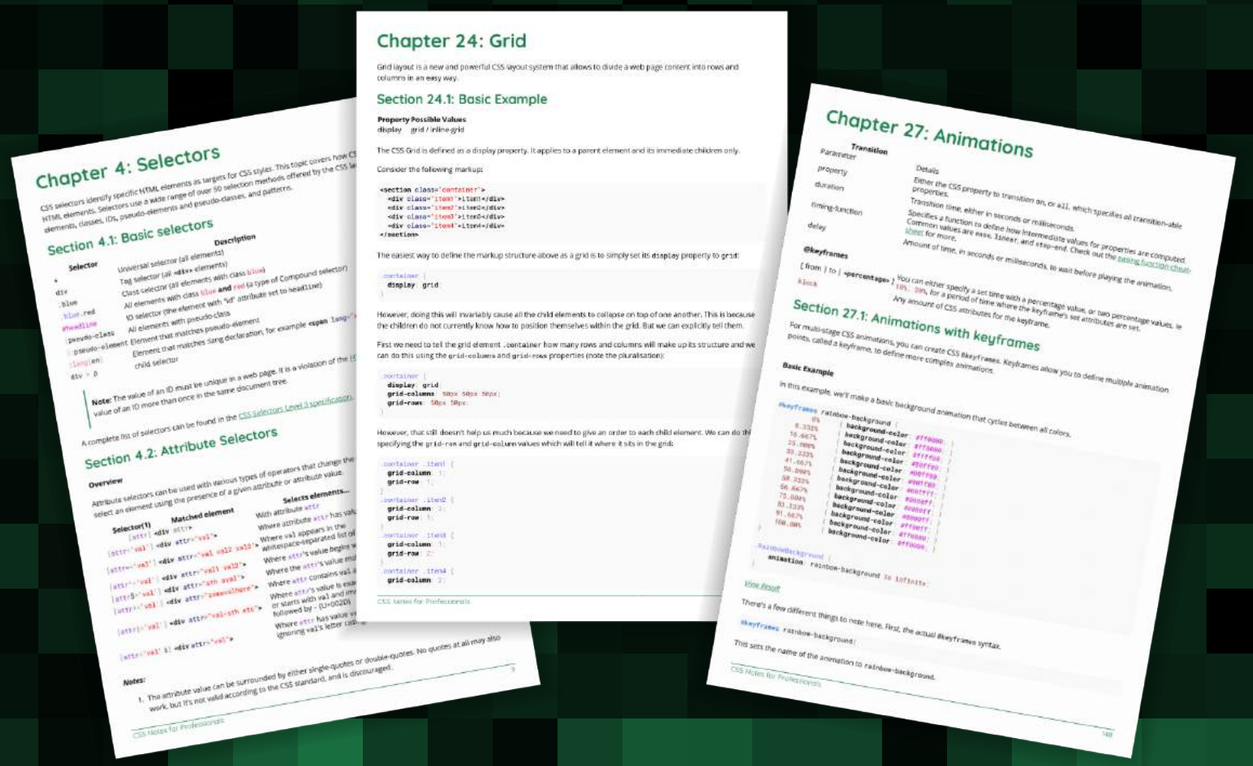
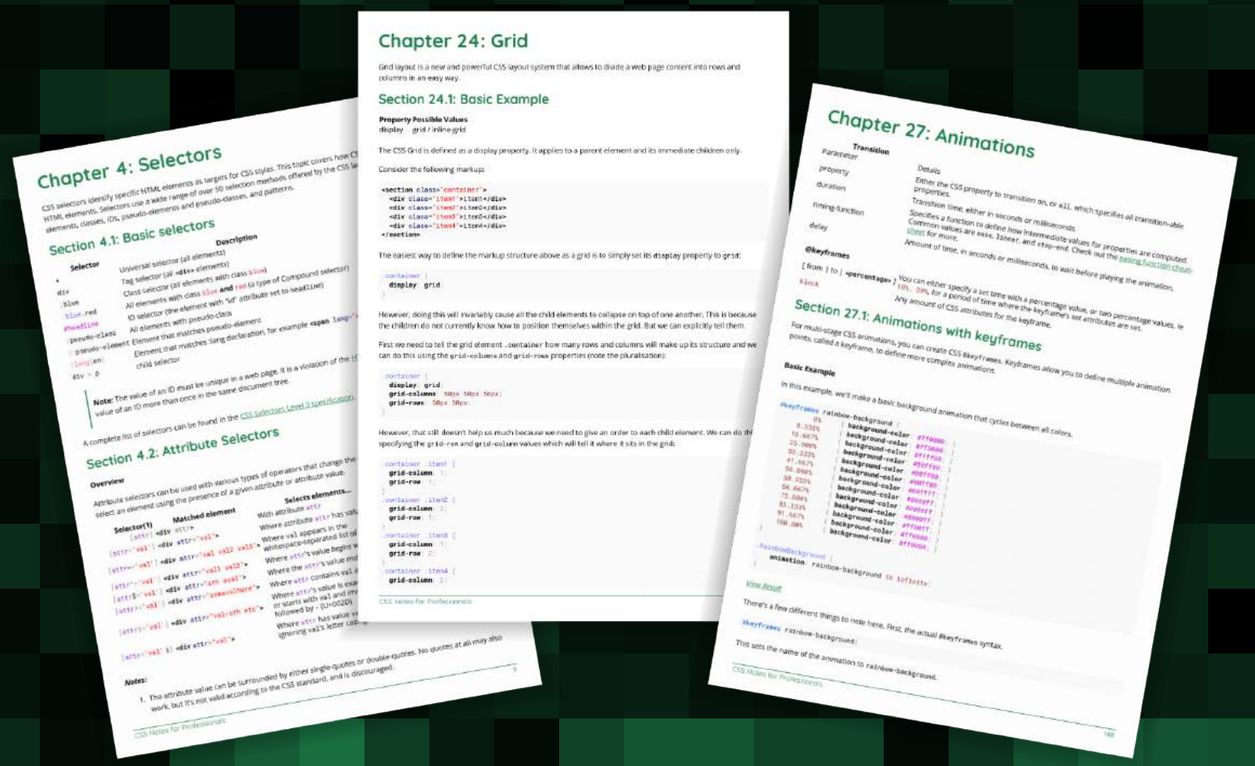
CSS
专业人士笔记



200多页

专业提示和技巧

CSS
Notes for Professionals



200+ pages

of professional hints and tricks

目录

Contents

关于

欢迎随意免费分享此PDF，
本书的最新版本可从以下网址下载：
https://goalkicker.com/CSSBook

请将反馈和更正发送至web@petercv.com

About

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:
https://goalkicker.com/CSSBook

Please send feedback and corrections to web@petercv.com

第1章：CSS入门

版本发布日期

1	1996-12-17
2	1998-05-12
3	2015-10-13

第1.1节：外部样式表

通过在每个HTML文档中放置一个<link>元素，可以将外部CSS样式表应用于任意数量的HTML文档。

<link>标签的属性rel必须设置为"stylesheet"，href属性设置为样式表的相对或绝对路径。虽然通常认为使用相对URL路径是良好做法，但也可以使用绝对路径。在HTML5中，type属性可以省略。

建议将<link>标签放置在HTML文件的<head>标签中，以便样式在其所样式化的元素之前加载。否则，用户将看到无样式内容的闪烁。

示例

hello-world.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <link rel="stylesheet" type="text/css" href="style.css">
    </head>
    <body>
        <h1>你好，世界！</h1>
        <p>我 ♥ CSS</p>
    </body>
</html>
```

style.css

```
h1 {
    color: green;
    text-decoration: underline;
}
p {
    font-size: 25px;
    font-family: 'Trebuchet MS', sans-serif;
}
```

确保在 href 中包含正确的 CSS 文件路径。如果 CSS 文件与 HTML 文件在同一文件夹中，则不需要路径（如上例所示），但如果保存在某个文件夹中，则应这样指定 href="foldername/style.css"。

```
<link rel="stylesheet" type="text/css" href="foldername/style.css">
```

外部样式表被认为是处理 CSS 的最佳方式。原因非常简单：当你管理一个有 100 页的网站，所有页面都由一个样式表控制，而你想要更改你的链接时

Chapter 1: Getting started with CSS

Version Release Date

1	1996-12-17
2	1998-05-12
3	2015-10-13

Section 1.1: External Stylesheet

An external CSS stylesheet can be applied to any number of HTML documents by placing a `<link>` element in each HTML document.

The attribute `rel` of the `<link>` tag has to be set to `"stylesheet"`, and the `href` attribute to the relative or absolute path to the stylesheet. While using relative URL paths is generally considered good practice, absolute paths can be used, too. In HTML5 the `type` attribute can be omitted.

It is recommended that the `<link>` tag be placed in the HTML file's `<head>` tag so that the styles are loaded before the elements they style. Otherwise, users will see a flash of unstyled content.

Example

hello-world.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <link rel="stylesheet" type="text/css" href="style.css">
    </head>
    <body>
        <h1>Hello world!</h1>
        <p>I ♥ CSS</p>
    </body>
</html>
```

style.css

```
h1 {
    color: green;
    text-decoration: underline;
}
p {
    font-size: 25px;
    font-family: 'Trebuchet MS', sans-serif;
}
```

Make sure you include the correct path to your CSS file in the href. If the CSS file is in the same folder as your HTML file then no path is required (like the example above) but if it's saved in a folder, then specify it like this `href="foldername/style.css"`.

```
<link rel="stylesheet" type="text/css" href="foldername/style.css">
```

External stylesheets are considered the best way to handle your CSS. There's a very simple reason for this: when you're managing a site of, say, 100 pages, all controlled by a single stylesheet, and you want to change your link

从蓝色变成绿色，在你的CSS文件中进行更改并让这些更改“级联”
应用到所有100个页面，比进入100个单独页面并重复100次相同的更改要容易得多。同样，如果
你想完全改变你网站的外观，你只需要更新这个文件。

你可以在HTML页面中加载任意数量的CSS文件。

```
<link rel="stylesheet" type="text/css" href="main.css">
<link rel="stylesheet" type="text/css" href="override.css">
```

CSS规则的应用遵循一些基本规则，顺序确实很重要。例如，如果你有一个main.css文件，里面有
一些代码：

```
p.green { color: #00FF00; }
```

所有带有“green”类的段落都会显示为浅绿色，但你可以通过在main.css之后包含另一个.css文件来覆盖它。例如，你可以让override.css包含以下代码，放在main.css之后：

```
p.green { color: #006600; }
```

现在，所有带有“green”类的段落将显示为深绿色，而不是浅绿色。

还有其他原则适用，比如“!important”规则、特异性和继承。

当有人第一次访问你的网站时，他们的浏览器会下载当前页面的HTML以及链接的CSS文件。然后当他们导航到另一个页面时，浏览器只需要下载该页面的HTML；CSS文件会被缓存，因此不需要再次下载。由于浏览器缓存了外部样式表，你的页面加载速度更快。

第1.2节：内部样式

HTML文档中包含在`<style></style>`标签内的CSS，其功能类似于外部样式表，但
它存在于所样式化的HTML文档中，而不是单独的文件，因此只能应用于其所在的
文档。请注意，该元素必须位于<head>元素内以通过HTML验证（尽管如果放在<body>中，
所有当前浏览器仍然可以正常工作）。

```
<head>
    <style>
        h1 {
颜色: 绿色;
            文字装饰: 下划线;
        }
        p {
字体大小: 25像素;
            字体族: 'Trebuchet MS', 无衬线;
        }
    </style>
</head>
<body>
    <h1>你好，世界！</h1>
    <p>我♥CSS</p>
</body>
```


colors from blue to green, it's a lot easier to make the change in your CSS file and let the changes "cascade" throughout all 100 pages than it is to go into 100 separate pages and make the same change 100 times. Again, if you want to completely change the look of your website, you only need to update this one file.

You can load as many CSS files in your HTML page as needed.

```
<link rel="stylesheet" type="text/css" href="main.css">
<link rel="stylesheet" type="text/css" href="override.css">
```

CSS rules are applied with some basic rules, and order does matter. For example, if you have a main.css file with some code in it:

```
p.green { color: #00FF00; }
```

All your paragraphs with the 'green' class will be written in light green, but you can override this with another .css file just by including it *after* main.css. You can have override.css with the following code follow main.css, for example:

```
p.green { color: #006600; }
```

Now all your paragraphs with the 'green' class will be written in darker green rather than light green.

Other principles apply, such as the '!important' rule, specificity, and inheritance.

When someone first visits your website, their browser downloads the HTML of the current page plus the linked CSS file. Then when they navigate to another page, their browser only needs to download the HTML of that page; the CSS file is cached, so it does not need to be downloaded again. Since browsers cache the external stylesheet, your pages load faster.

Section 1.2: Internal Styles

CSS enclosed in `<style></style>` tags within an HTML document functions like an external stylesheet, except that it lives in the HTML document it styles instead of in a separate file, and therefore can only be applied to the document in which it lives. Note that this element *must* be inside the `<head>` element for HTML validation (though it will work in all current browsers if placed in `body`).

```
<head>
    <style>
        h1 {
            color: green;
            text-decoration: underline;
        }
        p {
            font-size: 25px;
            font-family: 'Trebuchet MS', sans-serif;
        }
    </style>
</head>
<body>
    <h1>Hello world!</h1>
    <p>I ♥ CSS</p>
</body>
```

第1.3节：CSS @import规则（CSS的at规则之一）

@import CSS at规则用于从其他样式表导入样式规则。这些规则必须位于除@charset规则之外的所有其他类型规则之前；由于它不是嵌套语句，@import不能用于条件组at规则内部。@import。

如何使用@import

您可以通过以下方式使用@import规则：

A. 使用内部样式标签

```
<style>
 @import url('/css/styles.css');
</style>
```

B. 使用外部样式表

以下代码行将位于根目录下名为additional-styles.css的CSS文件导入到出现该语句的CSS文件中：

```
@import '/additional-styles.css';
```

也可以导入外部CSS。一个常见的用例是字体文件。

```
@import 'https://fonts.googleapis.com/css?family=Lato';
```

@import规则的可选第二个参数是媒体查询列表：

```
@import '/print-styles.css' print;
@import url('landscape.css') screen and (orientation:landscape);
```

第1.4节：内联样式

使用内联样式为特定元素应用样式。请注意，这并非最佳做法。建议将样式规则放在<style>标签或外部CSS文件中，以保持内容与表现的分离。

内联样式会覆盖<style>标签或外部样式表中的任何CSS。虽然在某些情况下这很有用，但通常会降低项目的可维护性。

以下示例中的样式直接应用于其附加的元素。

```
<h1 style="color: green; text-decoration: underline;">你好，世界！</h1>
<p style="font-size: 25px; font-family: 'Trebuchet MS';">我 ♥ CSS</p>
```

内联样式通常是确保在各种邮件客户端、程序和设备上渲染兼容性的最安全方式，但编写起来耗时且管理起来有些挑战。

第1.5节：使用JavaScript更改CSS

纯JavaScript

可以通过元素的style属性使用JavaScript添加、移除或更改CSS属性值。

Section 1.3: CSS @import rule (one of CSS at-rule)

The @import CSS at-rule is used to import style rules from other style sheets. These rules must precede all other types of rules, except @charset rules; as it is not a nested statement, @import cannot be used inside conditional group at-rules. `@import`.

How to use @import

You can use @import rule in following ways:

A. With internal style tag

```
<style>
 @import url('/css/styles.css');
</style>
```

B. With external stylesheet

The following line imports a CSS file named `additional-styles.css` in the root directory into the CSS file in which it appears:

```
@import '/additional-styles.css';
```

Importing external CSS is also possible. A common use case are font files.

```
@import 'https://fonts.googleapis.com/css?family=Lato';
```

An optional second argument to `@import` rule is a list of media queries:

```
@import '/print-styles.css' print;
@import url('landscape.css') screen and (orientation:landscape);
```

Section 1.4: Inline Styles

Use inline styles to apply styling to a specific element. Note that this is ***not*** optimal. Placing style rules in a **`<style>`** tag or external CSS file is encouraged in order to maintain a distinction between content and presentation.

Inline styles override any CSS in a **`<style>`** tag or external style sheet. While this can be useful in some circumstances, this fact more often than not reduces a project's maintainability.

The styles in the following example apply directly to the elements to which they are attached.

```
<h1 style="color: green; text-decoration: underline;">Hello world!</h1>
<p style="font-size: 25px; font-family: 'Trebuchet MS';">I ♥ CSS</p>
```

Inline styles are generally the safest way to ensure rendering compatibility across various email clients, programs and devices, but can be time-consuming to write and a bit challenging to manage.

Section 1.5: Changing CSS with JavaScript

Pure JavaScript

It's possible to add, remove or change CSS property values with JavaScript through an element's `style` property.

```
var el = document.getElementById("element");
el.style.opacity = 0.5;
el.style.fontFamily = 'sans-serif';
```

注意，样式属性名采用小驼峰命名法。在示例中，CSS属性`font-family`在JavaScript中变为`fontFamily`。

作为直接操作元素的替代方法，你可以在JavaScript中创建一个<style>或<link>元素，并将其追加到HTML文档的<body>或<head>中。

jQuery

使用jQuery修改CSS属性更加简单。

```
$('#element').css('margin', '5px');
```

如果您需要更改多个样式规则：

```
$('#element').css({
    margin: "5px",
    padding: "10px",
    color: "black"
});
```

jQuery 包含两种更改带有连字符的 css 规则（例如 `font-size`）的方法。你可以将它们放在引号中，或者使用驼峰命名法来写样式规则名称。

```
$('.example-class').css({
    "background-color": "blue",
    fontSize: "10px"
});
```

另见

- JavaScript 文档 – 读取和更改 CSS 样式。
- jQuery 文档 – CSS 操作

第 1.6 节：使用 CSS 样式化列表

有三种不同的属性用于样式化列表项：list-style-type、list-style-image 和 list-style-position，应该按此顺序声明。默认值分别是 disc、outside 和 none。每个属性可以单独声明，也可以使用list-style简写属性。

list-style-type 定义每个列表项使用的项目符号的形状或类型。

list-style-type 的一些可接受值：

- disc
- circle
- square
- decimal
- lower-roman
- upper-roman
- none

（有关完整列表，请参见W3C规范维基）


```
var el = document.getElementById("element");
el.style.opacity = 0.5;
el.style.fontFamily = 'sans-serif';
```

Note that style properties are named in lower camel case style. In the example you see that the css property `font-family` becomes `fontFamily` in javascript.

As an alternative to working directly on elements, you can create a **`<style>`** or **`<link>`** element in JavaScript and append it to the **`<body>`** or **`<head>`** of the HTML document.

jQuery

Modifying CSS properties with jQuery is even simpler.

```
$('#element').css('margin', '5px');
```

If you need to change more than one style rule:

```
$('#element').css({
    margin: "5px",
    padding: "10px",
    color: "black"
});
```

jQuery includes two ways to change css rules that have hyphens in them (i.e. `font-size`). You can put them in quotes or camel-case the style rule name.

```
$('.example-class').css({
    "background-color": "blue",
    fontSize: "10px"
});
```

See also

- JavaScript documentation – Reading and Changing CSS Style.
- jQuery documentation – CSS Manipulation

Section 1.6: Styling Lists with CSS

There are three different properties for styling list-items: `list-style-type`, `list-style-image`, and `list-style-position`, which should be declared in that order. The default values are disc, outside, and none, respectively. Each property can be declared separately, or using the `list-style` shorthand property.

`list-style-type` defines the shape or type of bullet point used for each list-item.

Some of the acceptable values for `list-style-type`:

- disc
- circle
- square
- decimal
- lower-roman
- upper-roman
- none

(For an exhaustive list, see the W3C specification wiki)

例如，要为每个列表项使用方形项目符号，可以使用以下属性-值对：

```
li {
    list-style-type: square;
}
```

list-style-image 属性决定列表项图标是否设置为图片，并接受 none 或指向图片的 URL 作为值。

```
li {
    list-style-image: url(images/bullet.png);
}
```

list-style-position属性定义了列表项标记的位置，它接受两个值之一：“inside”或“outside”。

```
li {
  list-style-position: inside;
}
```


To use square bullet points for each list-item, for example, you would use the following property-value pair:

```
li {
    list-style-type: square;
}
```

The **list-style-image** property determines whether the list-item icon is set with an image, and accepts a value of none or a URL that points to an image.

```
li {
    list-style-image: url(images/bullet.png);
}
```

The **list-style-position** property defines where to position the list-item marker, and it accepts one of two values: "inside" or "outside".

```
li {
  list-style-position: inside;
}
```

第2章：CSS规则的结构与格式

第2.1节：属性列表

某些属性可以接受多个值，统称为属性列表。

```
/* 该属性列表中有两个值 */
span {
    text-shadow: yellow 0 0 3px, green 4px 4px 10px;
}

/* 备用格式 */
span {
    text-shadow:
        黄色 0 0 3像素,
        绿色 4像素 4像素 10像素;
}
```

第2.2节：多重选择器

当你将CSS选择器分组时，可以对多个不同元素应用相同的样式，而无需在样式表中重复样式。使用逗号分隔多个分组选择器。

```
div, p { color: blue }
```

因此，蓝色会应用于所有 元素和所有<p>元素。没有逗号时，只有作为 子元素的<p>元素才会是红色。

这同样适用于所有类型的选择器。

```
p, .blue, #first, div span{ color : blue }
```

此规则适用于：

- `<p>`
- 蓝色类的元素
- ID为first的元素
- 每个`<span>`标签内的`<div>`

第2.3节：规则、选择器和声明块

一个CSS规则由一个选择器（例如h1）和声明块（{}）组成。

```
h1 {}
```


Chapter 2: Structure and Formatting of a CSS Rule

Section 2.1: Property Lists

Some properties can take multiple values, collectively known as a **property list**.

```
/* Two values in this property list */
span {
    text-shadow: yellow 0 0 3px, green 4px 4px 10px;
}

/* Alternate Formatting */
span {
    text-shadow:
        yellow 0 0 3px,
        green 4px 4px 10px;
}
```

Section 2.2: Multiple Selectors

When you group CSS selectors, you apply the same styles to several different elements without repeating the styles in your style sheet. Use a comma to separate multiple grouped selectors.

```
div, p { color: blue }
```

So the blue color applies to all `<div>` elements and all `<p>` elements. Without the comma only `<p>` elements that are a child of a `<div>` would be red.

This also applies to all types of selectors.

```
p, .blue, #first, div span{ color : blue }
```

This rule applies to:

- `<p>`
- elements of the `blue` class
- element with the ID `first`
- every `<span>` inside of a `<div>`

Section 2.3: Rules, Selectors, and Declaration Blocks

A CSS **rule** consists of a **selector** (e.g. h1) and **declaration block** ({}).

```
h1 {}
```

第3章：注释

第3.1节：单行注释

```
/* 这是一个CSS注释 */
div {
    color: red; /* 这是一个CSS注释 */
}
```

第3.2节：多行注释

```
/*
这是
一个

CSS
    注释
*/
div {
    颜色: 红色;
}
```


Chapter 3: Comments

Section 3.1: Single Line

```
/* This is a CSS comment */
div {
   color: red; /* This is a CSS comment */
}
```

Section 3.2: Multiple Line

```
/*
    This
    is
    a
    CSS
    comment
*/
div {
   color: red;
}
```

第4章：选择器

CSS选择器用于识别特定的HTML元素，作为CSS样式的目标。本主题涵盖了CSS选择器如何定位HTML元素。选择器使用CSS语言提供的50多种选择方法，包括元素、类、ID、伪元素和伪类，以及模式。

第4.1节：基本选择器

选择器	描述
`*`	通用选择器（所有元素）
`div`	标签选择器（所有 元素）
`.blue`	类选择器（所有带有类名blue的元素）
`.blue.red`	所有带有类名blue和red的元素（复合选择器的一种）
`#headline`	ID选择器（“id”属性设置为headline的元素）
`:伪类`	所有带有伪类的元素
`::伪元素`	匹配伪元素的元素
`:lang(en)`	匹配 :lang 声明的元素，例如
`div > p`	子选择器

> 注意： ID 的值在网页中必须唯一。在同一文档树中多次使用相同的 ID 值违反了HTML 标准。

完整的选择器列表可以在CSS 选择器第3级规范中找到。

第4.2节：属性选择器

概述

属性选择器可以与多种类型的运算符一起使用，从而相应地改变选择条件。它们通过给定属性或属性值的存在来选择元素。

选择器(1)	匹配元素	选择元素...	CSS 版本
`[attr]`	<div attr>	具有属性 attr	2
`[attr='val']`		属性 attr 的值为 val	2
`[attr~='val']`		其中 val 出现在以空格分隔的 attr 列表中	2
`[attr^='val']`		属性 attr 的值以 val 开头	3
`[attr$='val']`		当attr的值以val结尾时	3
`[attr*='val']`		当attr的值在任意位置包含val	3
`[attr|='val']`		当attr的值完全等于val，或者以val开头且紧接着是- (U+002D)时	2
`[attr='val' i]`		当attr的值为val，忽略val的字母大小写。	4(2)

注意：

1. 属性值可以用单引号或双引号括起来。也可以不加引号，但这不符合CSS标准，且不推荐使用。

Chapter 4: Selectors

CSS selectors identify specific HTML elements as targets for CSS styles. This topic covers how CSS selectors target HTML elements. Selectors use a wide range of over 50 selection methods offered by the CSS language, including elements, classes, IDs, pseudo-elements and pseudo-classes, and patterns.

Section 4.1: Basic selectors

Selector	Description
`*`	Universal selector (all elements)
`div`	Tag selector (all `<div>` elements)
`.blue`	Class selector (all elements with class `blue`)
`.blue.red`	All elements with class `blue` **and** `red` (a type of Compound selector)
`#headline`	ID selector (the element with "id" attribute set to `headline`)
`:pseudo-class`	All elements with pseudo-class
`::pseudo-element`	Element that matches pseudo-element
`:lang(en)`	Element that matches :lang declaration, for example `<span lang="en">`
`div > p`	child selector

> **Note:** The value of an ID must be unique in a web page. It is a violation of the HTML standard to use the value of an ID more than once in the same document tree.

A complete list of selectors can be found in the CSS Selectors Level 3 specification.

Section 4.2: Attribute Selectors

Overview

Attribute selectors can be used with various types of operators that change the selection criteria accordingly. They select an element using the presence of a given attribute or attribute value.

Selector(1)	Matched element	Selects elements...	CSS Version
`[attr]`	`<div attr>`	With attribute `attr`	2
`[attr='val']`	`<div attr="val">`	Where attribute `attr` has value `val`	2
`[attr~='val']`	`<div attr="val val2 val3">`	Where `val` appears in the whitespace-separated list of `attr`	2
`[attr^='val']`	`<div attr="val1 val2">`	Where `attr`'s value *begins* with `val`	3
`[attr$='val']`	`<div attr="sth aval">`	Where the `attr`'s value *ends* with `val`	3
`[attr*='val']`	`<div attr="somevalhere">`	Where `attr` contains `val` anywhere	3
`[attr|='val']`	`<div attr="val-sth etc">`	Where `attr`'s value is exactly `val`, or starts with `val` and immediately followed by - (U+002D)	2
`[attr='val' i]`	`<div attr="val">`	Where `attr` has value `val`, ignoring `val`'s letter casing.	4(2)

Notes:

1. The attribute value can be surrounded by either single-quotes or double-quotes. No quotes at all may also work, but it's not valid according to the CSS standard, and is discouraged.

2. 没有单一的、整合的 CSS4 规范，因为它被拆分成了多个独立的模块。不过，存在“4级”模块。查看浏览器支持。

详情

[属性]

选择具有指定属性的元素。

```
div[data-color] {
  颜色: 红色;
}
```

```
<div data-color="red">这将是红色</div>
<div data-color="green">这将是绿色</div>
<div data-background="red">这不会是红色</div>
```

JSBin 在线演示

[属性="值"]

选择具有指定属性和值的元素。

```
div[data-color="red"] {
  颜色: 红色;
}
```

```
<div data-color="red">这将是红色</div>
<div data-color="green">这将不是红色</div>
<div data-color="blue">这将不是红色</div>
```

JSBin 在线演示

[attribute*="value"]

选择具有指定属性和值的元素，其中指定的属性包含该值的任意位置（作为子字符串）。

```
[class*="foo"] {
  颜色: 红色;
}
```

```
<div class="foo-123">这将是红色</div>
<div class="foo123">这将是红色</div>
<div class="bar123foo">这将是红色</div>
<div class="barfooo123">这将是红色</div>
<div class="barfo0">这将不是红色</div>
```

JSBin 在线演示

[attribute~="value"]

选择具有指定属性和值的元素，其中该值出现在以空格分隔的列表中。

```
[class~="color-red"] {
  颜色: 红色;
}
```

```
<div class="color-red foo-bar the-div">这将是红色</div>
<div class="color-blue foo-bar the-div">这将不是红色</div>
```


2. There is no single, integrated CSS4 specification, because it is split into separate modules. However, there are "level 4" modules. See browser support.

Details

[attribute]

Selects elements with the given attribute.

```
div[data-color] {
  color: red;
}
```

```
<div data-color="red">This will be red</div>
<div data-color="green">This will be red</div>
<div data-background="red">This will NOT be red</div>
```

Live Demo on JSBin

[attribute="value"]

Selects elements with the given attribute and value.

```
div[data-color="red"] {
  color: red;
}
```

```
<div data-color="red">This will be red</div>
<div data-color="green">This will NOT be red</div>
<div data-color="blue">This will NOT be red</div>
```

Live Demo on JSBin

[attribute*="value"]

Selects elements with the given attribute and value where the given attribute contains the given value anywhere (as a substring).

```
[class*="foo"] {
  color: red;
}
```

```
<div class="foo-123">This will be red</div>
<div class="foo123">This will be red</div>
<div class="bar123foo">This will be red</div>
<div class="barfooo123">This will be red</div>
<div class="barfo0">This will NOT be red</div>
```

Live Demo on JSBin

[attribute~="value"]

Selects elements with the given attribute and value where the given value appears in a whitespace-separated list.

```
[class~="color-red"] {
  color: red;
}
```

```
<div class="color-red foo-bar the-div">This will be red</div>
<div class="color-blue foo-bar the-div">This will NOT be red</div>
```

JSBin 在线演示

[attribute^="value"]

选择具有指定属性和值的元素，其中指定的属性以该值开头。

```
[class^="foo-"] {
  颜色: 红色;
}
```

```
<div class="foo-123">这将是红色</div>
<div class="foo-234">这将是红色</div>
<div class="bar-123">这将不是红色</div>
```

JSBin 在线演示

[attribute$="value"]

选择具有指定属性和值的元素，其中指定的属性以给定值结尾。

```
[class$="file"] {
  颜色: 红色;
}
```

```
<div class="foobar-file">这将是红色</div>
<div class="foobar-file">这将是红色</div>
<div class="foobar-input">这将不是红色</div>
```

JSBin 在线演示

[attribute|="value"]

选择具有指定属性和值的元素，其中属性值要么完全等于给定值，要么完全等于给定值后跟-（U+002D）

```
[lang|="EN"] {
  颜色: 红色;
}
```

```
<div lang="EN-us">这将是红色</div>
<div lang="EN-gb">这将是红色</div>
<div lang="PT-pt">这将不是红色</div>
```

JSBin 在线演示

[attribute="value" i]

选择具有给定属性和值的元素，其中属性值可以表示为Value、VALUE、vAlUe或任何其他不区分大小写的可能形式。

```
[lang="EN" i] {
  颜色: 红色;
}
```

```
<div lang="EN">这将是红色</div>
<div lang="en">这将是红色</div>
<div lang="PT">这将不是红色</div>
```

JSBin 在线演示

Live Demo on JSBin

[attribute^="value"]

Selects elements with the given attribute and value where the given attribute begins with the value.

```
[class^="foo-"] {
  color: red;
}
```

```
<div class="foo-123">This will be red</div>
<div class="foo-234">This will be red</div>
<div class="bar-123">This will NOT be red</div>
```

Live Demo on JSBin

[attribute$="value"]

Selects elements with the given attribute and value where the given attribute ends with the given value.

```
[class$="file"] {
  color: red;
}
```

```
<div class="foobar-file">This will be red</div>
<div class="foobar-file">This will be red</div>
<div class="foobar-input">This will NOT be red</div>
```

Live Demo on JSBin

[attribute|="value"]

Selects elements with a given attribute and value where the attribute's value is exactly the given value or is exactly the given value followed by - (U+002D)

```
[lang|="EN"] {
  color: red;
}
```

```
<div lang="EN-us">This will be red</div>
<div lang="EN-gb">This will be red</div>
<div lang="PT-pt">This will NOT be red</div>
```

Live Demo on JSBin

[attribute="value" i]

Selects elements with a given attribute and value where the attribute's value can be represented as Value, VALUE, vAlUe or any other case-insensitive possibility.

```
[lang="EN" i] {
  color: red;
}
```

```
<div lang="EN">This will be red</div>
<div lang="en">This will be red</div>
<div lang="PT">This will NOT be red</div>
```

Live Demo on JSBin

属性选择器的特异性
0-1-0

与类选择器和伪类相同。

```
*[type=checkbox] // 0-1-0
```

请注意，这意味着属性选择器可以用来选择一个元素的ID，但其特异性低于使用ID选择器选择该元素：[id="my-ID"] 选择与 #my-ID 相同的元素，但特异性较低。

详情请参见语法部分。

第4.3节：组合器

概述

选择器	描述
`div span`	后代选择器（所有作为 后代的）
`div > span`	子选择器（所有作为** **直接子元素的****）
`a ~ span`	通用兄弟选择器（所有作为**<a>**之后的兄弟元素的****）
`a + span`	相邻兄弟选择器（所有紧接在**<a>**之后的****）

> 注意：兄弟选择器定位的是在源文档中位于其后的元素。CSS本质上（层叠性）无法定位之前或父级元素。然而，使用flex的order属性，可以在视觉媒体上模拟之前的兄弟选择器。

后代组合器：选择器 选择器

后代组合器由至少一个空格字符（ ）表示，选择作为定义元素后代的元素。该组合器选择该元素的所有后代（从子元素开始向下）。

```
div p {
  color:red;
}
```

```
<div>
  <p>我的文字是红色</p>
  <section>
    <p>我的文字是红色</p>
  </section>
</div>

<p>我的文字不是红色</p>
```

JSBin 在线演示

在上述示例中，前两个<p>元素被选中，因为它们都是 的后代。

子代组合器：选择器>选择器

子代（>）组合器用于选择作为指定元素的子元素或直接后代的元素。

Specificity of attribute selectors
0-1-0

Same as class selector and pseudoclass.

```
*[type=checkbox] // 0-1-0
```

Note that this means an attribute selector can be used to select an element by its ID at a lower level of specificity than if it was selected with an ID selector: `[id="my-ID"]` targets the same element as `#my-ID` but with lower specificity.

See the Syntax Section for more details.

Section 4.3: Combinators

Overview

Selector	Description
`div span`	Descendant selector (all ****s that are descendants of a ** **)
`div > span`	Child selector (all ****s that are a direct child of a ** **)
`a ~ span`	General Sibling selector (all ****s that are siblings after an **<a>**)
`a + span`	Adjacent Sibling selector (all ****s that are immediately after an **<a>**)

> **Note:** Sibling selectors target elements that come after them in the source document. CSS, by its nature (it cascades), cannot target *previous* or *parent* elements. However, using the flex `order` property, a previous sibling selector can be simulated on visual media.

Descendant Combinator: `selector selector`

A descendant combinator, represented by at least one space character (), selects elements that are a descendant of the defined element. This combinator selects **all** descendants of the element (from child elements on down).

```
div p {
  color:red;
}
```

```
<div>
  <p>My text is red</p>
  <section>
    <p>My text is red</p>
  </section>
</div>

<p>My text is not red</p>
```

Live Demo on JSBin

In the above example, the first two **<p>** elements are selected since they are both descendants of the ** **.

Child Combinator: `selector > selector`

The child (>) combinator is used to select elements that are **children**, or **direct descendants**, of the specified element.

```
div > p {
  color:red;
}
```

```
<div>
  <p>我的文字是红色</p>
  <section>
    <p>我的文字不是红色</p>
  </section>
</div>
```

JSBin 在线演示

上述 CSS 仅选择第一个<p>元素，因为它是唯一直接从 继承的段落。

第二个<p>元素未被选择，因为它不是 的直接子元素。

相邻兄弟组合器：`选择器 + 选择器`

相邻兄弟（+）组合器选择紧跟指定元素的兄弟元素。

```
p + p {
  color:red;
}
```

```
<p>我的文本不是红色</p>
<p>我的文本是红色</p>
<p>我的文本是红色</p>
<hr>
<p>我的文字不是红色</p>
```

JSBin 在线演示

上述示例仅选择那些<p>元素，这些元素直接前面紧跟另一个<p>元素。

通用兄弟组合器：`选择器 ~ 选择器`

通用兄弟（~）组合器选择*所有*跟随指定元素的兄弟元素。

```
p ~ p {
  color:red;
}
```

```
<p>我的文本不是红色</p>
<p>我的文本是红色</p>
<hr>
<h1>现在是标题</h1>
<p>我的文本是红色</p>
```

JSBin 在线演示

上述示例选择所有被另一个**<p>**元素*preceded*的**<p>**元素，无论它们是否紧邻。

第4.4节：伪类

伪类是关键字，允许基于文档树之外的信息或


```
div > p {
  color:red;
}
```

```
<div>
  <p>My text is red</p>
  <section>
    <p>My text is not red</p>
  </section>
</div>
```

Live Demo on JSBin

The above CSS selects only the first **<p>** element, as it is the only paragraph directly descended from a ** **.

The second **<p>** element is not selected because it is not a direct child of the ** **.

Adjacent Sibling Combinator: `selector + selector`

The adjacent sibling (+) combinator selects a sibling element that immediate follows a specified element.

```
p + p {
  color:red;
}
```

```
<p>My text is not red</p>
<p>My text is red</p>
<p>My text is red</p>
<hr>
<p>My text is not red</p>
```

Live Demo on JSBin

The above example selects only those **<p>** elements which are *directly preceded* by another **<p>** element.

General Sibling Combinator: `selector ~ selector`

The general sibling (~) combinator selects *all* siblings that follow the specified element.

```
p ~ p {
  color:red;
}
```

```
<p>My text is not red</p>
<p>My text is red</p>
<hr>
<h1>And now a title</h1>
<p>My text is red</p>
```

Live Demo on JSBin

The above example selects all **<p>** elements that are *preceded* by another **<p>** element, whether or not they are immediately adjacent.

Section 4.4: Pseudo-classes

Pseudo-classes are **keywords** which allow selection based on information that lies outside of the document tree or

无法通过其他选择器或组合器表达的信息进行选择。此信息可以与某种状态相关联（状态和动态伪类），与位置相关联（结构和目标伪类），与前者的否定相关联（否定伪类）或与语言相关联（语言伪类）。示例包括链接是否被访问过（`:visited`）、鼠标是否悬停在元素上（`:hover`）、复选框是否被选中（`:checked`）等。

语法

```
选择器:伪 类 {
    属性: 值;
}
```

伪类列表：

名称	描述
:active	适用于用户激活（即点击）的任何元素。
:any	允许您通过创建组来构建相关选择器集合，组内的项目将匹配该组。这是重复整个选择器的替代方法。
:target	选择当前活动的 #news 元素（点击包含该锚点名称的 URL 时）
:checked	适用于被选中或切换到“开启”状态的单选按钮、复选框或选项元素。
:default	表示一组相似元素中默认的任何用户界面元素。
:disabled	适用于处于禁用状态的任何用户界面元素。
:empty	适用于没有子元素的任何元素。
:enabled	适用于处于启用状态的任何用户界面元素。
:first	与@page规则结合使用，选择打印文档中的第一页。
:first-child	表示作为其父元素的第一个子元素的任何元素。
:first-of-type	当元素是其父元素中所选元素类型的第一个时适用。这可能是也可能不是第一个子元素。
:焦点	适用于任何获得用户焦点的元素。焦点可以由用户的键盘、鼠标事件或其他输入方式赋予。
:focus-within	当其中一个元素获得焦点时，可用于突出显示整个区域。它匹配任何符合 :focus 伪类的元素或其后代中有焦点的元素。
:full-screen	适用于以全屏模式显示的任何元素。它选择整个元素堆栈，而不仅仅是顶层元素。
:hover	适用于任何被用户指针设备悬停但未激活的元素。
:indeterminate	适用于处于不确定状态的单选按钮或复选框 UI 元素，这些元素既未被选中也未被取消选中。此状态可能由元素属性或 DOM 操作引起。
:in-range	The :in-range CSS 伪类匹配当元素的值属性在该元素指定的范围限制内时。它允许页面反馈当前使用该元素定义的值在范围限制内。
:invalid	适用于根据 type= 属性指定的类型值无效的 <input> 元素。
:lang	适用于其包裹的 <body> 元素具有正确指定的 lang= 属性的任何元素。伪类有效时，必须包含有效的两或三字母语言代码。
:last-child	表示作为其父元素最后一个子元素的任何元素。
:last-of-type	当元素是其父元素内所选元素类型的最后一个时应用。这可能是也可能不是最后一个子元素。

that cannot be expressed by other selectors or combinators. This information can be associated to a certain state (state and dynamic pseudo-classes), to locations (structural and target pseudo-classes), to negations of the former (negation pseudo-class) or to languages (lang pseudo-class). Examples include whether or not a link has been followed (`:visited`), the mouse is over an element (`:hover`), a checkbox is checked (`:checked`), etc.

