> 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。

盒模型

在 CSS 中，所有的元素都被一个个的“盒子”包围着，理解这些“盒子”的基本原理，是我们使用 CSS 实现准确布局、处理元素排列的关键。本文以 CSS 盒模型为主题，你将了解其工作原理和相关术语。

前提：	安装基本的软件，文件处理基本知识，HTML 基础知识（如果不了解 HTML，请移步 HTML 入门），了解 CSS 的工作原理（如果不了解 CSS，请移步 CSS 入门）
目标：	学习盒模型的基本理论、构成以及如何切换到替代盒模型。

区块盒子与行内盒子

在 CSS 中，我们有几种类型的盒子，一般分为**区块盒子**（block boxes）和**行内盒子**（inline boxes）。类型指的是盒子在页面流中的行为方式以及与页面上其他盒子的关系。盒子有**内部显示**（inner display type）和**外部显示**（outer display type）两种类型。

一般来说，可以使用 `display` 属性为显示类型设置各种值，该属性可以有多种值。

外部显示类型

一个拥有 `block` 外部显示类型的盒子会表现出以下行为：

- 盒子会产生换行。
- `width` 和 `height` 属性可以发挥作用。

- 内边距、外边距和边框会将其他元素从当前盒子周围“推开”。
- 如果未指定 `width`，方框将沿行向扩展，以填充其容器中的可用空间。在大多数情况下，盒子会变得与其容器一样宽，占据可用空间的 100%。

某些 HTML 元素，如 `<h1>` 和 `<p>`，默认使用 `block` 作为外部显示类型。

一个拥有 `inline` 外部显示类型的盒子会表现出以下行为：

- 盒子不会产生换行。
- `width` 和 `height` 属性将不起作用。
- 垂直方向的内边距、外边距以及边框会被应用但是不会把其他处于 `inline` 状态的盒子推开。
- 水平方向的内边距、外边距以及边框会被应用且会把其他处于 `inline` 状态的盒子推开。

某些 HTML 元素，如 `<a>`、`<span>`、`<em>` 以及 `<strong>`，默认使用 `inline` 作为外部显示类型。

内部显示类型

盒子还有*内部*显示类型，它决定了盒子内元素的布局方式。

区块和行内布局是网络上的默认行为方式。默认情况下，在没有任何其他指令的情况下，方框内的元素也会以**标准流**的方式布局，并表现为区块或行内盒子。

例如，可以通过设置 `display: flex;` 来更改内部显示类型。该元素仍将使用外部显示类型 `block` 但内部显示类型将变为 `flex`。该方框的任何直接子代都将成为弹性（flex）项，并按照弹性盒子规范执行。

