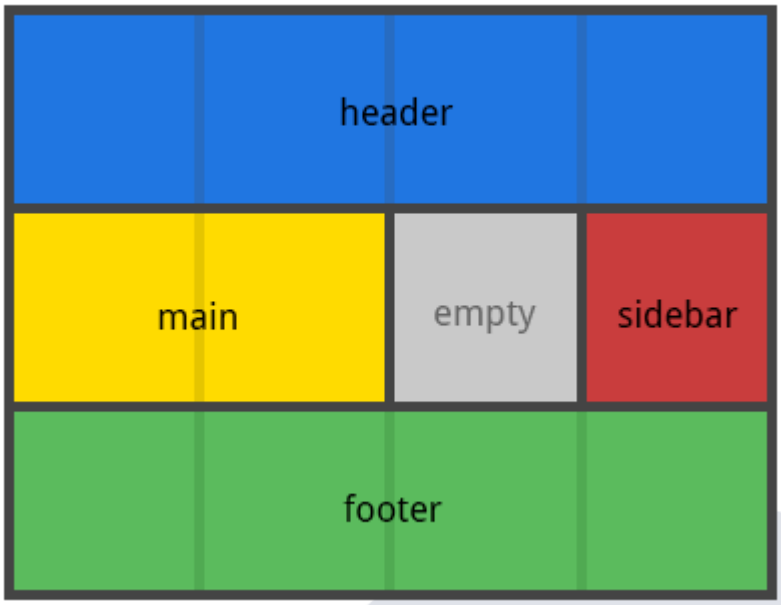
```
.item {
  grid-area: <name>;
}
```

实例

```
.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";
}
```

在上面的代码中，将创建一个三行四列的网格。其中第一行的四列组成header网格区域，第二行由三部分组成，第一个单元格和第二个单元格组成main区域，第三个单元格为空单元格，第四个单元格为sidebar;最后，第三行的四个单元格组成footer网格区域。

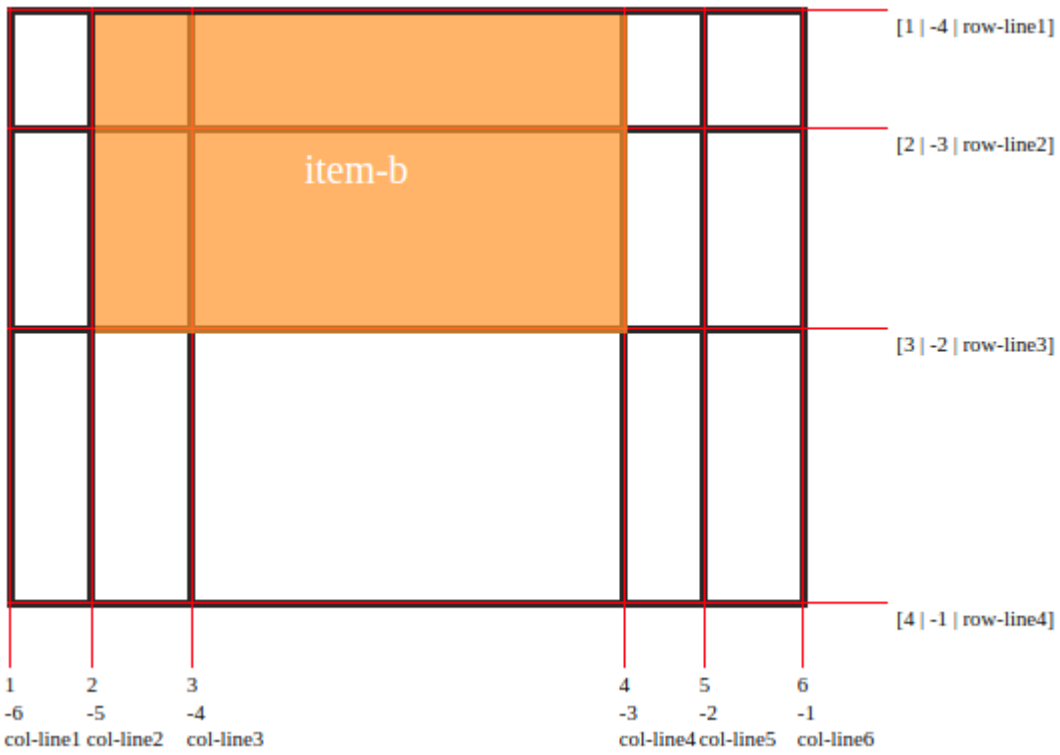


除了指定网格项的名称， `grid-area` 还可以作为 grid-row-start + grid-column-start+ grid-row-end + grid-column-end 的简写

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

实例

```
.item-b {
    grid-area: 1 / 2 / span 2 / span col-line4;
}
```


item-b
[1 | -4 | row-line1]
[2 | -3 | row-line2]
[3 | -2 | row-line3]
[4 | -1 | row-line4]
1
-6
col-line1
2
-5
col-line2
3
-4
col-line3
4
-3
col-line4
5
-2
col-line5
6
-1
col-line6

4.3. 网格项对齐

4.3.1. 沿着行轴对齐（水平对齐）

沿着内联（行）轴对齐网格项：

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

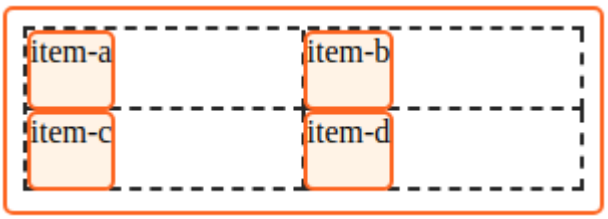
属性值：

- `start` - 将网格项与单元格的起始边缘对齐
- `end` - 将网格项与单元格的结束边缘对齐
- `center` - 将网格项与单元格中心的对齐
- `stretch` - 填充单元格的整个宽度（默认值）