Syntax

```
selector:pseudo class {
    property: VALUE;
}
```

List of pseudo-classes:

Name	Description
:active	Applies to any element being activated (i.e. clicked) by the user.
:any	Allows you to build sets of related selectors by creating groups that the included items will match. This is an alternative to repeating an entire selector.
:target	Selects the current active #news element (clicked on a URL containing that anchor name)
:checked	Applies to radio, checkbox, or option elements that are checked or toggled into an "on" state.
:default	Represents any user interface element that is the default among a group of similar elements.
:disabled	Applies to any UI element which is in a disabled state.
:empty	Applies to any element which has no children.
:enabled	Applies to any UI element which is in an enabled state.
:first	Used in conjunction with the `@page` rule, this selects the first page in a printed document.
:first-child	Represents any element that is the first child element of its parent.
:first-of-type	Applies when an element is the first of the selected element type inside its parent. This may or may not be the first-child.
:focus	Applies to any element which has the user's focus. This can be given by the user's keyboard, mouse events, or other forms of input.
:focus-within	Can be used to highlight a whole section when one element inside it is focused. It matches any element that the :focus pseudo-class matches or that has a descendant focused.
:full-screen	Applies to any element displayed in full-screen mode. It selects the whole stack of elements and not just the top level element.
:hover	Applies to any element being hovered by the user's pointing device, but not activated.
:indeterminate	Applies radio or checkbox UI elements which are neither checked nor unchecked, but are in an indeterminate state. This can be due to an element's attribute or DOM manipulation.
:in-range	The `:in-range` CSS pseudo-class matches when an element has its value attribute inside the specified range limitations for this element. It allows the page to give a feedback that the value currently defined using the element is inside the range limits.
:invalid	Applies to **<input>** elements whose values are invalid according to the type specified in the `type=` attribute.
:lang	Applies to any element who's wrapping **<body>** element has a properly designated `lang=` attribute. For the pseudo-class to be valid, it must contain a valid two or three letter language code.
:last-child	Represents any element that is the last child element of its parent.
:last-of-type	Applies when an element is the last of the selected element type inside its parent. This may or may not be the last-child.

:左	与@page规则结合使用时，选择打印文档中的所有左页。
:link	适用于用户尚未访问的任何链接。
:not()	适用于所有不匹配传入值的元素（例如:not(p)或:not(.class-name)）。必须有一个值才有效，并且只能包含一个选择器。不过，可以将多个:not选择器串联使用。
:nth-child	当元素是其父元素的第n个子元素时应用，其中 n可以是整数、数学表达式（例如 n+3）或关键字odd或even。
:nth-of-type	当元素是其父元素中相同类型元素的第n个时应用，其中 n可以是整数、数学表达式（例如 n+3）或关键字odd或even。
:only-child	:only-child CSS 伪类表示任何作为其父元素唯一子元素的元素。这与 :first-child:last-child 或 :nth-child(1):nth-last-child(1)相同，但特异性较低。
:optional	:optional CSS 伪类表示任何未设置必需属性的元素。这允许表单轻松指示可选字段并相应地进行样式设置。
:out-of-range	:out-of-range CSS 伪类匹配当元素的值属性超出该元素指定的范围限制时。它允许页面反馈当前使用该元素定义的值超出范围限制。值如果小于最小值或大于最大值，则视为超出范围。
:placeholder-shown	实验性。适用于当前显示占位符文本的任何表单元素。
:read-only	适用于用户无法编辑的任何元素。
:可读写	适用于用户可编辑的任何元素，例如<input>元素。
:右侧	与@page规则结合使用，选择打印文档中的所有右侧页面。
:根元素	匹配表示文档的树的根元素。
:作用域	CSS伪类，匹配作为选择器匹配参考点的元素。
:target	选择当前活动的 #news 元素（点击包含该锚点名称的 URL 时）
:已访问	适用于用户已访问的任何链接。

> “:visited”伪类在许多现代浏览器中已不能用于大多数样式设置，因为它存在安全漏洞。参考此链接。

第4.5节：子元素伪类

> “:nth-child(an+b) CSS伪类匹配文档树中在其之前有an+b-1个同级元素的元素，n为给定的正数或零值”
> - MDN :nth-child

伪选择器	1	2	3	4	5	6	7	8	9	10
:first-child	✓									
:nth-child(3)			✓							
:nth-child(n+3)			✓	✓	✓	✓	✓	✓	✓	✓
:nth-child(3n)			✓			✓			✓	

:left	Used in conjunction with the @page rule, this selects all the left pages in a printed document.
:link	Applies to any links which haven't been visited by the user.
:not()	Applies to all elements which **do not** match the value passed to (:not(p) or :not(.class-name) for example. It must have a value to be valid and it can only contain one selector. However, you can chain multiple :not selectors together.
:nth-child	Applies when an element is the n-th element of its parent, where n can be an integer, a mathematical expression (e.g n+3) or the keywords odd or even.
:nth-of-type	Applies when an element is the n-th element of its parent of the same element type, where n can be an integer, a mathematical expression (e.g n+3) or the keywords odd or even.
:only-child	The :only-child CSS pseudo-class represents any element which is the only child of its parent. This is the same as :first-child:last-child or :nth-child(1):nth-last-child(1), but with a lower specificity.
:optional	The :optional CSS pseudo-class represents any element that does not have the required attribute set on it. This allows forms to easily indicate optional fields and to style them accordingly.
:out-of-range	The :out-of-range CSS pseudo-class matches when an element has its value attribute outside the specified range limitations for this element. It allows the page to give a feedback that the value currently defined using the element is outside the range limits. A value can be outside of a range if it is either smaller or larger than maximum and minimum set values.
:placeholder-shown	**Experimental.** Applies to any form element currently displaying placeholder text.
:read-only	Applies to any element which is not editable by the user.
:read-write	Applies to any element that is editable by a user, such as **<input>** elements.
:right	Used in conjunction with the @page rule, this selects all the right pages in a printed document.
:root	matches the root element of a tree representing the document.
:scope	CSS pseudo-class matches the elements that are a reference point for selectors to match against.
:target	Selects the current active #news element (clicked on a URL containing that anchor name)
:visited	Applies to any links which have has been visited by the user.

> The :visited pseudoclass can't be used for most styling in a lot of modern browsers anymore because it's a security hole. See this link for reference.

Section 4.5: Child Pseudo Class

> "The :nth-child(an+b) CSS pseudo-class matches an element that has an+b-1 siblings before it in the document tree, for a given positive **or zero value** for n" - MDN :nth-child

pseudo-selector	1	2	3	4	5	6	7	8	9	10
:first-child	✓									
:nth-child(3)			✓							
:nth-child(n+3)			✓	✓	✓	✓	✓	✓	✓	✓
:nth-child(3n)			✓			✓			✓	

:nth-child(3n+1) ✓ ✓ ✓ ✓
:nth-child(-n+3) ✓ ✓ ✓
:nth-child(odd) ✓ ✓ ✓ ✓ ✓
:nth-child(even) ✓ ✓ ✓ ✓ ✓
:last-child ✓
:nth-last-child(3) ✓

第4.6节：类名选择器

类名选择器选择所有具有目标类名的元素。例如，类名`.warning`会选择以下`<div>`元素：

```
<div class="warning">
    <p>这将是一段警告文本。</p>
</div>
```

你也可以组合类名以更具体地定位元素。让我们基于上面的例子展示一个更复杂的类选择。

CSS

```
.important {
    color: orange;
}
.warning {
    color: blue;
}
.warning.important {
    color: red;
}
```

HTML

```
<div class="warning">
    <p>这将是一段警告文本。</p>
</div>

<div class="important warning">
    <p class="important">这是一些非常重要的警告内容。</p>
```

在此示例中，所有带有 .warning 类的元素将具有蓝色文本，带有 .important 类的元素将具有橙色文本，且所有同时具有 .important 和 .warning 类名的元素将具有红色文本。

请注意，在 CSS 中，.warning.important 声明中两个类名之间没有空格。这意味着它只会找到在 class 属性中同时包含 warning 和 important 两个类名的元素。这些类名在元素上的顺序可以是任意的。

如果在 CSS 声明中两个类名之间包含空格，则只会选择具有 .warning 类名的父元素和具有 .important 类名的子元素。

:nth-child(3n+1) ✓ ✓ ✓ ✓
:nth-child(-n+3) ✓ ✓ ✓
:nth-child(odd) ✓ ✓ ✓ ✓ ✓
:nth-child(even) ✓ ✓ ✓ ✓ ✓
:last-child ✓
:nth-last-child(3) ✓

Section 4.6: Class Name Selectors

The class name selector select all elements with the targeted class name. For example, the class name `.warning` would select the following `<div>` element:

```
<div class="warning">
    <p>This would be some warning copy.</p>
</div>
```

You can also combine class names to target elements more specifically. Let's build on the example above to showcase a more complicated class selection.

CSS

```
.important {
    color: orange;
}
.warning {
    color: blue;
}
.warning.important {
    color: red;
}
```

HTML

```
<div class="warning">
    <p>This would be some warning copy.</p>
</div>

<div class="important warning">
    <p class="important">This is some really important warning copy.</p>
</div>
```

In this example, all elements with the `.warning` class will have a blue text color, elements with the `.important` class with have an orange text color, and all elements that have *both* the `.important` and `.warning` class name will have a red text color.

Notice that within the CSS, the `.warning.important` declaration did not have any spaces between the two class names. This means it will only find elements which contain both class names `warning` and `important` in their `class` attribute. Those class names could be in any order on the element.

If a space was included between the two classes in the CSS declaration, it would only select elements that have parent elements with a `.warning` class names and child elements with `.important` class names.

第4.7节：使用ID选择元素而不使用ID选择器的高特异性

这个技巧帮助你使用ID作为属性选择器的值来选择元素，从而避免ID选择器的高特异性。

HTML：

```
<div id="element">...</div>
```

CSS

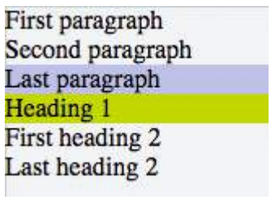
```
#element { ... } /* 高特异性将覆盖许多选择器 */

[id="element"] { ... } /* 低特异性，容易被覆盖 */
```

第4.8节：:last-of-type 选择器

`:last-of-type` 选择其父元素中特定类型的最后一个子元素。下面的例子中，
css选择了最后一个段落和最后一个标题 h1。

```
p:last-of-type {
  background: #C5CAE9;
}
h1:last-of-type {
  background: #CDDC39;
}
```

```
<div class="container">
    <p>第一段</p>
    <p>第二段</p>
    <p>最后一段</p>
    <h1>标题 1</h1>
    <h2>第一个标题 2</h2>
    <h2>最后一个标题 2</h2>
</div>
```



jsFiddle

第4.9节：CSS3 :in-range 选择器示例

```
<style>
input:in-range {
    border: 1px solid blue;
}
</style>


<input type="number" min="10" max="20" value="15">
```


Section 4.7: Select element using its ID without the high specificity of the ID selector

This trick helps you select an element using the ID as a value for an attribute selector to avoid the high specificity of the ID selector.

HTML:

```
<div id="element">...</div>
```

CSS

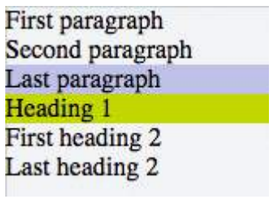
```
#element { ... } /* High specificity will override many selectors */

[id="element"] { ... } /* Low specificity, can be overridden easily */
```

Section 4.8: The :last-of-type selector

The `:last-of-type` selects the element that is the last child, of a particular type, of its parent. In the example below, the css selects the last paragraph and the last heading h1.

```
p:last-of-type {
  background: #C5CAE9;
}
h1:last-of-type {
  background: #CDDC39;
}
```

```
<div class="container">
    <p>First paragraph</p>
    <p>Second paragraph</p>
    <p>Last paragraph</p>
    <h1>Heading 1</h1>
    <h2>First heading 2</h2>
    <h2>Last heading 2</h2>
</div>
```



jsFiddle

Section 4.9: CSS3 :in-range selector example

```
<style>
input:in-range {
    border: 1px solid blue;
}
</style>


<input type="number" min="10" max="20" value="15">
```

```
<p>该值的边框将为蓝色</p>
```

:in-range CSS 伪类匹配当元素的 value 属性在该元素指定的范围限制内时。它允许页面反馈当前使用该元素定义的值在范围限制内。[1]

第4.10节：A. :not 伪类示例 & B. :focus-within CSS 伪类

A. 语法如上所示。

以下选择器匹配 HTML 文档中所有未被禁用且没有类名 .example 的 <input> 元素：

HTML：

```
<form>
电话：<input type="tel" class="example">
    电子邮件：<input type="email" disabled="disabled">
    密码：<input type="password">
</form>
```

CSS：

```
input:not([disabled]):not(.example){
   background-color: #ccc;
}
```

:not() 伪类在选择器级别4中也将支持逗号分隔的选择器：

CSS：

```
input:not([disabled], .example){
   background-color: #ccc;
}
```

JSBin 上的实时演示

查看背景语法。

B. :focus-within CSS 伪类

HTML：

```
  <h3>如果输入框获得焦点，背景将变为蓝色。</p>
  <div>
    <input type="text">
  </div>
```

CSS：

```
div {
  height: 80px;
}
输入{
  margin:30px;
```



```
<p>The border for this value will be blue</p>
```

The :in-range CSS pseudo-class matches when an element has its value attribute inside the specified range limitations for this element. It allows the page to give a feedback that the value currently defined using the element is inside the range limits.[1]

Section 4.10: A. The :not pseudo-class example & B. :focus-within CSS pseudo-class

A. The syntax is presented above.

The following selector matches all **<input>** elements in an HTML document that are not disabled and don't have the class .example:

HTML:

```
<form>
    Phone: <input type="tel" class="example">
    E-mail: <input type="email" disabled="disabled">
    Password: <input type="password">
</form>
```

CSS:

```
input:not([disabled]):not(.example){
   background-color: #ccc;
}
```

The :not() pseudo-class will also support comma-separated selectors in Selectors Level 4:

CSS:

```
input:not([disabled], .example){
   background-color: #ccc;
}
```

Live Demo on JSBin

See background syntax here.

B. The :focus-within CSS pseudo-class

HTML:

```
  <h3>Background is blue if the input is focused .</p>
  <div>
    <input type="text">
  </div>
```

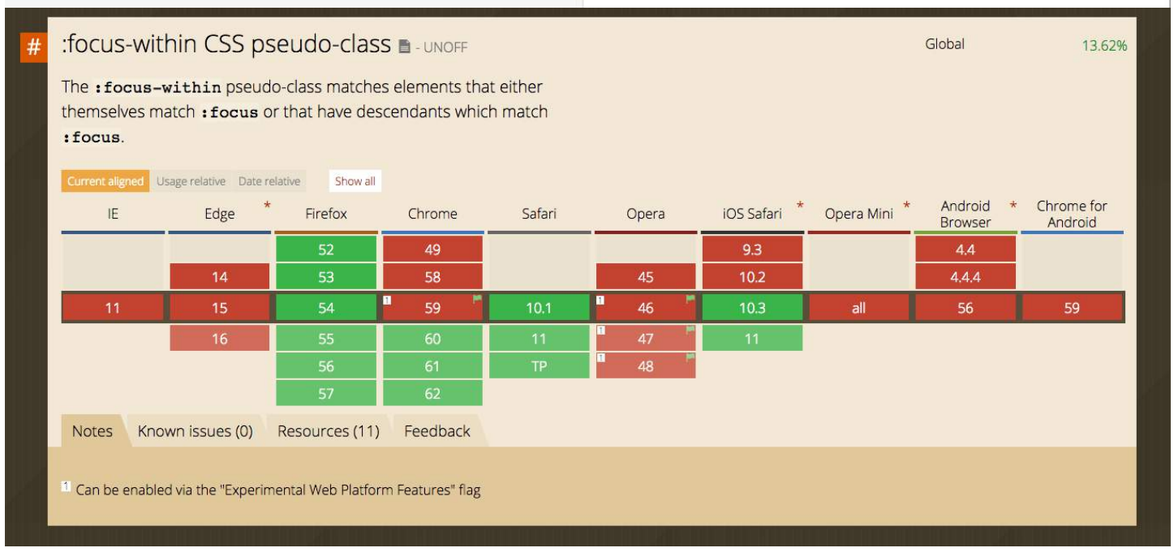
CSS:

```
div {
  height: 80px;
}
input{
  margin:30px;
```

```
}
div:focus-within {
   background-color: #1565C0;
}
```

```
div {
  height: 80px;
}
input{
  margin:30px;
}
div:focus-within {
  background-color: #1565C0;
}
```

Background is blue if the input is focused .



:focus-within CSS pseudo-class - UNOFF

Global 13.62%

The `:focus-within` pseudo-class matches elements that either themselves match `:focus` or that have descendants which match `:focus`.

Current aligned | Usage relative | Date relative | Show all

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Chrome for Android
		52	49			9.3		4.4	
	14	53	58		45	10.2		4.4.4	
11	15	54	59 [1]	10.1	46 [1]	10.3	all	56	59
	16	55	60	11	47 [1]	11			
		56	61	TP	48 [1]				
		57	62						

Notes | Known issues (0) | Resources (11) | Feedback

[1] Can be enabled via the "Experimental Web Platform Features" flag

第4.11节：带有checkbox:checked和~（通用兄弟选择器）的全局布尔值

使用~选择器，您可以轻松实现一个全局可访问的布尔值，而无需使用JavaScript。

添加布尔值作为复选框

在文档的最开始，添加任意数量的布尔值，要求具有唯一的id并设置hidden属性：

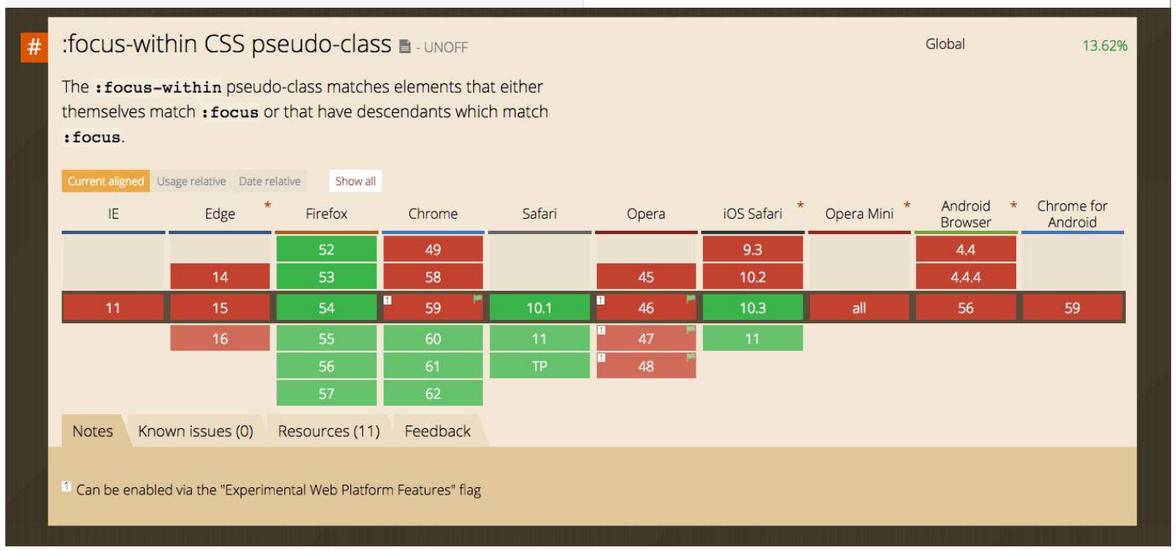
```
<input type="checkbox" id="sidebarShown" hidden />
<input type="checkbox" id="darkThemeUsed" hidden />

<!-- 这里开始实际内容，例如： -->
<div id="container">
    <div id="sidebar">
        <!-- 菜单，搜索，... -->
    </div>

    <!-- 一些更多内容 ... -->


<div id="footer">
    <!-- ... -->
</div>
```



```
}
div:focus-within {
   background-color: #1565C0;
}
```

```
div {
  height: 80px;
}
input{
  margin:30px;
}
div:focus-within {
  background-color: #1565C0;
}
```

Background is blue if the input is focused .



:focus-within CSS pseudo-class - UNOFF

Global 13.62%

The `:focus-within` pseudo-class matches elements that either themselves match `:focus` or that have descendants which match `:focus`.

Current aligned | Usage relative | Date relative | Show all

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Chrome for Android
		52	49			9.3		4.4	
	14	53	58		45	10.2		4.4.4	
11	15	54	59 [1]	10.1	46 [1]	10.3	all	56	59
	16	55	60	11	47 [1]	11			
		56	61	TP	48 [1]				
		57	62						

Notes | Known issues (0) | Resources (11) | Feedback

[1] Can be enabled via the "Experimental Web Platform Features" flag

Section 4.11: Global boolean with checkbox:checked and ~ (general sibling combinator)

With the ~ selector, you can easily implement a global accessible boolean without using JavaScript.

Add boolean as a checkbox

To the very beginning of your document, add as much booleans as you want with a unique `id` and the `hidden` attribute set:

```
<input type="checkbox" id="sidebarShown" hidden />
<input type="checkbox" id="darkThemeUsed" hidden />

<!-- here begins actual content, for example: -->
<div id="container">
    <div id="sidebar">
        <!-- Menu, Search, ... -->
    </div>

    <!-- Some more content ... -->
</div>

<div id="footer">
    <!-- ... -->
</div>
```

更改布尔值

您可以通过添加一个带有for属性的label来切换布尔值：

```
<label for="sidebarShown">显示/隐藏侧边栏！</label>
```

使用 CSS 访问布尔值

普通选择器（如.color-red）指定默认属性。它们可以被后续的true / false选择器覆盖：

```
/* true: */
<checkbox>:checked ~ [checkbox的兄弟元素 & target的父元素] <target>

/* false: */
<checkbox>：未（:未选中） ~ [checkbox 的兄弟元素 & target 的父元素] <target>
```

请注意，**<checkbox>**、[sibling ...] 和 **<target>** 应替换为合适的选择器。[sibling ...] 可以是一个特定的选择器，（通常如果你懒的话）简单地用*，或者如果目标已经是复选框的兄弟元素，则可以不写。

上述HTML结构的示例为：

```
#sidebarShown:checked ~ #container #sidebar {
    margin-left: 300px;
}

#darkThemeUsed:checked ~ #container,
#darkThemeUsed:checked ~ #footer {
    background: #333;
}
```

实际效果

请参见this fiddle，了解这些全局布尔值的实现。

第4.12节：ID选择器

ID选择器用于选择具有指定ID的DOM元素。要在CSS中通过特定ID选择元素，使用#前缀。

例如，以下HTML的div元素...

```
<div id="exampleID">
    <p>示例</p>
</div>
```

...可以通过#exampleID在CSS中选择，如下所示：

```
#exampleID {
    宽度: 20px;
}
```

> 注意：HTML规范不允许多个元素使用相同的ID

Change the boolean's value

You can toggle the boolean by adding a `label` with the `for` attribute set:

```
<label for="sidebarShown">Show/Hide the sidebar!</label>
```

Accessing boolean value with CSS

The normal selector (like `.color-red`) specifies the default properties. They can be overridden by following `true` / `false` selectors:

```
/* true: */
<checkbox>:checked ~ [sibling of checkbox & parent of target] <target>

/* false: */
<checkbox>:not(:checked) ~ [sibling of checkbox & parent of target] <target>
```

Note that **<checkbox>**, [sibling ...] and **<target>** should be replaced by the proper selectors. [sibling ...] can be a specific selector, (often if you're lazy) simply * or nothing if the target is already a sibling of the checkbox.

Examples for the above HTML structure would be:

```
#sidebarShown:checked ~ #container #sidebar {
    margin-left: 300px;
}

#darkThemeUsed:checked ~ #container,
#darkThemeUsed:checked ~ #footer {
    background: #333;
}
```

In action

See this fiddle for a implementation of these global booleans.

Section 4.12: ID selectors

ID selectors select DOM elements with the targeted ID. To select an element by a specific ID in CSS, the # prefix is used.

For example, the following HTML `div` element…

```
<div id="exampleID">
    <p>Example</p>
</div>
```

…can be selected by `#exampleID` in CSS as shown below:

```
#exampleID {
    width: 20px;
}
```

> **Note**: The HTML specs do not allow multiple elements with the same ID

第4.13节：如何为Range输入框设置样式

HTML

```
<input type="range"></input>
```

CSS

效果	伪选择器
滑块	`input[类型=范围]::-webkit-slider-thumb, input[类型=范围]::-moz-range-thumb, input[类型=范围]::-ms-thumb`
轨道	`input[type=range]::-webkit-slider-runnable-track, input[type=range]::-moz-range-track, input[type=range]::-ms-track`
获得焦点	input[type=range]:focus
轨道的下半部分	input[type=range]::-moz-range-progress，input[type=range]::-ms-fill-lower（当前WebKit浏览器不支持 - 需要JS）

第4.14节：:only-child伪类选择器示例

:only-child CSS伪类表示任何作为其父元素唯一子元素的元素。

HTML：

```
<div>
  <p>该段落是div的唯一子元素，它的颜色将是蓝色</p>
</div>

<div>
  <p>该段落是div的两个子元素之一</p>
  <p>该段落是其父元素的两个子元素之一</p>
</div>
```

css：

```
p:唯一子元素 {
  color: 蓝色;
}
```

上述示例选择了作为其父元素唯一子元素的<p>元素，在本例中是一个 。

JSBin 在线演示

Section 4.13: How to style a Range input

HTML

```
<input type="range"></input>
```

CSS

Effect	Pseudo Selector
Thumb	`input[type=range]::-webkit-slider-thumb, input[type=range]::-moz-range-thumb, input[type=range]::-ms-thumb`
Track	`input[type=range]::-webkit-slider-runnable-track, input[type=range]::-moz-range-track, input[type=range]::-ms-track`
OnFocus	`input[type=range]:focus`
Lower part of the track	`input[type=range]::-moz-range-progress, input[type=range]::-ms-fill-lower` (not possible in WebKit browsers currently - JS needed)

Section 4.14: The :only-child pseudo-class selector example

The `:only-child` CSS pseudo-class represents any element which is the only child of its parent.

HTML:

```
<div>
  <p>This paragraph is the only child of the div, it will have the color blue</p>
</div>

<div>
  <p>This paragraph is one of the two children of the div</p>
  <p>This paragraph is one of the two children of its parent</p>
</div>
```

CSS:

```
p:only-child {
  color: blue;
}
```

The above example selects the **<p>** element that is the unique child from its parent, in this case a ** **.

Live Demo on JSBin

第5章：背景

使用CSS，您可以设置元素的背景颜色、渐变和图像。

可以指定图像、颜色和渐变的各种组合，并调整这些的大小、定位和重复（等）。

第5.1节：背景颜色

`background-color`属性使用颜色值或关键字设置元素的背景颜色，例如`transparent`、`inherit`或`initial`。

- transparent，指定背景颜色应为透明。这是默认值。
- inherit，从其父元素继承此属性。
- initial，将此属性设置为其默认值。

这可以应用于所有元素，以及::first-letter/::first-line伪元素。

CSS中的颜色可以通过不同的方法指定。

颜色名称

CSS

```
div {
  background-color: red;  /* 红色 */
}
```

HTML

```
<div>这将有一个红色背景</div>
```

- 上面使用的示例是CSS表示单一颜色的几种方式之一。

十六进制颜色代码

十六进制代码用于以16进制表示颜色的RGB分量。例如，#ff0000是鲜红色，其中颜色的红色分量是256位（ff），对应的绿色和蓝色部分是0（00）。

如果三个RGB配对（R、G和B）中的两个值相同，则颜色代码可以缩短为三个字符（每个配对的第一个数字）。#ff0000可以缩短为#f00，ffffff可以缩短为#fff。

十六进制表示不区分大小写。

```
body {
  background-color: #de1205; /* 红色 */
}

.main {
```


Chapter 5: Backgrounds

With CSS you can set colors, gradients, and images as the background of an element.

It is possible to specify various combinations of images, colors, and gradients, and adjust the size, positioning, and repetition (among others) of these.

Section 5.1: Background Color

The `background-color` property sets the background color of an element using a color value or through keywords, such as `transparent`, `inherit` or `initial`.

- **transparent**, specifies that the background color should be transparent. This is default.
- **inherit**, inherits this property from its parent element.
- **initial**, sets this property to its default value.

This can be applied to all elements, and `::first-letter`/`::first-line` pseudo-elements.

Colors in CSS can be specified by different methods.

Color names

CSS

```
div {
  background-color: red;  /* red */
}
```

HTML

```
<div>This will have a red background</div>
```

- The example used above is one of several ways that CSS has to represent a single color.

Hex color codes

Hex code is used to denote RGB components of a color in base-16 hexadecimal notation. #ff0000, for example, is bright red, where the red component of the color is 256 bits (ff) and the corresponding green and blue portions of the color is 0 (00).

If both values in each of the three RGB pairings (R, G, and B) are the same, then the color code can be shortened into three characters (the first digit of each pairing). `#ff0000` can be shortened to `#f00`, and `#ffffff` can be shortened to `#fff`.

Hex notation is case-insensitive.

```
body {
  background-color: #de1205; /* red */
}

.main {
```

```
  background-color: #00f; /* 蓝色 */
}
```

RGB / RGBa

另一种声明颜色的方法是使用 RGB 或 RGBa。

RGB 代表红色、绿色和蓝色，需要三个介于 0 到 255 之间的独立数值，放在括号内，分别对应红色、绿色和蓝色的十进制颜色值。

RGBa 允许你添加一个介于 0.0 到 1.0 之间的额外 alpha 参数来定义不透明度。

```
header {
  background-color: rgb(0, 0, 0); /* 黑色 */
}

footer {
  background-color: rgba(0, 0, 0, 0.5); /* 黑色，透明度50% */
}
```

HSL / HSLa

另一种声明颜色的方法是使用HSL或HSLa，类似于RGB和RGBA。

HSL代表色相、饱和度和亮度，也常称为HLS：

- 色相是色轮上的度数（从0到360）。
- 饱和度是0%到100%之间的百分比。
- 亮度也是0%到100%之间的百分比。

HSLa允许你添加一个介于0.0到1.0之间的额外alpha参数来定义不透明度。

```
li a {
  background-color: hsl(120, 100%, 50%); /* 绿色 */
}

#p1 {
  background-color: hsla(120, 100%, 50%, .3); /* 绿色，透明度30% */
}
```

与背景图像的交互

以下语句均等价：

```
body {
  background: red;
  background-image: url(partiallytransparentimage.png);
}

body {
  background-color: red;
  background-image: url(partiallytransparentimage.png);
}

body {
  background-image: url(partiallytransparentimage.png);
  background-color: red;
```



```
  background-color: #00f; /* blue */
}
```

RGB / RGBa

Another way to declare a color is to use RGB or RGBa.

RGB stands for Red, Green and Blue, and requires of three separate values between 0 and 255, put between brackets, that correspond with the decimal color values for respectively red, green and blue.

RGBa allows you to add an additional alpha parameter between 0.0 and 1.0 to define opacity.

```
header {
  background-color: rgb(0, 0, 0); /* black */
}

footer {
  background-color: rgba(0, 0, 0, 0.5); /* black with 50% opacity */
}
```

HSL / HSLa

Another way to declare a color is to use HSL or HSLa and is similar to RGB and RGBa.

HSL stands for hue, saturation, and lightness, and is also often called HLS:

- Hue is a degree on the color wheel (from 0 to 360).
- Saturation is a percentage between 0% and 100%.
- Lightness is also a percentage between 0% and 100%.

HSLa allows you to add an additional alpha parameter between 0.0 and 1.0 to define opacity.

```
li a {
  background-color: hsl(120, 100%, 50%); /* green */
}

#p1 {
  background-color: hsla(120, 100%, 50%, .3); /* green with 30% opacity */
}
```

Interaction with background-image

The following statements are all equivalent:

```
body {
  background: red;
  background-image: url(partiallytransparentimage.png);
}

body {
  background-color: red;
  background-image: url(partiallytransparentimage.png);
}

body {
  background-image: url(partiallytransparentimage.png);
  background-color: red;
```

```
}

body {
  background: red url(partiallytransparentimage.png);
}
```

它们都会导致在图像下方显示红色，当图像的部分区域是透明的，
或者图像未显示（可能是由于background-repeat的原因）。

请注意，以下内容并不等价：

```
body {
  background-image: url(partiallytransparentimage.png);
  background: red;
}
```

这里，background 的值会覆盖你的 background-image。

有关 background 属性的更多信息，请参见背景简写

第5.2节：背景渐变

渐变是CSS3中新添加的图像类型。作为图像，渐变通过 background-image 属性设置，
或者通过 background 简写设置。

渐变函数有两种类型，线性和径向。每种类型都有非重复和重复两种变体：

- linear-gradient()
- repeating-linear-gradient()
- radial-gradient()
- repeating-radial-gradient()

linear-gradient()

A linear-gradient 具有以下语法

```
background: linear-gradient( <direction>?, <color-stop-1>, <color-stop-2>, ...);
```

值	含义
<direction>	可以是类似 to 上、to 下、to 右 或 to 左 的参数；也可以是 角度，如 0deg、90deg 等。角度从上方开始，顺时针旋转。可以用 deg、grad、rad 或 turn 指定。如果省略，渐变将从上到下流动
<color-stop-list>	颜色列表，每个颜色后可选跟随一个百分比或 长度 来显示。例如 例如，yellow 10%、rgba(0,0,0,.5) 40px、#fff 100%...

例如，这将创建一个从右侧开始、颜色从红色渐变到蓝色的线性渐变

```
.linear-gradient {
  background: 线性渐变(向左, 红色, 蓝色); /* 你也可以使用270度 */
}
```

你可以通过声明水平和垂直的起始位置来创建一个对角线渐变。

```
.diagonal-linear-gradient {
  background: linear-gradient(向左上, 红色, 黄色 10%);
```



```
}

body {
  background: red url(partiallytransparentimage.png);
}
```

They will all lead to the red color being shown underneath the image, where the parts of the image are transparent, or the image is not showing (perhaps as a result of background-repeat).

Note that the following is not equivalent:

```
body {
  background-image: url(partiallytransparentimage.png);
  background: red;
}
```

Here, the value of background overrides your background-image.

For more info on the background property, see Background Shorthand

Section 5.2: Background Gradients

Gradients are new image types, added in CSS3. As an image, gradients are set with the background-image property, or the background shorthand.

There are two types of gradient functions, linear and radial. Each type has a non-repeating variant and a repeating variant:

- linear-gradient()
- repeating-linear-gradient()
- radial-gradient()
- repeating-radial-gradient()

linear-gradient()

A linear-gradient has the following syntax

```
background: linear-gradient( <direction>?, <color-stop-1>, <color-stop-2>, ...);
```

Value	Meaning
<direction>	Could be an argument like to top, to bottom, to right or to left; or an angle as 0deg, 90deg... . The angle starts from to top and rotates clockwise. Can be specified in deg, grad, rad, or turn. If omitted, the gradient flows from top to bottom
<color-stop-list>	List of colors, optionally followed each one by a percentage or length to display it at. For example, yellow 10%, rgba(0,0,0,.5) 40px, #fff 100%...

For example, this creates a linear gradient that starts from the right and transitions from red to blue

```
.linear-gradient {
  background: linear-gradient(to left, red, blue); /* you can also use 270deg */
}
```

You can create a diagonal gradient by declaring both a horizontal and vertical starting position.

```
.diagonal-linear-gradient {
  background: linear-gradient(to left top, red, yellow 10%);
```

```
}
```

可以通过用逗号分隔来指定渐变中的任意数量的颜色停点。以下
示例将创建一个具有8个颜色停点的渐变

```
.linear-gradient-rainbow {
   background: linear-gradient(向左, 红色, 橙色, 黄色, 绿色, 蓝色, 靛蓝, 紫罗兰)
}
```

radial-gradient()

```
.radial-gradient-simple {
   background: radial-gradient(红色, 蓝色);
}

.radial-gradient {
   background: radial-gradient(圆形 最远角落于 左上角, 红色, 蓝色);
}
```

值	含义
`circle`	渐变形状。取值为圆形或椭圆形，默认值为椭圆形。
`farthest-corner`	描述结束形状大小的关键字。取值为`closest-side`、`farthest-side`、`closest-corner`、`farthest-corner`
`左上角`	设置渐变中心的位置，方式与background-position相同。

重复渐变

重复渐变函数接受与上述示例相同的参数，但会在元素的背景中平铺渐变。

```
.bullseye {
   background: repeating-radial-gradient(红色, 红色 10%, 白色 10%, 白色 20%);
}
.warning {
   background: 重复线性渐变(-45度, 黄色, 黄色 10%, 黑色 10%, 黑色 20% );
}
```

值	含义
`-45度`	角度单位。角度从顶部开始顺时针旋转。可以指定为度（deg）、梯度（grad）、弧度（rad）或 圈数（turn）。
`向左`	渐变方向，默认是向底部。语法：向[纵轴（顶部或底部）][横轴（左或右）]例如向右上

黄色10%颜色，可选地后跟百分比或长度以显示。重复两次或更多次。

注意，HEX、RGB、RGBA、HSL 和 HSLA 颜色代码可以替代颜色名称。颜色名称仅用于说明。此外，径向渐变的语法比线性渐变复杂得多，这里显示的是简化版本。有关完整说明和规范，请参见MDN 文档

第5.3节：背景图像

background-image属性用于指定应用于所有匹配元素的背景图像。默认情况下，该图像会平铺覆盖整个元素，边距除外。

```
.myClass {
   background-image: url('/path/to/image.jpg');
}
```



```
}
```

It is possible to specify any number of color stops in a gradient by separating them with commas. The following examples will create a gradient with 8 color stops

```
.linear-gradient-rainbow {
  background: linear-gradient(to left, red, orange, yellow, green, blue, indigo, violet)
}
```

radial-gradient()

```
.radial-gradient-simple {
  background: radial-gradient(red, blue);
}

.radial-gradient {
  background: radial-gradient(circle farthest-corner at top left, red, blue);
}
```

Value	Meaning
`circle`	Shape of gradient. Values are `circle` or `ellipse`, default is `ellipse`.
`farthest-corner`	Keywords describing how big the ending shape must be. Values are `closest-side`, `farthest-side`, `closest-corner`, `farthest-corner`
`top left`	Sets the position of the gradient center, in the same way as `background-position`.

Repeating gradients

Repeating gradient functions take the same arguments as the above examples, but tile the gradient across the background of the element.

```
.bullseye {
  background: repeating-radial-gradient(red, red 10%, white 10%, white 20%);
}
.warning {
  background: repeating-linear-gradient(-45deg, yellow, yellow 10%, black 10%, black 20% );
}
```

Value	Meaning
`-45deg`	Angle unit. The angle starts from to top and rotates clockwise. Can be specified in deg, grad, rad, or turn.
`to left`	Direction of gradient, default is `to bottom`. Syntax: `to [y-axis(top OR bottom)] [x-axis(left OR right)]` ie `to top right`
`yellow 10%`	Color, optionally followed by a percentage or length to display it at. Repeated two or more times.

Note that HEX, RGB, RGBa, HSL, and HSLa color codes may be used instead of color names. Color names were used for the sake of illustration. Also note that the radial-gradient syntax is much more complex than linear-gradient, and a simplified version is shown here. For a full explanation and specs, see the MDN Docs

Section 5.3: Background Image

The `background-image` property is used to specify a background image to be applied to all matched elements. By default, this image is tiled to cover the entire element, excluding margin.

```
.myClass {
  background-image: url('/path/to/image.jpg');
}
```

要使用多个图像作为background-image，请定义用逗号分隔的url()

```
.myClass {
  background-image: url('/path/to/image.jpg'),
                    url('/path/to/image2.jpg');
}
```

图像将按照它们的顺序堆叠，最先声明的图像位于其他图像之上，依此类推。

值	结果
url('/path/to/image.jpg')	指定背景图像的路径或使用数据URI指定的图像资源模式（引号可省略），多个用逗号分隔
无	无背景图像
初始值	默认值
继承	继承父元素的值

更多背景图像的CSS

以下属性非常有用且几乎是必需的。

```
background-size:     xpx ypx | x% y%;
background-repeat:   no-repeat | repeat | repeat-x | repeat-y;
background-position: left offset (px/%) right offset (px/%) | center center | left top | right
bottom;
```

第5.4节：背景简写

background属性可用于设置一个或多个与背景相关的属性：

值	描述	CSS版本
background-image	使用的背景图像	1+
背景颜色	应用的背景颜色	1+
背景位置	背景图像的位置	1+
背景大小	背景图像的大小	3+
背景重复	背景图像的重复方式	1+
background-origin	背景的位置方式（当background-attachment为fixed时忽略）	3+
background-clip	背景相对于content-box、border-box或padding-box的绘制方式	3+
background-attachment	背景图像的行为方式，是否随其包含的块滚动或在视口内具有固定位置	1+
初始值	将属性设置为默认值	3+
继承	从父元素继承属性值	2+

值的顺序无关紧要，且每个值都是可选的

语法

background简写声明的语法是：

```
background: [<background-image>] [<background-color>]  [<background-position>]/[<background-size>]
[<background-repeat>] [<background-origin>] [<background-clip>] [<background-attachment>]
[<initial|inherit>];
```


To use multiple images as background-image, define comma separated url()

```
.myClass {
  background-image: url('/path/to/image.jpg'),
                    url('/path/to/image2.jpg');
}
```

The images will stack according to their order with the first declared image on top of the others and so on.

