

此页面由社区从英文翻译而来。了解更多并加入 MDN Web Docs 社区。

# 网格

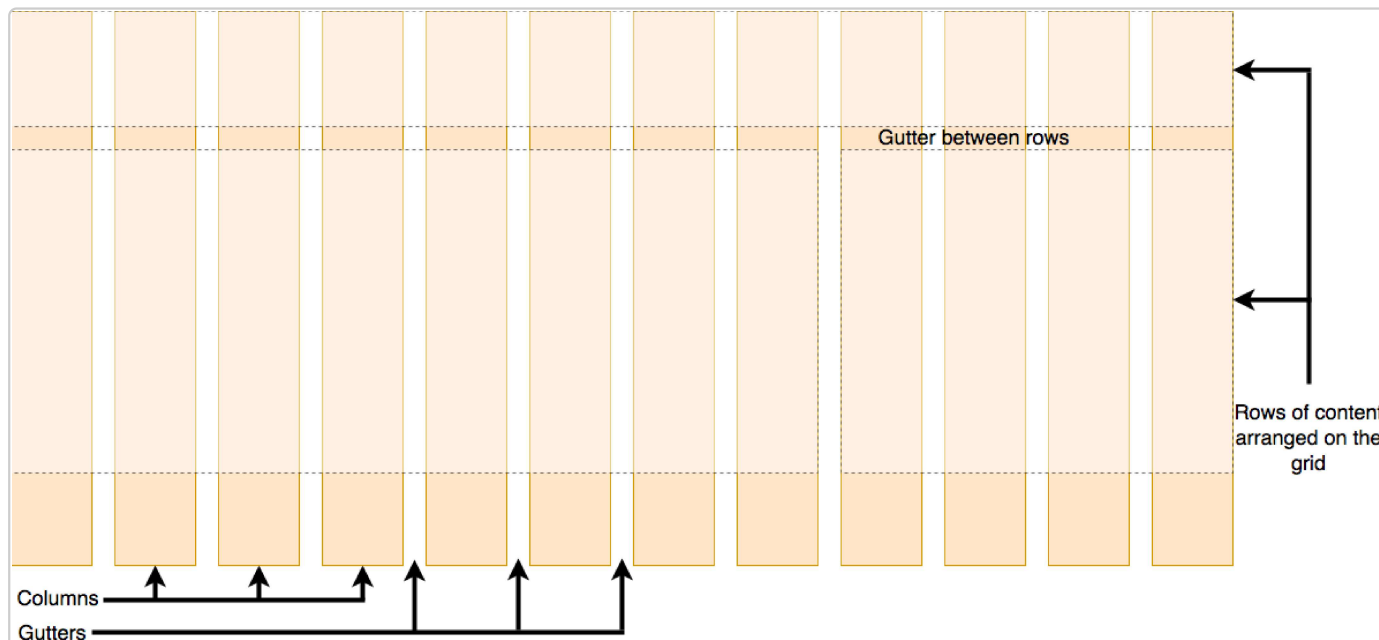
CSS 网格是一个用于 web 的二维布局系统。利用网格，你可以把内容按照行与列的格式进行排版。另外，网格还能非常轻松地实现一些复杂的布局。关于使用网格进行页面排版，这篇文章包含了你需要的一切知识。

前提:	HTML 基础 (学习 <a href="#">HTML 简介</a> )，以及了解 CSS 如何工作的 (学习 <a href="#">CSS 简介</a> 和 <a href="#">盒子样式</a> 。)
目标:	要了解网格布局系统背后的基本概念，以及如何在一個网页上实现一个网格布局。

## 什么是网格布局？

网格是由一系列水平及垂直的线构成的一种布局模式。根据网格，我们能够将设计元素进行排列，帮助我们设计一系列具有固定位置以及宽度的元素的页面，使我们的网站页面更加统一。