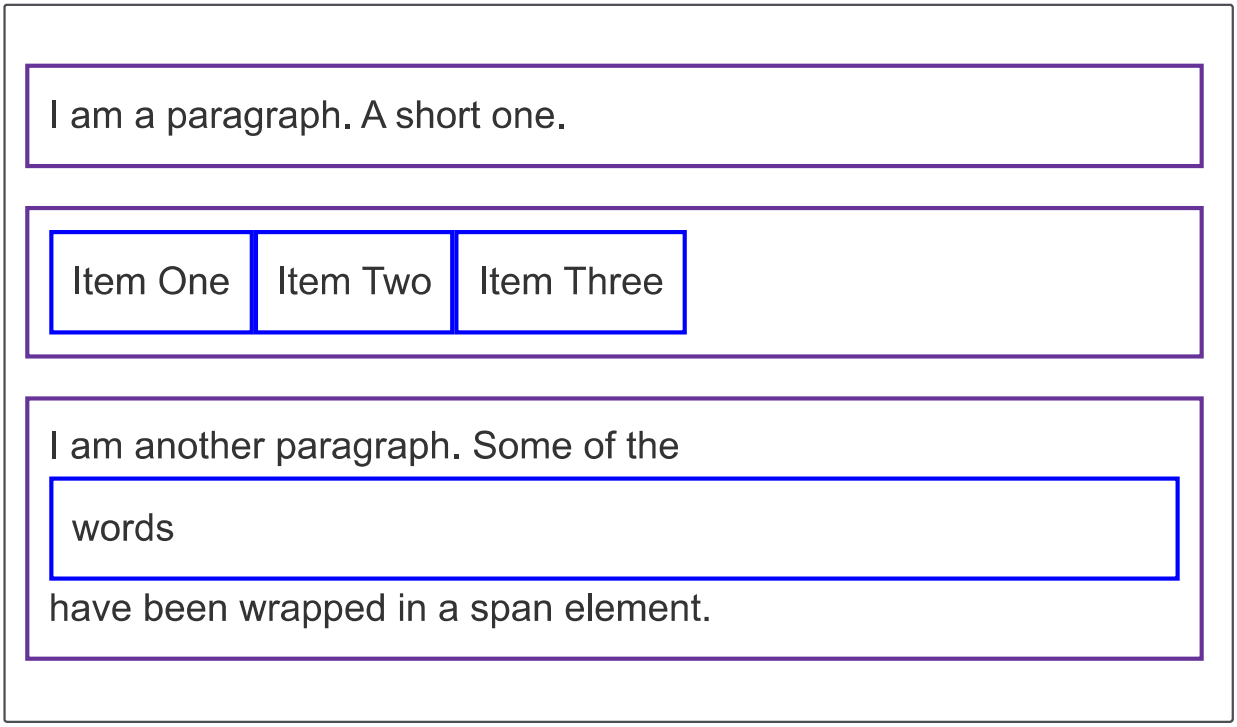
当你继续详细学习 CSS 布局时，将会遇到 `flex` 以及盒子可以具有的其他各种内部值，例如 `grid`。

> **备注：** 想要了解更多有关显示值以及盒子在区块和行内布局中的工作原理，请参阅常规流中的区块和行内布局。

不同显示类型的例子

下面的示例中有三个不同的 HTML 元素，它们的外部显示类型都是 `block`。

- 在 CSS 中添加了边框的段落。浏览器会将其渲染为一个盒子框。段落从新行开始，并扩展整个可用宽度。
- 使用 `display: flex` 布局的列表。这就为容器的子项（即弹性项）建立了弹性布局。列表本身是一个区块盒子，与段落一样，会扩展到整个容器的宽度，然后换行。
- 一个块级段落，内含两个 `<span>` 元素。这些元素通常是 `inline`，但是其中一个元素的类是 `block`，令其被设置为 `display: block`。



Interactive editor

```
p,
ul {
  border: 2px solid rebeccapurple;
  padding: .5em;
}

.block,
li {
  border: 2px solid blue;
  padding: .5em;
}

ul {
  display: flex;
  list-style: none;
}

.block {
  display: block;
}
```

```
<p>I am a paragraph. A short one.</p>
<ul>
  <li>Item One</li>
  <li>Item Two</li>
  <li>Item Three</li>
</ul>
<p>I am another paragraph. Some of the <span class="block">words</span> have been
wrapped in a <span>span element</span>.</p>
```

在下一个示例中，我们可以看到 `inline` 元素是如何表现的。

- 第一段中的 `<span>` 元素默认为行级，因此不会强制换行。
- 设置为 `display: inline-flex` 的 `<ul>` 元素会创建一个行向盒子，其中包含一些弹性项目。
- 这两个段落都设置为 `display: inline`。行向弹性容器和段落都在一行中流动，而不是分成两行（如果它们显示为块级元素，就会这样）。

要在显示模式之间切换，可以将 `display: inline` 更改为 `display: block`，或将 `display: inline-flex` 更改为 `display: flex`。

目前需要记住的关键是更改 `display` 属性的值可以改变框的外部显示类型是区块还是行内。这将改变它在布局中与其他元素一起显示的方式。

什么是 CSS 盒模型?