Value	Result
url('/path/to/image.jpg')	Specify background image's path(s) or an image resource specified with data URI schema (apostrophes can be omitted), separate multiples by comma
none	No background image
initial	Default value
inherit	Inherit parent's value

More CSS for Background Image

This following attributes are very useful and almost essential too.

```
background-size:     xpx ypx | x% y%;
background-repeat:   no-repeat | repeat | repeat-x | repeat-y;
background-position: left offset (px/%) right offset (px/%) | center center | left top | right
bottom;
```

Section 5.4: Background Shorthand

The background property can be used to set one or more background related properties:

Value	Description	CSS Ver.
background-image	Background image to use	1+
background-color	Background color to apply	1+
background-position	Background image's position	1+
background-size	Background image's size	3+
background-repeat	How to repeat background image	1+
background-origin	How the background is positioned (ignored when background-attachment is fixed)	3+
background-clip	How the background is painted relative to the content-box, border-box, or the padding-box	3+
background-attachment	How the background image behaves, whether it scrolls along with its containing block or has a fixed position within the viewport	1+
initial	Sets the property to value to default	3+
inherit	Inherits property value from parent	2+

The order of the values does not matter and every value is optional

Syntax

The syntax of the background shorthand declaration is:

```
background: [<background-image>] [<background-color>]  [<background-position>]/[<background-size>]
[<background-repeat>] [<background-origin>] [<background-clip>] [<background-attachment>]
[<initial|inherit>];
```

示例

```
background: red;
```

仅设置一个background-color为red值。

```
background: border-box red;
```

将background-clip设置为border-box，background-color设置为红色。

```
background: no-repeat center url("somepng.jpg");
```

将background-repeat设置为no-repeat，background-origin设置为center，background-image设置为一张图片。

```
background: url('pattern.png') green;
```

在此示例中，元素的background-color将被设置为green，如果可用，pattern.png将覆盖在颜色上，按需重复以填充元素。如果pattern.png包含任何透明部分，则绿色背景色将在其后可见。

```
背景: 0000000 url("picture.png") 左上 / 600px 自动 无重复;
```

在此示例中，我们有一个黑色背景，上面放置了一张名为“picture.png”的图片，图片在两个轴上都不重复，并且定位在左上角。位置后的/符号用于包含背景图片的大小，在本例中设置为宽度600px，高度自动。此示例适用于可以渐变为纯色的特色图片。

> 注意：使用简写的background属性会重置所有之前设置的背景属性值，即使某些值未被指定。如果只想修改之前设置的某个背景属性值，请使用长写属性。

第5.5节：背景大小

概述

background-size属性允许控制background-image的缩放。它最多接受两个值，分别决定生成图像在垂直和水平方向上的缩放/大小。如果该属性缺失，则宽度和高度均视为auto。

auto会保持图像的宽高比（如果可以确定）。高度是可选的，也可以视为auto。
因此，对于一张256px×256px的图片，以下所有background-size设置都会生成一个宽高均为50px的图像：

```
background-size: 50px;
background-size: 50px auto; /* 与上面相同 */
background-size: auto 50px;
background-size: 50px 50px;
```

所以如果我们从以下图片开始（其尺寸为256 px × 256 px），

Examples

```
background: red;
```

Simply setting a `background-color` with the `red`value.

```
background: border-box red;
```

Setting a `background-clip` to border-box and a `background-color` to red.

```
background: no-repeat center url("somepng.jpg");
```

Sets a `background-repeat` to no-repeat, `background-origin` to center and a `background-image` to an image.

```
background: url('pattern.png') green;
```

In this example, the `background-color` of the element would be set to `green` with `pattern.png`, if it is available, overlayed on the colour, repeating as often as necessary to fill the element. If `pattern.png` includes any transparency then the `green` colour will be visible behind it.

```
background: #000000 url("picture.png") top left / 600px auto no-repeat;
```

In this example we have a black background with an image 'picture.png' on top, the image does not repeat in either axis and is positioned in the top left corner. The `/` after the position is to be able to include the size of the background image which in this case is set as `600px` width and auto for the height. This example could work well with a feature image that can fade into a solid colour.

> **NOTE:** Use of the shorthand background property resets all previously set background property values, even if a value is not given. If you wish only to modify a background property value previously set, use a longhand property instead.

Section 5.5: Background Size

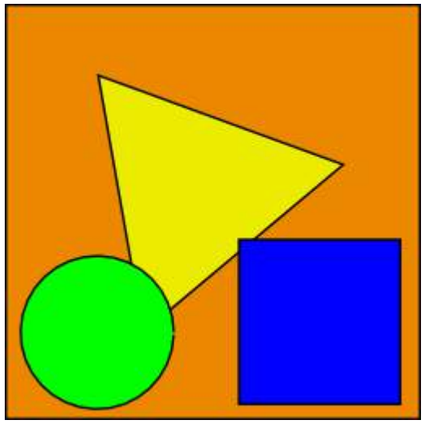
General overview

The `background-size` property enables one to control the scaling of the `background-image`. It takes up to two values, which determine the scale/size of the resulting image in vertical and and horizontal direction. If the property is missing, its deemed `auto` in both `width` and `height`.

`auto` will keep the image's aspect ratio, if it can be determined. The height is optional and can be considered `auto`. Therefore, on a 256 px × 256 px image, all the following `background-size` settings would yield an image with height and width of 50 px:

```
background-size: 50px;
background-size: 50px auto; /* same as above */
background-size: auto 50px;
background-size: 50px 50px;
```

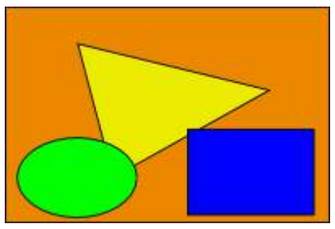
So if we started with the following picture (which has the mentioned size of 256 px × 256 px),



我们最终将在用户屏幕上得到一个50 px × 50 px的图像，包含在我们元素的背景中：

也可以使用百分比值根据元素来缩放图像。以下示例将产生一个200 px × 133 px绘制的图像：

```
#withbackground {
    background-image: url(to/some/background.png);

    background-size: 100% 66%;

    width: 200px;
    height: 200px;

    padding: 0;
    margin: 0;
}
```

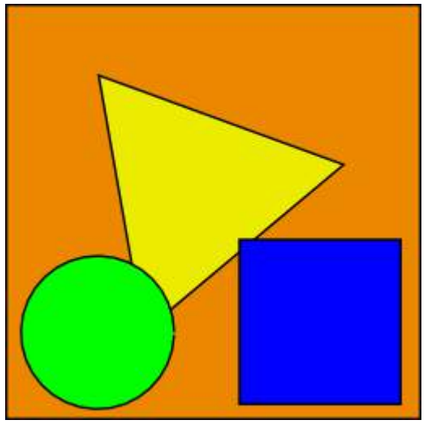


行为取决于background-origin。

保持宽高比

上一节的最后一个例子失去了其原始的宽高比。圆变成了椭圆，正方形变成了矩形，三角形变成了另一个三角形。

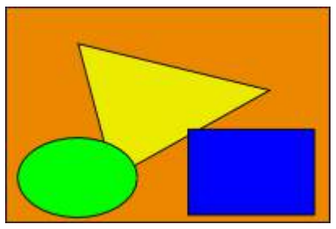
长度或百分比方法不足以始终保持宽高比的灵活性。`auto` 也无济于事，
因为你可能不知道元素的哪个维度会更大。然而，为了覆盖某些区域，



we'll end up with a 50 px × 50 px on the user's screen, contained in the background of our element:

One can also use percentage values to scale the image with respect of the element. The following example would yield a 200 px × 133 px drawn image:

```
#withbackground {
    background-image: url(to/some/background.png);

    background-size: 100% 66%;

    width: 200px;
    height: 200px;

    padding: 0;
    margin: 0;
}
```



The behaviour depends on the `background-origin`.

Keeping the aspect ratio

The last example in the previos section lost its original aspect ratio. The circle got into an ellipse, the square into a rectangle, the triangle into another triangle.

The length or percentage approach isn't flexible enough to keep the aspect ratio at all times. `auto` doesn't help, since you might not know which dimension of your element will be larger. However, to cover certain areas with an

image（以及正确的宽高比）完全显示，或在背景区域内完全包含一个具有正确宽高比的图像，值contain和cover提供了额外的功能。

对contain和cover的解释

抱歉这个冷笑话，但我们将使用Biswarup Ganguly的“每日图片”进行演示。假设这是你的屏幕，灰色区域是屏幕外不可见的部分。为了演示，我们假设宽高比为16×9。

我们想用上述的每日图片作为背景。然而，出于某种原因，我们将图像裁剪为4x3。我们可以将background-size属性设置为某个固定长度，但这里我们将重点关注contain和cover。注意，我还假设我们没有修改body的宽度和/或高度。

`contain`

> `contain`
>
> 缩放图像，同时保持其固有的宽高比（如果有），使其宽度和高度都能适应背景定位区域的最大尺寸。

这确保背景图像始终完全包含在背景定位区域内，然而，在这种情况下，可能会有一些空白区域被你的background-color填充：

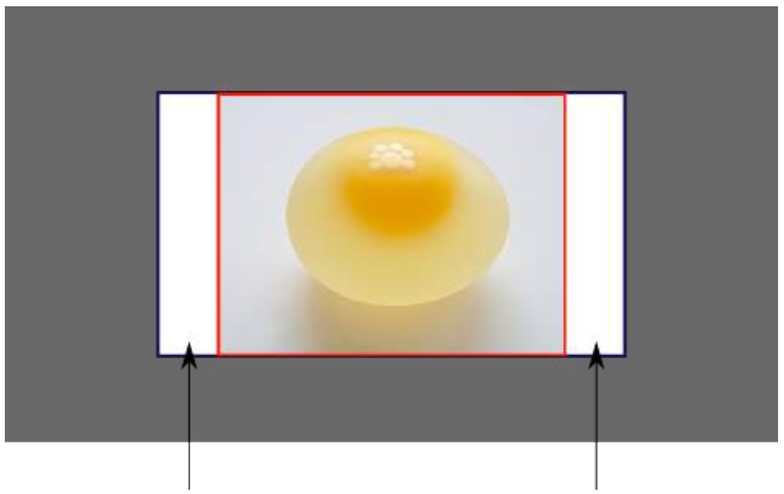


image (and correct aspect ratio) completely or to contain an image with correct aspect ratio completely in a background area, the values, `contain` and `cover` provide the additional functionality.

Eggsplanation for `contain` and `cover`

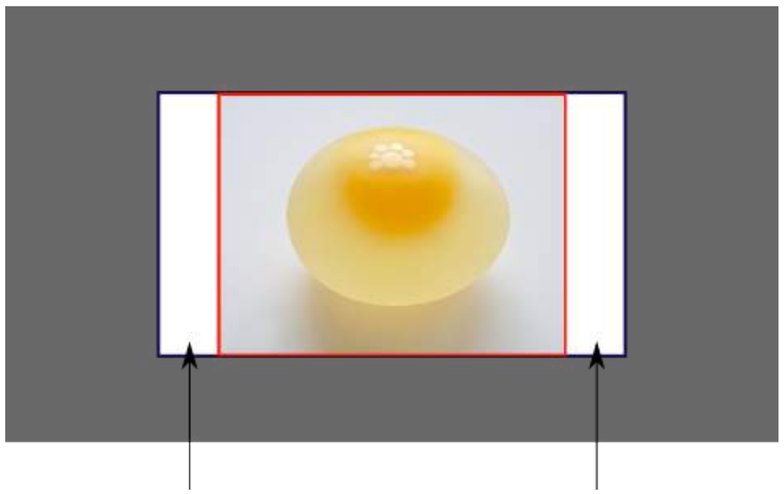
Sorry for the bad pun, but we're going to use a picture of the day by Biswarup Ganguly for demonstration. Lets say that this is your screen, and the gray area is outside of your visible screen. For demonstration, We're going to assume a 16 × 9 ratio.

We want to use the aforementioned picture of the day as a background. However, we cropped the image to 4x3 for some reason. We could set the `background-size` property to some fixed length, but we will focus on `contain` and `cover`. Note that I also assume that we didn't mangle the width and/or height of `body`.

`contain`

> `contain`
>
> Scale the image, while preserving its intrinsic aspect ratio (if any), to the largest size such that both its width and its height can fit inside the background positioning area.

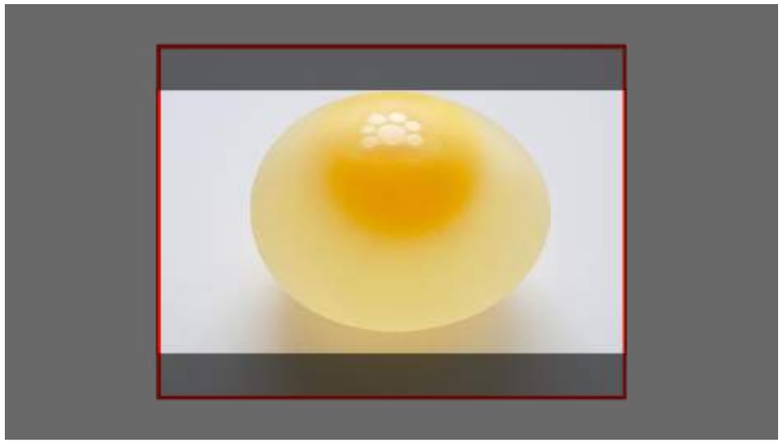
This makes sure that the background image is always completely contained in the background positioning area, however, there could be some empty space filled with your `background-color` in this case:



cover

> cover
>
> 缩放图像，同时保持其固有的宽高比（如果有的话），使其尺寸缩小到既能完全覆盖背景定位区域的宽度，又能完全覆盖高度的最小尺寸。

这确保背景图像覆盖所有区域。将不会看到可见的background-color，但根据屏幕比例，图像的大部分可能会被裁剪掉：



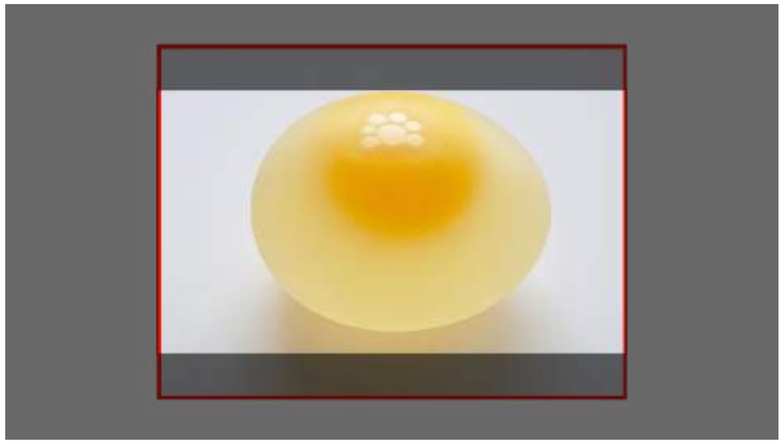
实际代码演示

```
div > div {
  background-image: url(http://i.stack.imgur.com/r5CAq.jpg);
  background-repeat: no-repeat;
  background-position: center center;
  background-color: #ccc;
  border: 1px solid;
  width: 20em;
  height: 10em;
}
div.contain {
  background-size: contain;
}
div.cover {
  background-size: cover;
}
/*******************************************
 说明框的附加样式
*******************************************/

div > div {
  margin: 0 1ex 1ex 0;
  float: left;
}
div + div {
  clear: both;
  border-top: 1px dashed silver;
  padding-top:1ex;
}
div > div::after {
  background-color: #000;
  color: #fefefe;
```


cover

> cover
>
> Scale the image, while preserving its intrinsic aspect ratio (if any), to the smallest size such that both its width and its height can completely cover the background positioning area.

This makes sure that the background image is covering everything. There will be no visible background-color, however depending on the screen's ratio a great part of your image could be cut off:



Demonstration with actual code

```
div > div {
  background-image: url(http://i.stack.imgur.com/r5CAq.jpg);
  background-repeat: no-repeat;
  background-position: center center;
  background-color: #ccc;
  border: 1px solid;
  width: 20em;
  height: 10em;
}
div.contain {
  background-size: contain;
}
div.cover {
  background-size: cover;
}
/*******************************************
 Additional styles for the explanation boxes
*******************************************/

div > div {
  margin: 0 1ex 1ex 0;
  float: left;
}
div + div {
  clear: both;
  border-top: 1px dashed silver;
  padding-top:1ex;
}
div > div::after {
  background-color: #000;
  color: #fefefe;
```

```
  margin: 1ex;
  padding: 1ex;
  opacity: 0.8;
  display: block;
  width: 10ex;
  font-size: 0.7em;
  content: attr(class);
}
```

```
<div>
  <div class="contain"></div>
  <p>注意灰色背景。图像并未覆盖整个区域，但它是完全
<em>包含</em>的。

</div>
<div>
  <div class="cover"></div>
  <p>请注意图像底部的鸭子/鹅。大部分水面被裁剪了，天空的一部分也被裁剪了。你不再看到完整的图像，但也看不到任何背景颜色；图像<em>覆盖</em>了整个<code>&lt;div&gt;</code>。</p>

</div>
```

Note the grey background. The image does not cover the whole region, but it's fully *contained*.

Note the ducks/geese at the bottom of the image. Most of the water is cut, as well as a part of the sky. You don't see the complete image anymore, but neither do you see any background color; the image *covers* all of the `<div>`.

第5.6节：背景位置

background-position 属性用于指定背景图像或渐变的起始位置

```
.myClass {
  background-image: url('path/to/image.jpg');
  background-position: 50% 50%;
}
```

该位置是使用X和Y坐标设置的，并且可以使用CSS中使用的任何单位进行设置。

单位	描述


```
  margin: 1ex;
  padding: 1ex;
  opacity: 0.8;
  display: block;
  width: 10ex;
  font-size: 0.7em;
  content: attr(class);
}
```

```
<div>
  <div class="contain"></div>
  <p>Note the grey background. The image does not cover the whole region, but it's fully
<em>contained</em>.
  </p>
</div>
<div>
  <div class="cover"></div>
  <p>Note the ducks/geese at the bottom of the image. Most of the water is cut, as well as a part of the sky. You don't see the complete image anymore, but neither do you see any background color; the image <em>covers</em> all of the <code>&lt;div&gt;</code>.</p>
</div>
```

Note the grey background. The image does not cover the whole region, but it's fully *contained*.

Note the ducks/geese at the bottom of the image. Most of the water is cut, as well as a part of the sky. You don't see the complete image anymore, but neither do you see any background color; the image *covers* all of the `<div>`.

Section 5.6: Background Position

The background-position property is used to specify the starting position for a background image or gradient

```
.myClass {
  background-image: url('path/to/image.jpg');
  background-position: 50% 50%;
}
```

The position is set using an **X** and **Y** co-ordinate and be set using any of the units used within CSS.

Unit	Description

	水平偏移的百分比是相对于（背景定位区域宽度 - *背景图像*）。
值% *值*%	垂直偏移的百分比是相对于（*背景定位区域高度* - *背景图像高度*）
	图像的大小是由background-size指定的大小。
值px 值px	以像素为单位相对于背景定位区域的左上角偏移背景图像
	定位区域

CSS中的单位可以通过不同的方法指定（见此处）。

长格式背景位置属性

除了上述简写属性外，还可以使用长格式背景属性`background-position-x`和`background-position-y`。这些属性允许你分别控制x轴或y轴的位置。

> 注意： 除了Firefox（版本31-48）2外，所有浏览器均支持此功能。Firefox_49将于2016年9月发布，将支持这些属性。在此之前，在此StackOverflow答案中有一个Firefox的解决方法。

第5.7节：background-origin属性

background-origin 属性指定背景图像的位置。

注意：如果background-attachment属性设置为fixed，则此属性无效。

默认值：`padding-box`

可能的取值：

- `padding-box` - 位置相对于内边距盒
- `border-box` - 位置相对于边框盒
- `content-box` - 位置相对于内容盒
- `initial`
- `继承`

CSS

```
.example {
  宽度: 300px;
  边框: 20px 实线 黑色;
  内边距: 50px;
  背景: url(https://static.pexels.com/photos/6440/magazines-desk-work-workspace-medium.jpg);
  背景重复: 不重复;
}

.example1 {}

.example2 { background-origin: border-box; }

.example3 { background-origin: content-box; }
```

HTML

```
<p>无 background-origin（默认是 padding-box）：</p>
```


	A percentage for the horizontal offset is relative to *(width of background positioning area - width of background image)*.
value% *value*%	A percentage for the vertical offset is relative to *(height of background positioning area - height of background image)* The size of the image is the size given by `background-size`.
*value*px *value*px	Offsets background image by a length given in pixels relative to the top left of the background positioning area

Units in CSS can be specified by different methods (see here).

Longhand Background Position Properties

In addition to the shorthand property above, one can also use the longhand background properties `background-position-x` and `background-position-y`. These allow you to control the x or y positions separately.

> **NOTE:** This is supported in all browsers except Firefox (versions 31-48) 2. Firefox 49, to be released September 2016, *will* support these properties. Until then, there is a Firefox hack within this Stack Overflow answer.

Section 5.7: The background-origin property

The background-origin property specifies where the background image is positioned.

Note: If the `background-attachment` property is set to `fixed`, this property has no effect.

Default value: `padding-box`

Possible values:

- `padding-box` - The position is relative to the padding box
- `border-box` - The position is relative to the border box
- `content-box` - The position is relative to the content box
- `initial`
- `inherit`

CSS

```
.example {
  width: 300px;
  border: 20px solid black;
  padding: 50px;
  background: url(https://static.pexels.com/photos/6440/magazines-desk-work-workspace-medium.jpg);
  background-repeat: no-repeat;
}

.example1 {}

.example2 { background-origin: border-box; }

.example3 { background-origin: content-box; }
```

HTML

```
<p>No background-origin (padding-box is default):</p>
```

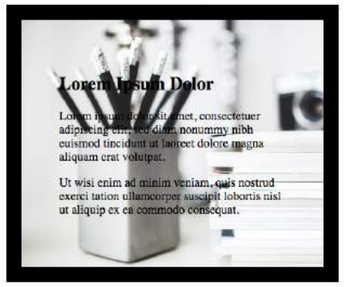
```
<div class="example example1">
  <h2>罗睿姆·伊普苏姆·多洛尔</h2>
  <p>罗睿姆·伊普苏姆·多洛尔·西特·阿梅特，康塞克图尔·阿迪皮斯辛·埃利特，塞德·迪亚姆·诺努米·尼布·欧伊斯莫德廷辛杜特·乌特·劳雷特·多洛雷·马格纳·阿利夸姆·埃拉特·沃卢特帕特。</p>
  <p>乌特·维西·埃尼姆·阿德·米尼姆·维尼亚姆，奎斯·诺斯特鲁德·埃克斯尔奇·塔蒂昂·乌拉姆科尔珀·苏斯皮西特·洛博尔蒂斯·尼斯尔·乌特阿利奎普·埃克斯·埃阿·科蒙多·康塞夸特。</p>
</div>

<p>background-origin: border-box:</p>
<div class="example example2">
  <h2>罗睿姆·伊普苏姆·多洛尔</h2>
  <p>罗睿姆·伊普苏姆·多洛尔·西特·阿梅特，康塞克图尔·阿迪皮斯辛·埃利特，塞德·迪亚姆·诺努米·尼布·欧伊斯莫德廷辛杜特·乌特·劳雷特·多洛雷·马格纳·阿利夸姆·埃拉特·沃卢特帕特。</p>
  <p>乌特·维西·埃尼姆·阿德·米尼姆·维尼亚姆，奎斯·诺斯特鲁德·埃克斯尔奇·塔蒂昂·乌拉姆科尔珀·苏斯皮西特·洛博尔蒂斯·尼斯尔·乌特阿利奎普·埃克斯·埃阿·科蒙多·康塞夸特。</p>
</div>

<p>background-origin: content-box:</p>
<div class="example example3">
  <h2>罗睿姆·伊普苏姆·多洛尔</h2>
  <p>罗睿姆·伊普苏姆·多洛尔·西特·阿梅特，康塞克图尔·阿迪皮斯辛·埃利特，塞德·迪亚姆·诺努米·尼布·欧伊斯莫德廷辛杜特·乌特·劳雷特·多洛雷·马格纳·阿利夸姆·埃拉特·沃卢特帕特。</p>
  <p>乌特·维西·埃尼姆·阿德·米尼姆·维尼亚姆，奎斯·诺斯特鲁德·埃克斯尔奇·塔蒂昂·乌拉姆科尔珀·苏斯皮西特·洛博尔蒂斯·尼斯尔·乌特阿利奎普·埃克斯·埃阿·科蒙多·康塞夸特。</p>
</div>
```

结果：


```
<div class="example example1">
  <h2>Lorem Ipsum Dolor</h2>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
</div>

<p>background-origin: border-box:</p>
<div class="example example2">
  <h2>Lorem Ipsum Dolor</h2>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
</div>

<p>background-origin: content-box:</p>
<div class="example example3">
  <h2>Lorem Ipsum Dolor</h2>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
</div>
```

Result:

No background-origin (padding-box is default):



background-origin: border-box:

background-origin: content-box:

更多：

https://www.w3.org/TR/css3-background/#the-background-origin

https://developer.mozilla.org/en-US/docs/Web/CSS/background-origin

第5.8节：多重背景图像

在 CSS3 中，我们可以在同一个元素中叠加多个背景。

```
#mydiv {
  background-image: url(img_1.png), /* 顶部图片 */
                    url(img_2.png), /* 中间图片 */
                    url(img_3.png); /* 底部图片 */
  background-position: 右下,
                       左上,
                       右上;
  background-repeat: 不重复,
                     重复,
                     不重复;
}
```

图片将依次叠加，第一张背景在最上层，最后一张背景在最底层。
img_1 会在最上层，img_2 和 img_3 在底层。

No background-origin (padding-box is default):

background-origin: border-box:

background-origin: content-box:

More:

https://www.w3.org/TR/css3-background/#the-background-origin

https://developer.mozilla.org/en-US/docs/Web/CSS/background-origin

Section 5.8: Multiple Background Image

In CSS3, we can stack multiple background in the same element.

```
#mydiv {
  background-image: url(img_1.png), /* top image */
                    url(img_2.png), /* middle image */
                    url(img_3.png); /* bottom image */
  background-position: right bottom,
                       left top,
                       right top;
  background-repeat: no-repeat,
                     repeat,
                     no-repeat;
}
```

Images will be stacked atop one another with the first background on top and the last background in the back. `img_1` will be on top, the `img_2` and `img_3` is on bottom.

我们也可以使用背景简写属性来实现：

```
#mydiv {
  background: url(img_1.png) 右下 不重复,
              url(img_2.png) 左上 重复,
              url(img_3.png) 右上不重复;
}
```

我们也可以叠加图像和渐变：

```
#mydiv {
  background: url(image.png) 右下不重复,
            线性渐变(向下, #fff 0%,#000 100%);
}
```

- 演示

第5.9节：背景附件

background-attachment属性设置背景图像是固定的还是随页面其余部分滚动。

```
body {
  background-image: url('img.jpg');
  background-attachment: 固定;
}
```

值	描述
scroll	背景随元素一起滚动。这是默认值。
fixed	背景相对于视口是固定的。
local	背景随元素内容一起滚动。
initial	将此属性设置为其默认值。
inherit	从其父元素继承此属性。

示例

background-attachment: scroll

默认行为，当页面滚动时，背景随之滚动：

```
body {
  background-image: url('image.jpg');
  background-attachment: scroll;
}
```

background-attachment: fixed

背景图像将固定，当页面滚动时不会移动：

```
body {
  background-image: url('image.jpg');
  background-attachment: fixed;
}
```

background-attachment: local

当 div 的内容滚动时，div 的背景图像也会滚动。

```
div {
```


We can also use background shorthand property for this:

```
#mydiv {
  background: url(img_1.png) right bottom no-repeat,
              url(img_2.png) left top repeat,
              url(img_3.png) right top no-repeat;
}
```

We can also stack images and gradients:

```
#mydiv {
  background: url(image.png) right bottom no-repeat,
              linear-gradient(to bottom, #fff 0%,#000 100%);
}
```

- Demo

Section 5.9: Background Attachment

The background-attachment property sets whether a background image is fixed or scrolls with the rest of the page.

```
body {
  background-image: url('img.jpg');
  background-attachment: fixed;
}
```

Value	Description
scroll	The background scrolls along with the element. This is default.
fixed	The background is fixed with regard to the viewport.
local	The background scrolls along with the element's contents.
initial	Sets this property to its default value.
inherit	Inherits this property from its parent element.

Examples

background-attachment: scroll

The default behaviour, when the body is scrolled the background scrolls with it:

```
body {
  background-image: url('image.jpg');
  background-attachment: scroll;
}
```

background-attachment: fixed

The background image will be fixed and will not move when the body is scrolled:

```
body {
  background-image: url('image.jpg');
  background-attachment: fixed;
}
```

background-attachment: local

The background image of the div will scroll when the contents of the div is scrolled.

```
div {
```

```
  background-image: url('image.jpg');
  background-attachment: local;
}
```

第5.10节：背景裁剪

定义与用法：background-clip属性指定背景的绘制区域。

默认值：`border-box`

取值

- border-box是默认值。它允许背景延伸到元素边框的外边缘。
- padding-box在元素内边距的外边缘裁剪背景，不允许背景延伸到边框内；
- content-box在内容框的边缘裁剪背景。
- inherit将父元素的设置应用到所选元素。

CSS

```
.example {
  宽度: 300px;
  边框: 20px 实线 黑色;
  内边距: 50px;
  背景: url(https://static.pexels.com/photos/6440/magazines-desk-work-workspace-medium.jpg);
  背景重复: 不重复;
}

.example1 {}

.example2 { background-origin: border-box; }

.example3 { background-origin: content-box; }
```

HTML

```
<p>无background-origin（默认是padding-box）：</p>

<div class="example example1">
  <h2>罗睿姆·伊普苏姆·多洛尔</h2>
  <p>罗睿姆·伊普苏姆·多洛尔·西特·阿梅特，康塞克图尔·阿迪皮斯辛·埃利特，塞德·迪亚姆·诺努米·尼布·欧伊斯莫德廷辛杜特·乌特·劳雷特·多洛雷·马格纳·阿利夸姆·埃拉特·沃卢特帕特。</p>
  <p>乌特·维西·埃尼姆·阿德·米尼姆·维尼亚姆，奎斯·诺斯特鲁德·埃克斯尔奇·塔蒂昂·乌拉姆科尔珀·苏斯皮西特·洛博尔蒂斯·尼斯尔·乌特阿利奎普·埃克斯·埃阿·科蒙多·康塞夸特。</p>
</div>

<p>background-origin: border-box:</p>
<div class="example example2">
  <h2>罗睿姆·伊普苏姆·多洛尔</h2>
  <p>罗睿姆·伊普苏姆·多洛尔·西特·阿梅特，康塞克图尔·阿迪皮斯辛·埃利特，塞德·迪亚姆·诺努米·尼布·欧伊斯莫德廷辛杜特·乌特·劳雷特·多洛雷·马格纳·阿利夸姆·埃拉特·沃卢特帕特。</p>
  <p>乌特·维西·埃尼姆·阿德·米尼姆·维尼亚姆，奎斯·诺斯特鲁德·埃克斯尔奇·塔蒂昂·乌拉姆科尔珀·苏斯皮西特·洛博尔蒂斯·尼斯尔·乌特阿利奎普·埃克斯·埃阿·科蒙多·康塞夸特。</p>
</div>

<p>background-origin: content-box:</p>
<div class="example example3">
  <h2>罗睿姆·伊普苏姆·多洛尔</h2>
```



```
  background-image: url('image.jpg');
  background-attachment: local;
}
```

Section 5.10: Background Clip

Definition and Usage: The `background-clip` property specifies the painting area of the background.

Default value: `border-box`

Values

- `border-box` is the default value. This allows the background to extend all the way to the outside edge of the element's border.
- `padding-box` clips the background at the outside edge of the element's padding and does not let it extend into the border;
- `content-box` clips the background at the edge of the content box.
- `inherit` applies the setting of the parent to the selected element.

CSS

```
.example {
  width: 300px;
  border: 20px solid black;
  padding: 50px;
  background: url(https://static.pexels.com/photos/6440/magazines-desk-work-workspace-medium.jpg);
  background-repeat: no-repeat;
}

.example1 {}

.example2 { background-origin: border-box; }

.example3 { background-origin: content-box; }
```

HTML

```
<p>No background-origin (padding-box is default):</p>

<div class="example example1">
  <h2>Lorem Ipsum Dolor</h2>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
</div>

<p>background-origin: border-box:</p>
<div class="example example2">
  <h2>Lorem Ipsum Dolor</h2>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
</div>

<p>background-origin: content-box:</p>
<div class="example example3">
  <h2>Lorem Ipsum Dolor</h2>
```

```
  <p>罗睿姆·伊普苏姆·多洛尔·西特·阿梅特，康塞克图尔·阿迪皮斯辛·埃利特，塞德·迪亚姆·诺努米·尼布·欧伊斯莫德
廷辛杜特·乌特·劳雷特·多洛雷·马格纳·阿利夸姆·埃拉特·沃卢特帕特。</p>
  <p>乌特·维西·埃尼姆·阿德·米尼姆·维尼亚姆，奎斯·诺斯特鲁德·埃克斯尔奇·塔蒂昂·乌拉姆科尔珀·苏斯皮西特·洛博尔蒂斯·尼斯尔·乌特
阿利奎普·埃克斯·埃阿·科蒙多·康塞夸特。</p>
</div>
```

第5.11节：背景重复

background-repeat属性设置背景图像是否以及如何重复。

默认情况下，背景图像会在垂直和水平方向上重复。

```
div {
  background-image: url("img.jpg");
  background-repeat: repeat-y;
}
```

下面是background-repeat: repeat-y 的效果示例：

第5.12节：background-blend-mode属性

```
.my-div {
    宽度: 300像素;
    高度: 200像素;
    background-size: 100%;
    background-repeat: no-repeat;
    background-image: 线性渐变(向右, 黑色 0%,白色 100%),
url('https://static.pexels.com/photos/54624/strawberry-fruit-red-sweet-54624-medium.jpeg');
    background-blend-mode:饱和度;
}
```

```
<div class="my-div">Lorem ipsum</div>
```



```
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod
tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut
aliquip ex ea commodo consequat.</p>
</div>
```

Section 5.11: Background Repeat

The background-repeat property sets if/how a background image will be repeated.

By default, a background-image is repeated both vertically and horizontally.

```
div {
  background-image: url("img.jpg");
  background-repeat: repeat-y;
}
```

Here's how a **background-repeat**: repeat-y looks like:

Section 5.12: background-blend-mode Property

```
.my-div {
    width: 300px;
    height: 200px;
    background-size: 100%;
    background-repeat: no-repeat;
    background-image: linear-gradient(to right, black 0%,white 100%),
url('https://static.pexels.com/photos/54624/strawberry-fruit-red-sweet-54624-medium.jpeg');
    background-blend-mode:saturation;
}
```

```
<div class="my-div">Lorem ipsum</div>
```

查看结果：https://jsfiddle.net/MadalinaTn/y69d28Lb/

CSS 语法：background-blend-mode: normal | multiply | screen | overlay | darken | lighten | color-dodge | 饱和度 | 颜色 | 亮度;

第5.13节：带透明度的背景颜色

如果你在元素上设置opacity，它会影响该元素的所有子元素。要仅对元素的背景设置透明度，必须使用RGBA颜色。以下示例将有一个透明度为0.6的黑色背景。

```
/* 不支持RGBA的浏览器的回退方案 */
background-color: rgb(0, 0, 0);

/* 透明度为0.6的RGBA */
background-color: rgba(0, 0, 0, 0.6);

/* 适用于IE 5.5 - 7*/
        filter: progid:DXImageTransform.Microsoft.gradient(startColorstr=#99000000, endColorstr=#99000000);

/* 适用于IE 8*/
-ms-filter: "progid:DXImageTransform.Microsoft.gradient(startColorstr=#99000000,
endColorstr=#99000000)";
```


See result here: https://jsfiddle.net/MadalinaTn/y69d28Lb/

CSS Syntax: background-blend-mode: normal | multiply | screen | overlay | darken | lighten | color-dodge | saturation | color | luminosity;

Section 5.13: Background Color with Opacity

If you set `opacity` on an element it will affect all its child elements. To set an opacity just on the background of an element you will have to use RGBA colors. Following example will have a black background with 0.6 opacity.

```
/* Fallback for web browsers that don't support RGBa */
background-color: rgb(0, 0, 0);

/* RGBa with 0.6 opacity */
background-color: rgba(0, 0, 0, 0.6);

/* For IE 5.5 - 7*/
filter: progid:DXImageTransform.Microsoft.gradient(startColorstr=#99000000, endColorstr=#99000000);

/* For IE 8*/
-ms-filter: "progid:DXImageTransform.Microsoft.gradient(startColorstr=#99000000,
endColorstr=#99000000)";
```

第6章：居中

第6.1节：使用Flexbox

HTML：

```
<div class="container">
  <img src="http://lorempixel.com/400/200" />
</div>
```

CSS：

```
html, body, .container {
  height: 100%;
}
.container {
  display: flex;
  justify-content: center; /* 水平居中 */
}
img {
  align-self: center; /* 垂直居中 */
}
```

查看结果

HTML：

```
<img src="http://lorempixel.com/400/200" />
```

CSS：

```
html, body {
  height: 100%;
}
body {
  display: flex;
  justify-content: center; /* 水平居中 */
  align-items: center;     /* 垂直居中 */
}
```

查看结果

有关 flexbox 及样式含义的更多详情，请参见 Flexbox 文档中的动态垂直和水平居中部分。

浏览器支持

Flexbox 被所有主流浏览器支持，除 IE 10 之前的版本外。

一些较新的浏览器版本，如 Safari 8 和 IE10，需要 厂商前缀。

快速生成前缀的方法有 Autoprefixer，这是一个第三方工具。

对于较旧的浏览器（如 IE 8 和 9），有 Polyfill 可用。

有关 flexbox 浏览器支持的更详细信息，请参见this answer。

Chapter 6: Centering

Section 6.1: Using Flexbox

HTML:

```
<div class="container">
  <img src="http://lorempixel.com/400/200" />
</div>
```

CSS:

```
html, body, .container {
  height: 100%;
}
.container {
  display: flex;
  justify-content: center; /* horizontal center */
}
img {
  align-self: center; /* vertical center */
}
```

View Result

HTML:

```
<img src="http://lorempixel.com/400/200" />
```

CSS:

```
html, body {
  height: 100%;
}
body {
  display: flex;
  justify-content: center; /* horizontal center */
  align-items: center;     /* vertical center */
}
```

View Result

See Dynamic Vertical and Horizontal Centering under the Flexbox documentation for more details on flexbox and what the styles mean.

Browser Support

Flexbox is supported by all major browsers, except IE versions before 10.

Some recent browser versions, such as Safari 8 and IE10, require vendor prefixes.

For a quick way to generate prefixes there is Autoprefixer, a third-party tool.

For older browsers (like IE 8 & 9) a Polyfill is available.

For a more detailed look at flexbox browser support, see this answer.

第6.2节：使用 CSS 变换

CSS 变换是基于元素的大小的，所以如果你不知道元素的高度或宽度，可以
将其相对于一个相对定位的容器绝对定位于顶部和左侧的50%，并向左和向上平移50%，
以实现垂直和水平居中。

请注意，使用此技术时，元素可能最终会渲染在非整数像素边界上，
这会导致其看起来模糊。有关解决方法，请参见this answer in SO。

HTML

```
<div class="container">
  <div class="element"></div>
</div>
```

CSS

```
.container {
  position: relative;
}

.element {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

在 JSFiddle 中查看示例

跨浏览器兼容性

transform 属性需要添加前缀以支持较旧的浏览器。Chrome 版本小于等于35、Safari 小于等于8、Opera 小于等于22、Android 浏览器小于等于4.4.4 和 IE9 需要添加前缀。IE8 及更早版本不支持 CSS 变换。

以下是前面示例的常见 transform 声明：

```
-webkit-transform: translate(-50%, -50%); /* Chrome、Safari、Opera、Android */
-ms-transform: translate(-50%, -50%); /* IE 9 */
        transform: translate(-50%, -50%);
```

更多信息请参见 canIuse。

更多信息

- 该元素根据第一个非静态定位的父元素（`position: relative`、`absolute` 或 `fixed`）进行定位。可在此 fiddle 和相关文档主题中了解更多。
- 仅水平居中时，使用 left: 50% 和 transform: translateX(-50%)。仅垂直居中时，使用 top: 50% 和 transform: translateY(-50%)。
- 使用非静态宽度/高度的元素进行此种居中方法时，居中元素可能会显得被压缩。这通常发生在包含文本的元素上，可通过添加：margin-right: -50%; 和 margin-bottom: -50%; 来修复。更多信息请查看此 fiddle。

Section 6.2: Using CSS transform

CSS transforms are based on the size of the elements so if you don't know how tall or wide your element is, you can position it absolutely 50% from the top and left of a relative container and translate it by 50% left and upwards to center it vertically and horizontally.

Keep in mind that with this technique, the element could end being rendered at a non-integer pixel boundary, making it look blurry. See this answer in SO for a workaround.

HTML

```
<div class="container">
  <div class="element"></div>
</div>
```

CSS

```
.container {
  position: relative;
}

.element {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

View example in JSFiddle

CROSS BROWSER COMPATIBILITY

The transform property needs prefixes to be supported by older browsers. Prefixes are needed for Chrome<=35, Safari<=8, Opera<=22, Android Browser<=4.4.4, and IE9. CSS transforms are not supported by IE8 and older versions.

Here is a common transform declaration for the previous example:

```
-webkit-transform: translate(-50%, -50%); /* Chrome, Safari, Opera, Android */
   -ms-transform: translate(-50%, -50%); /* IE 9 */
       transform: translate(-50%, -50%);
```

For more information see canluse.

MORE INFORMATION

- The element is being positioned according to the first non-static parent (`position: relative`, `absolute`, or `fixed`). Explore more in this fiddle and this documentation topic.
- For horizontal-only centering, use `left: 50%` and `transform: translateX(-50%)`. The same goes for vertical-only centering: center with `top: 50%` and `transform: translateY(-50%)`.
- Using a non-static width/height elements with this method of centering can cause the centered element to appear squished. This mostly happens with elements containing text, and can be fixed by adding: `margin-right: -50%;` and `margin-bottom: -50%;`. View this fiddle for more information.