一个网格通常具有许多的**列 (column)** 与**行 (row)**，以及行与行、列与列之间的间隙，这个间隙一般被称为**沟槽 (gutter)**。



**备注：** 任何有设计背景的人似乎都感到惊讶，CSS 没有内置的网格系统，而我们似乎使用各种次优方法来创建网格状的设计。正如你将在本文的最后一部分中发现的那样，这将被改变，但是你可能需要知道在未来一段时间内创建网格的现有方法。

## 在 CSS 中创建自己的网格

决定好你的设计所需要的网格后，你可以创建一个 CSS 网格版面并放入各类元素。我们先来看看网格的基础功能，然后尝试做一个简单的网格系统。

下面这个视频提供了一个很好的解释：

## Build a Classic Layout FAST in CSS Grid



### 定义一个网格

和往常一样，你可以下载，然后在文本编辑器中打开并浏览教程的[起始文件](#)（你可以[在这里查看实时的效果](#)）。你会看到一个带有容器的示例，容器中有一些子项。默认情况下，子项按照正常布局流自顶而下排布。在这篇教程的第一部分，我们会从这开始，通过对这个文件做一些改变，来了解网格是如何工作的。

首先，我们通过把容器的 `display` 属性设置为 `grid`，来定义一个网格。与弹性盒子一样，将父容器改为网格布局后，他的直接子项会变为网格项。把下面的 css 规则加到你的文件中。

CSS

Play

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

与弹性盒子不同的是，在定义网格后，网页并不会马上发生变化。因为 `display: grid` 的声明只创建了一个只有一列的网格，所以子项还是会像正常布局流那样，自上而下、一个接一个的排布。

为了让我们的容器看起来更像一个网格，我们要给刚定义的网格加一些列。那就让我们加三个宽度为 `200px` 的列。当然，这里可以用任何长度单位，包括百分比。

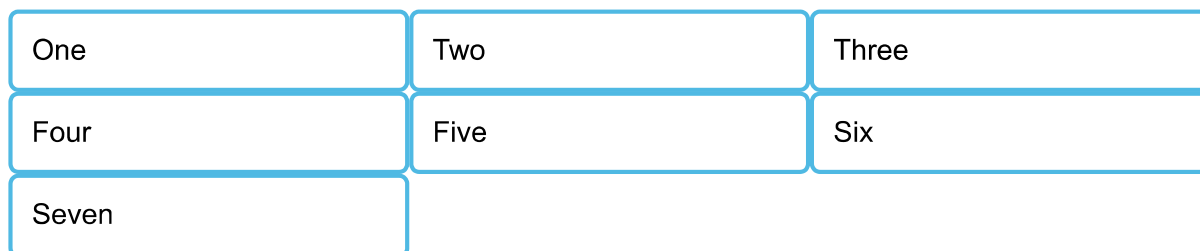
CSS

Play

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

在你的 CSS 中加入第二个规则。刷新页面后，你会看到子项们排进了新定义的网格中。

Play



## 使用 fr 单位的灵活网格

除了长度和百分比，我们也可以用 `fr` 这个单位来灵活地定义网格的行与列的大小。这个单位代表网格容器中可用空间的一份，可能有点抽象，看看下面的例子吧。

使用下面的规则来修改你的网格轨道，创建 3 个宽度为 `1fr` 的列：

CSS

Play

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

将窗口调窄（由于示例中设定了 [max-width](#)，可能需要很窄），你应该能看到每一列的宽度可以会随着可用空间变小而变小。fr 单位按比例划分了可用空间，如果没有理解，可以试着改一下数值，看看会发生什么，比如下面的代码：

CSS

Play

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

这个定义里，第一列被分配了 2fr 可用空间，余下的两列各被分配了 1fr 的可用空间，这会使得第一列的宽度更大。另外，fr 可以与一般的长度单位混合使用。比如设置 `grid-template-columns: 300px 2fr 1fr`，那么第一列宽度是 300px，剩下的两列会根据剩余的可用空间按比例分配。