CSS 盒模型整体上适用于区块盒子，它定义了盒子的不同部分（外边距、边框、内边距和内容）如何协同工作，以创建一个在页面上可以看到的盒子。行内盒子使用的只是盒模型中定义的*部分*行为。

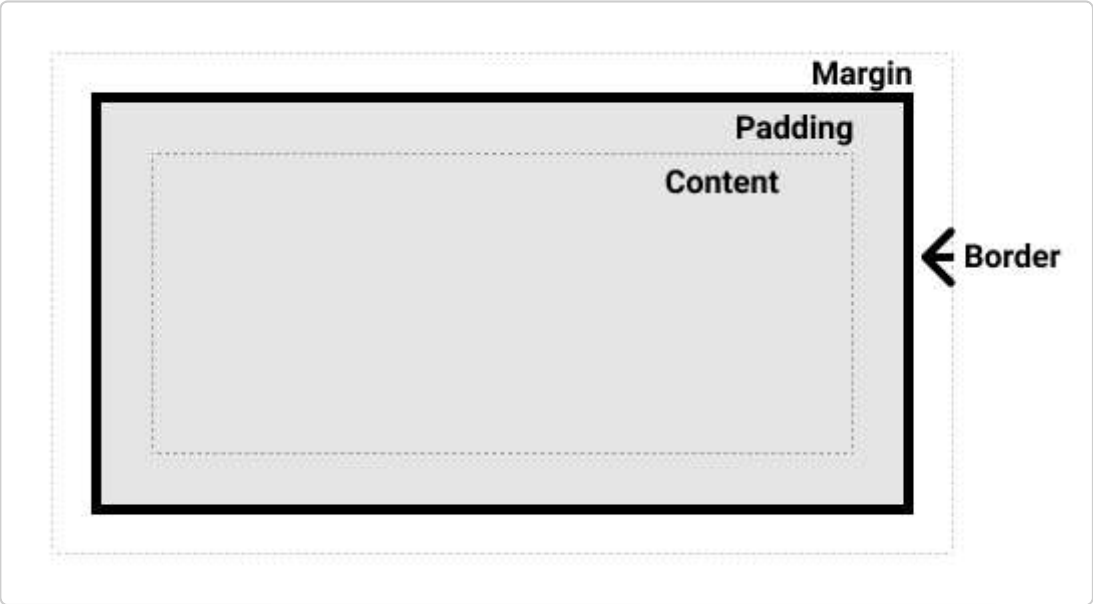
为了增加复杂性，有一种标准盒模型和一种替代盒模型。默认情况下，浏览器使用标准盒模型。

盒模型的各个部分

CSS 中组成一个区块盒子需要：

- **内容盒子**：显示内容的区域；使用 `inline-size` 和 `block-size` 或 `width` 和 `height` 等属性确定其大小。
- **内边距盒子**：填充位于内容周围的空白处；使用 `padding` 和相关属性确定其大小。
- **边框盒子**：边框盒子包住内容和任何填充；使用 `border` 和相关属性确定其大小。
- **外边距盒子**：外边距是最外层，其包裹内容、内边距和边框，作为该盒子与其他元素之间的空白；使用 `margin` 和相关属性确定其大小。