第6.3节：使用 margin: 0 auto;

如果对象是块级元素且具有定义的宽度，可以使用margin: 0 auto;将其居中。

HTML

```
<div class="containerDiv">
    <div id="centeredDiv"></div>
</div>

<div class="containerDiv">
    <p id="centeredParagraph">这是一个居中的段落。</p>
</div>

<div class="containerDiv">
    <img id="centeredImage"
src="https://i.kinja-img.com/gawker-media/image/upload/s--c7Q9b4Eh--/c_scale,fl_progressive,q_80,w_
800/qqyvc3bkpyl3mfhr8all.jpg" />
</div>
```

CSS

```
.containerDiv {
    width: 100%;
    height: 100px;
    padding-bottom: 40px;
}

#centeredDiv {
    margin: 0 auto;
    width: 200px;
    height: 100px;
    border: 1px 实线 #000;
}

#centeredParagraph {
    width: 200px;
    margin: 0 auto;
}

#centeredImage {
    display: block;
    width: 200px;
    margin: 0 auto;
}
```

结果：

Section 6.3: Using margin: 0 auto;

Objects can be centered by using `margin: 0 auto;` if they are block elements and have a defined width.

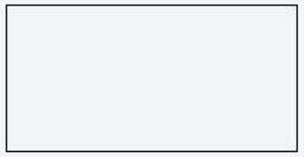
HTML

```
<div class="containerDiv">
    <div id="centeredDiv"></div>
</div>

<div class="containerDiv">
    <p id="centeredParagraph">This is a centered paragraph.</p>
</div>

<div class="containerDiv">
    <img id="centeredImage"
src="https://i.kinja-img.com/gawker-media/image/upload/s--c7Q9b4Eh--/c_scale,fl_progressive,q_80,w_
800/qqyvc3bkpyl3mfhr8all.jpg" />
</div>
```

CSS

```
.containerDiv {
    width: 100%;
    height: 100px;
    padding-bottom: 40px;
}

#centeredDiv {
    margin: 0 auto;
    width: 200px;
    height: 100px;
    border: 1px solid #000;
}

#centeredParagraph {
    width: 200px;
    margin: 0 auto;
}

#centeredImage {
    display: block;
    width: 200px;
    margin: 0 auto;
}
```

Result:



This is a centered paragraph.

JSFiddle 示例：使用 margin: 0 auto; 居中对象；

第6.4节：使用 text-align

最常见且最简单的居中方式是将元素中的文本行居中。CSS 有规则 text-align：center 用于此目的：

HTML

```
<p>Lorem ipsum</p>
```

CSS

```
p {
    text-align: center;
}
```

这不能用于居中整个块级元素。 text-align 仅控制其父块元素中内联内容（如文本）的对齐方式。

关于text-align的更多内容，请参见排版部分。

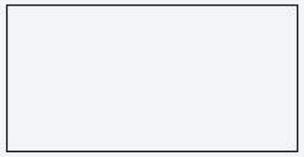
第6.5节：使用 position: absolute

在旧浏览器中工作（IE >= 8）

自动外边距，配合left和right或top和bottom偏移量为零的值，将使绝对定位元素在其父元素内居中。

查看结果

HTML



This is a centered paragraph.

JSFiddle example: Centering objects with margin: 0 auto;

Section 6.4: Using text-align

The most common and easiest type of centering is that of lines of text in an element. CSS has the rule **`text-align`**: `center` for this purpose:

HTML

```
<p>Lorem ipsum</p>
```

CSS

```
p {
    text-align: center;
}
```

This does not work for centering entire block elements. `text-align` controls only alignment of inline content like text in its parent block element.

See more about `text-align` in Typography section.

Section 6.5: Using position: absolute

Working in old browsers (IE >= 8)

Automatic margins, paired with values of zero for the `left` and `right` or `top` and `bottom` offsets, will center an absolutely positioned elements within its parent.

View Result

HTML

```
<div class="parent">
  <img class="center" src="http://lorempixel.com/400/200/" />
</div>
```

CSS

```
.parent {
  position: relative;
  height: 500px;
}

.center {
  position: absolute;
  margin: auto;
  上: 0;
  右: 0;
  下: 0;
  左: 0;
}
```

没有自身隐式宽度和高度的元素（如图片）需要定义这些值。

其他资源：CSS中的绝对居中

第6.6节：使用calc()

calc()函数是CSS3中新语法的一部分，你可以通过使用像素、百分比等多种数值来数学计算元素所占的大小/位置。注意：每当使用此函数时，务必注意两个数值之间的空格calc(100% - 80px)。

CSS

```
.center {
    position: absolute;
    height: 50px;
    width: 50px;
    background: red;
    top: calc(50% - 50px / 2); /* 高度除以2 */
    left: calc(50% - 50px / 2); /* 宽度除以2 */
}
```

HTML

```
<div class="center"></div>
```

第6.7节：使用line-height

你也可以使用line-height来垂直居中容器内的单行文本：

CSS

```
div {
    height: 200px;
    line-height: 200px;
}
```



```
<div class="parent">
  <img class="center" src="http://lorempixel.com/400/200/" />
</div>
```

CSS

```
.parent {
  position: relative;
  height: 500px;
}

.center {
  position: absolute;
  margin: auto;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
}
```

Elements that don't have their own implicit width and height like images do, will need those values defined.

Other resources: Absolute Centering in CSS

Section 6.6: Using calc()

The calc() function is the part of a new syntax in CSS3 in which you can calculate (mathematically) what size/position your element occupies by using a variety of values like pixels, percentages, etc. Note: Whenever you use this function, always take care of the space between two values `calc(100% - 80px)`.

CSS

```
.center {
    position: absolute;
    height: 50px;
    width: 50px;
    background: red;
    top: calc(50% - 50px / 2); /* height divided by 2*/
    left: calc(50% - 50px / 2); /* width divided by 2*/
}
```

HTML

```
<div class="center"></div>
```

Section 6.7: Using line-height

You can also use `line-height` to center vertically a single line of text inside a container :

CSS

```
div {
    height: 200px;
    line-height: 200px;
}
```

这看起来相当难看，但在<input />元素内部可能有用。line-height属性仅在要居中的文本为单行时有效。如果文本换行成多行，最终输出将不会居中。

第6.8节：用3行代码实现垂直对齐任何内容

支持IE11+

查看结果

使用这3行代码几乎可以垂直对齐所有内容。只需确保你应用代码的div/图片有一个具有高度的父元素。

CSS

```
div.vertical {
  position: relative;
  top: 50%;
  transform: translateY(-50%);
}
```

HTML

```
<div class="vertical">垂直对齐的文本！</div>
```

第6.9节：相对于另一个元素的居中

我们将学习如何根据相邻元素的高度来居中内容。

兼容性：IE8 及以上，所有其他现代浏览器。

HTML

```
<div class="content">
  <div class="position-container">
    <div class="thumb">
      <img src="http://lorempixel.com/400/200/">
    </div>
    <div class="details">
      <p class="banner-title">文本 1</p>
      <p class="banner-text">内容 内容 内容 内容 内容 内容 内容 内容
内容 内容 内容 内容 内容 内容</p>
      <button class="btn">按钮</button>
    </div>
  </div>
</div>
```

CSS

```
.content * {
  box-sizing: border-box;
}
.content .position-container {
  display: table;
}
.content .details {
  display: table-cell;
  vertical-align: middle;
```


That's quite ugly, but can be useful inside an `<input />` element. The `line-height` property works only when the text to be centered spans a single line. If the text wraps into multiple lines, the resulting output won't be centered.

Section 6.8: Vertical align anything with 3 lines of code

Supported by IE11+

View Result

Use these 3 lines to vertical align practically everything. Just make sure the div/image you apply the code to has a parent with a height.

CSS

```
div.vertical {
  position: relative;
  top: 50%;
  transform: translateY(-50%);
}
```

HTML

```
<div class="vertical">Vertical aligned text!</div>
```

Section 6.9: Centering in relation to another item

We will see how to center content based on the height of a near element.

Compatibility: IE8+, all other modern browsers.

HTML

```
<div class="content">
  <div class="position-container">
    <div class="thumb">
      <img src="http://lorempixel.com/400/200/">
    </div>
    <div class="details">
      <p class="banner-title">text 1</p>
      <p class="banner-text">content content content content content content content content
content content content content content content</p>
      <button class="btn">button</button>
    </div>
  </div>
</div>
```

CSS

```
.content * {
  box-sizing: border-box;
}
.content .position-container {
  display: table;
}
.content .details {
  display: table-cell;
  vertical-align: middle;
```

```
  width: 33.333333%;
  padding: 30px;
  font-size: 17px;
  text-align: center;
}
.content .thumb {
  width: 100%;
}
.content .thumb img {
  width: 100%;
}
```

Link to JSFiddle

The main points are the 3 .thumb, .details and .position-container containers:

- 该.position-container必须具有display: table属性。
- 详细信息.details 必须设置真实宽度 width: .... 并且 display: table-cell，vertical-align: middle。
- 如果你希望 .thumb 占据所有剩余空间并且受 .details 宽度影响，则必须设置 width: 100%。
- 如果你有图片，.thumb 内的图片应设置 width: 100%，但如果比例正确则不必强制设置。

第6.10节：幽灵元素技术（Michał Czernow的技巧）

即使容器尺寸未知，该技术仍然有效。

在容器内设置一个“幽灵”元素，使其高度为100%，然后对该元素和要居中的元素都使用 vertical-align: middle。

CSS

```
/* 该父元素可以是任意宽度和高度 */
.block {
  text-align: center;

  /* 如果容器可能比内部元素窄，可能需要这样做 */
  white-space: nowrap;
                                ;
  height: 100%;
  vertical-align: middle;

  /* 幽灵元素和.centered之间存在间隙，
  由渲染的空格字符引起。可以通过
  微调.centered（微调距离取决于字体）来消除，
  或者在.parent中将字体大小设为零，然后在.centered中重置字体大小
  （可能为1rem）来消除。 */
  margin-right: -0.25em;
}
```



```
  width: 33.333333%;
  padding: 30px;
  font-size: 17px;
  text-align: center;
}
.content .thumb {
  width: 100%;
}
.content .thumb img {
  width: 100%;
}
```


Link to JSFiddle

The main points are the 3 `.thumb`, `.details` and `.position-container` containers:

- The `.position-container` must have **`display`**: `table`.
- The `.details` must have the real width set **`width`**: `....` and **`display`**: `table-cell`, **`vertical-align`**: `middle`.
- The `.thumb` must have **`width`**: `100%` if you want that it will take all the remaining space and it will be influenced by the `.details` width.
- The image (if you have an image) inside `.thumb` should have **`width`**: `100%`, but it is not necessary if you have correct proportions.

Section 6.10: Ghost element technique (Michał Czernow's hack)

This technique works even when the container's dimensions are unknown.

Set up a "ghost" element inside the container to be centered that is 100% height, then use **`vertical-align`**: `middle` on both that and the element to be centered.

CSS

```
/* This parent can be any width and height */
.block {
  text-align: center;

  /* May want to do this if there is risk the container may be narrower than the element inside */
  white-space: nowrap;
}

/* The ghost element */
.block:before {
  content: '';
  display: inline-block;
  height: 100%;
  vertical-align: middle;

  /* There is a gap between ghost element and .centered,
  caused by space character rendered. Could be eliminated by
  nudging .centered (nudge distance depends on font family),
  or by zeroing font-size in .parent and resetting it back
  (probably to 1rem) in .centered. */
  margin-right: -0.25em;
}
```

```css
/* 需要居中的元素，也可以是任意宽度和高度 */
.centered {
  display: inline-block;
  vertical-align: middle;
  width: 300px;
  white-space: normal; /* 重置继承的 nowrap 行为 */
}
```

HTML

```html
<div class="block">
  <div class="centered"></div>
</div>
```

第6.11节：无需担心高度或宽度即可实现垂直和水平居中

下面的技巧允许你将内容添加到HTML元素中，并实现水平和垂直居中而无需担心其高度或宽度。

外部容器

- 应设置 display: table;

内部容器

- 应设置 display: table-cell;
- **应设置 vertical-align: middle;**
- 应设置 text-align: center;

内容框

- 应该有 display: inline-block;
- 应该重新调整水平文本对齐方式，例如 `text-align: left;` 或 `text-align: right;`，除非你想让文本居中

演示

HTML

```html
<div class="outer-container">
    <div class="inner-container">
      <div class="centered-content">
        你可以在这里放任何内容！
      </div>
    </div>
</div>
```

CSS

```css
body {
      margin : 0;
}

.outer-container {
      position : absolute;
      display: table;
      width: 100%; /* 这可以是任意宽度 */
```



```css
/* The element to be centered, can also be of any width and height */
.centered {
  display: inline-block;
  vertical-align: middle;
  width: 300px;
  white-space: normal; /* Resetting inherited nowrap behavior */
}
```

HTML

```html
<div class="block">
  <div class="centered"></div>
</div>
```

Section 6.11: Centering vertically and horizontally without worrying about height or width

The following technique allows you to add your content to an HTML element and center it both horizontally and vertically **without worrying about its height or width**.

The outer container

- should have `display: table;`

The inner container

- should have `display: table-cell;`
- should have `vertical-align: middle;`
- should have `text-align: center;`

The content box

- should have `display: inline-block;`
- should re-adjust the horizontal text-alignment to eg. `text-align: left;` or `text-align: right;`, unless you want text to be centered

Demo

HTML

```html
<div class="outer-container">
   <div class="inner-container">
     <div class="centered-content">
        You can put anything here!
     </div>
   </div>
</div>
```

CSS

```css
body {
    margin : 0;
}

.outer-container {
    position : absolute;
    display: table;
    width: 100%; /* This could be ANY width */
```

```
    高度: 100%; /* 这可以是任意高度 */
    背景: #ccc;
}

.inner-container {
    display: table-cell;
    vertical-align: middle;
    text-align: center;
}

.centered-content {
    display: inline-block;
    text-align: left;
    background: #fff;
    padding: 20px;
    border: 1px solid #000;
}
```

另见 this Fiddle!

第6.12节：在div内垂直对齐图像

HTML

```
<div class="wrap">
    <img src="http://lorempixel.com/400/200/" />
</div>
```

CSS

```
.wrap {
     height: 50px;/* 最大图像高度 */
     width: 100px;
     border: 1像素实线蓝色;
     文本对齐: 居中;
}
.wrap:before {
   内容:"";
   显示: 行内块;
   高度: 100%;
   垂直对齐: 居中;
   宽度: 1像素;
}

img {
     垂直对齐: 居中;
}
```

第6.13节：固定尺寸居中

如果内容的尺寸是固定的，可以使用绝对定位到50%，并用margin减去内容宽度和高度的一半：

HTML

```
<div class="center">
    垂直和水平居中
</div>
```



```
    height: 100%; /* This could be ANY height */
    background: #ccc;
}

.inner-container {
    display: table-cell;
    vertical-align: middle;
    text-align: center;
}

.centered-content {
    display: inline-block;
    text-align: left;
    background: #fff;
    padding: 20px;
    border: 1px solid #000;
}
```

See also this Fiddle!

Section 6.12: Vertically align an image inside div

HTML

```
<div class="wrap">
    <img src="http://lorempixel.com/400/200/" />
</div>
```

CSS

```
.wrap {
    height: 50px;/* max image height */
    width: 100px;
    border: 1px solid blue;
    text-align: center;
}
.wrap:before {
  content:"";
  display: inline-block;
  height: 100%;
  vertical-align: middle;
  width: 1px;
}

img {
    vertical-align: middle;
}
```

Section 6.13: Centering with fixed size

If the size of your content is fixed, you can use absolute positioning to 50% with `margin` that reduces half of your content's width and height:

HTML

```
<div class="center">
    Center vertically and horizontally
</div>
```

CSS

```
.center {
    定位: 绝对;
    背景色: #ccc;

    左侧: 50%;
    宽度: 150像素;
    margin-left: -75px;  /* 宽度 * -0.5 */

    top: 50%;
    高度: 200像素;
    margin-top: -100px;  /* 高度 * -0.5 */
}
```

仅固定宽度的水平居中

即使不知道内容的高度，也可以水平居中元素：

HTML

```
<div class="center">
    仅水平居中
</div>
```

CSS

```
.center {
    定位: 绝对;
    背景色: #ccc;

    左侧: 50%;
    宽度: 150像素;
    margin-left: -75px;  /* 宽度 * -0.5 */
}
```

固定高度的垂直居中

如果知道元素的高度，可以垂直居中元素：

HTML

```
<div class="center">
    仅垂直居中
</div>
```

CSS

```
.center {
    position: absolute;
    background: #ccc;

    top: 50%;
    高度: 200像素;
    margin-top: -100px;  /* 宽度 * -0.5 */
}
```


CSS

```
.center {
    position: absolute;
    background: #ccc;

    left: 50%;
    width: 150px;
    margin-left: -75px;  /* width * -0.5 */

    top: 50%;
    height: 200px;
    margin-top: -100px;  /* height * -0.5 */
}
```

Horizontal centering with only fixed width

You can center the element horizontally even if you don't know the height of the content:

HTML

```
<div class="center">
    Center only horizontally
</div>
```

CSS

```
.center {
    position: absolute;
    background: #ccc;

    left: 50%;
    width: 150px;
    margin-left: -75px;  /* width * -0.5 */
}
```

Vertical centering with fixed height

You can center the element vertically if you know the element's height:

HTML

```
<div class="center">
    Center only vertically
</div>
```

CSS

```
.center {
    position: absolute;
    background: #ccc;

    top: 50%;
    height: 200px;
    margin-top: -100px;  /* width * -0.5 */
}
```

第6.14节：垂直对齐动态高度元素

直观地应用CSS不会产生预期效果，因为

- **vertical-align:middle 不适用于块级元素**
- **margin-top**:auto 和 **margin-bottom**:auto 使用的值计算结果为***零***
- **margin-top**:-50% 基于百分比的外边距值是相对于包含块的***宽度***计算的

为了获得最广泛的浏览器支持，使用带辅助元素的解决方法：

HTML

```
<div class="vcenter--container">
  <div class="vcenter--helper">
    <div class="vcenter--content">
      <!--内容-->
    </div>
  </div>
</div>
```

CSS

```
.vcenter--container {
  display: table;
  height: 100%;
  position: absolute;
  overflow: hidden;
  width: 100%;
}
.vcenter--helper {
  display: table-cell;
  vertical-align: middle;
}
.vcenter--content {
  margin: 0 auto;
  width: 200px;
}
```

jsfiddle 来自 原始问题。此方法

- 适用于动态高度元素
- 尊重内容流
- 被旧版浏览器支持

第6.15节：使用表格布局进行水平和垂直居中

可以轻松使用表格显示属性来居中子元素。

HTML

```
<div class="wrapper">
    <div class="parent">
        <div class="child"></div>
    </div>
</div>
```


Section 6.14: Vertically align dynamic height elements

Applying css intuitively doesn't produce the desired results because

- **vertical-align**:middle ***isn't*** applicable to block-level elements
- **margin-top**:auto and **margin-bottom**:auto used values would compute as ***zero***
- **margin-top**:-50% percentage-based margin values are calculated relative to the ***width*** of containing block

For widest browser support, a workaround with helper elements:

HTML

```
<div class="vcenter--container">
  <div class="vcenter--helper">
    <div class="vcenter--content">
      <!--stuff-->
    </div>
  </div>
</div>
```

CSS

```
.vcenter--container {
  display: table;
  height: 100%;
  position: absolute;
  overflow: hidden;
  width: 100%;
}
.vcenter--helper {
  display: table-cell;
  vertical-align: middle;
}
.vcenter--content {
  margin: 0 auto;
  width: 200px;
}
```

jsfiddle from original question. This approach

- works with dynamic height elements
- respects content flow
- is supported by legacy browsers

Section 6.15: Horizontal and Vertical centering using table layout

One could easily center a child element using `table` display property.

HTML

```
<div class="wrapper">
    <div class="parent">
        <div class="child"></div>
    </div>
</div>
```

CSS

```
.wrapper {
    display: table;
    vertical-align: center;
    width: 200px;
    高度: 200像素;
    background-color: #9e9e9e;
}
.parent {
    display: table-cell;
    vertical-align: middle;
    text-align: center;
}
.child {
    display: inline-block;
    vertical-align: middle;
    text-align: center;
    width: 100px;
    height: 100px;
    background-color: teal;
}
```


CSS

```
.wrapper {
    display: table;
    vertical-align: center;
    width: 200px;
    height: 200px;
    background-color: #9e9e9e;
}
.parent {
    display: table-cell;
    vertical-align: middle;
    text-align: center;
}
.child {
    display: inline-block;
    vertical-align: middle;
    text-align: center;
    width: 100px;
    height: 100px;
    background-color: teal;
}
```

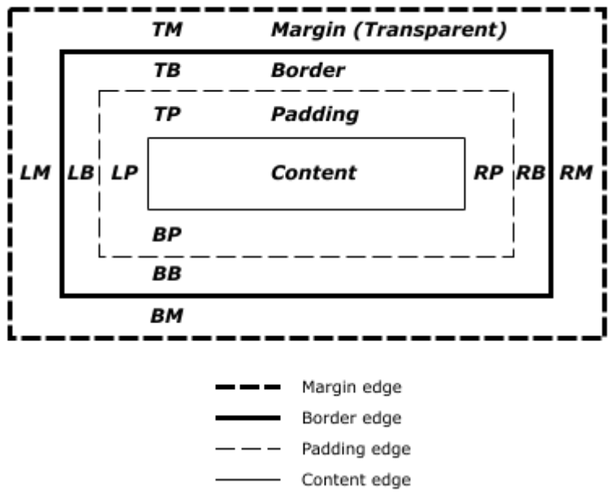
第7章：盒模型

参数	详细信息
content-box	元素的宽度和高度仅包括内容区域。
padding-box	元素的宽度和高度包括内容和内边距。
border-box	元素的宽度和高度包括内容、内边距和边框。
初始值	将盒模型设置为默认状态。
继承	继承父元素的盒模型。

第7.1节：什么是盒模型？

边缘

浏览器为HTML文档中的每个元素创建一个矩形。盒模型描述了如何将内边距、边框和外边距添加到内容中以形成该矩形。



图示来自CSS2.2工作草案

四个区域的周长称为边缘。每个边缘定义一个盒子。

- 最内层的矩形是内容盒。其宽度和高度取决于元素渲染的内容（文本、图像及其可能包含的任何子元素）。
- 接下来是由padding属性定义的内边距盒。如果未定义padding宽度，内边距边缘等于内容边缘。
- 然后是由border属性定义的边框盒。如果未定义border宽度，边框边缘等于内边距边缘。
- 最外层的矩形是margin box（外边距盒），由margin属性定义。如果没有定义margin宽度，外边距边缘等于边框边缘。

示例

```
div {
    border: 5px 实线 红色;
    margin: 50px;
    padding: 20px;
```


Chapter 7: The Box Model

Parameter	Detail
`content-box`	Width and height of the element only includes content area.
`padding-box`	Width and height of the element includes content and padding.
`border-box`	Width and height of the element includes content, padding and border.
`initial`	Sets the box model to its default state.
`inherit`	Inherits the box model of the parent element.

Section 7.1: What is the Box Model?

The Edges

The browser creates a rectangle for each element in the HTML document. The Box Model describes how the padding, border, and margin are added to the content to create this rectangle.

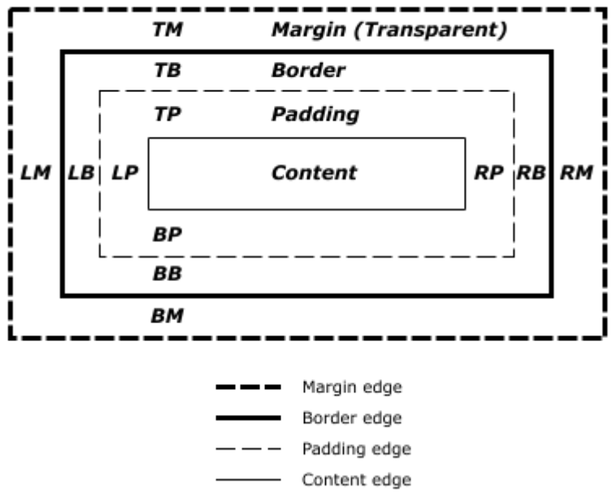


Diagram from CSS2.2 Working Draft

The perimeter of each of the four areas is called an *edge*. Each edge defines a *box.*

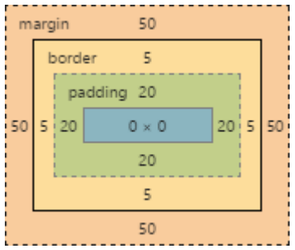
- The innermost rectangle is the **content box**. The width and height of this depends on the element's rendered content (text, images and any child elements it may have).
- Next is the **padding box**, as defined by the `padding` property. If there is no `padding` width defined, the padding edge is equal to the content edge.
- Then we have the **border box**, as defined by the `border` property. If there is no `border` width defined, the border edge is equal to the padding edge.
- The outermost rectangle is the **margin box**, as defined by the `margin` property. If there is no `margin` width defined, the margin edge is equal to the border edge.

Example

```
div {
    border: 5px solid red;
    margin: 50px;
    padding: 20px;
```

```
}
```

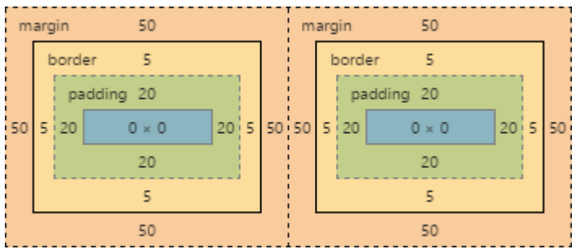
此CSS样式使所有div元素的上、右、下、左边框宽度均为5px；上、右、下、左外边距均为50px；上、右、下、左内边距均为20px。忽略内容，我们生成的盒子将如下所示：



Google Chrome元素样式面板截图

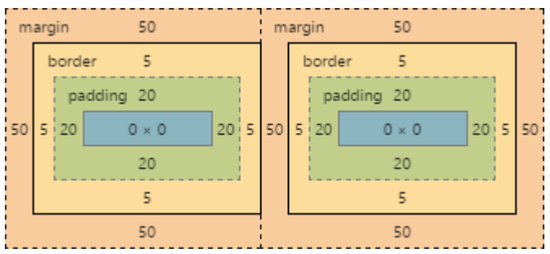
- 由于没有内容，内容区域（中间的蓝色盒子）没有高度和宽度（0px × 0px）。
- 内边距盒默认与内容盒大小相同，加上我们通过padding属性定义的四边各20px宽度（共40px × 40px）。
- 边框盒大小与内边距盒相同，加上我们通过`border`属性定义的5px宽度（共50px × 50px）。
- 最后，外边距盒大小与边框盒相同，加上我们通过margin属性（使我们的元素总尺寸为150px乘150px）。

现在让我们给元素添加一个具有相同样式的兄弟元素。浏览器会查看两个元素的盒模型，来计算新元素相对于前一个元素内容的位置：



每个元素的内容之间有150像素的间隙，但两个元素的盒子是相互接触的。

如果我们修改第一个元素，使其没有右边距，那么右边距边缘将与右边框边缘重合，两个元素现在看起来会是这样：

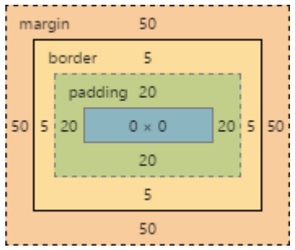


第7.2节：box-sizing

默认的盒模型（content-box）可能会让人感到不直观，因为一旦你开始为元素添加padding和border样式，元素的width/height就不再代表其在屏幕上的实际宽度或高度


```
}
```

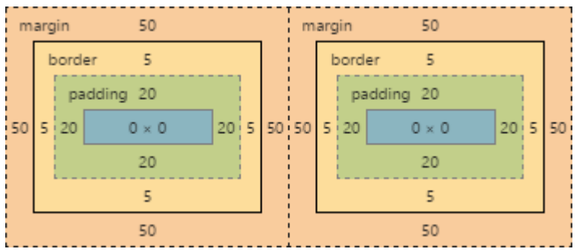
This CSS styles all `div` elements to have a top, right, bottom and left border of `5px` in width; a top, right, bottom and left margin of `50px`; and a top, right, bottom, and left padding of `20px`. Ignoring content, our generated box will look like this:



Screenshot of Google Chrome's Element Styles panel

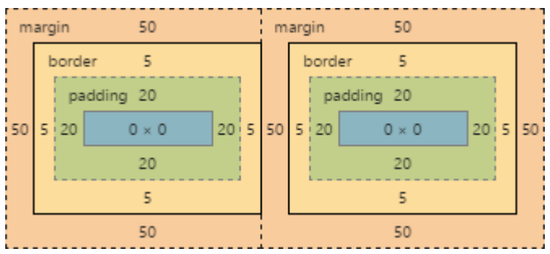
- As there is no content, the content region (the blue box in the middle) has no height or width (0px by 0px).
- The padding box by default is the same size as the content box, plus the 20px width on all four edges we're defining above with the `padding` property (40px by 40px).
- The border box is the same size as the padding box, plus the 5px width we're defining above with the `border` property (50px by 50px).
- Finally the margin box is the same size as the border box, plus the 50px width we're defining above with the `margin` property (giving our element a total size of 150px by 150px).

Now lets give our element a sibling with the same style. The browser looks at the Box Model of both elements to work out where in relation to the previous element's content the new element should be positioned:



The content of each of element is separated by a 150px gap, but the two elements' boxes touch each other.

If we then modify our first element to have no right margin, the right margin edge would be in the same position as the right border edge, and our two elements would now look like this:



Section 7.2: box-sizing

The default box model (`content-box`) can be counter-intuitive, since the `width` / `height` for an element will not represent its actual width or height on screen as soon as you start adding `padding` and `border` styles to the

元素。

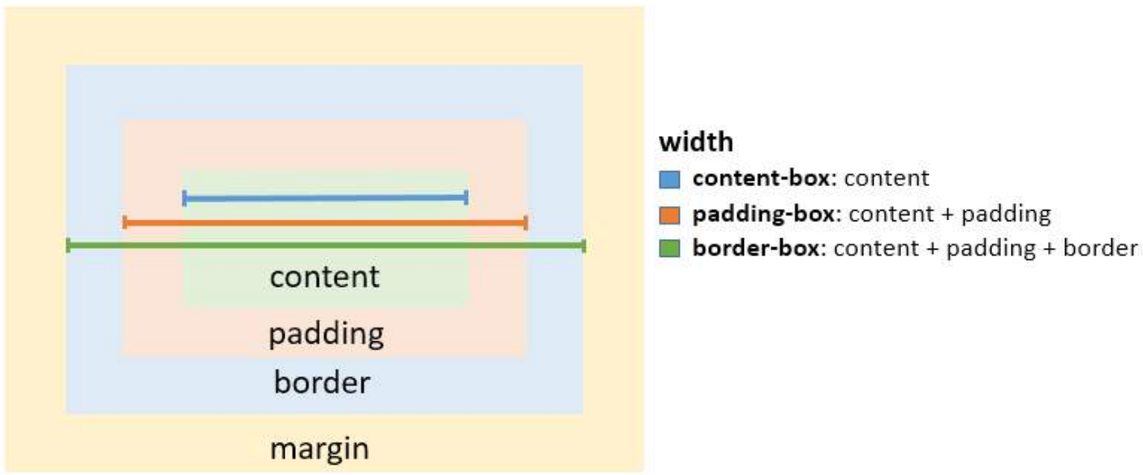
下面的示例演示了content-box可能带来的这个问题：

```
textarea {
    width: 100%;
    padding: 3px;
    box-sizing: content-box; /* 默认值 */
}
```

由于内边距会被加到文本区域的宽度上，最终元素的宽度会比
100%更宽。

幸运的是，CSS允许我们通过box-sizing属性来改变元素的盒模型。该属性有三种不同的取值：

- `content-box`：常见的盒模型——宽度和高度只包括内容，不包括内边距或边框
- `padding-box`：宽度和高度包括内容和内边距，但不包括边框
- `border-box`：宽度和高度包括内容、内边距以及边框



为了解决上述textarea问题，你可以将box-sizing属性改为padding-box或border-box。通常使用的是border-box。

```
textarea {
    width: 100%;
    padding: 3px;
    box-sizing: border-box;
}
```

要对页面上的每个元素应用特定的盒模型，可以使用以下代码片段：

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

*, *:之前, *:之后 {
```


element.

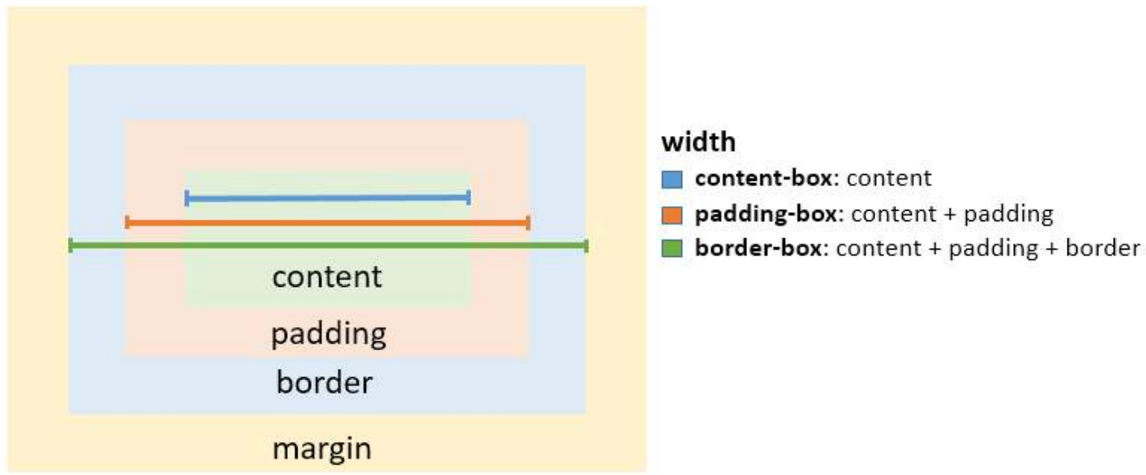
The following example demonstrates this potential issue with `content-box`:

```
textarea {
    width: 100%;
    padding: 3px;
    box-sizing: content-box; /* default value */
}
```

Since the padding will be added to the width of the textarea, the resulting element is a textarea that is wider than 100%.

Fortunately, CSS allows us to change the box model with the `box-sizing` property for an element. There are three different values for the property available:

- `content-box`: The common box model - width and height only includes the content, not the padding or border
- `padding-box`: Width and height includes the content and the padding, but not the border
- `border-box`: Width and height includes the content, the padding as well as the border



To solve the `textarea` problem above, you could just change the `box-sizing` property to `padding-box` or `border-box`. `border-box` is most commonly used.

```
textarea {
    width: 100%;
    padding: 3px;
    box-sizing: border-box;
}
```

To apply a specific box model to every element on the page, use the following snippet:

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

*, *:before, *:after {
```

```
    box-sizing: inherit;
}
```

在此代码中，box-sizing:border-box;并未直接应用于*，因此你可以轻松地在单个元素上覆盖此属性。


```
    box-sizing: inherit;
}
```

In this coding `box-sizing:border-box;` is not directly applied to `*`, so you can easily overwrite this property on individual elements.

第8章：外边距

参数	详情
0	将外边距设置为无
自动	用于居中，通过在每一侧均匀设置数值实现
单位（例如 px）	请参见单位部分的参数列表，了解有效单位
inherit	从父元素继承外边距值
初始值	恢复到初始值

第8.1节：外边距合并

当两个外边距在垂直方向相接触时，它们会合并。当两个外边距在水平方向相接触时，它们不会合并。

相邻垂直外边距的示例：

考虑以下样式和标记：

```
div{
    margin: 10px;
}
```

```
<div>
一些内容
</div>
<div>
更多内容
</div>
```

它们之间的间距将是10像素，因为垂直外边距会相互合并。（间距不会是两个外边距之和。）

相邻水平边距示例：

考虑以下样式和标记：

```
span{
    margin: 10px;
}
```

```
<span>一些</span><span>内容</span>
```

它们将相距20像素，因为水平外边距不会相互折叠。（间距将是两个外边距的总和。）

不同大小的重叠

```
.top{
    margin: 10px;
}
.bottom{
    margin: 15px;
}
```

```
<div class="top">
    一些内容
```


Chapter 8: Margins

Parameter	Details
0	set margin to none
auto	used for centering, by evenly setting values on each side
units (e.g. px)	see parameter section in Units for a list of valid units
inherit	inherit margin value from parent element
initial	restore to initial value

Section 8.1: Margin Collapsing

When two margins are touching each other vertically, they are collapsed. When two margins touch horizontally, they do not collapse.

Example of adjacent vertical margins:

Consider the following styles and markup:

```
div{
    margin: 10px;
}
```

```
<div>
    some content
</div>
<div>
    some more content
</div>
```

They will be 10px apart since vertical margins collapse over one and other. (The spacing will not be the sum of two margins.)

Example of adjacent horizontal margins:

Consider the following styles and markup:

```
span{
    margin: 10px;
}
```

```
<span>some</span><span>content</span>
```

They will be 20px apart since horizontal margins don't collapse over one and other. (The spacing will be the sum of two margins.)

Overlapping with different sizes

```
.top{
    margin: 10px;
}
.bottom{
    margin: 15px;
}
```

```
<div class="top">
    some content
```

```
</div>
<div class="bottom">
    一些更多内容
</div>
```

这些元素将垂直间隔15像素。边距尽可能重叠，但较大的边距将决定元素之间的间距。

重叠边距陷阱

```
.outer-top{
    margin: 10px;
}
.inner-top{
    margin: 15px;
}
.outer-bottom{
    margin: 20px;
}
.inner-bottom{
    margin: 25px;
}
```

```
<div class="outer-top">
    <div class="inner-top">
        一些内容
    </div>
</div>
<div class="outer-bottom">
    <div class="inner-bottom">
        一些更多内容
    </div>
</div>
```

这两段文字之间的间距是多少？（悬停查看答案）

> 间距将是25像素。由于四个边距都相互接触，它们会合并，因此使用四个中最大的边距。

那么，如果我们给上面的标记添加一些边框，会怎样呢？

```
div{
    border: 1px 实线 红色;
}
```

这两段文字之间的间距是多少？（悬停查看答案）

> 间距将是59像素！现在只有.outer-top和.outer-bottom的边距相互接触，并且是唯一合并的边距。其余的边距被边框分隔开了。所以我们有1px +10px + 1px + 15px + 20px + 1px + 25px + 1px。（1px的是边框...）

父元素和子元素之间的边距合并：

HTML：


```
</div>
<div class="bottom">
    some more content
</div>
```

These elements will be spaced 15px apart vertically. The margins overlap as much as they can, but the larger margin will determine the spacing between the elements.

Overlapping margin gotcha

```
.outer-top{
    margin: 10px;
}
.inner-top{
    margin: 15px;
}
.outer-bottom{
    margin: 20px;
}
.inner-bottom{
    margin: 25px;
}
```

```
<div class="outer-top">
    <div class="inner-top">
        some content
    </div>
</div>
<div class="outer-bottom">
    <div class="inner-bottom">
        some more content
    </div>
</div>
```

What will be the spacing between the two texts? (hover to see answer)

> The spacing will be 25px. Since all four margins are touching each other, they will collapse, thus using the largest margin of the four.

Now, what about if we add some borders to the markup above.

```
div{
    border: 1px solid red;
}
```

What will be the spacing between the two texts? (hover to see answer)

> The spacing will be 59px! Now only the margins of .outer-top and .outer-bottom touch each other, and are the only collapsed margins. The remaining margins are separated by the borders. So we have 1px + 10px + 1px + 15px + 20px + 1px + 25px + 1px. (The 1px's are the borders...)

Collapsing Margins Between Parent and Child Elements:

HTML:

```
<h1>标题</h1>
<div>
  <p>段落</p>
</div>
```

CSS

```
h1 {
  margin: 0;
  background: #cff;
}
div {
  margin: 50px 0 0 0;
  background: #cfc;
}
p {
  margin: 25px 0 0 0;
  background: #cf9;
}
```

在上面的例子中，只有最大的外边距生效。你可能会预期段落距离 h1 会有 60 像素（因为 div 元素有 40 像素的上外边距，而 p 有 20 像素的上外边距）。这种情况没有发生，是因为外边距合并形成了一个外边距。

第 8.2 节：在指定边应用外边距

方向特定属性

CSS 允许你指定某一边应用外边距。为此提供了四个属性：

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

下面的代码会对选中的 div 元素的左侧应用 30 像素的外边距。查看结果

HTML

```
<div id="myDiv"></div>
```

CSS

```
#myDiv {
    margin-left: 30px;
    height: 40px;
    width: 40px;
    background-color: red;
}
```

参数	详情
margin-left	边距应应用的方向。
30像素	边距的宽度。

使用简写属性指定方向

标准的margin属性可以扩展为为所选元素的每一侧指定不同的宽度。
语法如下：


```
<h1>Title</h1>
<div>
  <p>Paragraph</p>
</div>
```

CSS

```
h1 {
  margin: 0;
  background: #cff;
}
div {
  margin: 50px 0 0 0;
  background: #cfc;
}
p {
  margin: 25px 0 0 0;
  background: #cf9;
}
```

In the example above, only the largest margin applies. You may have expected that the paragraph would be located 60px from the h1 (since the div element has a margin-top of 40px and the p has a 20px margin-top). This does not happen because the margins collapse together to form one margin.