Play

**备注：** fr 单位分配的是可用空间而非所有空间，所以如果某一格包含的内容变多了，那么整个可用空间就会减少，可用空间是不包括那些已经确定被占用的空间的。

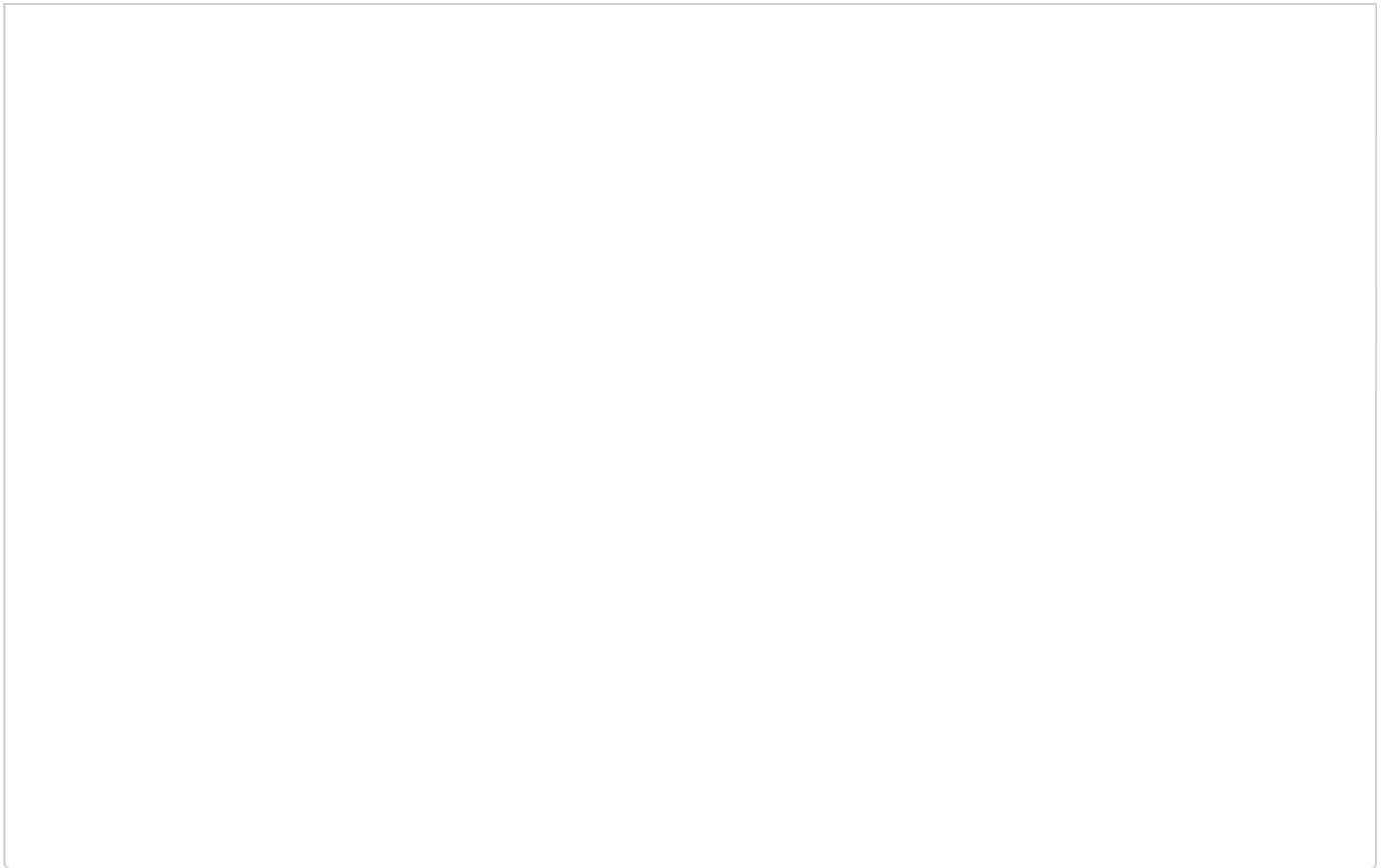
# 网格间隙

使用 [grid-column-gap \(en-US\)](#) 属性来定义列间隙；使用 [grid-row-gap \(en-US\)](#) 来定义行间隙；使用 [grid-gap \(en-US\)](#) 可以同时设定两者。

```
.container {  
  display: grid;  
  grid-template-columns: 2fr 1fr 1fr;  
  grid-gap: 20px;  
}
```

间隙距离可以用任何长度单位包括百分比来表示，但不能使用 fr 单位。

Play



**备注：** gap 属性曾经有一个 grid- 前缀，不过后来的标准进行了修改，目的是让他们能够在不同的布局方法中都能起作用。尽管现在这个前缀不会影响语义，但为了代码的健壮性，你可以把两个属性都写上。

```
.container {  
  display: grid;  
  grid-template-columns: 2fr 1fr 1fr;  
  grid-gap: 20px;  
  gap: 20px;  
}
```