下图显示了这些层次：



CSS 标准盒模型

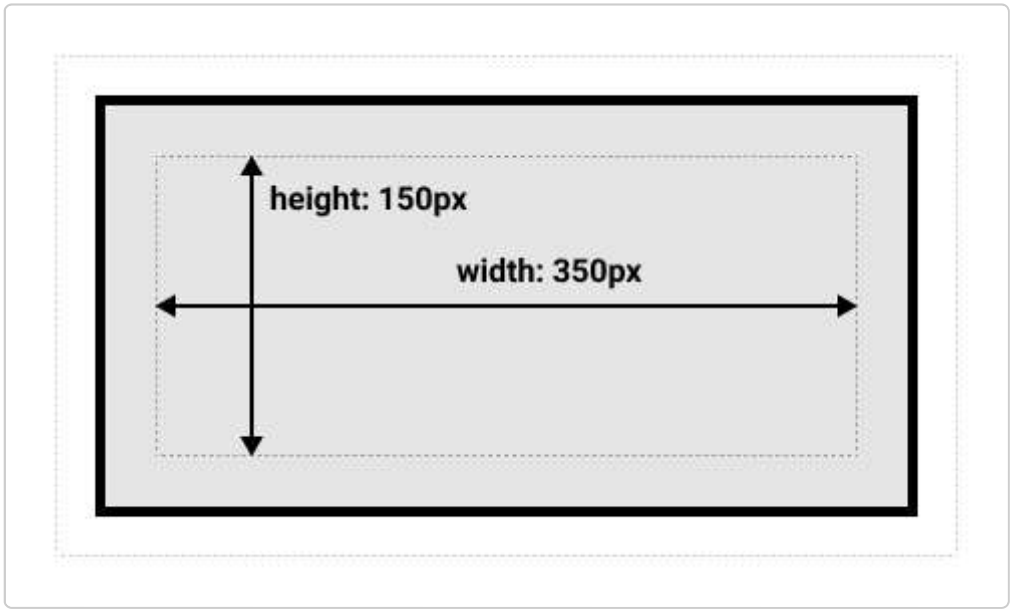
在标准盒模型中，如果在盒子上设置了 `inline-size` 和 `block-size`（或 `width` 和 `height`）属性值，这些值就定义了*内容盒子*的 `inline-size` 和 `block-size`（水平语言中为 `width` 和 `height`）。然后将任何内边距和边框添加到这些尺寸中，以获得盒子所占的总大小（见下图）。

假设一个盒子的 CSS 如下：

CSS

```css
.box {
  width: 350px;
  height: 150px;
  margin: 10px;
  padding: 25px;
  border: 5px solid black;
}
```

方框*实际*占用的空间宽为 410px（350 + 25 + 25 + 5 + 5），高为 210px（150 + 25 + 25 + 5 + 5）。



> **备注：** 外边距不计入盒子的实际大小——当然，它影响盒子在页面上所占的总空间，但只影响盒子外的空间。盒子的面积止于边框，不会延伸到外边距中。

CSS 替代盒模型

在替代盒模型中，任何宽度都是页面上可见方框的宽度。内容区域的宽度是该宽度减去填充和边框的宽度（见下图）。无需将边框和内边距相加，即可获得盒子的实际大小。

要为某个元素使用替代模型，可对其设置 `box-sizing: border-box`：

CSS

```css
.box {
  box-sizing: border-box;
}
```

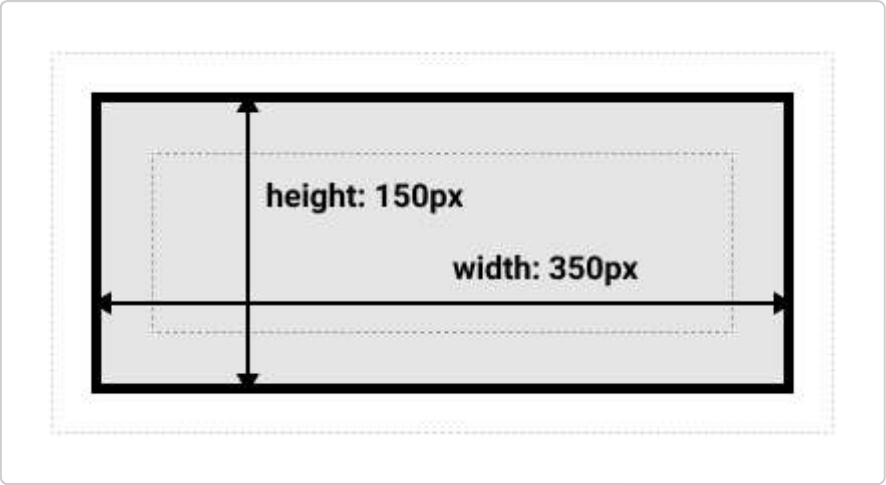
假设一个盒子的 CSS 与上例相同：

CSS

```css
.box {
  width: 350px;
  inline-size: 350px;
  height: 150px;
  block-size: 150px;
  margin: 10px;
  padding: 25px;
```

```
  border: 5px solid black;
}
```