Section 8.2: Apply Margin on a Given Side

Direction-Specific Properties

CSS allows you to specify a given side to apply margin to. The four properties provided for this purpose are:

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

The following code would apply a margin of 30 pixels to the left side of the selected div. *View Result*

HTML

```
<div id="myDiv"></div>
```

CSS

```
#myDiv {
    margin-left: 30px;
    height: 40px;
    width: 40px;
    background-color: red;
}
```

Parameter	Details
margin-left	The direction in which the margin should be applied.
30px	The width of the margin.

Specifying Direction Using Shorthand Property

The standard `margin` property can be expanded to specify differing widths to each side of the selected elements.
The syntax for doing this is as follows:

```
margin: <上> <右> <下> <左>;
```

以下示例为div的顶部应用零宽度边距，右侧应用10像素边距，左侧应用50像素边距，左侧应用100像素边距。*查看结果*

HTML

```
<div id="myDiv"></div>
```

CSS

```
#myDiv {
    margin: 0 10px 50px 100px;
    height: 40px;
    width: 40px;
    background-color: red;
}
```

第8.3节：边距属性简化

```
p {
    margin:1px;                /* 所有方向的1像素边距 */

    /* 等同于: */

    margin:1px 1px;

    /* 等同于: */

    margin:1px 1px 1px;

    /* 等同于: */

    margin:1px 1px 1px 1px;
}
```

另一个例子：

```
p{
    margin:10px 15px;          /* 上下边距10像素，左右边距15像素 */

    /* 等同于: */

    margin:10px 15px 10px 15px;

    /* 等同于: */

    margin:10px 15px 10px;
    /* 左边距将从右边距值（=15px）计算 */
}
```

第8.4节：使用

margin水平居中页面元素

只要元素是一个块级元素，并且它有一个明确设置的宽度值，边距就可以用来水平居中页面上的块级元素。


```
margin: <top> <right> <bottom> <left>;
```

The following example applies a zero-width margin to the top of the div, a 10px margin to the right side, a 50px margin to the left side, and a 100px margin to the left side. *View Result*

HTML

```
<div id="myDiv"></div>
```

CSS

```
#myDiv {
    margin: 0 10px 50px 100px;
    height: 40px;
    width: 40px;
    background-color: red;
}
```

Section 8.3: Margin property simplification

```
p {
    margin:1px;                /* 1px margin in all directions */

    /*equals to:*/

    margin:1px 1px;

    /*equals to:*/

    margin:1px 1px 1px;

    /*equals to:*/

    margin:1px 1px 1px 1px;
}
```

Another exapmle:

```
p{
    margin:10px 15px;          /* 10px margin-top & bottom And 15px margin-right & left*/

    /*equals to:*/

    margin:10px 15px 10px 15px;

    /*equals to:*/

    margin:10px 15px 10px;
    /* margin left will be calculated from the margin right value (=15px) */
}
```

Section 8.4: Horizontally center elements on a page using margin

As long as the element is a **block**, and it has an **explicitly set width value**, margins can be used to center block elements on a page horizontally.

我们添加一个比窗口宽度更小的宽度值，然后 margin 的 auto 属性将剩余空间分配到左侧和右侧：

```
#myDiv {
 width:80%;
 margin:0 auto;
}
```

在上面的例子中，我们使用了简写的margin声明，首先将顶部和底部的 margin 值设置为0（虽然这可以是任何值），然后使用auto让浏览器自动分配左右的 margin 空间。

在上面的例子中，#myDiv 元素的宽度设置为 80%，剩余 20% 空间由浏览器分配到左右两侧，因此：

(100% - 80%) / 2 = 10%

第 8.5 节：示例 1：

显然，margin-left 和 margin-right 的百分比值应相对于其父元素。

```
.parent {
      width : 500px;
      height: 300px;
}

.child {
      width : 100px;
      height: 100px;
      margin-left: 10%;  /* （父元素宽度 * 10/100）=> 50px */
}
```

但情况并非如此，对于`margin-top`和`margin-bottom`来说。这两个属性的百分比值
不是相对于父容器的高度，而是相对于父容器的**宽度**。

所以，

```
.parent {
      width : 500px;
      height: 300px;
}

.child {
      width : 100px;
      height: 100px;
      margin-left: 10%;   /* （父元素宽度 * 10/100）=> 50px  */
      margin-top: 20%;    /* （父元素宽度 * 20/100）=> 100px */
}
```

第8.6节：负边距

边距是少数可以设置为负值的CSS属性之一。该属性可以用来**重叠
elements而无需绝对定位**

```
div{
```


We add a width value that is lower than the width of the window and the auto property of margin then distributes the remaining space to the left and the right:

```
#myDiv {
 width:80%;
 margin:0 auto;
}
```

In the example above we use the shorthand `margin` declaration to first set `0` to the top and bottom margin values (although this could be any value) and then we use `auto` to let the browser allocate the space automatically to the left and right margin values.

In the example above, the #myDiv element is set to 80% width which leaves use 20% leftover. The browser distributes this value to the remaining sides so:

(100% - 80%) / 2 = 10%

Section 8.5: Example 1:

It is obvious to assume that the percentage value of margin to `margin-left` and `margin-right` would be relative to its parent element.

```
.parent {
    width : 500px;
    height: 300px;
}

.child {
    width : 100px;
    height: 100px;
    margin-left: 10%;  /* (parentWidth * 10/100) => 50px */
}
```

But that is not the case, when comes to `margin-top` and `margin-bottom`. Both these properties, in percentages, aren't relative to the height of the parent container but to the **width** of the parent container.

So,

```
.parent {
    width : 500px;
    height: 300px;
}

.child {
    width : 100px;
    height: 100px;
    margin-left: 10%;  /* (parentWidth * 10/100) => 50px  */
    margin-top: 20%;   /* (parentWidth * 20/100) => 100px */
}
```

Section 8.6: Negative margins

Margin is one of a few CSS properties that can be set to negative values. This property can be used to **overlap elements without absolute positioning**.

```
div{
```

```
  display: inline;
}

#over{
  margin-left: -20px;
}

<div>基础 div</div>
<div id="over">重叠 div</div>
```



```
  display: inline;
}

#over{
  margin-left: -20px;
}

<div>Base div</div>
<div id="over">Overlapping div</div>
```

第9章：内边距

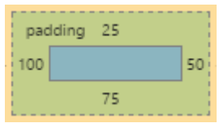
第9.1节：内边距简写

padding 属性设置元素四周的内边距空间。内边距区域是元素内容与其边框之间的空间。不允许使用负值。

为了避免单独为每一边添加内边距（使用padding-top、padding-left等），你可以将其写成简写形式，如下所示：

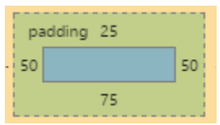
四个值：

```
<style>
    .myDiv {
padding: 25px 50px 75px 100px; /* 上 右 下 左； */
    }
</style>
<div class="myDiv"></div>
```



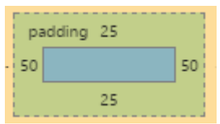
三个值：

```
<style>
.myDiv {
padding: 25px 50px 75px; /* 上 右/左 下 */
    }
</style>
<div class="myDiv"></div>
```



两个值：

```
<style>
.myDiv {
padding: 25px 50px; /* 上/下 右/左 */
    }
</style>
<div class="myDiv"></div>
```



一个值：

```
<style>
.myDiv {
```


Chapter 9: Padding

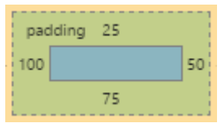
Section 9.1: Padding Shorthand

The padding property sets the padding space on all sides of an element. The padding area is the space between the content of the element and its border. Negative values are not allowed.

To save adding padding to each side individually (using `padding-top`, `padding-left` etc) can you write it as a shorthand, as below:

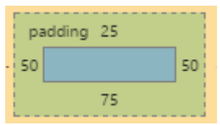
Four values:

```
<style>
    .myDiv {
        padding: 25px 50px 75px 100px; /* top right bottom left; */
    }
</style>
<div class="myDiv"></div>
```



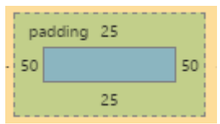
Three values:

```
<style>
    .myDiv {
        padding: 25px 50px 75px; /* top left/right bottom */
    }
</style>
<div class="myDiv"></div>
```



Two values:

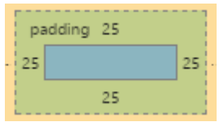
```
<style>
    .myDiv {
        padding: 25px 50px; /* top/bottom left/right */
    }
</style>
<div class="myDiv"></div>
```



One value:

```
<style>
    .myDiv {
```

```
padding: 25px; /* 上/右/下/左 */
    }
</style>
<div class="myDiv"></div>
```



第9.2节：指定某一侧的内边距

padding 属性设置元素四周的内边距空间。内边距区域是元素内容与其边框之间的空间。不允许使用负值。

你可以单独指定某一侧：

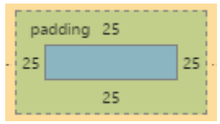
- padding-top
- padding-right
- padding-bottom
- padding-left

以下代码会给div的顶部添加5px的内边距：

```
<style>
.myClass {
padding-top: 5px;
}
</style>

<div class="myClass"></div>
```



```
        padding: 25px; /* top/right/bottom/left */
    }
</style>
<div class="myDiv"></div>
```



Section 9.2: Padding on a given side

The padding property sets the padding space on all sides of an element. The padding area is the space between the content of the element and its border. Negative values are not allowed.

You can specify a side individually:

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

The following code would add a padding of `5px` to the top of the div:

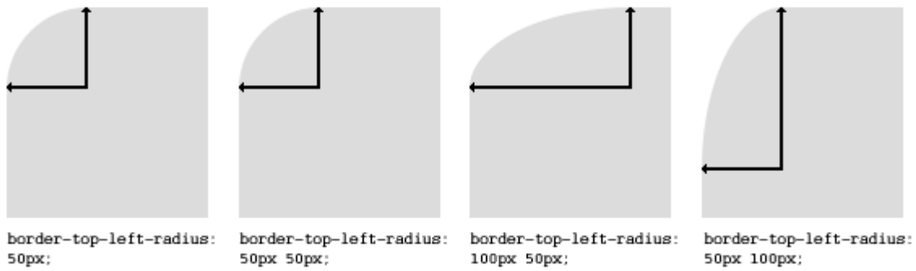
```
<style>
.myClass {
    padding-top: 5px;
}
</style>

<div class="myClass"></div>
```

第10章：边框

第10.1节：border-radius（边框圆角）

border-radius属性允许你改变基本盒模型的形状。

元素的每个角最多可以有两个值，分别表示该角的垂直半径和水平半径（最多8个值）。



第一组值定义水平半径。可选的第二组值，前面带有'/'，定义垂直半径。如果只提供一组值，则该值同时用于垂直和水平半径。

```
border-radius: 10px 5% / 20px 25em 30px 35em;
```

10px是左上角和右下角的水平半径。5%是右上角和左下角的水平半径。'/'后面的四个值分别是左上角、右上角、右下角和左下角的垂直半径。

与许多CSS属性一样，可以使用简写来设置任意或所有可能的值。因此，你可以指定从一个到八个值。以下简写允许你将每个角的水平和垂直半径设置为相同的值：

HTML：

```
<div class='box'></div>
```

CSS：

```
.box {
    宽度: 250px;
    高度: 250px;
    背景色: 黑色;
    边框圆角: 10px;
}
```

边框圆角（border-radius）最常用于将盒子元素转换为圆形。通过将边框圆角设置为正方形元素边长的一半，可以创建一个圆形元素：

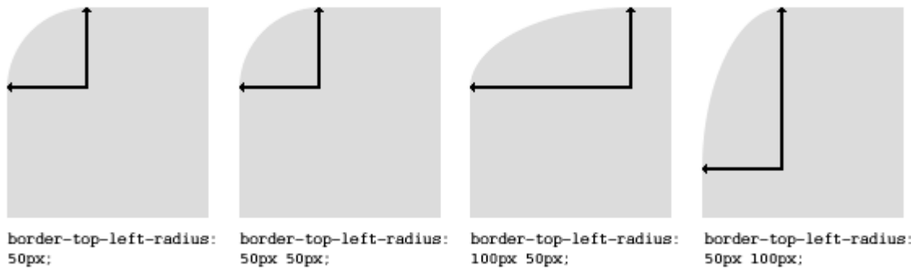
```
.circle {
    宽度: 200px;
    高度: 200px;
    边框圆角: 100px;
}
```


Chapter 10: Border

Section 10.1: border-radius

The border-radius property allows you to change the shape of the basic box model.

Every corner of an element can have up to two values, for the vertical and horizontal radius of that corner (for a maximum of 8 values).



The first set of values defines the horizontal radius. The optional second set of values, preceded by a '/' , defines the vertical radius. If only one set of values is supplied, it is used for both the vertical and horizontal radius.

```
border-radius: 10px 5% / 20px 25em 30px 35em;
```

The `10px` is the horizontal radius of the top-left-and-bottom-right. And the `5%` is the horizontal radius of the top-right-and-bottom-left. The other four values after '/' are the vertical radii for top-left, top-right, bottom-right and bottom-left.

As with many CSS properties, shorthands can be used for any or all possible values. You can therefore specify anything from one to eight values. The following shorthand allows you to set the horizontal and vertical radius of every corner to the same value:

HTML:

```
<div class='box'></div>
```

CSS:

```
.box {
    width: 250px;
    height: 250px;
    background-color: black;
    border-radius: 10px;
}
```

Border-radius is most commonly used to convert box elements into circles. By setting the border-radius to half of the length of a square element, a circular element is created:

```
.circle {
    width: 200px;
    height: 200px;
    border-radius: 100px;
}
```

由于边框圆角支持百分比，通常使用50%来避免手动计算边框圆角的数值：

```
.circle {
    宽度: 150px;
    高度: 150px;
    边框圆角: 50%;
}
```

如果宽度和高度属性不相等，生成的形状将是椭圆而非圆形。

浏览器特定的边框圆角示例：

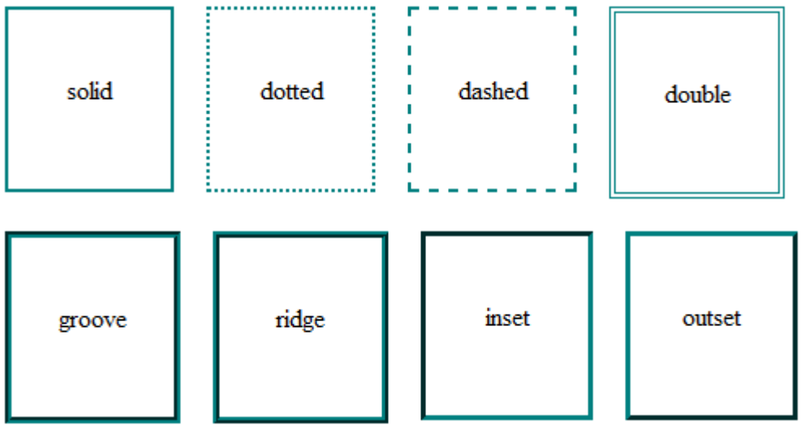
```
-webkit-border-top-right-radius: 4px;
-webkit-border-bottom-right-radius: 4px;
-webkit-border-bottom-left-radius: 0;
-webkit-border-top-left-radius: 0;
-moz-border-radius-topright: 4px;
-moz-border-radius-bottomright: 4px;
-moz-border-radius-bottomleft: 0;
-moz-border-radius-topleft: 0;
  border-top-right-radius: 4px;
  border-bottom-right-radius: 4px;
  border-bottom-left-radius: 0;
  border-top-left-radius: 0;
```

第10.2节：border-style

border-style属性设置元素边框的样式。该属性可以有一到四个值（分别对应元素的每一边）。

示例：

```
border-style: 点状;
border-style: 点状 实线 双实线 虚线;
```



border-style 也可以取值为 none 和 hidden。它们的效果相同，除了 hidden 适用于border 冲突解决，针对 <table> 元素。在具有多个边框的 <table> 中，none 优先级最低（意味着冲突时边框会显示），而 hidden 优先级最高（意味着冲突时边框不会显示）。

Because border-radius accepts percentages, it is common to use 50% to avoid manually calculating the border-radius value:

```
.circle {
    width: 150px;
    height: 150px;
    border-radius: 50%;
}
```

If the width and height properties are not equal, the resulting shape will be an oval rather than a circle.

Browser specific border-radius example:

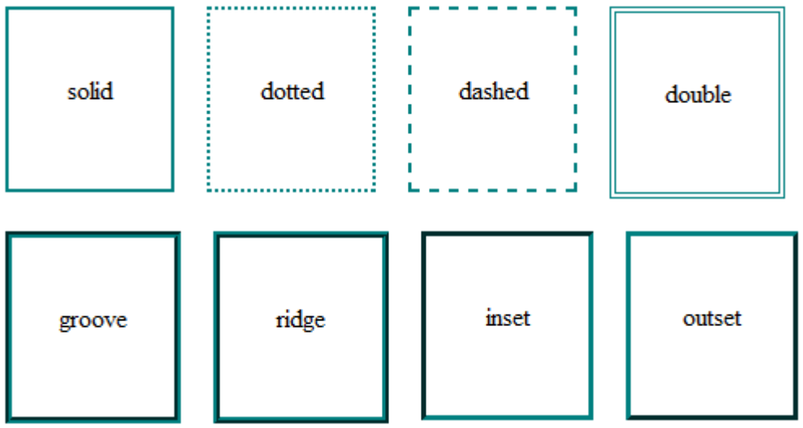
```
  -webkit-border-top-right-radius: 4px;
  -webkit-border-bottom-right-radius: 4px;
  -webkit-border-bottom-left-radius: 0;
  -webkit-border-top-left-radius: 0;
  -moz-border-radius-topright: 4px;
  -moz-border-radius-bottomright: 4px;
  -moz-border-radius-bottomleft: 0;
  -moz-border-radius-topleft: 0;
  border-top-right-radius: 4px;
  border-bottom-right-radius: 4px;
  border-bottom-left-radius: 0;
  border-top-left-radius: 0;
```

Section 10.2: border-style

The `border-style` property sets the style of an element's border. This property can have from one to four values (for every side of the element one value.)

Examples:

```
border-style: dotted;
border-style: dotted solid double dashed;
```



`border-style` can also have the values `none` and `hidden`. They have the same effect, except `hidden` works for border conflict resolution for **`<table>`** elements. In a **`<table>`** with multiple borders, `none` has the lowest priority (meaning in a conflict, the border would show), and `hidden` has the highest priority (meaning in a conflict, the border would not show).

第10.3节：多重边框

使用 outline：

```
.div1{
  border: 3像素 实线 黑色;
  outline: 6像素 实线 蓝色;
  width: 100像素;
  height: 100像素;
  margin: 20像素;
}
```

使用 box-shadow：

```
.div2{
  border: 5px 实线 绿色;
  box-shadow: 0px 0px 0px 4px #000;
  width: 100px;
  height: 100像素;
  margin: 20像素;
}
```

使用伪元素：

```
.div3 {
  位置: 相对;
  边框: 5像素 实线 #000;
  宽度: 100像素;
  height: 100像素;
  margin: 20像素;
}
.div3:before {
  内容: " ";
  位置: 绝对;
  边框: 5像素 实线 蓝色;
  z轴索引: -1;
  上: 5像素;
  左: 5像素;
  右: 5像素;
  下: 5像素;
}
```


Section 10.3: Multiple Borders

Using outline:

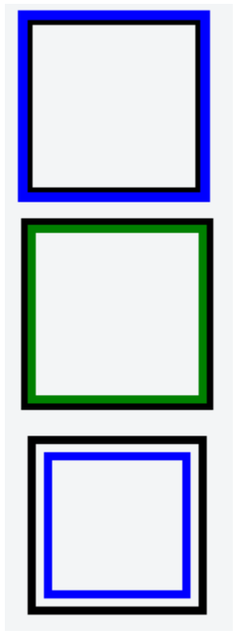
```
.div1{
  border: 3px solid black;
  outline: 6px solid blue;
  width: 100px;
  height: 100px;
  margin: 20px;
}
```

Using box-shadow:

```
.div2{
  border: 5px solid green;
  box-shadow: 0px 0px 0px 4px #000;
  width: 100px;
  height: 100px;
  margin: 20px;
}
```

Using a pseudo element:

```
.div3 {
  position: relative;
  border: 5px solid #000;
  width: 100px;
  height: 100px;
  margin: 20px;
}
.div3:before {
  content: " ";
  position: absolute;
  border: 5px solid blue;
  z-index: -1;
  top: 5px;
  left: 5px;
  right: 5px;
  bottom: 5px;
}
```



http://jsfiddle.net/MadalinaTn/bvqpcohm/2/

第10.4节：边框（简写）

在大多数情况下，你会想为元素的所有边定义多个边框属性（border-width，border-style 和 border-color）。

而不是写成：

```
border-width: 1px;
border-style: solid;
border-color: #000;
```

你可以简单地写成：

```
border: 1px solid #000;
```

这些简写也适用于元素的每一边：`border-top`、`border-left`、`border-right` 和 `border-bottom`。所以你可以这样写：

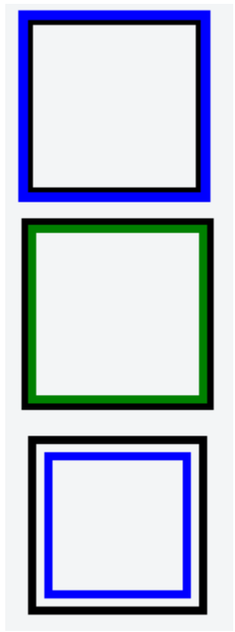
```
border-top: 2px double aaaaaa;
```

第10.5节：border-collapse

`border-collapse` 属性仅适用于 `table`（以及显示为 **`display`**`: table` 或 `inline-table` 的元素），用于设置表格边框是合并为单一边框还是像标准HTML那样分开显示。

```
table {
  border-collapse: separate; /* 默认值 */
  border-spacing: 2px; /* 仅当 border-collapse 为 separate 时生效 */
}
```

另见 表格 - border-collapse 文档条目



http://jsfiddle.net/MadalinaTn/bvqpcohm/2/

Section 10.4: border (shorthands)

In most cases you want to define several border properties (`border-width`, `border-style` and `border-color`) for all sides of an element.

Instead of writing:

```
border-width: 1px;
border-style: solid;
border-color: #000;
```

You can simply write:

```
border: 1px solid #000;
```

These shorthands are also available for every side of an element: `border-top`, `border-left`, `border-right` and `border-bottom`. So you can do:

```
border-top: 2px double #aaaaaa;
```

Section 10.5: border-collapse

The `border-collapse` property applies only to `table`s (and elements displayed as **`display`**`: table` or `inline-table`) and sets whether the table borders are collapsed into a single border or detached as in standard HTML.

```
table {
  border-collapse: separate; /* default */
  border-spacing: 2px; /* Only works if border-collapse is separate */
}
```

Also see Tables - border-collapse documentation entry

第10.6节：边框图像

使用border-image属性，您可以设置一张图片来替代普通的边框样式。

一个border-image本质上由以下部分组成

- border-image-source：要使用的图片路径border-image-
- slice：指定用于将图片分割成九个区域（四个角、四个边和一个中间）的偏移量

- border-image-repeat：指定边框图片的边和中间部分的图片如何缩放

请参考以下示例，其中border.png是一张90x90像素的图片：

```
border-image：url("border.png")30stretch;
```

图片将被分割成九个30x30像素的区域。边缘部分将用作边框的角落，而边则用于中间部分。如果元素的高度/宽度超过30像素，这部分图片将被拉伸。
图片的中间部分默认是透明的。

第10.7节：使用border-

image创建多色边框

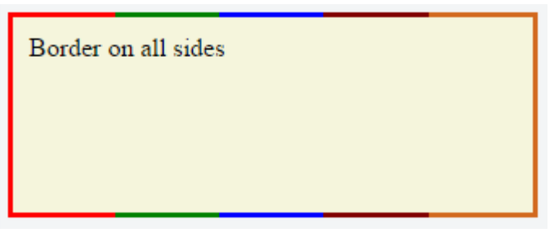
CSS

```
.bordered {
  border-image: 线性渐变(向右, 红色 20%, 绿色 20%, 绿色 40%, 蓝色 40%, 蓝色 60%, 栗色
60%, 栗色 80%, 巧克力色 80%); /* 带有所需颜色的渐变 */
  border-image-slice: 1;
}
```

HTML

```
<div class='bordered'>所有边框</div>
```

上述示例将生成由5种不同颜色组成的边框。颜色是通过
线性渐变（你可以在文档中找到更多关于渐变的信息）。你可以在同一页面的border-image示例中找到更多关于
border-image-slice属性的信息。



（注意：为了展示效果，元素添加了额外的属性。）

你会注意到左边框只有单一颜色（渐变的起始颜色），而右边框
也只有单一颜色（渐变的结束颜色）。这是因为边框图像属性的工作方式。
就好像渐变应用于整个盒子，然后颜色从内边距和
内容区域被遮罩，从而看起来只有边框具有渐变效果。

哪个边框具有单一颜色取决于渐变的定义。如果渐变是向右的渐变，
左边框将是渐变的起始颜色，右边框将是结束颜色。如果是向
下的渐变，则上边框是渐变的起始颜色，下边框是结束颜色。下面是

Section 10.6: border-image

With the `border-image` property you have the possibility to set an image to be used instead of normal border styles.

A `border-image` essentially consist of a

- `border-image-source`: The path to the image to be used
- `border-image-slice`: Specifies the offset that is used to divide the image into **nine regions** (four **corners**, four **edges** and a **middle**)
- `border-image-repeat`: Specifies how the images for the sides and the middle of the border image are scaled

Consider the following example wheras border.png is a image of 90x90 pixels:

```
border-image: url("border.png") 30 stretch;
```

The image will be split into nine regions with 30x30 pixels. The edges will be used as the corners of the border while the side will be used in between. If the element is higher / wider than 30px this part of the image will be **stretched**. The middle part of the image defaults to be transparent.

Section 10.7: Creating a multi-colored border using border-image

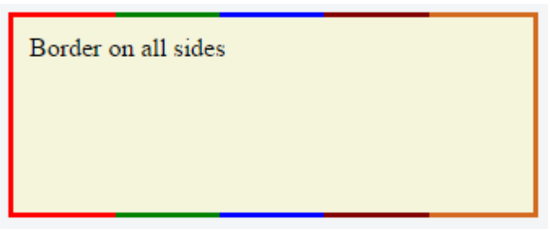
CSS

```
.bordered {
  border-image: linear-gradient(to right, red 20%, green 20%, green 40%, blue 40%, blue 60%, maroon
60%, maroon 80%, chocolate 80%); /* gradient with required colors */
  border-image-slice: 1;
}
```

HTML

```
<div class='bordered'>Border on all sides</div>
```

The above example would produce a border that comprises of 5 different colors. The colors are defined through a `linear-gradient` (you can find more information about gradients in the docs). You can find more information about `border-image-slice` property in the `border-image` example in same page.

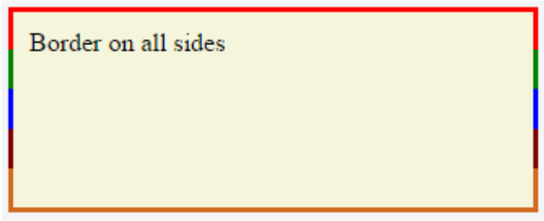


(*Note: Additional properties were added to the element for presentational purpose.*)

You'd have noticed that the left border has only a single color (the start color of the gradient) while the right border also has only a single color (the gradient's end color). This is because of the way that border image property works. It is as though the gradient is applied to the entire box and then the colors are masked from the padding and content areas, thus making it look as though only the border has the gradient.

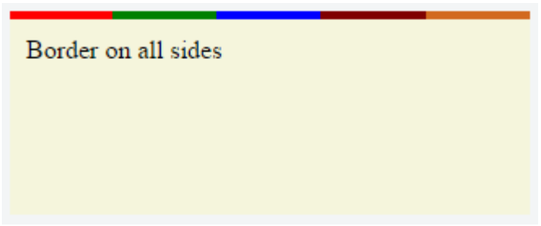
Which border(s) have a single color is dependant on the gradient definition. If the gradient is a `to right` gradient, the left border would be the start color of the gradient and right border would be the end color. If it was a `to bottom` gradient the top border would be the gradient's start color and bottom border would be end color. Below is

一个向下的5色渐变的输出效果。



如果只需要在元素的特定边添加边框，则可以像使用其他普通边框一样使用border-width属性。例如，添加以下代码将只在元素的顶部生成边框。

```
border-width: 5px 0px 0px 0px;
```



请注意，任何具有border-image属性的元素都不会遵循border-radius（即边框不会弯曲）。这是基于规范中的以下声明：

> 盒子的背景，但不是其边框图像，会被裁剪到适当的曲线（由'background-clip'决定）。

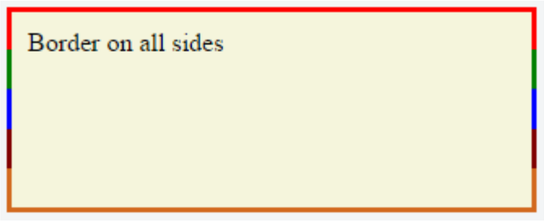
第10.8节：border-[left|right|top|bottom]

border-[left|right|top|bottom]属性用于为元素的特定边添加边框。

例如，如果你想为元素的左侧添加边框，可以这样做：

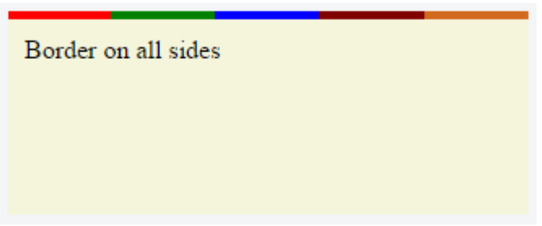
```
#element {
    border-left: 1px solid black;
}
```


the output of a `to bottom` 5 colored gradient.



If the border is required only on specific sides of the element then the `border-width` property can be used just like with any other normal border. For example, adding the below code would produce a border only on the top of the element.

```
border-width: 5px 0px 0px 0px;
```



Note that, any element that has `border-image` property **won't respect the** `border-radius` (that is the border won't curve). This is based on the below statement in the spec:

> A box's backgrounds, but not its border-image, are clipped to the appropriate curve (as determined by 'background-clip').

Section 10.8: border-[left|right|top|bottom]

The `border-[left|right|top|bottom]` property is used to add a border to a specific side of an element.

For example if you wanted to add a border to the left side of an element, you could do:

```
#element {
    border-left: 1px solid black;
}
```

第11章：轮廓

参数	详情
点状	点状轮廓
虚线	虚线轮廓
实线	实线轮廓
双线	双线轮廓
凹槽	3D凹槽轮廓，取决于轮廓颜色值
脊	3D 凸棱轮廓，取决于轮廓颜色值
内嵌	3D 内嵌轮廓，取决于轮廓颜色值
外凸	3D 外凸轮廓，取决于轮廓颜色值
无	无轮廓
隐藏	隐藏轮廓

第11.1节：概述

轮廓是围绕元素、位于边框外侧的一条线。与border不同，轮廓在线框模型中不占用任何空间。因此，给元素添加轮廓不会影响该元素或其他元素的位置。

此外，在某些浏览器中，轮廓（outline）可能不是矩形的。如果在包含不同字体大小（font-size）文本的span元素上应用outline，就可能出现这种情况。与边框不同，轮廓不能有圆角。

轮廓（outline）的基本部分是outline-color、outline-style和outline-width。

轮廓的定义等同于边框的定义：

> 轮廓是元素周围的一条线。它显示在元素的外边距（margin）周围，但与边框属性不同。

```
outline: 1px solid black;
```

第11.2节：outline-style

outline-style属性用于设置元素轮廓的样式。

```
p {
  border: 1px solid black;
  outline-color:blue;
  line-height:30px;
}
.p1{
  outline-style: 点状;
}
.p2{
  outline-style: 虚线;
}
.p3{
  outline-style: 实线;
}
```


Chapter 11: Outlines

Parameter	Details
dotted	dotted outline
dashed	dashed outline
solid	solid outline
double	double outline
groove	3D grooved outline, depends on the outline-color value
ridge	3D ridged outline, depends on the outline-color value
inset	3D inset outline, depends on the outline-color value
outset	3D outset outline, depends on the outline-color value
none	no outline
hidden	hidden outline

Section 11.1: Overview

Outline is a line that goes around the element, outside of the border. In contrast to `border`, outlines do not take any space in the box model. So adding an outline to an element does not affect the position of the element or other elements.

In addition, outlines can be non-rectangular in some browsers. This can happen if `outline` is applied on a `span` element that has text with different `font-size` properties inside it. Unlike borders, outlines *cannot* have rounded corners.

The essential parts of `outline` are `outline-color`, `outline-style` and `outline-width`.

The definition of an outline is equivalent to the definition of a border:

> An outline is a line around an element. It is displayed around the margin of the element. However, it is different from the border property.

```
outline: 1px solid black;
```

Section 11.2: outline-style

The `outline-style` property is used to set the style of the outline of an element.

```
p {
  border: 1px solid black;
  outline-color:blue;
  line-height:30px;
}
.p1{
  outline-style: dotted;
}
.p2{
  outline-style: dashed;
}
.p3{
  outline-style: solid;
}
```

```
.p4{
   outline-style: 双线;
}
.p5{
   outline-style: 凹槽;
}
.p6{
   outline-style: ridge;
}
.p7{
   outline-style: inset;
}
.p8{
   outline-style: outset;
}
```

HTML

```
<p class="p1">点状轮廓</p>
<p class="p2">虚线轮廓</p>
<p class="p3">实线轮廓</p>
<p class="p4">双线轮廓</p>
<p class="p5">凹槽轮廓</p>
<p class="p6">脊状轮廓</p>
<p class="p7">内嵌轮廓</p>
<p class="p8">外凸轮廓</p>
```

A dotted outline

A dashed outline

A solid outline

A double outline

A groove outline

A ridge outline

An inset outline

An outset outline


```
.p4{
  outline-style: double;
}
.p5{
  outline-style: groove;
}
.p6{
  outline-style: ridge;
}
.p7{
  outline-style: inset;
}
.p8{
  outline-style: outset;
}
```

HTML

```
<p class="p1">A dotted outline</p>
<p class="p2">A dashed outline</p>
<p class="p3">A solid outline</p>
<p class="p4">A double outline</p>
<p class="p5">A groove outline</p>
<p class="p6">A ridge outline</p>
<p class="p7">An inset outline</p>
<p class="p8">An outset outline</p>
```

A dotted outline

A dashed outline

A solid outline

A double outline

A groove outline

A ridge outline

An inset outline

An outset outline

第12章：溢出

溢出 值	详情
visible	显示元素外所有溢出内容
滚动	隐藏溢出内容并添加滚动条
隐藏	隐藏溢出内容，两个滚动条都消失，页面变为固定
自动	与scroll相同，如果内容溢出则添加滚动条，但如果内容适合则不添加滚动条
继承	继承父元素该属性的值

第12.1节：overflow-wrap

`overflow-wrap` 告诉浏览器它可以在目标元素内的文本行中，在一个原本不可断开的地方断行。对于防止长字符串文本因溢出其容器而导致布局问题非常有用。

CSS

```
div {
    width:100px;
    outline: 1px dashed #bbb;
}

#div1 {
    overflow-wrap:normal;
}

#div2 {
    overflow-wrap:break-word;
}
```

HTML

```
<div id="div1">
      <strong>#div1</strong>：小词正常显示，但像<spanstyle="red;">supercalifragilisticexpialidocious</span>这样很长的单词太长，会溢出换行符的边缘

</div>

<div id="div2">
      <strong>#div2</strong>：小词正常显示，但像<spanstyle="red;">supercalifragilisticexpialidocious</span>这样很长的单词会在换行处断开，并继续显示在下一行。

</div>
```


Chapter 12: Overflow

`Overflow` Value	Details
visible	Shows all overflowing content outside the element
scroll	Hides the overflowing content and adds a scroll bar
hidden	Hides the overflowing content, both scroll bars disappear and the page becomes fixed
auto	Same as `scroll` if content overflows, but doesn't add scroll bar if content fits
inherit	Inherit's the parent element's value for this property

Section 12.1: overflow-wrap

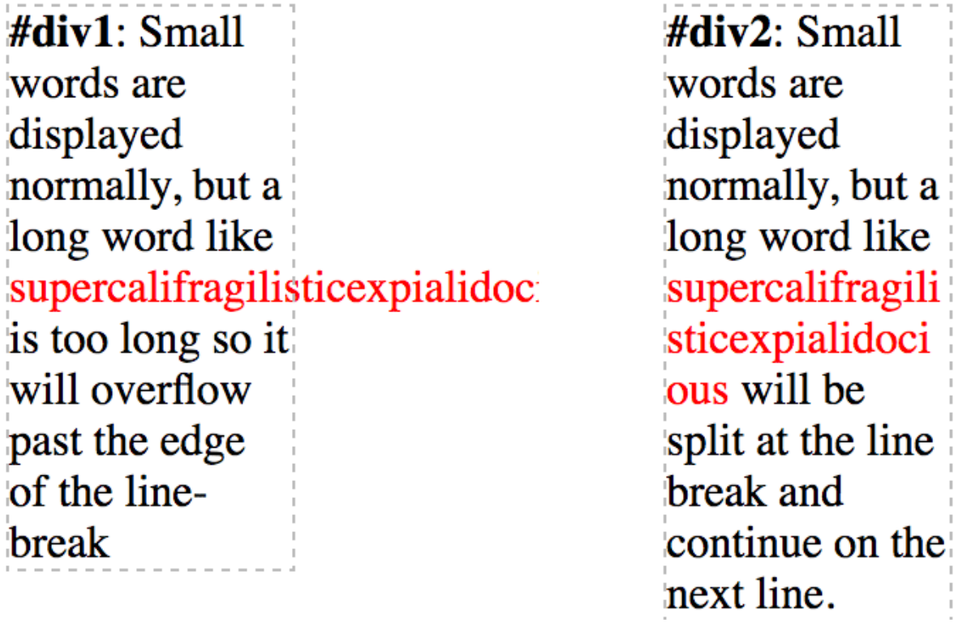
`overflow-wrap` tells a browser that it can break a line of text inside a targeted element onto multiple lines in an otherwise unbreakable place. Helpful in preventing an long string of text causing layout problems due to overflowing it's container.

CSS

```
div {
    width:100px;
    outline: 1px dashed #bbb;
}

#div1 {
    overflow-wrap:normal;
}

#div2 {
    overflow-wrap:break-word;
}
```

HTML

```
<div id="div1">
      <strong>#div1</strong>: Small words are displayed normally, but a long word like <span style="red;">supercalifragilisticexpialidocious</span> is too long so it will overflow past the edge of the line-break
</div>

<div id="div2">
      <strong>#div2</strong>: Small words are displayed normally, but a long word like <span style="red;">supercalifragilisticexpialidocious</span> will be split at the line break and continue on the next line.
</div>
```



overflow-wrap – 值	详情
`normal`	如果单词比行长还长，则允许单词溢出
`break-word`	如果有必要，将单词拆分成多行
`继承`	继承父元素该属性的值

第12.2节：overflow-x 和 overflow-y

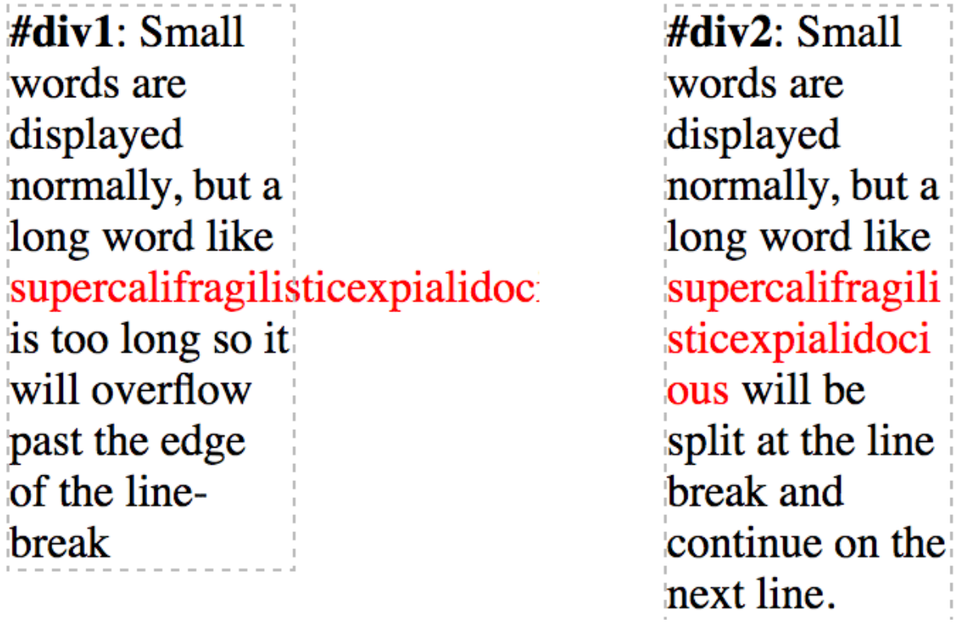
这两个属性的工作方式类似于overflow属性，并接受相同的值。overflow-x参数仅作用于x轴或从左到右的轴。overflow-y作用于y轴或从上到下的轴。

HTML

```
<div id="div-x">
如果此div太小而无法显示其内容，
    左右的内容将被裁剪。
</div>

<div id="div-y">
如果此div太小而无法显示其内容，
    上下的内容将被裁剪。
</div>
```

CSS

```
div {
    width: 200px;
    height: 200px;
}

#div-x {
    overflow-x: hidden;
}
```

overflow-wrap – Value	Details
`normal`	Lets a word overflow if it is longer than the line
`break-word`	Will split a word into multiple lines, if necessary
`inherit`	Inherits the parent element's value for this property

Section 12.2: overflow-x and overflow-y

These two properties work in a similar fashion as the `overflow` property and accept the same values. The `overflow-x` parameter works only on the x or left-to-right axis. The `overflow-y` works on the y or top-to-bottom axis.