## 重复构建轨道组

你可以使用 `repeat` 来重复构建具有某些宽度配置的某些列。举个例子，如果要创建多个等宽轨道，可以用下面的方法。

CSS

---

```
.container {  
  display: grid;  
  grid-template-columns: repeat(3, 1fr);  
  grid-gap: 20px;  
}
```

和之前一样，你仍然得到了 3 个 `1fr` 的列。第一个传入 `repeat` 函数的值（3）表明了后续列宽的配置要重复多少次，而第二个值（`1fr`）表示需要重复的构建配置，这个配置可以具有多个长度设定。例如 `repeat(2, 2fr 1fr)`，如果你仍然不明白，可以实际测试一下效果，这相当于填入了 `2fr 1fr 2fr 1fr`。

## 显式网格与隐式网格

到目前为止，我们定义过了列，但还没有管过行。但在这之前，我们要来理解一下显式网格和隐式网格。显式网格是我们用 `grid-template-columns` 或 `grid-template-rows` 属性创建的。而隐式网格则是当有内容被放到网格外时才会生成的。显式网格与隐式网格的关系与弹性盒子的 `main` 和 `cross` 轴的关系有些类似。

隐式网格中生成的行/列大小是参数默认是 `auto`，大小会根据放入的内容自动调整。当然，你也可以使用 [grid-auto-rows](#) 和 [grid-auto-columns](#) 属性手动设定隐式网格轨道的大小。下面的例子将 `grid-auto-rows` 设为了 `100px`，然后你可以看到那些隐式网格中的行（因为这个例子里没有设定 [grid-template-rows](#)，因此，所有行都位于隐式网格内）现在都是 100 像素高了。

译者注：简单来说，隐式网格就是为了放显式网格放不下的元素，浏览器根据已经定义的显式网格自动生成的网格部分。

```
.container {  
  display: grid;  
  grid-template-columns: repeat(3, 1fr);  
  grid-auto-rows: 100px;  
  grid-gap: 20px;  
}
```

Play



## minmax() 函数

100 像素高的轨道有时可能会不够用，因为时常会有比 100 像素高的内容加进去。所以，我们可以将其设定为至少 100 像素，并且能够跟随内容来自动拓展尺寸，从而保证能容纳所有内容。显而易见，你很难知道网页上某个元素的尺寸在不同情况下会变成多少，一些额外的内容或者更大的字号就会导致许多能做到像素级精准的设计出现问题。所以，我们有了 [minmax\(\)](#) 函数。

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



```
.container {  
  display: grid;  
  grid-template-columns: repeat(3, 1fr);  
  grid-auto-rows: minmax(100px, auto);  
  grid-gap: 20px;  
}
```

如果所有网格内的内容均小于 100 像素，看起来不会有变化。但如果在某一项中放入很长的内容或者图片，你可以看到这个格子所在的哪一行的高度变成能刚好容纳内容的高度了。注意我们修改的是 `grid-auto-rows`，因此只会作用于隐式网格。当然，这一项属性也可以应用于显式网格，更多内容可以参考 [minmax\(\).](#) 页面。

## 自动使用多列填充

现在来试试把学到的关于网格的一切，包括 `repeat` 与 `minmax` 函数，组合起来，来实现一个非常有用的功能。某些情况下，我们需要让网格自动创建很多列来填满整个容器。通过设置 `grid-template-columns` 属性，我们可以实现这个效果，不过这一次我们会用到 [repeat\(\).](#) 函数中的一个关键字 `auto-fill` 来替代确定的重复次数。而函数的第二个参数，我们使用 [minmax\(\).](#) 函数来设定一个行/列的最小值，以及最大值 `1fr`。


在你的文件中试试看，你也许可以用到以下的代码。

CSS

Play

```
.container {  
  display: grid;  
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));  
  grid-auto-rows: minmax(100px, auto);  
  grid-gap: 20px;  
}
```