现在，盒子*实际*占用的空间在行向为 350px，在块向为 150px。



要在所有元素中使用替代方框模型（这是开发人员的常见选择），请在 `<html>` 元素上设置 `box-sizing` 属性，并将所有其他元素设置为继承该值：

CSS

```
html {
  box-sizing: border-box;
}

*,
*::before,
*::after {
  box-sizing: inherit;
}
```

要了解基本概念，可以阅读 CSS Tricks 关于盒子尺寸的文章 。

玩转盒模型

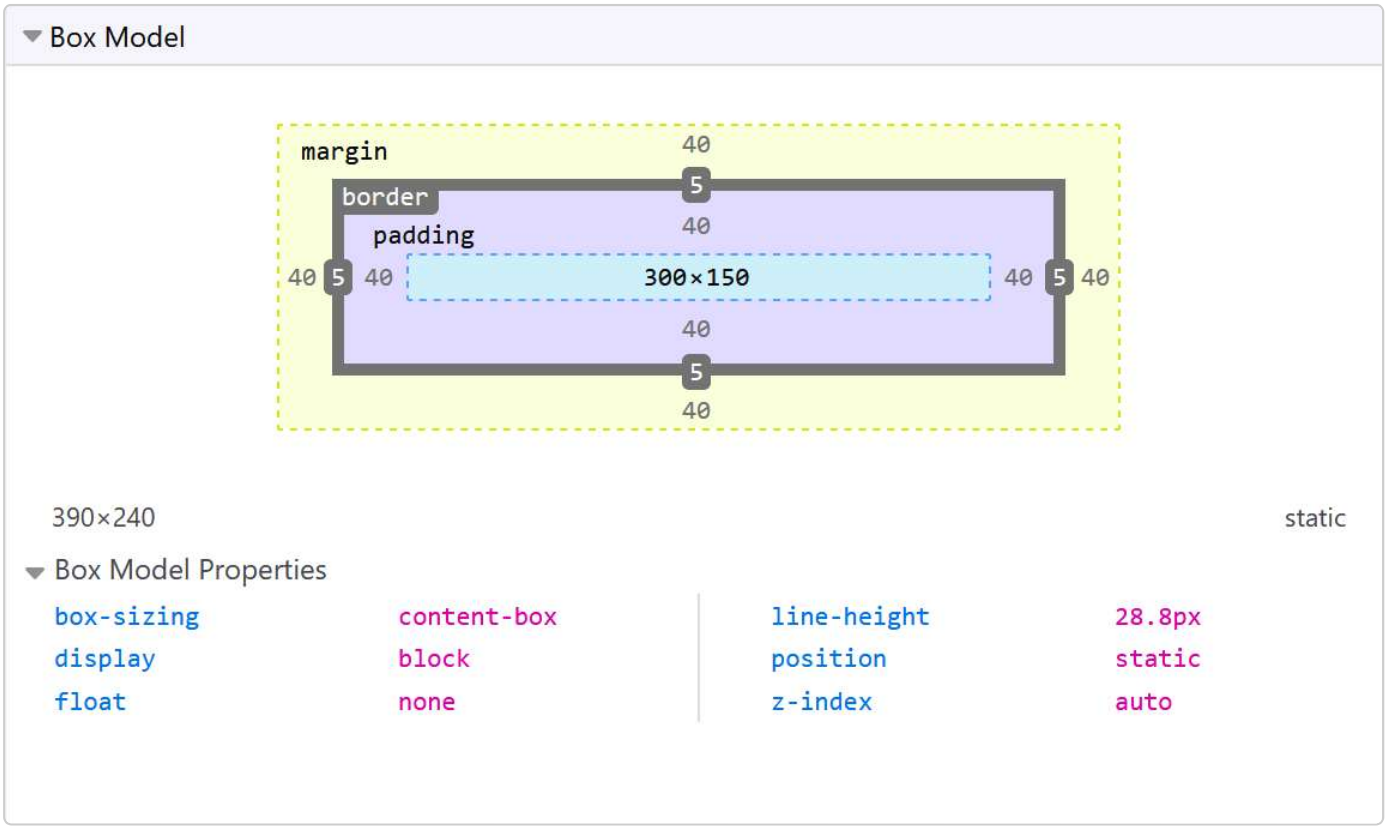
在下面的示例中，可以看到两个盒子。两个盒子的类都是 `.box`，因此具有相同的 `width`、`height`、`margin`、`border` 和 `padding`。唯一不同的是，第二个方框被设置为使用替代盒模型。