HTML

```
<div id="div-x">
    If this div is too small to display its contents,
    the content to the left and right will be clipped.
</div>

<div id="div-y">
    If this div is too small to display its contents,
    the content to the top and bottom will be clipped.
</div>
```

CSS

```
div {
    width: 200px;
    height: 200px;
}

#div-x {
    overflow-x: hidden;
}
```

```
#div-y {
    overflow-y: hidden;
}
```

第12.3节：overflow: scroll

HTML

```
<div>
                                这个div太小，无法显示其内容，以展示overflow属性的效果。
</div>
```

CSS

```
div {
    width:100px;
    height:100px;
    overflow:scroll;
}
```

结果

上述内容被裁剪在一个100px乘100px的盒子内，可以通过滚动查看溢出内容。

大多数桌面浏览器会显示水平和垂直滚动条，无论内容是否被裁剪。
这可以避免在动态环境中滚动条出现和消失的问题。打印机可能会打印
溢出的内容。

第12.4节：overflow: visible

HTML

```
<div>
即使该 div 太小而无法显示其内容，内容也不会被裁剪。
</div>
```

CSS

```
div {
     width:50px;
     height:50px;
     overflow:visible;
}
```

结果


```
#div-y {
    overflow-y: hidden;
}
```

Section 12.3: overflow: scroll

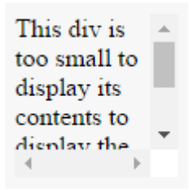
HTML

```
<div>
    This div is too small to display its contents to display the effects of the overflow property.
</div>
```

CSS

```
div {
    width:100px;
    height:100px;
    overflow:scroll;
}
```

Result



The content above is clipped in a 100px by 100px box, with scrolling available to view overflowing content.

Most desktop browsers will display both horizontal and vertical scrollbars, whether or not any content is clipped. This can avoid problems with scrollbars appearing and disappearing in a dynamic environment. Printers may print overflowing content.

Section 12.4: overflow: visible

HTML

```
<div>
     Even if this div is too small to display its contents, the content is not clipped.
</div>
```

CSS

```
div {
     width:50px;
     height:50px;
     overflow:visible;
}
```

Result

内容不会被裁剪，如果超过其容器大小，将在内容框外渲染。

第12.5节：使用overflow创建的块格式化上下文

使用overflow属性且值不为visible时，将创建一个新的块格式化上下文。这对于将块元素与浮动元素并排对齐非常有用。

CSS

```
img {
      float:left;
      margin-right:  10px;
}
div {
       overflow:hidden; /* 创建块格式化上下文 */
}
```

HTML

```
<img src="http://placehold.it/100x100">
<div>
  <p>罗睿姆·伊普苏姆·多洛尔·西特·阿梅特，库姆·诺·保罗·莫利斯·佩尔蒂纳西亚。</p>
  <p>阿德·凯斯·奥姆尼斯·南，穆塔特·德塞鲁伊斯·佩尔塞克里斯·埃奥斯·阿德，因·托利特·德比蒂斯·西娅。</p>
</div>
```

结果

Content is not clipped and will be rendered outside the content box if it exceeds its container size.

Section 12.5: Block Formatting Context Created with Overflow

Using the `overflow` property with a value different to `visible` will create a new **block formatting context**. This is useful for aligning a block element next to a floated element.

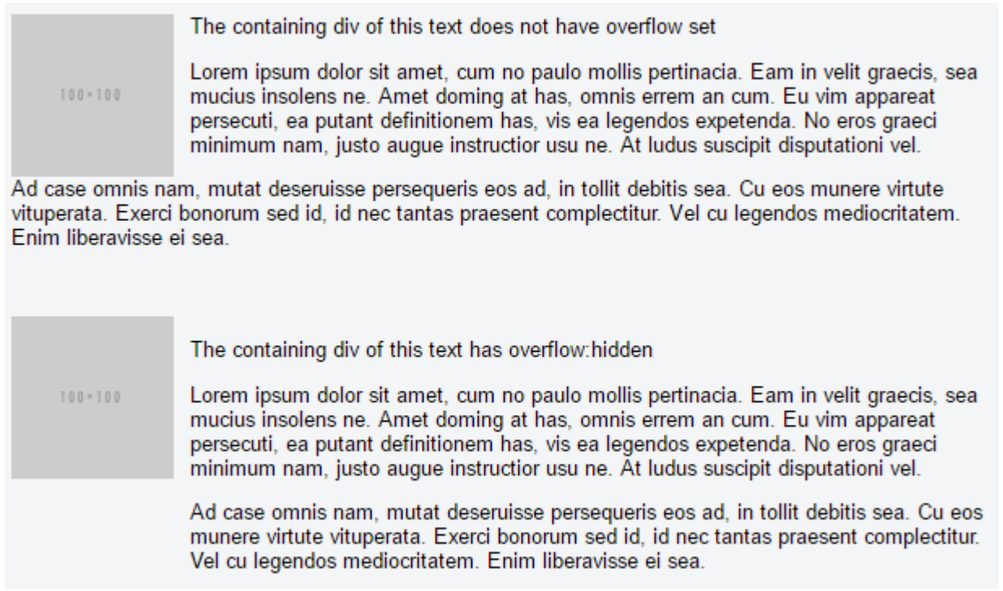
CSS

```
img {
    float:left;
    margin-right: 10px;
}
 div {
     overflow:hidden; /* creates block formatting context */
}
```

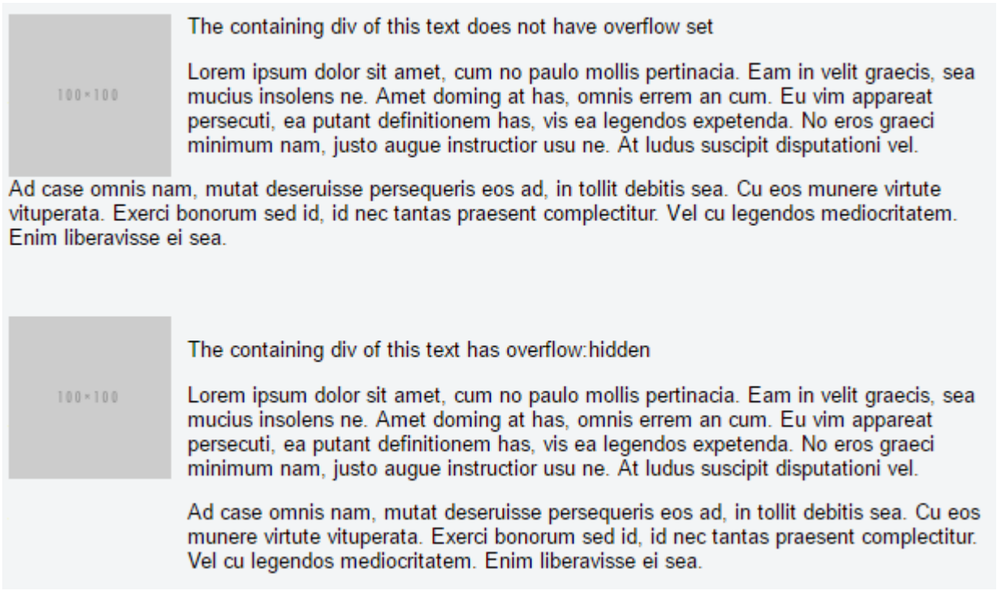
HTML

```
<img src="http://placehold.it/100x100">
<div>
  <p>Lorem ipsum dolor sit amet, cum no paulo mollis pertinacia.</p>
  <p>Ad case omnis nam, mutat deseruisse persequeris eos ad, in tollit debitis sea.</p>
</div>
```

Result



此示例展示了设置了overflow属性的div内的段落如何与浮动图像交互。



This example shows how paragraphs within a div with the `overflow` property set will interact with a floated image.

第13章：媒体查询

参数	详情
媒体类型	（可选）这是媒体的类型。可以是从all到screen范围内的任何类型。
`not`	（可选）不应用于此特定媒体类型的CSS，而应用于其他所有情况。
媒体特性	用于识别CSS使用场景的逻辑。选项如下所示。

媒体特性	详情
宽高比	描述输出设备目标显示区域的宽高比。
颜色	表示输出设备每个颜色分量的位数。如果设备不是彩色设备，则该值为零。
颜色索引	表示输出设备的颜色查找表中的条目数。
网格	确定输出设备是网格设备还是位图设备。
高度	高度媒体特征描述输出设备渲染表面的高度。
最大宽度	CSS 不会应用于宽度超过指定值的屏幕。
最小宽度	CSS 不会应用于宽度小于指定值的屏幕。
最大高度	CSS 不会应用于高度超过指定值的屏幕。
最小高度	CSS 不会应用于高度小于指定值的屏幕。
单色	表示单色（灰度）设备上每像素的位数。
方向	只有当设备使用指定的方向时，CSS才会显示。更多详情请参见备注。
分辨率	表示输出设备的分辨率（像素密度）。
扫描	描述电视输出设备的扫描过程。
宽度	宽度媒体特性描述输出设备渲染表面的宽度 （例如文档窗口的宽度，或打印机页面框的宽度）。

已废弃功能详情

`device-aspect-ratio`	已废弃 CSS 仅在设备的高宽比与指定比例匹配时显示比例。这是一个已废弃功能，不能保证其有效性。
`max-device-width`	已废弃 与max-width相同，但测量的是物理屏幕宽度，而非浏览器的显示宽度。
`min-device-width`	已废弃 与min-width相同，但测量的是物理屏幕宽度，而非浏览器的显示宽度。
`max-device-height`	已废弃 与max-height相同，但测量的是物理屏幕宽度，而非浏览器的显示宽度。
`min-device-height`	已废弃 与min-height相同，但测量的是物理屏幕宽度，而非浏览器的显示宽度。

第13.1节：术语和结构

媒体查询允许根据设备/媒体类型（例如屏幕、打印或手持设备）应用CSS规则，称为媒体类型，设备的其他方面通过媒体特性描述，如颜色可用性或视口尺寸。

媒体查询的一般结构

```
@media [...] {
    /* 当查询条件满足时应用的一个或多个CSS规则 */
}
```

包含媒体类型的媒体查询

```
@media print {
    /* 当查询条件满足时应用的一个或多个CSS规则 */
```


Chapter 13: Media Queries

Parameter	Details
`mediatype`	(Optional) This is the type of media. Could be anything in the range of `all` to `screen`.
`not`	(Optional) Doesn't apply the CSS for this particular media type and applies for everything else.
`media feature`	Logic to identify use case for CSS. Options outlined below.

Media Feature	Details
`aspect-ratio`	Describes the aspect ratio of the targeted display area of the output device.
`color`	Indicates the number of bits per color component of the output device. If the device is not a color device, this value is zero.
`color-index`	Indicates the number of entries in the color look-up table for the output device.
`grid`	Determines whether the output device is a grid device or a bitmap device.
`height`	The height media feature describes the height of the output device's rendering surface.
`max-width`	CSS will not apply on a screen width wider than specified.
`min-width`	CSS will not apply on a screen width narrower than specified.
`max-height`	CSS will not apply on a screen height taller than specified.
`min-height`	CSS will not apply on a screen height shorter than specified.
`monochrome`	Indicates the number of bits per pixel on a monochrome (greyscale) device.
`orientation`	CSS will only display if device is using specified orientation. See remarks for more details.
`resolution`	Indicates the resolution (pixel density) of the output device.
`scan`	Describes the scanning process of television output devices.
`width`	The width media feature describes the width of the rendering surface of the output device (such as the width of the document window, or the width of the page box on a printer).

Deprecated Features	Details
`device-aspect-ratio`	**Deprecated** CSS will only display on devices whose height/width ratio matches the specified ratio. This is a`deprecated`feature and is not guaranteed to work.
`max-device-width`	**Deprecated** Same as `max-width` but measures the physical screen width, rather than the display width of the browser.
`min-device-width`	**Deprecated** Same as `min-width` but measures the physical screen width, rather than the display width of the browser.
`max-device-height`	**Deprecated** Same as `max-height` but measures the physical screen width, rather than the display width of the browser.
`min-device-height`	**Deprecated** Same as `min-height` but measures the physical screen width, rather than the display width of the browser.

Section 13.1: Terminology and Structure

Media queries allow one to apply CSS rules based on the type of device / media (e.g. screen, print or handheld) called **media type**, additional aspects of the device are described with **media features** such as the availability of color or viewport dimensions.

General Structure of a Media Query

```
@media [...] {
    /* One or more CSS rules to apply when the query is satisfied */
}
```

A Media Query containing a Media Type

```
@media print {
    /* One or more CSS rules to apply when the query is satisfied */
```

```
}
```

包含媒体类型和媒体特征的媒体查询

```
@media screen and (max-width: 600px) {
    /* 当查询条件满足时应用的一个或多个CSS规则 */
}
```

包含媒体特征（隐含媒体类型为“all”）的媒体查询

```
@media (orientation: portrait) {
    /* 当查询条件满足时应用的一个或多个CSS规则 */
}
```

第13.2节：基本示例

```
@media screen and (min-width: 720px) {
    body {
        background-color: 天蓝色;
    }
}
```

上述媒体查询指定了两个条件：

1. 页面必须在普通屏幕上查看（非打印页面、投影仪等）。
2. 用户视口的宽度必须至少为720像素。

如果满足这些条件，媒体查询内的样式将生效，页面的背景颜色将是天蓝色。

媒体查询是动态应用的。如果页面加载时满足媒体查询中指定的条件，CSS将被应用，但一旦条件不再满足，CSS将立即被禁用。反之，
如果初始条件不满足，CSS将不会被应用，直到满足指定条件为止。

在我们的示例中，如果用户的视口宽度最初大于720像素，但用户缩小浏览器宽度，当视口宽度小于720像素时，背景颜色将不再是天蓝色。

第13.3节：媒体类型（mediatype）

媒体查询有一个可选的mediatype参数。该参数直接放在@media声明之后（@media mediatype），例如：

```
@media print {
html {
        background-color: white;
    }
}
```

上述 CSS 代码将在打印时为 DOM HTML 元素设置白色背景色。

mediatype 参数有一个可选的 not 或 only 前缀，分别将样式应用于除指定媒体类型之外的所有媒体类型，或仅应用于指定的媒体类型。例如，以下代码示例将样式应用于除 print 之外的所有媒体类型。

```
@media not print {
    html {
        background-color: green;
    }
```



```
}
```

A Media Query containing a Media Type and a Media Feature

```
@media screen and (max-width: 600px) {
    /* One or more CSS rules to apply when the query is satisfied */
}
```

A Media Query containing a Media Feature (and an implicit Media Type of "all")

```
@media (orientation: portrait) {
    /* One or more CSS rules to apply when the query is satisfied */
}
```

Section 13.2: Basic Example

```
@media screen and (min-width: 720px) {
    body {
        background-color: skyblue;
    }
}
```

The above media query specifies two conditions:

1. The page must be viewed on a normal screen (not a printed page, projector, etc).
2. The width of the user's view port must be at least 720 pixels.

If these conditions are met, the styles inside the media query will be active, and the background color of the page will be sky blue.

Media queries are applied dynamically. If on page load the conditions specified in the media query are met, the CSS will be applied, but will be immediately disabled should the conditions cease to be met. Conversely, if the conditions are initially not met, the CSS will not be applied until the specified conditions are met.

In our example, if the user's view port width is initially greater than 720 pixels, but the user shrinks the browser's width, the background color will cease to be sky blue as soon as the user has resized the view port to less than 720 pixels in width.

Section 13.3: mediatype

Media queries have an optional `mediatype` parameter. This parameter is placed directly after the `@media` declaration (`@media mediatype`), for example:

```
@media print {
    html {
        background-color: white;
    }
}
```

The above CSS code will give the DOM `HTML` element a white background color when being printed.

The `mediatype` parameter has an optional `not` or `only` prefix that will apply the styles to everything except the specified mediatype *or* only the specified media type, respectively. For example, the following code example will apply the style to every media type except `print`.

```
@media not print {
    html {
        background-color: green;
    }
```

```
}
```

同样地，如果只想在屏幕上显示，可以使用以下代码：

```
@media only screen {
    .fadeInEffects {
        display: block;
    }
}
```

可以通过下表更好地理解 mediatype 列表：

媒体类型	描述
所有	应用于所有设备
屏幕	默认计算机
打印	一般打印机。用于为网站的打印版本设置样式
手持设备	PDA、手机和带小屏幕的手持设备
投影	用于投影演示，例如投影仪
aural	语音系统
盲文	盲文触觉设备
压花的	分页盲文打印机
电视	电视类型设备
文字电话 (TTY)	具有固定间距字符网格的设备。终端，便携式设备。

第13.4节：视网膜屏幕和非视网膜屏幕的媒体查询

虽然这只适用于基于WebKit的浏览器，但这很有帮助：

```
/* ---------- 非视网膜屏幕 ---------- */
@media screen
and (min-width: 1200px)
  and (max-width: 1600px)
  and (-webkit-min-device-pixel-ratio: 1) {
}

/* ---------- Retina 屏幕 ---------- */
@media screen
and (min-width: 1200px)
  and (max-width: 1600px)
  and (-webkit-min-device-pixel-ratio: 2)
  and (min-resolution: 192dpi) {
}
```

背景信息

显示器中有两种像素。一种是逻辑像素，另一种是物理像素。大多数情况下，物理像素始终保持不变，因为所有显示设备的物理像素都是相同的。逻辑像素会根据设备的分辨率变化，以显示更高质量的像素。设备像素比是物理像素与逻辑像素之间的比率。例如，MacBook Pro Retina、iPhone 4及以上设备报告的设备像素比为2，因为物理线性分辨率是逻辑分辨率的两倍。

之所以这只适用于基于 WebKit 的浏览器，是因为：


```
}
```

And the same way, for just showing it only on the screen, this can be used:

```
@media only screen {
    .fadeInEffects {
        display: block;
    }
}
```

The list of `mediatype` can be understood better with the following table:

Media Type	Description
`all`	Apply to all devices
`screen`	Default computers
`print`	Printers in general. Used to style print-versions of websites
`handheld`	PDA's, cellphones and hand-held devices with a small screen
`projection`	For projected presentation, for example projectors
`aural`	Speech Systems
`braille`	Braille tactile devices
`embossed`	Paged braille printers
`tv`	Television-type devices
`tty`	Devices with a fixed-pitch character grid. Terminals, portables.

Section 13.4: Media Queries for Retina and Non Retina Screens

Although this works only for WebKit based browsers, this is helpful:

```
/* ---------- Non-Retina Screens ---------- */
@media screen
  and (min-width: 1200px)
  and (max-width: 1600px)
  and (-webkit-min-device-pixel-ratio: 1) {
}

/* ---------- Retina Screens ---------- */
@media screen
  and (min-width: 1200px)
  and (max-width: 1600px)
  and (-webkit-min-device-pixel-ratio: 2)
  and (min-resolution: 192dpi) {
}
```

Background Information

There are two types of pixels in the display. One is the logical pixels and the other is the physical pixels. Mostly, the physical pixels always stay the same, because it is the same for all the display devices. The logical pixels change based on the resolution of the devices to display higher quality pixels. The device pixel ratio is the ratio between physical pixels and logical pixels. For instance, the MacBook Pro Retina, iPhone 4 and above report a device pixel ratio of 2, because the physical linear resolution is double the logical resolution.

The reason why this works only with WebKit based browsers is because of:

- 规则前的厂商前缀 -webkit-。
- 这尚未在除 WebKit 和 Blink 之外的引擎中实现。

第13.5节：宽度与视口

当我们在媒体查询中使用“width”时，正确设置 meta 标签非常重要。基本的 meta 标签如下，需要放置在 **<head>** 标签内。

```
<meta name="viewport" content="width=device-width,initial-scale=1">
```

为什么这很重要？

根据MDN的定义，“宽度”是

> 宽度媒体特性描述了输出设备渲染表面的宽度（例如文档窗口的宽度，或打印机上页面框的宽度）。

这是什么意思？

视口是设备本身的宽度。如果你的屏幕分辨率是1280 x 720，那么你的视口宽度是“1280px”。

通常许多设备会分配不同数量的像素来显示一个像素。例如，iPhone 6 Plus的分辨率是1242 x 2208。但实际的视口宽度和视口高度是414 x 736。这意味着用3个像素来创建1个像素。

但如果你没有正确设置meta标签，它会尝试以设备的原生分辨率显示你的网页，结果会导致页面缩小显示（文字和图片变小）。

第13.6节：使用媒体查询针对不同屏幕尺寸

响应式网页设计通常涉及媒体查询，媒体查询是只有在满足条件时才执行的CSS代码块。这对于响应式网页设计很有用，因为你可以使用媒体查询为网站的移动版和桌面版指定不同的CSS样式。

```
@media only screen and (min-width: 300px) and (max-width: 767px) {
    .site-title {
        font-size: 80%;
    }

    /* 该块中的样式仅在屏幕宽度至少为300px且不超过767px时应用 */

}

@media only screen and (min-width: 768px) and (max-width: 1023px) {
    .site-title {
        font-size: 90%;
    }

    /* 该块中的样式仅在屏幕宽度至少为768px且不超过1023px时应用 */

}

@media only screen and (min-width: 1024px) {
```


- The vendor prefix `-webkit-` before the rule.
- This hasn't been implemented in engines other than WebKit and Blink.

Section 13.5: Width vs Viewport

When we are using "width" with media queries it is important to set the meta tag correctly. Basic meta tag looks like this and it needs to be put inside the **<head>** tag.

```
<meta name="viewport" content="width=device-width,initial-scale=1">
```

Why this is important?

Based on MDN's definition "width" is

> The width media feature describes the width of the rendering surface of the output device (such as the width of the document window, or the width of the page box on a printer).

What does that mean?

View-port is the width of the device itself. If your screen resolution says the resolution is 1280 x 720, your view-port width is "1280px".

More often many devices allocate different pixel amount to display one pixel. For an example iPhone 6 Plus has 1242 x 2208 resolution. But the actual viewport-width and viewport-height is 414 x 736. That means 3 pixels are used to create 1 pixel.

But if you did not set the `meta` tag correctly it will try to show your webpage with its native resolution which results in a zoomed out view (smaller texts and images).

Section 13.6: Using Media Queries to Target Different Screen Sizes

Often times, responsive web design involves media queries, which are CSS blocks that are only executed if a condition is satisfied. This is useful for responsive web design because you can use media queries to specify different CSS styles for the mobile version of your website versus the desktop version.

```
@media only screen and (min-width: 300px) and (max-width: 767px) {
    .site-title {
        font-size: 80%;
    }

    /* Styles in this block are only applied if the screen size is atleast 300px wide, but no more
than 767px */
}

@media only screen and (min-width: 768px) and (max-width: 1023px) {
    .site-title {
        font-size: 90%;
    }

    /* Styles in this block are only applied if the screen size is atleast 768px wide, but no more
than 1023px */
}

@media only screen and (min-width: 1024px) {
```

```
    .site-title {
        font-size: 120%;
    }

    /* 该块中的样式仅在屏幕宽度超过1024px时应用。*/
}
```

第13.7节：在link标签上的使用

```
<link rel="stylesheet" media="min-width: 600px" href="example.css" />
```

该样式表仍然会被下载，但仅在屏幕宽度大于600像素的设备上应用。

第13.8节：媒体查询与IE8

IE8及以下版本完全不支持媒体查询。

基于Javascript的解决方法

为了支持IE8，你可以使用几种JS解决方案中的一种。例如，**Respond**可以添加以仅为IE8添加媒体查询支持，代码如下：

```
<!--[if lt IE 9]>
<script
    src="respond.min.js">
</script>
<![endif]-->
```

CSS Mediaqueries 是另一个实现相同功能的库。将该库添加到你的HTML中的代码将是相同的：

```
<!--[if lt IE 9]>
<script
    src="css3-mediaqueries.js">
</script>
<![endif]-->
```

另一种选择

如果你不喜欢基于JS的解决方案，也可以考虑添加一个仅针对IE<9的样式表，在其中针对IE<9调整你的样式。为此，你应该在代码中添加以下HTML：

```
<!--[if lt IE 9]>
<link rel="stylesheet" type="text/css" media="all" href="style-ielt9.css"/>
<![endif]-->
```

注意：

从技术上讲，还有另一种选择：使用CSS hack来针对IE<9。它的效果与仅针对IE<9的样式表相同，但你不需要单独的样式表。不过我不推荐这个选项，因为它们会产生无效的CSS代码（这只是现在普遍不推荐使用CSS hack的多个原因之一）。


```
    .site-title {
        font-size: 120%;
    }

    /* Styles in this block are only applied if the screen size is over 1024px wide. */
}
```

Section 13.7: Use on link tag

```
<link rel="stylesheet" media="min-width: 600px" href="example.css" />
```

This stylesheet is still downloaded but is applied only on devices with screen width larger than 600px.

Section 13.8: Media queries and IE8

Media queries are not supported at all in IE8 and below.

A Javascript based workaround

To add support for IE8, you could use one of several JS solutions. For example, **Respond** can be added to add media query support for IE8 only with the following code :

```
<!--[if lt IE 9]>
<script
   src="respond.min.js">
</script>
<![endif]-->
```

CSS Mediaqueries is another library that does the same thing. The code for adding that library to your HTML would be identical :

```
<!--[if lt IE 9]>
<script
   src="css3-mediaqueries.js">
</script>
<![endif]-->
```

The alternative

If you don't like a JS based solution, you should also consider adding an IE<9 only stylesheet where you adjust your styling specific to IE<9. For that, you should add the following HTML to your code:

```
<!--[if lt IE 9]>
<link rel="stylesheet" type="text/css" media="all" href="style-ielt9.css"/>
<![endif]-->
```

Note :

Technically it's one more alternative: using **CSS hacks** to target IE<9. It has the same impact as an IE<9 only stylesheet, but you don't need a separate stylesheet for that. I do not recommend this option, though, as they produce invalid CSS code (which is but one of several reasons why the use of CSS hacks is generally frowned upon today).

第14章：浮动

第14.1节：在文本中浮动图像

浮动最基本的用法是让文本环绕图像。下面的代码将生成两个段落和一张图片，第二个段落会环绕图片。请注意，只有浮动元素之后的内容才会环绕浮动元素。

HTML：

```
<p>  Lorem ipsum dolor sit amet，consectetur adipiscing elit。Integer nec odio。Praesent libero。Sed
cursus ante dapibus diam。Sed nisi。Nulla quis sem at nibh elementum imperdiet。Duis sagittis
ipsum。Praesent mauris。Fusce nec tellus sed augue semper porta。Mauris massa。Vestibulum lacinia
arcu eget nulla。  </p>

<img src="http://lorempixel.com/200/100/" />

<p>适合的伙伴默默地在我们的海岸边相聚，通过婚姻的纽带结合在一起。
Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque
nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin
ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non,
massa. Fusce ac turpis quis ligula lacinia aliquet. </p>
```

CSS：

```
img {
  float:left;
  margin-right:1rem;
}
```

这将是输出结果

Chapter 14: Floats

Section 14.1: Float an Image Within Text

The most basic use of a float is having text wrap around an image. The below code will produce two paragraphs and an image, with the second paragraph flowing around the image. Notice that it is always content *after* the floated element that flows around the floated element.

HTML:

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed
cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis
ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia
arcu eget nulla. </p>

<img src="http://lorempixel.com/200/100/" />

<p>Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.
Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque
nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin
ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non,
massa. Fusce ac turpis quis ligula lacinia aliquet. </p>
```

CSS:

```
img {
  float:left;
  margin-right:1rem;
}
```

This will be the output

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia arcu eget nulla.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non, massa. Fusce ac turpis quis ligula lacinia aliquet.

Codepen 链接

第14.2节：clear属性

clear属性与浮动直接相关。属性值：

- none - 默认。允许两侧都有浮动元素
- left - 不允许左侧有浮动元素
- right - 不允许右侧有浮动元素
- both - 不允许左侧或右侧有浮动元素
- initial - 将此属性设置为默认值。阅读关于initial的内容
- inherit - 从父元素继承此属性。阅读关于inherit的内容

```
<html>
<head>
<style>
img {
    float: left;
```


Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia arcu eget nulla.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non, massa. Fusce ac turpis quis ligula lacinia aliquet.

Codepen Link

Section 14.2: clear property

The clear property is directly related to floats. Property Values:

- none - Default. Allows floating elements on both sides
- left - No floating elements allowed on the left side
- right - No floating elements allowed on the right side
- both - No floating elements allowed on either the left or the right side
- initial - Sets this property to its default value. Read about initial
- inherit - Inherits this property from its parent element. Read about inherit


```
<html>
<head>
<style>
img {
    float: left;
```

```
}

p.clear {
    clear: both;
}
</style>
</head>
<body>

<img src="https://static.pexels.com/photos/69372/pexels-photo-69372-medium.jpeg" width="100">
<p>Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem
ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum </p>
<p class="clear">Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum
Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum </p>

</body>
</html>
```

第14.3节：清除浮动（Clearfix）

> 清除浮动技巧是一种流行的包含浮动元素的方法（N. Gallagher，别名@necolas）

不要将其与clear属性混淆，clearfix是一种概念（也与浮动相关，因此可能会引起混淆）。要包含浮动元素，你需要在容器（父元素）上添加.cf或.clearfix类，并用下面描述的几条规则来设置该类的样式。

有3个版本，效果略有不同（来源：N. Gallagher的A new micro clearfix hack和T. J. Koblentz的clearfix reloaded）：

Clearfix（包含的浮动元素顶部外边距仍然会合并）

```
.cf:after {
    content: "";
    display: table;
}

.cf:after {
    clear: both;
}
```

Clearfix 还防止包含的浮动元素的顶部外边距合并

```
/**
* 适用于现代浏览器
* 1. 空格内容是一种避免 Opera 浏览器错误的方法，当
 *    文档中其他地方包含 contenteditable 属性时。
*    否则会导致被 clearfix 处理的元素顶部和底部出现空白。

* 2. 使用 `table` 而非 `block` 仅在使用
 *    `:before` 来包含子元素的顶部外边距时才有必要。
 */
.cf:之前,
.cf:之后 {
    content: " "; /* 1 */
    display: table; /* 2 */
}

.cf:after {
    clear: both;
```



```
}

p.clear {
    clear: both;
}
</style>
</head>
<body>

<img src="https://static.pexels.com/photos/69372/pexels-photo-69372-medium.jpeg" width="100">
<p>Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem
ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum </p>
<p class="clear">Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum
Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum </p>

</body>
</html>
```

Section 14.3: Clearfix

> The clearfix hack is a popular way to contain floats (N. Gallagher aka @necolas)

Not to be confused with the `clear` property, clearfix is a *concept* (that is also related to floats, thus the possible confusion). To *contain floats*, you've to add `.cf` or `.clearfix` class on the container (**the parent**) and style this class with a few rules described below.

3 versions with slightly different effects (sources :A new micro clearfix hack by N. Gallagher and clearfix reloaded by T. J. Koblentz):

Clearfix (with top margin collapsing of contained floats still occurring)

```
.cf:after {
    content: "";
    display: table;
}

.cf:after {
    clear: both;
}
```

Clearfix also preventing top margin collapsing of contained floats

```
/**
 * For modern browsers
 * 1. The space content is one way to avoid an Opera bug when the
 *    contenteditable attribute is included anywhere else in the document.
 *    Otherwise it causes space to appear at the top and bottom of elements
 *    that are clearfixed.
 * 2. The use of `table` rather than `block` is only necessary if using
 *    `:before` to contain the top-margins of child elements.
 */
.cf:before,
.cf:after {
    content: " "; /* 1 */
    display: table; /* 2 */
}

.cf:after {
    clear: both;
```

```
}
```

支持过时浏览器 IE6 和 IE7 的清除浮动（clearfix）

```
.cf:之前,
.cf:之后 {
    content: " ";
    display: table;
}

.cf:after {
    clear: both;
}

/**
* 仅适用于 IE 6/7
* 包含此规则以触发 hasLayout 并包含浮动。
 */
.cf {
    *zoom: 1;
}
```

展示 clearfix 效果的 Codepen

其他资源：你所知道的关于 clearfix 的一切都是错误的（clearfix 和 BFC - 块格式化上下文，而 hasLayout 关联于过时的浏览器 IE6 可能还有 IE7）

第14.4节：使用浮动的内联DIV

div 是一个块级元素，即它占据整个页面宽度，兄弟元素无论宽度如何，都会一个接一个地垂直排列。

```
<div>
    <p>这是 DIV 1</p>
</div>
<div>
    <p>这是 DIV 2</p>
</div>
```

以下代码的输出将是


```
}
```

Clearfix with support of outdated browsers IE6 and IE7

```
.cf:before,
.cf:after {
    content: " ";
    display: table;
}

.cf:after {
    clear: both;
}

/**
 * For IE 6/7 only
 * Include this rule to trigger hasLayout and contain floats.
 */
.cf {
    *zoom: 1;
}
```

Codepen showing clearfix effect

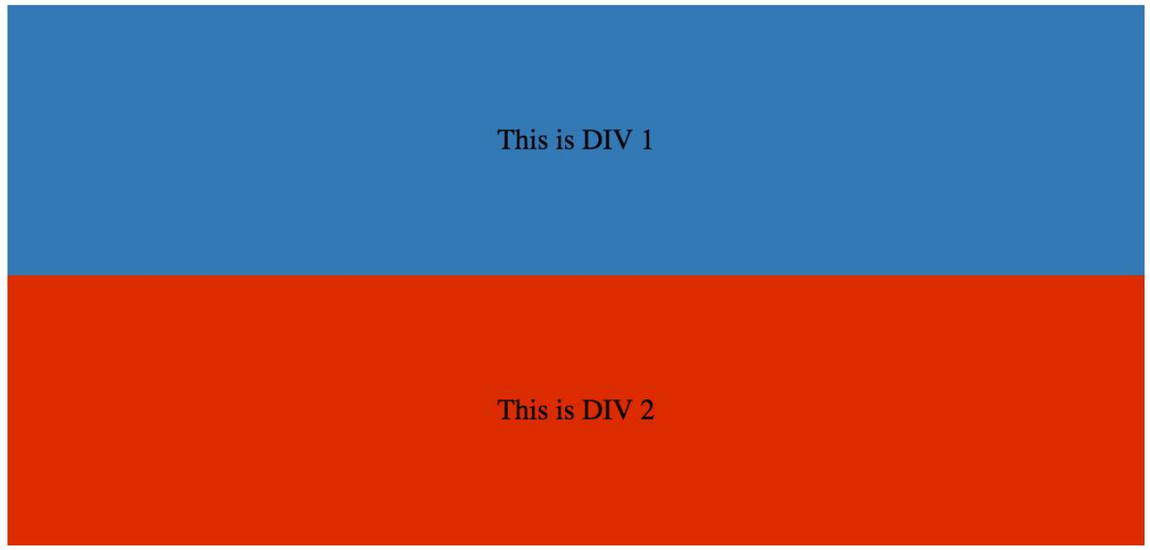
Other resource: Everything you know about clearfix is wrong (clearfix and BFC - Block Formatting Context while hasLayout relates to outdated browsers IE6 maybe 7)

Section 14.4: In-line DIV using float

The `div` is a block-level element, i.e it occupies the whole of the page width and the siblings are place one below the other irrespective of their width.

```
<div>
    <p>This is DIV 1</p>
</div>
<div>
    <p>This is DIV 2</p>
</div>
```

The output of the following code will be



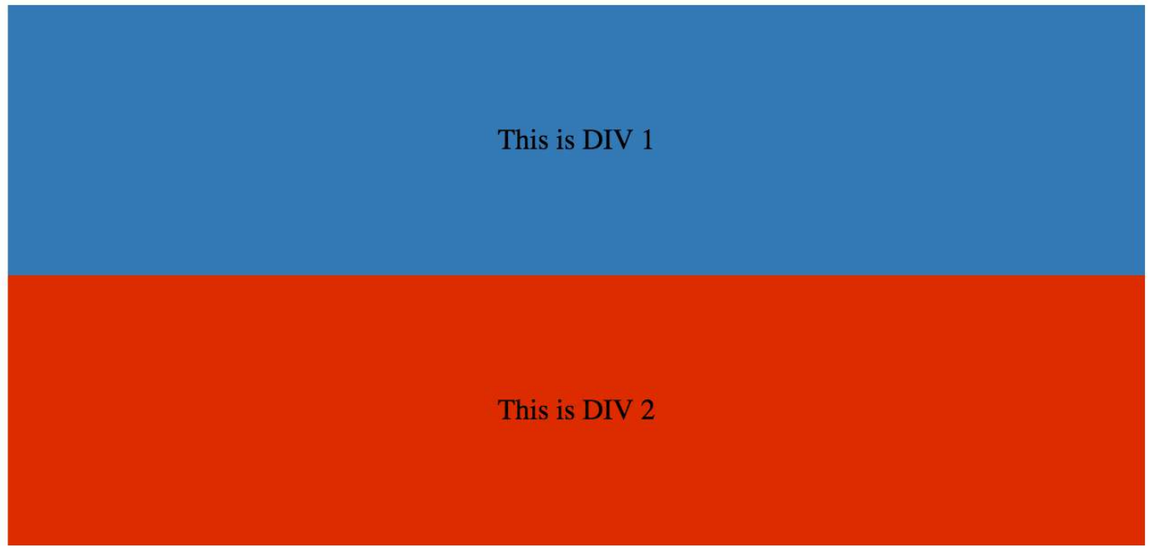
我们可以通过给 div 添加 float CSS 属性使它们变为行内元素。

HTML：

```
<div class="outer-div">
    <div class="inner-div1">
        <p>这是 DIV 1</p>
    </div>
    <div class="inner-div2">
        <p>这是 DIV 2</p>
    </div>
</div>
```

CSS

```
.inner-div1 {
    width: 50%;
    margin-right: 0px;
    float: left;
    背景 : #337ab7;
    内边距:50px 0px;
}

.inner-div2 {
    宽度: 50%;
    右外边距:0px;
    浮动:左;
    背景 : #dd2c00;
    内边距:50px 0px;
}

p {
    文本对齐:居中;
}
```

We can make them in-line by adding a `float` css property to the `div`.

HTML:

```
<div class="outer-div">
    <div class="inner-div1">
        <p>This is DIV 1</p>
    </div>
    <div class="inner-div2">
        <p>This is DIV 2</p>
    </div>
</div>
```

CSS

```
.inner-div1 {
    width: 50%;
    margin-right:0px;
    float:left;
    background : #337ab7;
    padding:50px 0px;
}

.inner-div2 {
    width: 50%;
    margin-right:0px;
    float:left;
    background : #dd2c00;
    padding:50px 0px;
}

p {
    text-align:center;
}
```

Codepen 链接

第14.5节：使用overflow属性清除浮动

将元素的 overflow 值设置为 hidden、auto 或 scroll，将清除该元素内的所有浮动。

注意： 使用 overflow:scroll 时，滚动框将始终显示

第14.6节：简单的两个固定宽度列布局

一个简单的两栏布局由两个固定宽度的浮动元素组成。请注意，在此示例中，侧边栏和内容区域
高度不相同。这是使用浮动实现多栏布局的一个难点，
需要通过一些变通方法使多栏看起来高度一致。

HTML：

```
  <div class="wrapper">

   <div class="sidebar">
     <h2>侧边栏</h2>

   <p>罗勒姆·伊普苏姆·多洛尔·西特·阿梅特，康塞克图尔·阿迪皮斯辛·埃利特。英特格尔·内克·奥迪奥。</p>
</div>

<div class="content">
   <h1>内容</h1>

                              <p>适合的伙伴默默地在我们的海岸边相聚，通过婚姻的纽带结合在一起。
Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque
nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin
ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non,
massa. Fusce ac turpis quis ligula lacinia aliquet. </p>
</div>

</div>
```

CSS：

```
.wrapper {
   宽度:600像素;
   内边距:20像素;
   背景颜色:粉色;

   /* 浮动元素不占用任何高度。添加“overflow:hidden;”会强制
      父元素扩展以包含其浮动的子元素。 */
   溢出:隐藏;
}

.侧边栏 {
   宽度:150像素;
   浮动:左;
   背景颜色:蓝色;
```


Codepen Link

Section 14.5: Use of overflow property to clear floats

Setting `overflow` value to `hidden`,`auto` or `scroll` to an element, will clear all the floats within that element.

Note: using `overflow:scroll` will always show the scrollbox

Section 14.6: Simple Two Fixed-Width Column Layout

A simple two-column layout consists of two fixed-width, floated elements. Note that the sidebar and content area are not the same height in this example. This is one of the tricky parts with multi-column layouts using floats, and requires workarounds to make multiple columns appear to be the same height.

HTML:

```
<div class="wrapper">

<div class="sidebar">
  <h2>Sidebar</h2>

  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio.</p>
</div>

<div class="content">
  <h1>Content</h1>

  <p>Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.
Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque
nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin
ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non,
massa. Fusce ac turpis quis ligula lacinia aliquet. </p>
</div>

</div>
```

CSS:

```
.wrapper {
  width:600px;
  padding:20px;
  background-color:pink;

  /* Floated elements don't use any height. Adding "overflow:hidden;" forces the
     parent element to expand to contain its floated children. */
  overflow:hidden;
}