实例

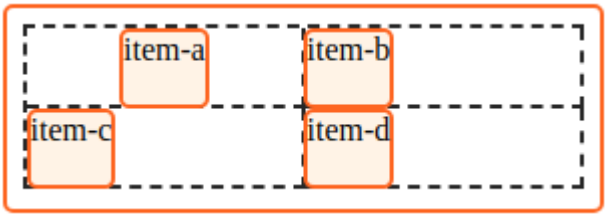
```
.container {
    grid-template-columns:1fr 1fr;
    grid-auto-rows:40px;
    justify-items:start;
}

.item-a {
  justify-self: start;
}
```



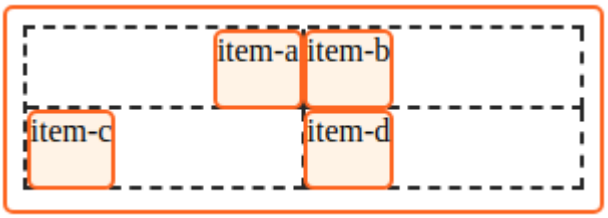
```
.container {
    grid-template-columns:1fr 1fr;
    grid-auto-rows:40px;
    justify-items:start;
}

.item-a {
  justify-self: center;
}
```



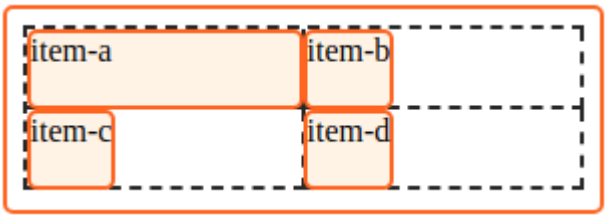

```
.container {
    grid-template-columns:1fr 1fr;
    grid-auto-rows:40px;
    justify-items:start;
}

.item-a {
  justify-self: end;
}
```




```
.container {
    grid-template-columns:1fr 1fr;
    grid-auto-rows:40px;
    justify-items:start;
}

.item-a {
  justify-self: stretch;
}
```



4.3.2. 沿着列轴对齐（垂直对齐）

沿着内联（行）轴对齐网格项：

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

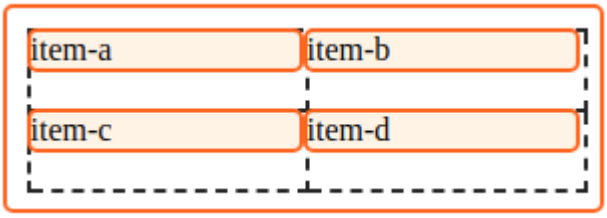
属性值：

- start - 将网格项与单元格的起始边缘对齐
- end - 将网格项与单元格的结束边缘对齐
- center - 将网格项与单元格中心的对齐
- stretch - 填充单元格的整个宽度（默认值）

实例

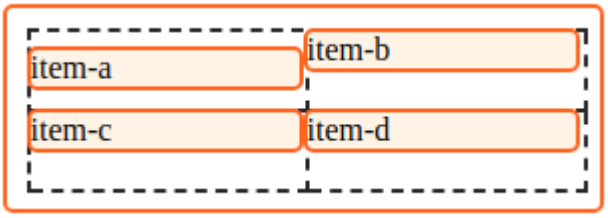
```
.container {
    grid-template-columns:1fr 1fr;
    grid-auto-rows:40px;
    align-items:start;
}

.item-a {
  align-self: start;
}
```



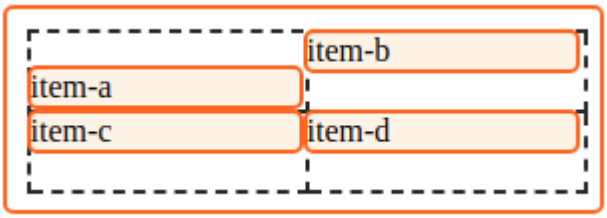
```
.container {
    grid-template-columns:1fr 1fr;
    grid-auto-rows:40px;
    align-items:start;
}

.item-a {
  align-self: center;
}
```



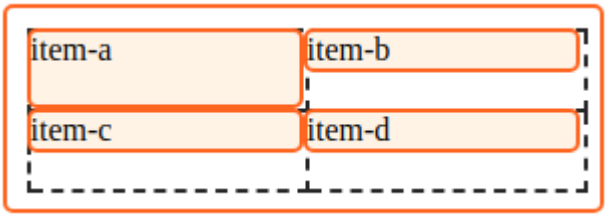

```
.container {
    grid-template-columns:1fr 1fr;
    grid-auto-rows:40px;
    align-items:start;
}

.item-a {
  align-self: end;
}
```




```
.container {
    grid-template-columns:1fr 1fr;
    grid-auto-rows:40px;
    align-items:start;
}

.item-a {
  align-self: stretch;
}
```



对齐简写

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

注： 如果省略第二个值，则将第一个值分配给这两个属性

5. 参考资料

- A Complete Guide to Grid
- CSS Grid Layout
- Grid by Example
- Auto-Sizing Columns in CSS Grid: `auto-fill` vs `auto-fit`
- Responsive layout with CSS grid, part 2: auto-fill & auto-fit