你能改变第二个盒子的大小（通过添加 CSS 到 `.alternate` 类中）让它和第一个盒子宽高一样吗？

> **备注：** 在这里 查看该任务的解答。

使用浏览器开发者工具来查看盒模型

浏览器开发者工具可以使你更容易地理解盒模型。如果你在 Firefox 的 DevTools 中查看一个元素，你可以看到元素的大小以及它的外边距、内边距和边框。这是一个很好的检查元素大小的方式，可以便捷的判断你的盒子大小是否符合预期！



外边距、内边距和边框

在上面的示例中，你已经看到了 `margin` 、 `padding` 和 `border` 属性的作用。该示例中使用了**简写属性**，允许我们一次性设置盒子的所有边。这些简写属性也有等效的普通属性，可以单独控制盒子的不同边。

接下来，让我们更详细地探究这些属性。

外边距

外边距是盒子周围一圈看不到的空间。它会把其他元素退推离盒子。外边距属性值可以为正也可以为负。在盒子一侧设置负值会导致盒子和页面上的其他内容重叠。无论使用标准模型还是替代模型，外边距总是在计算可见部分后额外添加。

我们可以使用 `margin` 属性一次性控制一个元素的所有外边距，或者每边单独使用等价的普通属性控制：

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

在下面的示例中，尝试更改外边距的值，来查看当前元素和其包含元素，在外边距设置为正时如何推开周边元素，以及设置为负时，是如何收缩空间的。

外边距折叠

根据外边距相接触的两个元素是正边距还是负边距，结果会有所不同：

- 两个正外边距将合并为一个外边距。其大小等于最大的单个外边距。
- 两个负外边距会折叠，并使用最小（离零最远）的值。

- 如果其中一个外边距为负值，其值将从总值中*减去*。

在下面的示例中，我们有两个段落。最上面一段的 `margin-bottom` 为 50 像素，另一段的 `margin-top` 为 30 像素。页边距折叠在一起，因此方框之间的实际页边距是 50 像素，而不是两个页边距的总和。

你可以通过将第 2 段的 `margin-top` 设置为 0 来测试它。两个段落之间的可见边距不会改变——它保留了第一个段落 `margin-bottom` 设置的 50 像素。如果将其设置为 -10px，你会发现总边距变成了 40px（从 50px 中减去该负值）。

外边距何时折叠，何时不折叠，由许多规则决定。有关详细信息，请参阅掌握外边距折叠。需要记住的主要一点是，外边距折叠是指在使用外边距创建空间时，如果没有获得预期的空间，就会发生外边距折叠。

边框

边框是在边距和填充盒子之间绘制的。如果你正在使用标准的盒模型，边框的大小将添加到框的宽度和高度。如果你使用的是替代盒模型，边框越大会使内容框越小，因为它会占用一些可用的宽度和高度。