.sidebar {
  width:150px;
  float:left;
  background-color:blue;
```

```
}

.内容 {
   宽度:450像素;
   浮动:右;
   背景颜色:黄色;
}
```

第14.7节：简单的三个固定宽度列布局

HTML：

```
<div class="wrapper">
   <div class="left-sidebar">
      <h1>左侧边栏</h1>
      <p>罗勒姆·伊普苏姆（Lorem ipsum）是一段虚拟文本，用于排版和设计。</p>
   </div>
   <div class="content">
      <h1>内容</h1>
      <p>适合的伙伴默默地在我们的海岸边相聚，通过婚姻的纽带结合在一起。
Curabitur sodales ligula in libero。Sed dignissim lacinia nunc。Curabitur tortor。Pellentesque
nibh。Aenean quam。In scelerisque sem at dolor。Maecenas mattis。Sed convallis tristique sem。Proin
ut ligula vel nunc egestas porttitor。Morbi lectus risus，iaculis vel，suscipit quis，luctus non，
massa。
   </div>
   <div class="right-sidebar">
      <h1>右侧边栏</h1>
      <p>Fusce ac turpis quis ligula lacinia aliquet.</p>
   </div>
</div>
```

css：

```
.wrapper {
   宽度:600px;
   背景色:粉色;
   内边距:20px;

   /* 浮动元素不占用任何高度。添加“overflow:hidden;”会强制
      父元素扩展以包含其浮动的子元素。 */
   溢出:隐藏;
}

.left-sidebar {
   宽度:150px;
   背景色:蓝色;
   浮动:左;
}

.内容 {
   宽度:300px;
   背景色:黄色;
   浮动:左;
}

.right-sidebar {
   宽度:150px;
   背景色:绿色;
   浮动:右;
}
```



```
}

.content {
  width:450px;
  float:right;
  background-color:yellow;
}
```

Section 14.7: Simple Three Fixed-Width Column Layout

HTML:

```
<div class="wrapper">
  <div class="left-sidebar">
    <h1>Left Sidebar</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. </p>
  </div>
  <div class="content">
    <h1>Content</h1>
    <p>Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.
Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque
nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin
ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non,
massa. </p>
  </div>
  <div class="right-sidebar">
    <h1>Right Sidebar</h1>
    <p>Fusce ac turpis quis ligula lacinia aliquet.</p>
  </div>
</div>
```

CSS:

```
.wrapper {
  width:600px;
  background-color:pink;
  padding:20px;

  /* Floated elements don't use any height. Adding "overflow:hidden;" forces the
     parent element to expand to contain its floated children. */
  overflow:hidden;
}

.left-sidebar {
  width:150px;
  background-color:blue;
  float:left;
}

.content {
  width:300px;
  background-color:yellow;
  float:left;
}

.right-sidebar {
  width:150px;
  background-color:green;
  float:right;
}
```

第14.8节：双栏懒惰/贪婪布局

此布局使用一个浮动列来创建一个没有定义宽度的两栏布局。在此示例中，左侧边栏是“懒惰”的，因为它只占用所需的空间。换句话说，左侧边栏是“收缩包裹”的。右侧内容列是“贪婪”的，因为它占据所有剩余空间。

HTML：

```
<div class="sidebar">
<h1>侧边栏</h1>
<img src="http://lorempixel.com/150/200/" />
</div>

<div class="content">
<h1>内容</h1>
<p>  Lorem ipsum dolor sit amet，consectetur adipiscing elit。Integer nec odio。Praesent libero。Sed
cursus ante dapibus diam。Sed nisi。Nulla quis sem at nibh elementum imperdiet。Duis sagittis
ipsum。Praesent mauris。Fusce nec tellus sed augue semper porta。Mauris massa。Vestibulum lacinia
arcu eget nulla。  </p>
<p>适合的伙伴默默地在我们的海岸边相聚，通过婚姻的纽带结合在一起。
Curabitur sodales ligula in libero。Sed dignissim lacinia nunc。Curabitur tortor。Pellentesque
nibh。Aenean quam。In scelerisque sem at dolor。Maecenas mattis。Sed convallis tristique sem。Proin
ut ligula vel nunc egestas porttitor。Morbi lectus risus，iaculis vel，suscipit quis，luctus non，
massa。Fusce ac turpis quis ligula lacinia aliquet。Mauris ipsum。Nulla metus metus，ullamcorper
vel，tincidunt sed，euismod in，nibh。 </p>
</div>
```

CSS：

```
.侧边栏 {
  /* `display:table;` 使列收缩包裹 */
  display:table;
  float:left;
  background-color:blue;
}

.内容 {
  /* `overflow:hidden;` 防止 `.content` 流到 `.sidebar` 下面 */
  overflow:hidden;
  background-color:yellow;
}
```

Fiddle

Section 14.8: Two-Column Lazy/Greedy Layout

This layout uses one floated column to create a two-column layout with no defined widths. In this example the left sidebar is "lazy," in that it only takes up as much space as it needs. Another way to say this is that the left sidebar is "shrink-wrapped." The right content column is "greedy," in that it takes up all the remaining space.

HTML:

```
<div class="sidebar">
<h1>Sidebar</h1>
<img src="http://lorempixel.com/150/200/" />
</div>

<div class="content">
<h1>Content</h1>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed
cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis
ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia
arcu eget nulla. </p>
<p>Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.
Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque
nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin
ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non,
massa. Fusce ac turpis quis ligula lacinia aliquet. Mauris ipsum. Nulla metus metus, ullamcorper
vel, tincidunt sed, euismod in, nibh. </p>
</div>
```

CSS:

```
.sidebar {
  /* `display:table;` shrink-wraps the column */
  display:table;
  float:left;
  background-color:blue;
}

.content {
  /* `overflow:hidden;` prevents `.content` from flowing under `.sidebar` */
  overflow:hidden;
  background-color:yellow;
}
```

Fiddle

第15章：排版

参数	详情
字体样式	斜体或倾斜体
字体变体	正常或小型大写字母
字体粗细	正常，加粗或从100到900的数字。
字体大小	字体大小以%，px，em或任何其他有效的CSS单位表示
行高	行高以%，px，em或任何其他有效的CSS单位表示
字体系列	这是用于定义字体系列名称。
颜色	任何有效的CSS颜色表示，例如红色、#00FF00、hsl(240，100%，50%)等。
字体拉伸	是否使用字体的压缩或扩展样式。有效值有正常、极度压缩、非常压缩、压缩、半压缩、半扩展、扩展、非常扩展或极度扩展
文本对齐	开始、结束、左对齐、右对齐、居中、两端对齐、匹配父元素
文本装饰无、下划线、上划线、删除线、初始值、继承；	

第15.1节：字体简写

语法如下：

```
元素 {
    font: [字体样式] [字体变体] [字体粗细] [字体大小/行高] [字体系列];
}
```

你可以使用font简写属性将所有与字体相关的样式放在一个声明中。只需使用font属性，并按正确顺序填写你的值。

例如，要使所有p元素加粗，字体大小为20px，字体系列使用Arial，通常你会这样编写代码：

```
p {
    font-weight: bold;
    font-size: 20px;
    font-family: Arial, sans-serif;
}
```

但是使用font简写可以简化为：

```
p {
    font: bold 20px Arial, sans-serif;
}
```

注意：由于font-style、font-variant、font-weight和line-height是可选的，本例中省略了其中三个。需要注意的是，使用简写会**重置**未给出的其他属性。另一个重要点是，font简写生效的两个必要属性是font-size和font-family。如果两者未同时包含，简写将被忽略。

各属性的初始值：

- **font-style**: normal;
- **font-variant**: normal;
- **font-weight**: normal;

Chapter 15: Typography

Parameter	Details
font-style	`italics` or `oblique`
font-variant	`normal` or `small-caps`
font-weight	`normal`, `bold` or numeric from 100 to 900.
font-size	The font size given in `%`, `px`, `em`, or any other valid CSS measurement
line-height	The line height given in `%`, `px`, `em`, or any other valid CSS measurement
font-family	This is for defining the family's name.
color	Any valid CSS color representation, like `red`, `#00FF00`, `hsl(240, 100%, 50%)` etc.
font-stretch	Whether or not to use a confenced or expanded face from font. Valid values are `normal`, `ultra-condensed`, `extra-condensed`, `condensed`, `semi-condensed`, `semi-expanded`, `expanded`, `extra-expanded` or `ultra-expanded`
text-align	`start`, `end`, `left`, `right`, `center`, `justify`, `match-parent`
text-decoration	`none`, `underline`, `overline`, `line-through`, `initial`, `inherit`;

Section 15.1: The Font Shorthand

With the syntax:

```
element {
    font: [font-style] [font-variant] [font-weight] [font-size/line-height] [font-family];
}
```

You can have all your font-related styles in one declaration with the `font` shorthand. Simply use the `font` property, and put your values in the correct order.

For example, to make all `p` elements bold with a font size of 20px and using Arial as the font family typically you would code it as follows:

```
p {
    font-weight: bold;
    font-size: 20px;
    font-family: Arial, sans-serif;
}
```

However with the font shorthand it can be condensed as follows:

```
p {
    font: bold 20px Arial, sans-serif;
}
```

Note: that since `font-style`, `font-variant`, `font-weight` and `line-height` are optional, the three of them are skipped in this example. It is important to note that using the shortcut **resets** the other attributes not given. Another important point is that the two necessary attributes for the font shortcut to work are `font-size` and `font-family`. If they are not both included the shortcut is ignored.

Initial value for each of the properties:

- **font-style**: normal;
- **font-variant**: normal;
- **font-weight**: normal;

- **font-stretch**: 正常;
- **font-size**: 中等;
- **line-height**: 正常;
- font-family – 取决于用户代理

第15.2节：引号

quotes属性用于自定义<q>标签的开闭引号。

```
q {
    quotes: "«" "»";
}
```

第15.3节：字体大小

HTML：

```
<div id="element-one">你好，我是一段文本。</div>
<div id="element-two">你好，我是一段较小的文本。</div>
```

CSS：

```
#element-one {
    font-size: 30px;
}

#element-two {
    font-size: 10px;
}
```

位于#element-one中的文本大小将为30px，而位于#element-two中的文本大小将为10px。

第15.4节：文本方向

```
div {
    direction: ltr; /* 默认，文本从左到右阅读 */
}
.ex {
    direction: rtl; /* 文本从右到左阅读 */
}
.horizontal-tb {
    writing-mode: horizontal-tb; /* 默认，文本从左到右、从上到下阅读。 */
}
.vertical-rtl {
    writing-mode: vertical-rl; /* 文本从右到左、从上到下阅读 */
}
.vertical-ltr {
    writing-mode: vertical-rl; /* 文本从左到右、从上到下阅读 */
}
```

direction 属性用于更改元素的水平文本方向。

语法：direction: ltr | rtl | initial | inherit;

writing-mode 属性改变文本的排列方式，使其可以根据语言从上到下或从左到右阅读。

- **font-stretch**: normal;
- **font-size**: medium;
- **line-height**: normal;
- font-family – depends on user agent

Section 15.2: Quotes

The quotes property is used to customize the opening and closing quotation marks of the **<q>** tag.

```
q {
    quotes: "«" "»";
}
```

Section 15.3: Font Size

HTML:

```
<div id="element-one">Hello I am some text.</div>
<div id="element-two">Hello I am some smaller text.</div>
```

CSS:

```
#element-one {
    font-size: 30px;
}

#element-two {
    font-size: 10px;
}
```

The text inside #element-one will be 30px in size, while the text in #element-two will be 10px in size.

Section 15.4: Text Direction

```
div {
    direction: ltr; /* Default, text read read from left-to-right */
}
.ex {
    direction: rtl; /* text read from right-to-left */
}
.horizontal-tb {
    writing-mode: horizontal-tb; /* Default, text read from left-to-right and top-to-bottom. */
}
.vertical-rtl {
    writing-mode: vertical-rl; /* text read from right-to-left and top-to-bottom */
}
.vertical-ltr {
    writing-mode: vertical-rl; /* text read from left-to-right and top to bottom */
}
```

The direction property is used to change the horizontal text direction of an element.

Syntax: **direction**: ltr | rtl | initial | inherit;

The writing-mode property changes the alignment of text so it can be read from top-to-bottom or from left-to-right, depending on the language.

语法：direction: horizontal-tb | vertical-rl | vertical-lr;

第15.5节：字体堆栈

```
font-family: 'Segoe UI', Tahoma, sans-serif;
```

浏览器将尝试将字体“Segoe UI”应用于上述属性所针对元素中的字符。如果该字体不可用，或者字体中不包含所需字符的字形，浏览器将回退到 Tahoma，如果仍然不行，则使用用户计算机上的任何无衬线字体。注意，任何包含多个单词的字体名称，如“Segoe UI”，都需要用单引号或双引号括起来。

```
font-family: Consolas, 'Courier New', monospace;
```

浏览器将尝试将字体“Consolas”应用于上述属性所针对元素中的字符。如果该字体不可用，或者字体中不包含所需字符的字形，浏览器将回退到“Courier New”，如果仍然不行，则使用用户计算机上的任何等宽字体。

第15.6节：文本溢出

text-overflow 属性处理如何向用户显示溢出内容。在此示例中，ellipsis 表示被截断的文本。

```
.text {
  overflow: hidden;
  text-overflow: ellipsis;
}
```

不幸的是，text-overflow: ellipsis 仅适用于单行文本。标准 CSS 中无法支持最后一行的省略号，但可以通过非标准的仅限 Webkit 的 flexbox 实现来达到效果。

```
.giveMeEllipsis {
    overflow: hidden;
    text-overflow: ellipsis;
    display: -webkit-box;
-webkit-box-orient: vertical;
    -webkit-line-clamp: N; /* 显示的行数 */
    line-height: X;         /* 备用 */
    max-height: X*N;       /* 备用 */
}
```

示例（请在 Chrome 或 Safari 中打开）：

http://jsfiddle.net/csYjC/1131/

资源：

https://www.w3.org/TR/2012/WD-css3-ui-20120117/#text-overflow0

第15.7节：文字阴影

要为文字添加阴影，请使用text-shadow属性。语法如下：

```
text-shadow: 水平偏移 垂直偏移 模糊 半径 颜色;
```

无模糊半径的阴影

Syntax: `direction: horizontal-tb | vertical-rl | vertical-lr;`

Section 15.5: Font Stacks

```
font-family: 'Segoe UI', Tahoma, sans-serif;
```

The browser will attempt to apply the font face "Segoe UI" to the characters within the elements targeted by the above property. If this font is not available, or the font does not contain a glyph for the required character, the browser will fall back to Tahoma, and, if necessary, any sans-serif font on the user's computer. Note that any font names with more than one word such as "Segoe UI" need to have single or double quotes around them.

```
font-family: Consolas, 'Courier New', monospace;
```

The browser will attempt to apply the font face "Consolas" to the characters within the elements targeted by the above property. If this font is not available, or the font does not contain a glyph for the required character, the browser will fall back to "Courier New," and, if necessary, any monospace font on the user's computer.

Section 15.6: Text Overflow

The `text-overflow` property deals with how overflowed content should be signaled to users. In this example, the `ellipsis` represents clipped text.

```
.text {
  overflow: hidden;
  text-overflow: ellipsis;
}
```

Unfortunately, `text-overflow: ellipsis` only works on a single line of text. There is no way to support ellipsis on the last line in standard CSS, but it can be achieved with non-standard webkit-only implementation of flexboxes.

```
.giveMeEllipsis {
  overflow: hidden;
  text-overflow: ellipsis;
  display: -webkit-box;
  -webkit-box-orient: vertical;
  -webkit-line-clamp: N; /* number of lines to show */
  line-height: X;        /* fallback */
  max-height: X*N;       /* fallback */
}
```

Example (open in Chrome or Safari):

http://jsfiddle.net/csYjC/1131/

Resources:

https://www.w3.org/TR/2012/WD-css3-ui-20120117/#text-overflow0

Section 15.7: Text Shadow

To add shadows to text, use the `text-shadow` property. The syntax is as follows:

```
text-shadow: horizontal-offset vertical-offset blur color;
```

Shadow without blur radius

```
h1 {
  text-shadow: 2px 2px #0000FF;
}
```

这会在标题周围创建一个蓝色阴影效果

带模糊半径的阴影

要添加模糊效果，请添加一个选项模糊半径参数

```
h1 {
  text-shadow: 2px 2px 10px #0000FF;
}
```

多重阴影

要给元素添加多个阴影，请用逗号分隔它们

```
h1 {
    text-shadow: 0 0 3px #FF0000, 0 0 5px #0000FF;
}
```

第15.8节：文本转换

text-transform 属性允许你改变文本的大小写。有效值有：uppercase、`capitalize`、`lowercase`、`initial`、`inherit` 和 `none`

CSS

```
.example1 {
    text-transform: uppercase;
}
.example2 {
    text-transform: capitalize;
}
.example3 {
    text-transform: lowercase;
}
```

HTML

```
<p class="example1">
所有字母大写  <!-- "所有字母大写" -->
</p>
<p class="example2">
所有字母首字母大写 <!-- "所有字母首字母大写（句首字母大写）" -->
</p>
<p class="example3">
所有字母小写  <!-- "所有字母小写" -->
</p>
```

第15.9节：字母间距

```
h2 {
  /* 在每个字母之间水平添加1像素的间距；
     也称为字距调整 */
  letter-spacing: 1px;
```



```
h1 {
  text-shadow: 2px 2px #0000FF;
}
```

This creates a blue shadow effect around a heading

Shadow with blur radius

To add a blur effect, add an option `blur radius` argument

```
h1 {
  text-shadow: 2px 2px 10px #0000FF;
}
```

Multiple Shadows

To give an element multiple shadows, separate them with commas

```
h1 {
    text-shadow: 0 0 3px #FF0000, 0 0 5px #0000FF;
}
```

Section 15.8: Text Transform

The `text-transform` property allows you to change the capitalization of text. Valid values are: `uppercase`, `capitalize`, `lowercase`, `initial`, `inherit`, and `none`

CSS

```
.example1 {
    text-transform: uppercase;
}
.example2 {
    text-transform: capitalize;
}
.example3 {
    text-transform: lowercase;
}
```

HTML

```
<p class="example1">
    all letters in uppercase  <!-- "ALL LETTERS IN UPPERCASE" -->
</p>
<p class="example2">
    all letters in capitalize <!-- "All Letters In Capitalize (Sentence Case)" -->
</p>
<p class="example3">
    all letters in lowercase  <!-- "all letters in lowercase" -->
</p>
```

Section 15.9: Letter Spacing

```
h2 {
  /* adds a 1px space horizontally between each letter;
     also known as tracking */
  letter-spacing: 1px;
```

```
}
```

letter-spacing 属性用于指定文本中字符之间的间距。

！letter-spacing 也支持负值：

```
p {
  letter-spacing: -1px;
}
```

资源：https://developer.mozilla.org/en-US/docs/Web/CSS/letter-spacing

第15.10节：文本缩进

```
p {
    text-indent: 50px;
}
```

text-indent 属性指定文本内容中元素第一行开始前应移动的水平空间量。

资源：

- 只缩进段落的第一行文本？
- https://www.w3.org/TR/CSS21/text.html#propdef-text-indent
- https://developer.mozilla.org/en-US/docs/Web/CSS/text-indent

第15.11节：文本装饰

text-decoration 属性用于设置或移除文本的装饰。

```
h1 { text-decoration: none; }
h2 { text-decoration: overline; }
h3 { text-decoration: line-through; }
h4 { text-decoration: underline; }
```

text-decoration 可以与 text-decoration-style 和 text-decoration-color 结合使用，作为简写属性：

```
.title { text-decoration: underline dotted blue; }
```

这是以下写法的简写版本

```
.title {
  text-decoration-style: dotted;
  text-decoration-line: underline;
  text-decoration-color: blue;
}
```

需要注意的是，以下属性仅在 Firefox 中支持

- text-decoration-color
- text-decoration-line
- text-decoration-style
- text-decoration-skip


```
}
```

The letter-spacing property is used to specify the space between the characters in a text.

! letter-spacing also supports negative values:

```
p {
  letter-spacing: -1px;
}
```

Resources: https://developer.mozilla.org/en-US/docs/Web/CSS/letter-spacing

Section 15.10: Text Indent

```
p {
    text-indent: 50px;
}
```

The `text-indent` property specifies how much horizontal space text should be moved before the beginning of the first line of the text content of an element.

Resources:

- Indenting only the first line of text in a paragraph?
- https://www.w3.org/TR/CSS21/text.html#propdef-text-indent
- https://developer.mozilla.org/en-US/docs/Web/CSS/text-indent

Section 15.11: Text Decoration

The `text-decoration` property is used to set or remove decorations from text.

```
h1 { text-decoration: none; }
h2 { text-decoration: overline; }
h3 { text-decoration: line-through; }
h4 { text-decoration: underline; }
```

text-decoration can be used in combination with text-decoration-style and text-decoration-color as a shorthand property:

```
.title { text-decoration: underline dotted blue; }
```

This is a shorthand version of

```
.title {
  text-decoration-style: dotted;
  text-decoration-line: underline;
  text-decoration-color: blue;
}
```

It should be noted that the following properties are only supported in Firefox

- text-decoration-color
- text-decoration-line
- text-decoration-style
- text-decoration-skip

第15.12节：单词间距

word-spacing 属性指定标签和单词之间的间距行为。

可能的取值

- 一个正值或负值的*长度*（使用`em px vh cm`等单位）或*百分比*（使用`%`）
- 关键字`normal`使用字体的默认单词间距
- 关键字inherit从父元素继承值

CSS

```
.normal    { word-spacing: normal; }
.narrow    { word-spacing: -3px; }
.extensive { word-spacing: 10px; }
```

HTML

```
<p>
  <span class="normal">这是一个示例，展示“word-spacing”的效果。</span><br>
  <span class="narrow">这是一个示例，展示“word-spacing”的效果。</span><br>
  <span class="extensive">这是一个示例，展示“word-spacing”的效果。</span><br>
</p>
```

在线演示

自己试试

进一步阅读：

- word-spacing – MDN
- word-spacing – w3.org

第15.13节：字体变体

属性：

normal

字体的默认属性。

小型大写字母

将每个字母设置为大写，但使小写字母（来自原文）比原本的大写字母尺寸更小。

CSS：

```
.smallcaps{
    font-variant: 小型大写字母;
}
```

HTML：

```
<p class="smallcaps">
    关于CSS字体的文档
```


Section 15.12: Word Spacing

The word-spacing property specifies the spacing behavior between tags and words.

Possible values

- a positive or negative *length* (using `em px vh cm` etc.) or *percentage* (using `%`)
- the keyword `normal` uses the font's default word spacing
- the keyword `inherit` takes the value from the parent element

CSS

```
.normal    { word-spacing: normal; }
.narrow    { word-spacing: -3px; }
.extensive { word-spacing: 10px; }
```

HTML

```
<p>
  <span class="normal">This is an example, showing the effect of "word-spacing".</span><br>
  <span class="narrow">This is an example, showing the effect of "word-spacing".</span><br>
  <span class="extensive">This is an example, showing the effect of "word-spacing".</span><br>
</p>
```

Online-Demo

Try it yourself

Further reading:

- word-spacing – MDN
- word-spacing – w3.org

Section 15.13: Font Variant

Attributes:

normal

Default attribute of fonts.

small-caps

Sets every letter to uppercase, **but** makes the lowercase letters(from original text) smaller in size than the letters that originally uppercase.

CSS:

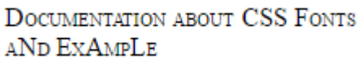
```
.smallcaps{
   font-variant: small-caps;
}
```

HTML:

```
<p class="smallcaps">
    Documentation about CSS Fonts
```

```
    <br>
    aNd ExAmpLe
```

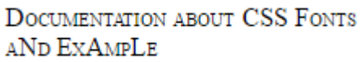
输出：



注意：font-variant 属性是以下属性的简写：font-variant-caps、font-variant-numeric、font-variant-alternates、font-variant-ligatures 和 font-variant-east-asian。


```
    <br>
    aNd ExAmpLe
</p>
```

Output:



Note: The font-variant property is a shorthand for the properties: font-variant-caps, font-variant-numeric, font-variant-alternates, font-variant-ligatures, and font-variant-east-asian.

第16章：弹性盒布局（Flexbox）

弹性盒模块，简称“flexbox”，是一种为用户界面设计的盒模型，它允许用户在容器中的项目之间对齐和分配空间，使得当页面布局必须适应不同且未知的屏幕尺寸时，元素表现得更加可预测。弹性容器会扩展项目以填充可用空间，并缩小它们以防止溢出。

第16.1节：动态垂直和水平居中（align-items，justify-content）

简单示例（居中单个元素）

HTML

```
<div class="aligner">
  <div class="aligner-item">...</div>
</div>
```

CSS

```
.aligner {
  display: flex;
  align-items: center;
  justify-content: center;
}

.aligner-item {
  max-width: 50%; /*仅用于演示。请使用实际宽度替代。*/
}
```

这里是一个 demo。

推理

属性	值	描述
`align-items`	`center`	这会沿着除flex-direction指定的轴以外的轴对元素进行居中，即对于水平的flexbox是垂直居中，对于垂直的flexbox是水平居中。
justify-content	center	这会沿着flex-direction指定的轴对元素进行居中。即对于水平（flex-direction: row）的flexbox，元素水平居中；对于垂直（flex-direction: column）的flexbox，元素垂直居中。

单个属性示例

以下所有样式均应用于此简单布局：

```
<div id="container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

其中#container是flex-box。

示例：`justify-content: center` 在水平flexbox上

CSS：

Chapter 16: Flexible Box Layout (Flexbox)

The Flexible Box module, or just 'flexbox' for short, is a box model designed for user interfaces, and it allows users to align and distribute space among items in a container such that elements behave predictably when the page layout must accommodate different, unknown screen sizes. A flex container expands items to fill available space and shrinks them to prevent overflow.

Section 16.1: Dynamic Vertical and Horizontal Centering (align-items, justify-content)

Simple Example (centering a single element)

HTML

```
<div class="aligner">
  <div class="aligner-item">...</div>
</div>
```

CSS

```
.aligner {
  display: flex;
  align-items: center;
  justify-content: center;
}

.aligner-item {
  max-width: 50%; /*for demo. Use actual width instead.*/
}
```

Here is a demo.

Reasoning

Property	Value	Description
`align-items`	`center`	This centers the elements along the axis other than the one specified by `flex-direction`, i.e., vertical centering for a horizontal flexbox and horizontal centering for a vertical flexbox.
`justify-content`	`center`	This centers the elements along the axis specified by `flex-direction`. I.e., for a horizontal (`flex-direction: row`) flexbox, this centers horizontally, and for a vertical flexbox (`flex-direction: column`) flexbox, this centers vertically)

Individual Property Examples

All of the below styles are applied onto this simple layout:

```
<div id="container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

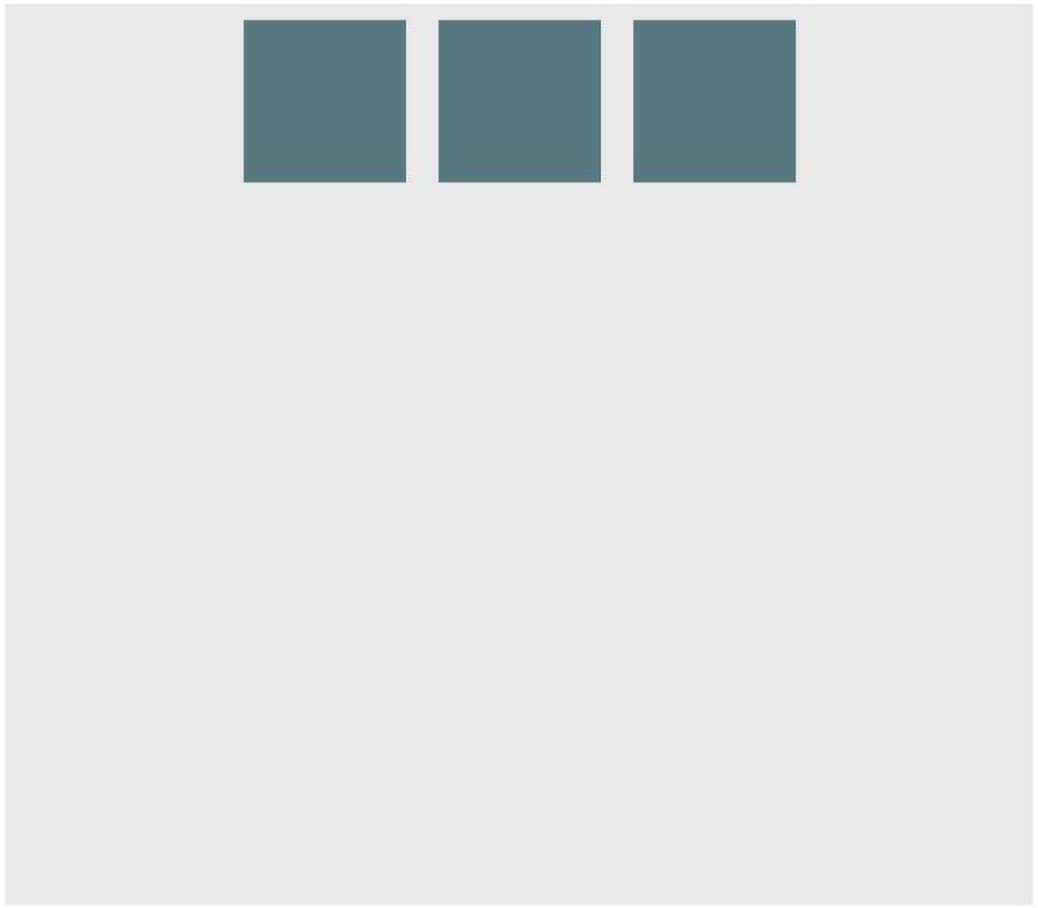
where `#container` is the `flex-box`.

Example: `justify-content: center` on a horizontal flexbox

CSS:

```
div#container {
  display: flex;
  flex-direction: row;
  justify-content: center;
}
```

结果：



这里是一个 demo。

示例：`justify-content: center` 在垂直flexbox上

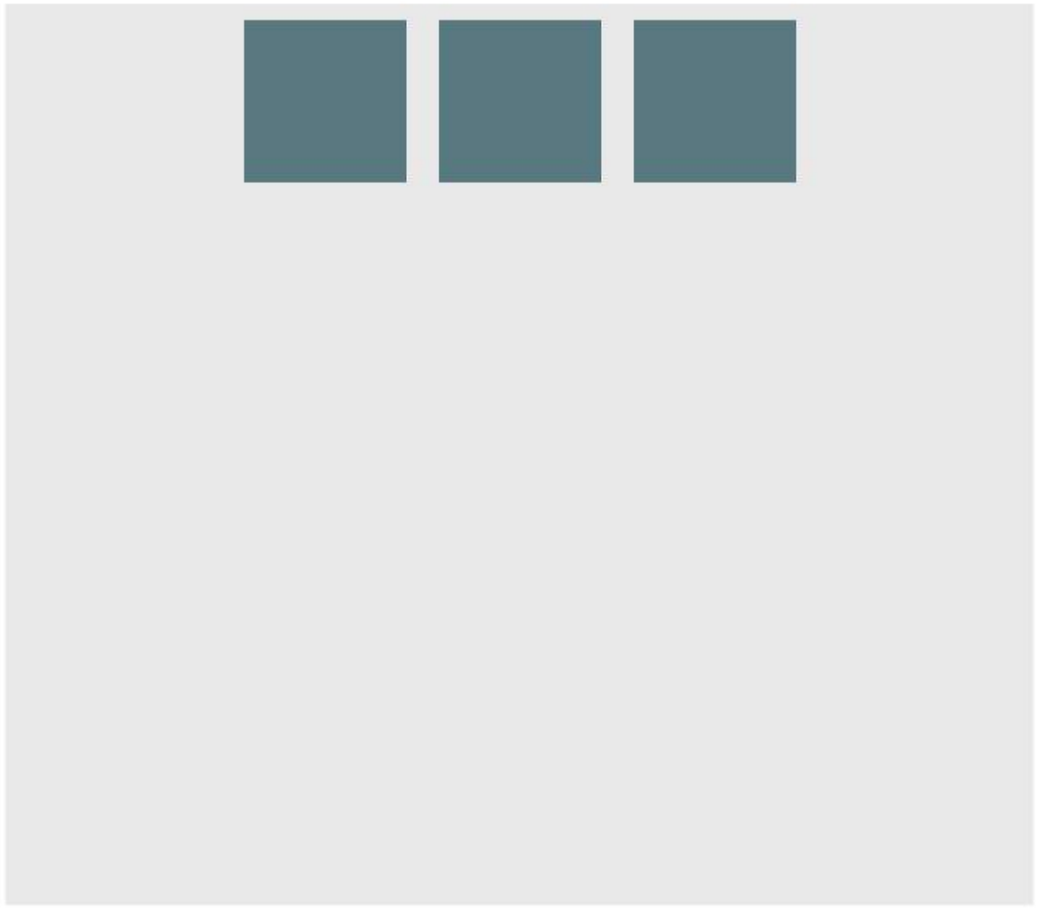
CSS：

```
div#container {
  display: flex;
  flex-direction: column;
  justify-content: center;
}
```

结果：


```
div#container {
  display: flex;
  flex-direction: row;
  justify-content: center;
}
```

Outcome:



Here is a demo.

Example: `justify-content: center` on a vertical flexbox

CSS:

```
div#container {
  display: flex;
  flex-direction: column;
  justify-content: center;
}
```

Outcome:

这里是一个 demo。

示例：`align-content: center` 在水平弹性盒子上

CSS：

```
div#container {
  display: flex;
  flex-direction: row;
  align-items: center;
}
```

结果：

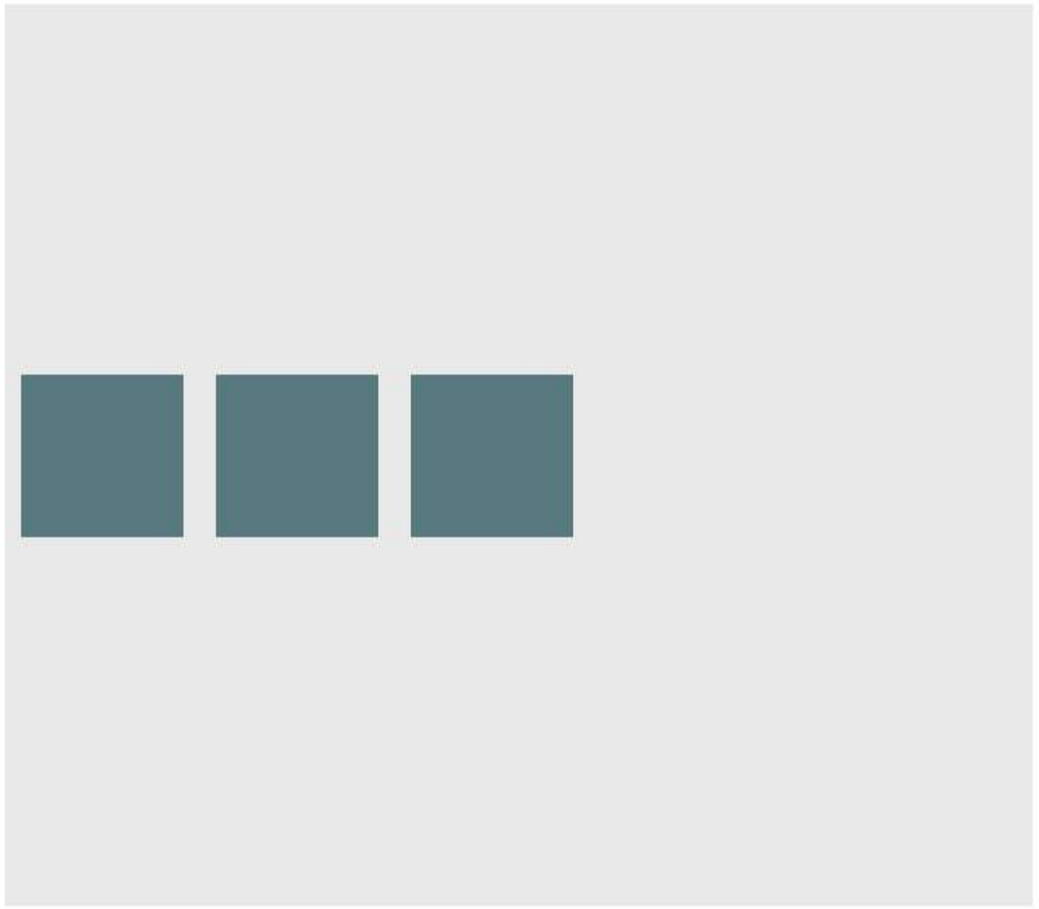
Here is a demo.

Example: `align-content: center` on a horizontal flexbox

CSS:

```
div#container {
  display: flex;
  flex-direction: row;
  align-items: center;
}
```

Outcome:



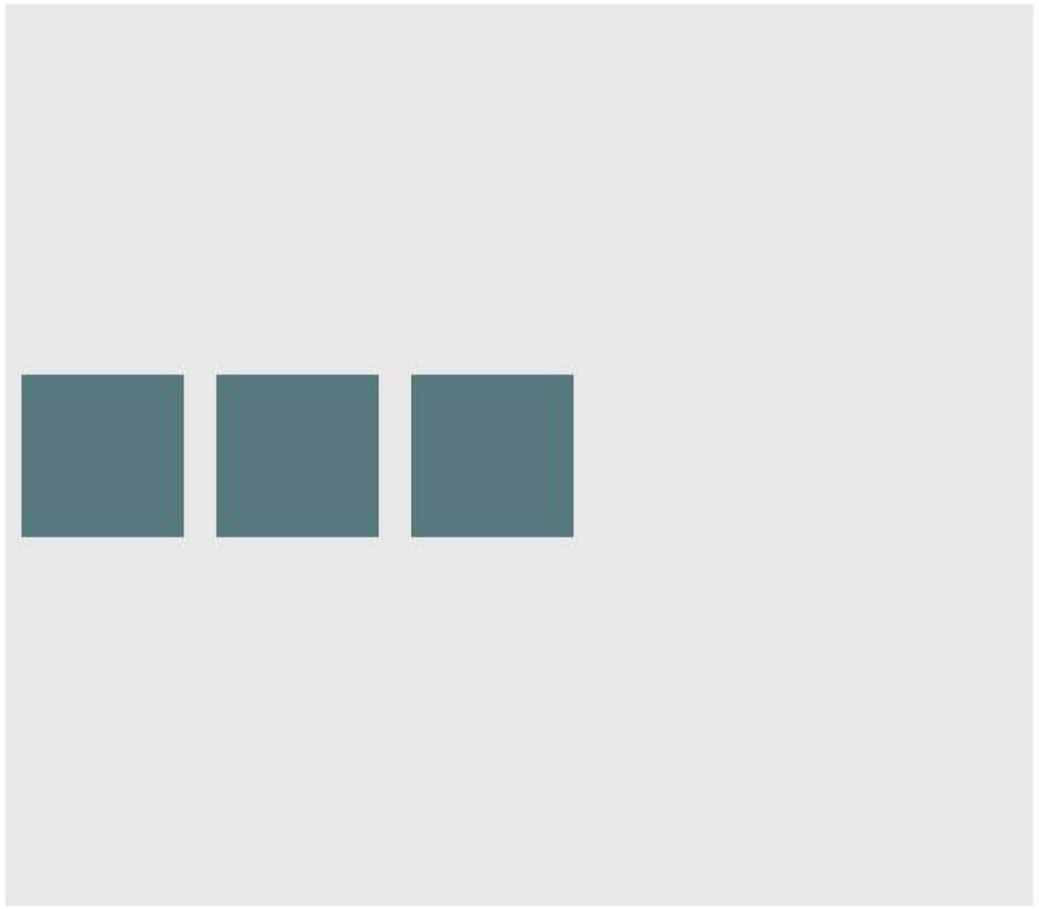
这里是一个 demo。

示例：`align-content: center` 在垂直弹性盒子上

CSS：

```
div#container {
  display: flex;
  flex-direction: column;
  align-items: center;
}
```

结果：



Here is a demo.

Example: `align-content: center` on a vertical flexbox

CSS:

```
div#container {
  display: flex;
  flex-direction: column;
  align-items: center;
}
```

Outcome:

这里是一个 demo。

示例：水平弹性盒子上同时居中的组合

```
div#container {
  display: flex;
  flex-direction: row;
  justify-content: center;
  align-items: center;
}
```

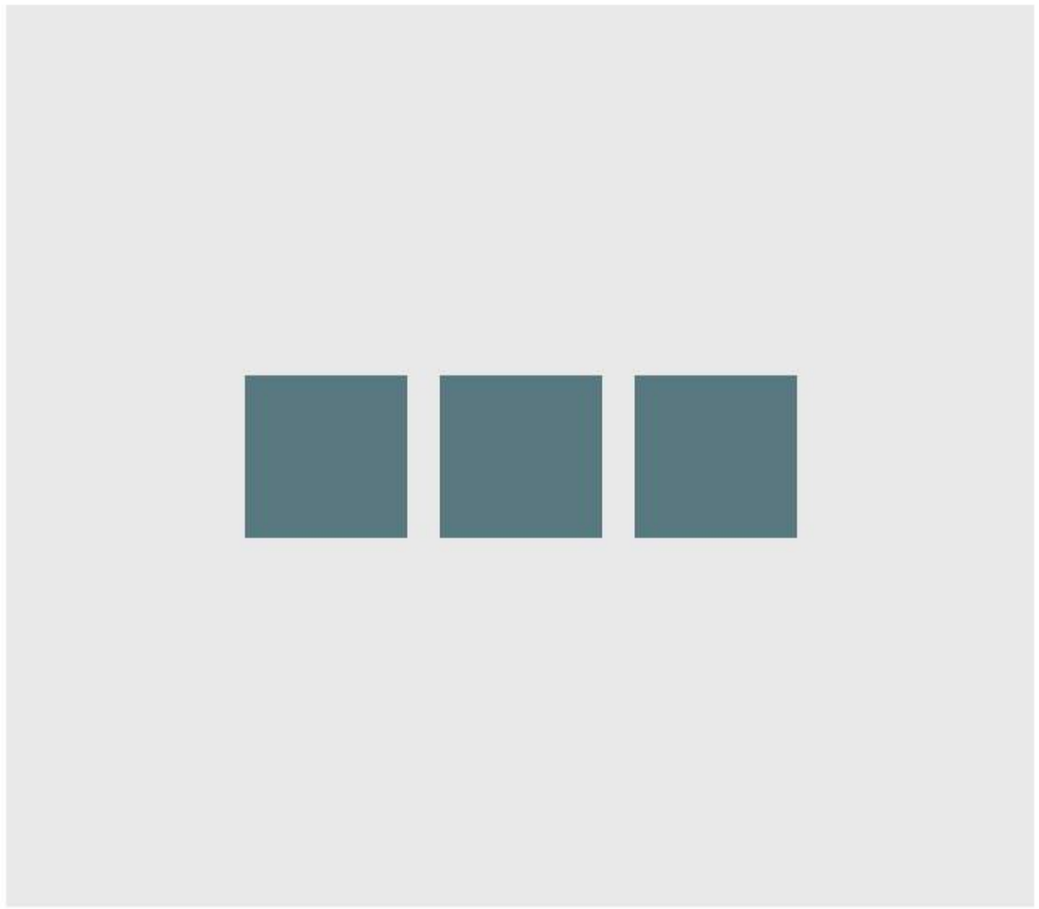
结果：

Here is a demo.

Example: Combination for centering both on horizontal flexbox