Play



你应该能看到形成了一个包含了许多至少 200 像素宽的列的网格，将容器填满。随着容器宽度的改变，网格会自动根据容器宽度进行调整，每一列的宽度总是大于 200 像素，并且容器总会被列填满。（This works because grid is creating as many 200 pixel columns as will fit into the container, then sharing whatever space is leftover between all of the columns — the maximum is 1fr which, as we already know, distributes space evenly between tracks.）

## 基于线的元素放置

在定义完了网格之后，我们要把元素放入网格中。我们的网格有许多分隔线，第一条线的起始点与文档书写模式相关。在英文中，第一条列分隔线（即网格边缘线）在网格的最左边而第一条行分隔线在网格的最上面。而对于阿拉伯语，第一条列分隔线在网格的最右边，因为阿拉伯文是从右往左书写的。

我们根据这些分隔线来放置元素，通过以下属性来指定从那条线开始到哪条线结束。

- [grid-column-start <sub>\(en-US\)</sub>](#)
- [grid-column-end <sub>\(en-US\)</sub>](#)
- [grid-row-start <sub>\(en-US\)</sub>](#)
- [grid-row-end <sub>\(en-US\)</sub>](#)

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

- [grid-column](#)
- [grid-row](#)

注意开始与结束的线的序号要使用 / 符号分开。

下载[这个文件](#)（或者查看[在线预览](#)）。文件中已经定义了一个网格以及一篇简单的文章位于网格之外。你可以看到元素已经被自动放置到了我们创建的网格中。

接下来，尝试用定义网格线的方法将所有元素放置到网格中。将以下规则加入到你的 css 的末尾：

CSS

Play

```
header {  
  grid-column: 1 / 3;  
  grid-row: 1;  
}
```

```
article {  
  grid-column: 2;  
  grid-row: 2;  
}
```

```
aside {  
  grid-column: 1;  
  grid-row: 2;  
}
```

```
footer {  
  grid-column: 1 / 3;  
  grid-row: 3;  
}
```

Play

**备注：** 你也可以用 `-1` 来定位到最后一列分隔线或是行分隔线，并且可以用负数来指定倒数的某一条分隔线。但是这只能用于显式网格，对于隐式网格 `-1` 不一定能定位到最后一条分隔线。

## 使用 `grid-template-areas` 属性放置元素

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

将之前基于线的元素放置代码删除（或者重新下载一份新的文件），然后加入以下 CSS 规则：

CSS

Play

```
.container {  
  display: grid;  
  grid-template-areas:  
    "header header"  
    "sidebar content"  
    "footer footer";  
  grid-template-columns: 1fr 3fr;  
  gap: 20px;
```

```
}

header {
  grid-area: header;
}

article {
  grid-area: content;
}

aside {
  grid-area: sidebar;
}

footer {
  grid-area: footer;
}
```

刷新页面，然后你应该能看到的元素会被放到与之前相同的地方，整个过程不需要我们指定任何分隔线序号。

Play



grid-template-areas 属性的使用规则如下：

- 你需要填满网格的每个格子
- 对于某个横跨多个格子的元素，重复写上那个元素 grid-area 属性定义的区域名字
- 所有名字只能出现在一个连续的区域，不能在不同的位置出现
- 一个连续的区域必须是一个矩形
- 使用 . 符号，让一个格子留空

你可以在文件中尽情发挥你的想象来测试各种网格排版，比如把页脚放在内容之下，或者把侧边栏一直延伸到最底。这种直观的元素放置方式很棒，你在 CSS 中看到的就是实际会出现的排版效果。

## 一个用 CSS 网格实现的网格排版框架

网格排版框架一般由 12 到 16 列的网格构成，你可以用 CSS 网格系统直接实现而不需要任何第三方的工具，毕竟这是标准定义好了的。

下载这个[初始文件](#)，文件中包含了一个定义了 12 列网格的容器。文件中的一些内容我们曾在前面两个示例中使用过，我们暂时可以先用基于线的元素放置模式来将我们的内容放到这个 12 列的网格中。