为边框设置样式时，有大量的属性可以使用——有四个边框，每个边框都有样式、宽度和颜色，我们可能需要对它们进行操作。

可以使用 `border` 属性一次性设置所有四个边框的宽度、颜色和样式。

欲分别设置每边的属性，可以使用：

- `border-top`
- `border-right`
- `border-bottom`
- `border-left`

欲设置所有边的宽度、样式或颜色，可以使用：

- `border-width`
- `border-style`
- `border-color`

欲设置单条边的宽度、样式或颜色，可以使用最细粒度的普通属性之一：

- `border-top-width`
- `border-top-style`
- `border-top-color`
- `border-right-width`
- `border-right-style`
- `border-right-color`
- `border-bottom-width`
- `border-bottom-style`

- `border-bottom-color`
- `border-left-width`
- `border-left-style`
- `border-left-color`

在下面的示例中，我们使用了各种简写属性和普通属性来创建边框。请尝试使用不同的属性，以了解它们的工作原理。有关边框属性的 MDN 页面提供了有关不同可用边框样式的信息。

内边距

内边距位于边框和内容区域之间，用于将内容推离边框。与外边距不同，内边距不能为负数。任何应用于元素的背景都会显示在内边距后面。

我们可以使用 `padding` 简写属性一次性控制元素所有边，或者每边单独使用等价的普通属性：

- `padding-top`
- `padding-right`
- `padding-bottom`
- `padding-left`

在下面的示例中，你可以更改类 `.box` 上的内边距值，从而看到文本开始的位置与盒子的关系发生了变化。你还可以更改类 `.container` 的内边距，在容器和盒子之间创建空间。你可以更改任何元素的内边距，在其边框和元素内部的任何内容之间创建空间。

盒子模型和行内盒子

以上所有的方法都完全适用于块级盒子。某些属性也适用于行内盒子，例如由 `<span>` 元素创建的盒子。

在下面的示例中，我们在一个段落中使用了 `<span>`，并对其应用了 `width`、 `height`、 `margin`、 `border` 和 `padding`。可以看到，宽度和高度都被忽略了。上下外边距、内边距边框都得到了应用，但不会改变其他内容与行内盒子之间的关系。内边距和边框与段落中的其他文字重叠。左右内边距、外边距和边框会将其他内容从方框中推开。

使用 display: inline-block

`display: inline-block` 是 `display` 的一个特殊值，它提供了介于 `inline` 和 `block` 之间的中间位置。如果不希望项目换行，但又希望它使用 `width` 和 `height` 值并避免出现上述重叠现象，请使用它。

一个元素使用 `display: inline-block`，实现我们需要的块级的部分效果：

- 设置 `width` 和 `height` 属性会生效。

- `padding`、 `margin` 和 `border` 会推开其他元素。

不过，它不会换行，只有在明确添加 `width` 和 `height` 属性后，才会变得比其内容大。

在下一个示例中，我们将 `display: inline-block` 添加到 `<span>` 元素中。尝试将此更改为 `display: block` 或完全删除此行，以查看显示模型中的差异。

当你想通过添加 `padding` 来扩大链接的点击范围时，这个功能就派上用场了。 `<a>` 和 `<span>` 一样是一个行内元素；可以使用 `display: inline-block` 在其上设置内边距，使用户更容易点击链接。

这种情况在导航栏中很常见。下面的导航使用弹性盒显示在同一行中，我们为 `<a>` 元素添加了内边距，因为我们希望能够在 `<a>` 在鼠标移动到上面时改变背景色。内边距似乎覆盖了 `<ul>` 元素

上的边框。这是因为 `<a>` 是一个内联元素。

在带有 `.links-list a` 选择器的规则中添加 `display: inline-block`，你就会看到它是如何通过使其他元素考虑内边距来解决这个问题的。

技能测试！

本文已经结束，你还记得最重要的信息吗？在继续学习之前，可以找到一些进一步的测试来验证你是否保留了这些信息——请参阅技能测试：盒模型 (en-US)。

总结

这就是你需要了解的关于盒子模型的大部分内容。今后，如果你对布局中盒子的大小感到困惑，不妨再回来看看这些内容。

在下一篇文章中，我们将看看如何使用背景和边框来使你的普通盒子看起来更有趣。