CSS

Play

```
header {
  grid-column: 1 / 13;
  grid-row: 1;
}

article {
  grid-column: 4 / 13;
  grid-row: 2;
}

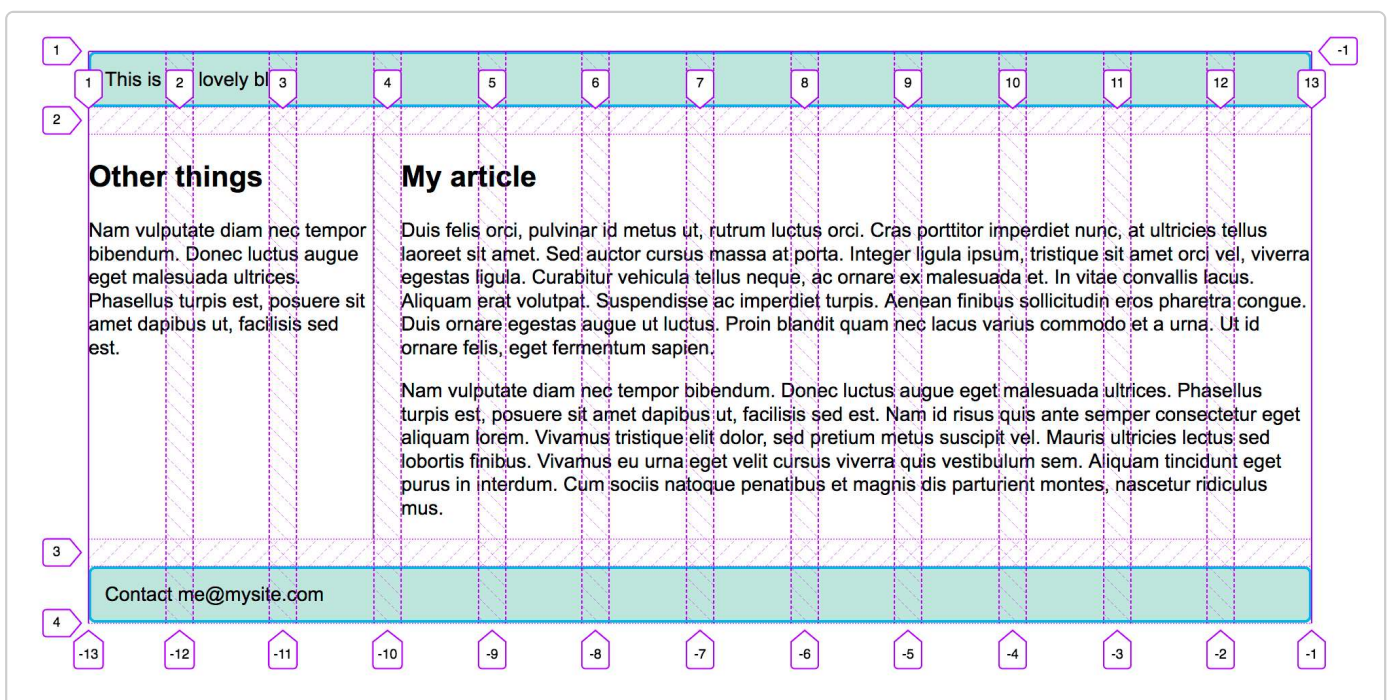
aside {
  grid-column: 1 / 4;
  grid-row: 2;
}

footer {
```

```
grid-column: 1 / 13;  
grid-row: 3;  
}
```

Play

你可以使用[Firefox Grid Inspector \(en-US\)](#) 去查看页面中的网格线，你应该能看到这 12 列的网格是如何工作的。



# 技能测试！

你已经读完了这篇教程，那你记住那些最重要的内容了么？在继续之前，你可以通过一些其他测试来验证你是否真正学习到了这些知识，参见[技能测试：网格](#)。

## 小结

我们在这篇文章中接触了 CSS 网格版面的主要特性，你现在应该可以在你自己的设计中使用了。想深入了解这些内容，你可以读一读下面关于网格版面的文章，可以下面的推荐阅读里看到。

## 推荐阅读

- [CSS 网格指南](#)
- [CSS 网格检查器：检查的你的网格版面 \(en-US\)](#)

### Help improve MDN

Was this page helpful to you?

Yes

No

[Learn how to contribute.](#)

This page was last modified on 2023年11月30日 by [MDN contributors](#).