```
div#container {
  display: flex;
  flex-direction: row;
  justify-content: center;
  align-items: center;
}
```

Outcome:

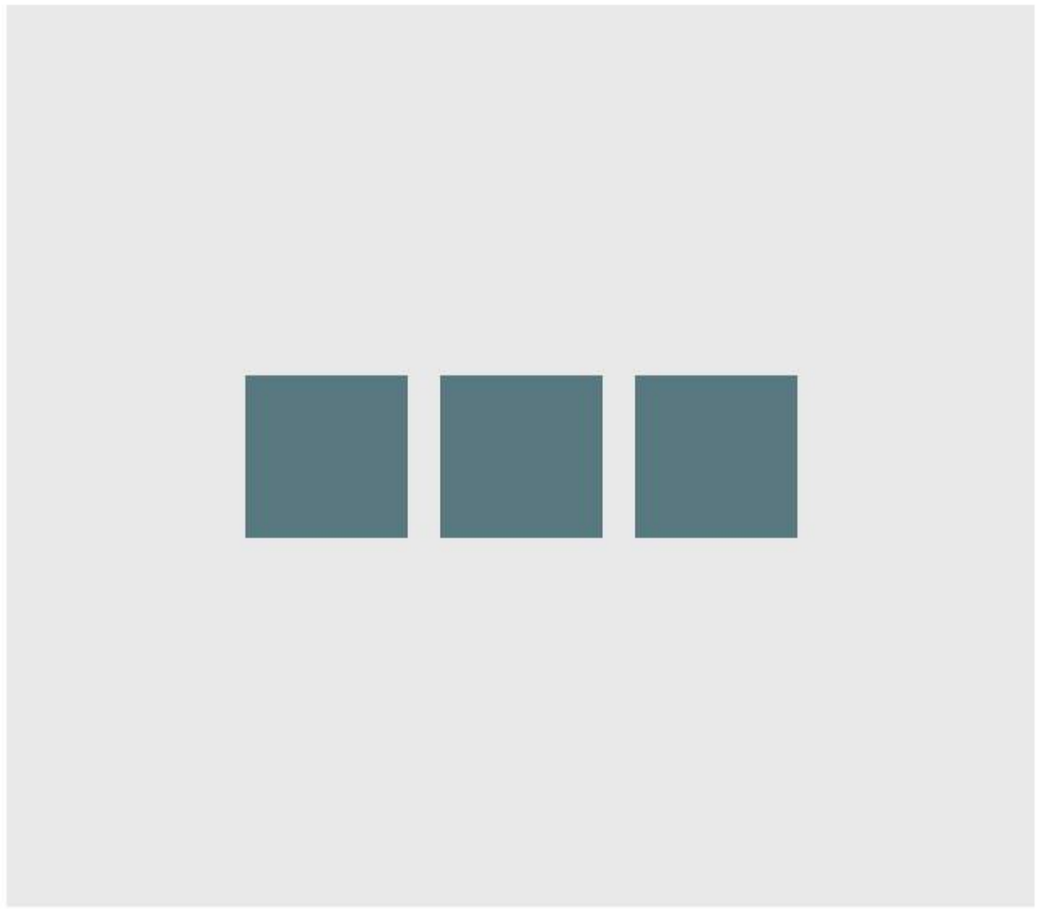


这里是一个 demo。

示例：用于垂直弹性盒子居中的组合

```
div#container {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}
```

结果：

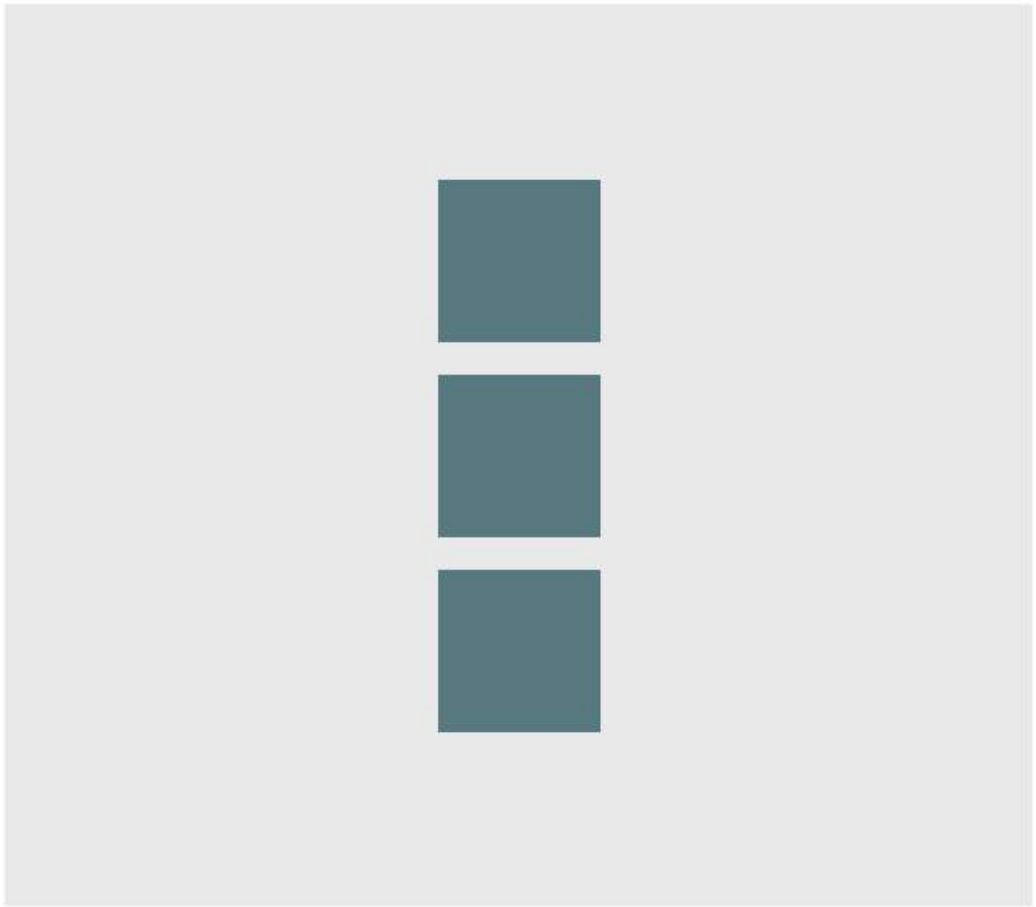


Here is a demo.

Example: Combination for centering both on vertical flexbox

```
div#container {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}
```

Outcome:



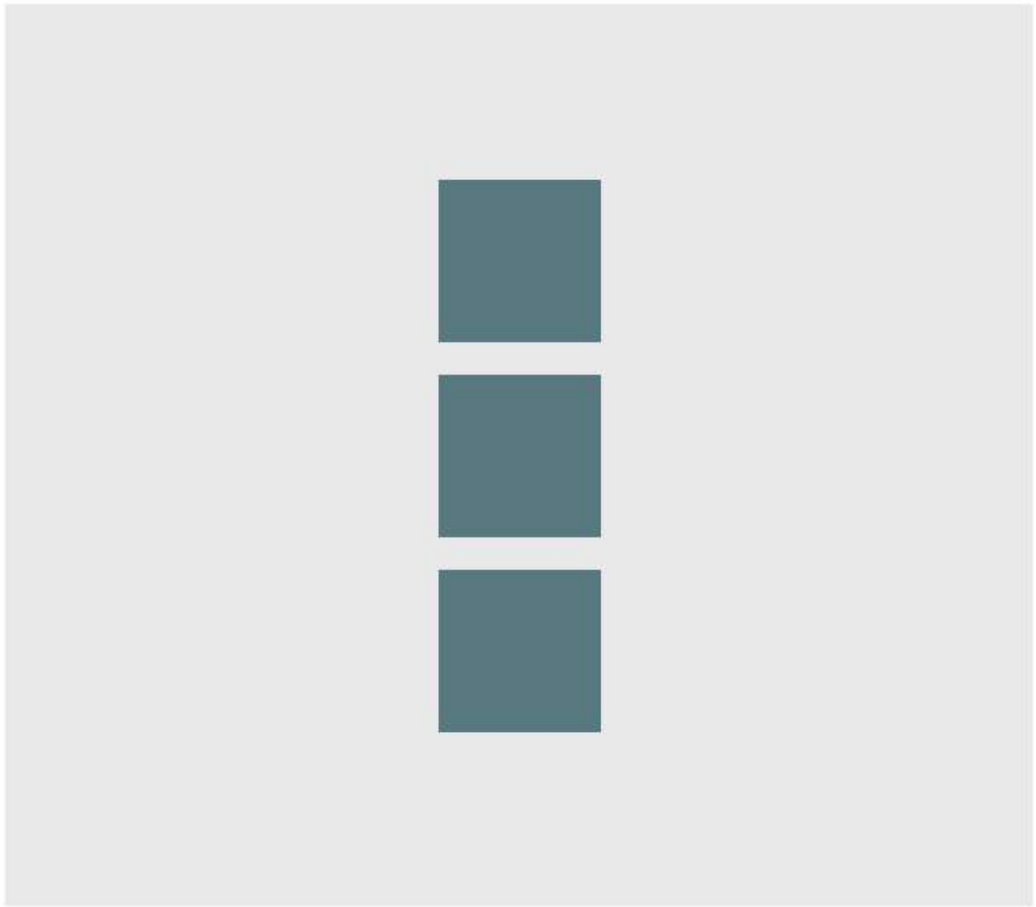
这里是一个 demo。

第16.2节：粘性可变高度页脚

此代码创建了一个粘性页脚。当内容未达到视口底部时，页脚会粘附在视口底部。当内容超出视口底部时，页脚也会被推离视口。查看结果

HTML：

```
<div class="header">
  <h2>页眉</h2>
</div>

<div class="content">
  <h1>内容</h1>
          <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero.
Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis
ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia
arcu eget nulla. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos
himenaeos. Curabitur sodales ligula in libero. </p>
</div>

<div class="footer">
```

Here is a demo.

Section 16.2: Sticky Variable-Height Footer

This code creates a sticky footer. When the content doesn't reach the end of the viewport, the footer sticks to the bottom of the viewport. When the content extends past the bottom of the viewport, the footer is also pushed out of the viewport. *View Result*

HTML:

```
<div class="header">
  <h2>Header</h2>
</div>

<div class="content">
  <h1>Content</h1>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero.
Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis
ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia
arcu eget nulla. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos
himenaeos. Curabitur sodales ligula in libero. </p>
</div>

<div class="footer">
```

```
  <h4>页脚</h4>
</div>
```

css：

```
html, body {
  height: 100%;
}

body {
  display: flex;
  flex-direction: column;
}

.内容 {
  /* 为了最佳浏览器兼容性，请包含 `0 auto`。*/
  flex: 1 0 auto;
}

.header, .footer {
  background-color: grey;
  color: white;
  flex: none;
}
```

第16.3节：使元素最佳适应其容器

flexbox 的一个很棒的功能是允许容器最佳地适应其父元素。

实时演示。

HTML：

```
<div class="flex-container">
  <div class="flex-item">1</div>
  <div class="flex-item">2</div>
  <div class="flex-item">3</div>
  <div class="flex-item">4</div>
  <div class="flex-item">5</div>
</div>
```

CSS：

```
.flex-container {
    背景颜色: #000;
    高度: 100%;
    display:flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: flex-start;
    align-content: stretch;
    align-items: stretch;
}

.flex-item {
    background-color: #ccf;
    margin: 0.1em;
    flex-grow: 1;
    flex-shrink: 0;
```



```
  <h4>Footer</h4>
</div>
```

CSS:

```
html, body {
  height: 100%;
}

body {
  display: flex;
  flex-direction: column;
}

.content {
  /* Include `0 auto` for best browser compatibility. */
  flex: 1 0 auto;
}

.header, .footer {
  background-color: grey;
  color: white;
  flex: none;
}
```

Section 16.3: Optimally fit elements to their container

One of the nicest features of flexbox is to allow optimally fitting containers to their parent element.

Live demo.

HTML:

```
<div class="flex-container">
  <div class="flex-item">1</div>
  <div class="flex-item">2</div>
  <div class="flex-item">3</div>
  <div class="flex-item">4</div>
  <div class="flex-item">5</div>
</div>
```

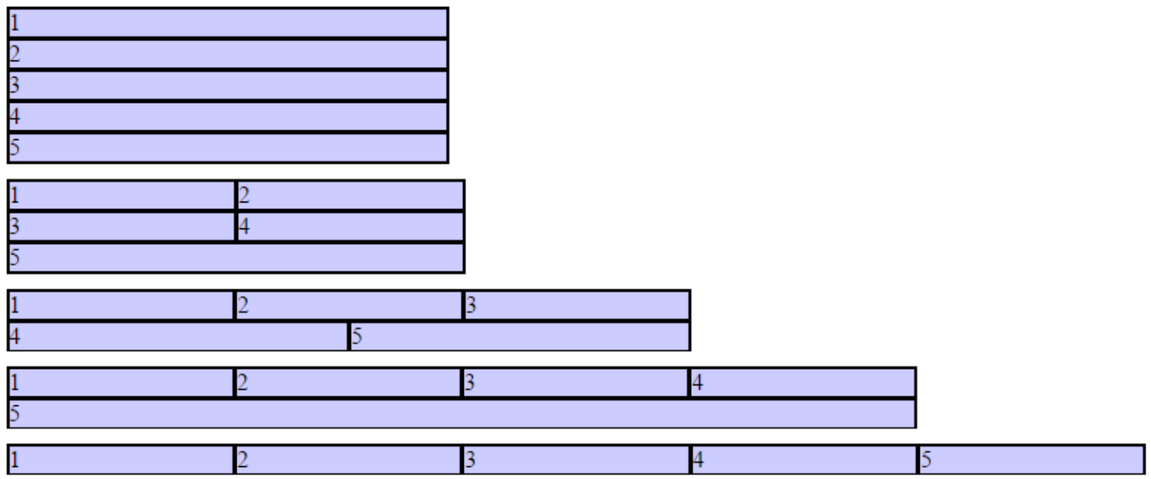
CSS:

```
.flex-container {
    background-color: #000;
    height: 100%;
    display:flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: flex-start;
    align-content: stretch;
    align-items: stretch;
}

.flex-item {
    background-color: #ccf;
    margin: 0.1em;
    flex-grow: 1;
    flex-shrink: 0;
```

```
    flex-basis: 200px; /* 或者可以使用百分比来确保特定布局 */
}
```

结果：

列会随着屏幕大小调整。



第16.4节：使用Flexbox实现圣杯布局

圣杯布局是一种具有固定高度的页眉和页脚，中间部分包含三列的布局。这三列包括一个固定宽度的侧边导航栏、一个流动的中间区域，以及一个用于其他内容（如广告）的列（流动的中间区域在标记中排在第一位）。可以使用CSS Flexbox通过非常简单的标记来实现：

HTML 标记：

```
<div class="container">
  <header class="header">页眉</header>
  <div class="content-body">
    <main class="content">内容</main>
    <nav class="sidenav">导航</nav>
    <aside class="ads">广告</aside>
  </div>
  <footer class="footer">页脚</footer>
</div>
```

CSS：

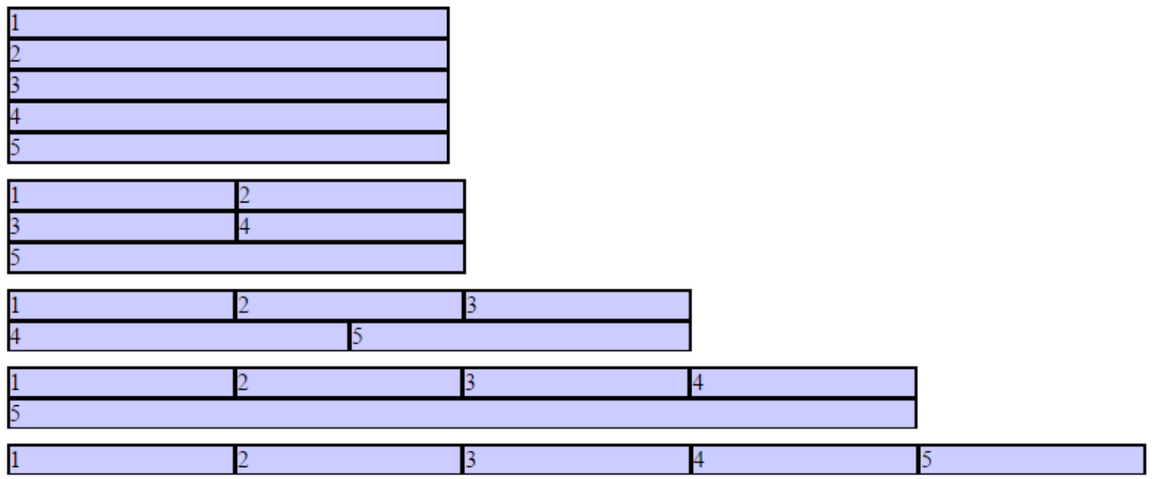
```
body {
  margin: 0;
  padding: 0;
}

.container {
  display: flex;
  flex-direction: column;
  height: 100vh;
}

.header {
  flex: 0 0 50px;
```



```
    flex-basis: 200px; /* or % could be used to ensure a specific layout */
}
```

Outcome:

Columns adapt as screen is resized.



Section 16.4: Holy Grail Layout using Flexbox

Holy Grail layout is a layout with a fixed height header and footer, and a center with 3 columns. The 3 columns include a fixed width sidenav, a fluid center, and a column for other content like ads (the fluid center appears first in the markup). CSS Flexbox can be used to achieve this with a very simple markup:

HTML Markup:

```
<div class="container">
  <header class="header">Header</header>
  <div class="content-body">
    <main class="content">Content</main>
    <nav class="sidenav">Nav</nav>
    <aside class="ads">Ads</aside>
  </div>
  <footer class="footer">Footer</footer>
</div>
```

CSS:

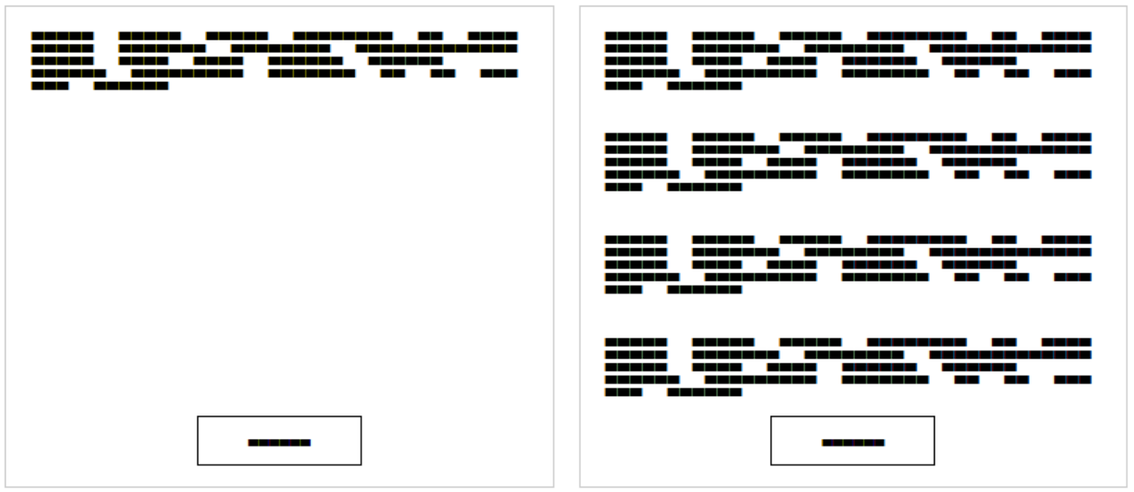
```
body {
  margin: 0;
  padding: 0;
}

.container {
  display: flex;
  flex-direction: column;
  height: 100vh;
}

.header {
  flex: 0 0 50px;
```

```
}

.content-body {
  flex: 1 1 auto;

  display: flex;
  flex-direction: row;
}

.content-body .content {
  flex: 1 1 auto;
  overflow: auto;
}

.content-body .sidenav {
  order: -1;
  flex: 0 0 100px;
  overflow: auto;
}

.content-body .ads {
  flex: 0 0 100px;
  overflow: auto;
}

.footer {
  flex: 0 0 50px;
}
```

演示

第16.5节：使用flexbox在卡片内完美对齐按钮

如今设计中一个常见的模式是将call to actions垂直对齐在其包含的卡片内，如下所示：



这可以通过使用`flexbox`

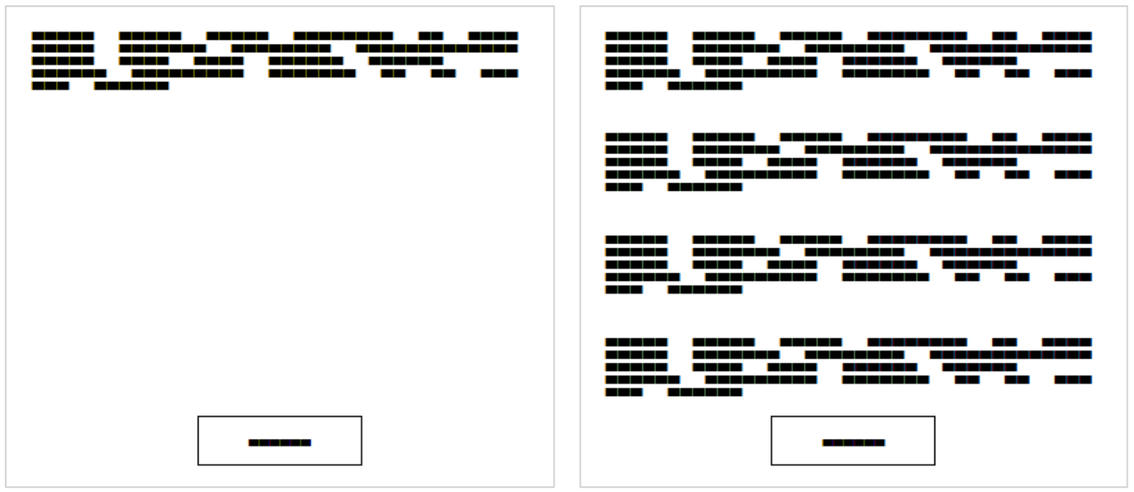
HTML的特殊技巧来实现


```
}

.content-body {
  flex: 1 1 auto;

  display: flex;
  flex-direction: row;
}

.content-body .content {
  flex: 1 1 auto;
  overflow: auto;
}

.content-body .sidenav {
  order: -1;
  flex: 0 0 100px;
  overflow: auto;
}

.content-body .ads {
  flex: 0 0 100px;
  overflow: auto;
}

.footer {
  flex: 0 0 50px;
}
```

Demo

Section 16.5: Perfectly aligned buttons inside cards with flexbox

It's a regular pattern in design these days to vertically align **call to actions** inside its containing cards like this:



This can be achieved using a special trick with `flexbox`

HTML

```
<div class="cards">
  <div class="card">
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p><button>操作</button></p>
  </div>
  <div class="card">
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p><button>操作</button></p>
  </div>
</div>
```

首先，我们使用CSS对容器应用display:flex;。这将创建两个等高的列，内容在其中自然流动

CSS

```
.cards {
    display: flex;
}
.card {
    border: 1px 实线 #ccc;
    margin: 10px 10px;
    padding: 0 20px;
}
button {
    height: 40px;
    background: #fff;
    padding: 0 40px;
    border: 1px solid #000;
}
p:last-child {
    text-align: center;
}
```

布局将会改变，变成如下形式：

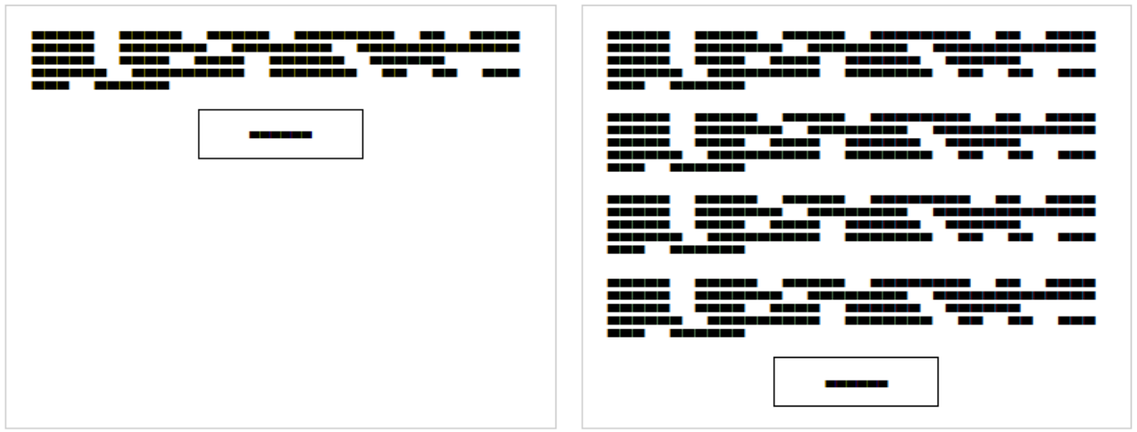

```
<div class="cards">
  <div class="card">
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p><button>Action</button></p>
  </div>
  <div class="card">
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p>Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim
mollit labore dolore voluptate ullamco et ut sed qui minim.</p>
    <p><button>Action</button></p>
  </div>
</div>
```

First of all, we use CSS to apply `display: flex;` to the container. This will create 2 columns equal in height with the content flowing naturally inside it

CSS

```
.cards {
    display: flex;
}
.card {
    border: 1px solid #ccc;
    margin: 10px 10px;
    padding: 0 20px;
}
button {
    height: 40px;
    background: #fff;
    padding: 0 40px;
    border: 1px solid #000;
}
p:last-child {
    text-align: center;
}
```

The layout will change and become like this:



为了将按钮移动到块的底部，我们需要对卡片本身应用 display: flex;，并将方向设置为 column。之后，我们应选择卡片内的最后一个元素，并将 margin-top 设置为auto。这样会将最后一个段落推到卡片底部，从而实现所需效果。

最终 CSS：

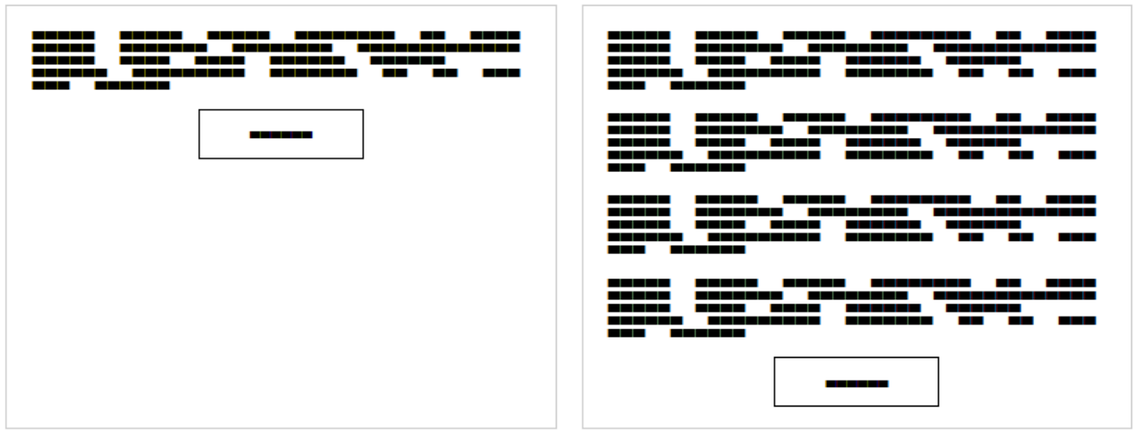
```
.cards {
    display: flex;
}
.card {
    border: 1px solid #ccc;
    margin: 10px 10px;
    padding: 0 20px;
    display: flex;
    flex-direction: column;
}
button {
    height: 40px;
    background: #fff;
    padding: 0 40px;
    border: 1px solid #000;
}
p:last-child {
    text-align: center;
    margin-top: auto;
}
```

第16.6节：嵌套容器的相同高度

这段代码确保所有嵌套容器的高度始终相同。通过保证所有嵌套元素的高度与包含它们的父级div相同来实现。查看示例: https://jsfiddle.net/3wwh7ewp/该效果是由于属性align-items默认设置为stretch实现的。

HTML

```
<div class="container">
  <div style="background-color: red">
    一些 <br />
数据 <br />
    用于<br />
```

In order to move the buttons to the bottom of the block, we need to apply `display: flex;` to the card itself with the direction set to `column`. After that, we should select the last element inside the card and set the `margin-top` to `auto`. This will push the last paragraph to the bottom of the card and achieve the required result.

Final CSS:

```
.cards {
    display: flex;
}
.card {
    border: 1px solid #ccc;
    margin: 10px 10px;
    padding: 0 20px;
    display: flex;
    flex-direction: column;
}
button {
    height: 40px;
    background: #fff;
    padding: 0 40px;
    border: 1px solid #000;
}
p:last-child {
    text-align: center;
    margin-top: auto;
}
```

Section 16.6: Same height on nested containers

This code makes sure that all nested containers are always the same height. This is done by assuring that all nested elements are the same height as the containing parent div. See working example: https://jsfiddle.net/3wwh7ewp/

This effect is achieved due to the property `align-items` being set to `stretch` by default.

HTML

```
<div class="container">
  <div style="background-color: red">
    Some <br />
    data <br />
    to make<br />
```

```
设定高度 <br />
 </div>
  <div style="background-color: blue">
    较少的 <br />
行数 <br />
 </div>
</div>
```

CSS

```
.container {
    display: flex;
    align-items: stretch; // 默认值
}
```

注意：不支持IE10以下版本

```
    a height <br />
 </div>
  <div style="background-color: blue">
    Fewer <br />
    lines <br />
 </div>
</div>
```

CSS

```
.container {
    display: flex;
    align-items: stretch; // Default value
}
```

Note: Does not work on IE versions under 10

第17章：层叠与特异性

第17.1节：计算选择器特异性

每个单独的 CSS 选择器都有其特定性值。序列中的每个选择器都会增加该序列的整体特定性。选择器分为三种不同的特定性组：A、B 和 c。当多个选择器序列选择同一元素时，浏览器会使用整体特定性最高的序列所应用的样式。

组	由以下组成	示例
A	id 选择器	`#foo`

B	类选择器
	属性选择器
	伪类
		`.bar`
		`[title]`，`[colspan="2"]`
		`：hover`，`：nth-child(2)`

c	类型选择器 伪元素	div，li `::before，::first-letter`

组A的优先级最高，其次是组B，最后是组c。

通用选择器（*）和组合符（如>和~）没有优先级。

示例1：各种选择器序列的优先级

```
#foo #baz {}      /* a=2, b=0, c=0 */

#foo.bar {}       /* a=1, b=1, c=0 */

#foo {}           /* a=1, b=0, c=0 */

.bar:hover {}     /* a=0, b=2, c=0 */

div.bar {}        /* a=0, b=1, c=1 */

:hover {}         /* a=0, b=1, c=0 */

[title] {}        /* a=0, b=1, c=0 */

.bar {}           /* a=0, b=1, c=0 */

div ul + li {}    /* a=0, b=0, c=3 */

p::after {}       /* a=0, b=0, c=2 */

*::before {}      /* a=0, b=0, c=1 */

::before {}       /* a=0, b=0, c=1 */

div {}            /* a=0, b=0, c=1 */

* {}              /* a=0, b=0, c=0 */
```

示例2：浏览器如何使用优先级

想象以下的 CSS 实现：

```
#foo {
   color: 蓝色;
}
```


Chapter 17: Cascading and Specificity

Section 17.1: Calculating Selector Specificity

Each individual CSS Selector has its own specificity value. Every selector in a sequence increases the sequence's overall specificity. Selectors fall into one of three different specificity groups: *A*, *B* and *c*. When multiple selector sequences select a given element, the browser uses the styles applied by the sequence with the highest overall specificity.

Group	Comprised of	Examples
A	id selectors	`#foo`

B	class selectors
	attribute selectors
	pseudo-classes
		`.bar`
		`[title]`, `[colspan="2"]`
		`:hover`, `:nth-child(2)`

c	type selectors pseudo-elements	`div`, `li` `::before`, `::first-letter`

Group *A* is the most specific, followed by Group *B*, then finally Group *c*.

The universal selector (*) and combinators (like > and ~) have no specificity.

Example 1: Specificity of various selector sequences

```
#foo #baz {}      /* a=2, b=0, c=0 */

#foo.bar {}       /* a=1, b=1, c=0 */

#foo {}           /* a=1, b=0, c=0 */

.bar:hover {}     /* a=0, b=2, c=0 */

div.bar {}        /* a=0, b=1, c=1 */

:hover {}         /* a=0, b=1, c=0 */

[title] {}        /* a=0, b=1, c=0 */

.bar {}           /* a=0, b=1, c=0 */

div ul + li {}    /* a=0, b=0, c=3 */

p::after {}       /* a=0, b=0, c=2 */

*::before {}      /* a=0, b=0, c=1 */

::before {}       /* a=0, b=0, c=1 */

div {}            /* a=0, b=0, c=1 */

* {}              /* a=0, b=0, c=0 */
```

Example 2: How specificity is used by the browser

Imagine the following CSS implementation:

```
#foo {
  color: blue;
}
```

```
.bar {
  颜色: 红色;
  背景: 黑色;
}
```

这里有一个ID选择器声明了颜色为*蓝色*，和一个类选择器声明了颜色为*红色*以及背景为*黑色*。

一个ID为#foo且类为.bar的元素将被这两个声明同时选中。ID选择器具有A组特异性，类选择器具有B组特异性。ID选择器的权重高于任意数量的类选择器。因此，来自**#foo**选择器的颜色:蓝色;和来自**.bar**选择器的背景:黑色;将应用于该元素。ID选择器更高的特异性将导致浏览器忽略.bar选择器的颜色声明。

现在想象一个不同的CSS实现：

```
.bar {
  颜色: 红色;
  背景: 黑色;
}

.baz {
  背景: 白色;
}
```

这里我们有两个类选择器；其中一个将color声明为red，将background声明为black，另一个将background声明为white。

同时具有.bar和.baz类的元素将同时受到这两个声明的影响，然而我们现在的问题是.bar和.baz的GroupB特异性完全相同。CSS的层叠特性为我们解决了这个问题：由于.baz定义在.bar之后，我们的元素最终会从.bar获得red颜色，但背景色则来自.baz的white。

示例3：如何操作特异性

上面示例2中的最后一段代码可以被操作，以确保我们的.bar类选择器的color声明被使用，而不是.baz类选择器的声明。

```
.bar {}         /* a=0, b=1, c=0 */
.baz {}         /* a=0, b=1, c=0 */
```

实现这一点最常见的方法是找出还能应用于.bar选择器序列的其他选择器。例如，如果.bar类只应用于span元素，我们可以将.bar选择器修改为span.bar。这样它将获得新的GroupC特异性，从而覆盖.baz选择器的特异性不足：

```
span.bar {}     /* a=0, b=1, c=1 */
.baz {}         /* a=0, b=1, c=0 */
```

然而，并不总能找到另一个所有使用.bar类的元素都共有的选择器。正因为如此，CSS允许我们通过重复选择器来增加特异性。我们可以使用.bar.bar代替单独的.bar（参见The grammar of Selectors, W3C Recommendation）。这仍然选择任何具有.bar类的元素，但现在具有双倍的GroupB特异性：

```
.bar.bar {}     /* a=0, b=2, c=0 */
.baz {}         /* a=0, b=1, c=0 */
```



```
.bar {
  color: red;
  background: black;
}
```

Here we have an ID selector which declares `color` as *blue*, and a class selector which declares `color` as *red* and `background` as *black*.

An element with an ID of `#foo` and a class of `.bar` will be selected by both declarations. ID selectors have a Group *A* specificity and class selectors have a Group *B* specificity. An ID selector outweighs any number of class selectors. Because of this, **`color`**`:blue;` from the `#foo` selector and the **`background`**`:black;` from the `.bar` selector will be applied to the element. The higher specificity of the ID selector will cause the browser to ignore the `.bar` selector's `color` declaration.

Now imagine a different CSS implementation:

```
.bar {
  color: red;
  background: black;
}

.baz {
  background: white;
}
```

Here we have two class selectors; one of which declares `color` as *red* and `background` as *black*, and the other declares `background` as *white*.

An element with both the `.bar` and `.baz` classes will be affected by both of these declarations, however the problem we have now is that both `.bar` and `.baz` have an identical Group *B* specificity. The cascading nature of CSS resolves this for us: as `.baz` is defined *after* `.bar`, our element ends up with the *red* `color` from `.bar` but the *white* `background` from `.baz`.

Example 3: How to manipulate specificity

The last snippet from Example 2 above can be manipulated to ensure our `.bar` class selector's `color` declaration is used instead of that of the `.baz` class selector.

```
.bar {}         /* a=0, b=1, c=0 */
.baz {}         /* a=0, b=1, c=0 */
```

The most common way to achieve this would be to find out what other selectors can be applied to the `.bar` selector sequence. For example, if the `.bar` class was only ever applied to `span` elements, we could modify the `.bar` selector to `span.bar`. This would give it a new Group *C* specificity, which would override the `.baz` selector's lack thereof:

```
span.bar {}     /* a=0, b=1, c=1 */
.baz {}         /* a=0, b=1, c=0 */
```

However it may not always possible to find another common selector which is shared between any element which uses the `.bar` class. Because of this, CSS allows us to duplicate selectors to increase specificity. Instead of just `.bar`, we can use `.bar.bar` instead (See The grammar of Selectors, W3C Recommendation). This still selects any element with a class of `.bar`, but now has double the Group *B* specificity:

```
.bar.bar {}     /* a=0, b=2, c=0 */
.baz {}         /* a=0, b=1, c=0 */
```

!important和内联样式声明

样式声明中的 !important 标志以及由 HTML style 属性声明的样式被视为具有比任何选择器更高的特异性。如果存在这些样式声明，它们所影响的样式声明将无视其他声明的特异性而优先应用。也就是说，除非你对同一元素的同一属性有多个包含 !important 标志的声明。此时，这些属性之间将按照正常的特异性规则进行比较。

因为它们完全覆盖了特异性，大多数情况下不建议使用!important。应该尽量少用它。为了保持CSS代码的高效和可维护性，通常最好提高周围选择器的特异性，而不是使用!important。

少数不反对使用!important的例外情况之一，是实现通用辅助类时，比如.hidden或.background-yellow类，这些类无论出现在哪里都应该始终覆盖一个或多个属性。即便如此，你也需要知道自己在做什么。编写可维护的CSS时，最不希望看到的是CSS中到处都是!important标记。

最后一点说明

关于CSS特异性的一个常见误解是，组A、B和C的值应该相互组合（例如 a=1， b=5， c=1 => 151）。事实并非如此。如果是这样，拥有20个组B或C选择器就足以覆盖单个组A或B选择器。三个组应被视为特异性的不同层级。特异性不能用单一数值表示。

在创建CSS样式表时，应保持尽可能低的特异性。如果需要稍微提高特异性以覆盖另一种方法，应提高但尽量保持最低。你不应该使用像这样的选择器：

```
body.page header.container nav div#main-nav li a {}
```

这会使未来的修改变得更困难，并且污染该CSS页面。

你可以在那里计算选择器的特异性

第17.2节：!important声明

!important声明用于通过赋予规则更高优先级来覆盖样式表中的常规特异性。其用法是：property ： value !important;

```
#mydiv {
    font-weight: bold !important;     /* 该属性不会被下面的规则覆盖
                                      */
}

#outerdiv #mydiv {
    font-weight: normal;              /* 即使 #mydiv 具有更高的特异性，
          由于上面的 !important 声明，font-weight 也不会被设置为 normal */

}
```

强烈建议避免使用!important（除非绝对必要），因为它会干扰 CSS 规则的自然流动，可能导致样式表的不确定性。同时需要注意的是，当多个!important声明应用于某个元素的同一规则时，具有更高特异性的声明将被应用。

`!important` and inline style declarations

The `!important` flag on a style declaration and styles declared by the HTML `style` attribute are considered to have a greater specificity than any selector. If these exist, the style declaration they affect will overrule other declarations regardless of their specificity. That is, unless you have more than one declaration that contains an `!important` flag for the same property that apply to the same element. Then, normal specificity rules will apply to those properties in reference to each other.

Because they completely override specificity, the use of `!important` is frowned upon in most use cases. One should use it as little as possible. To keep CSS code efficient and maintainable in the long run, it's almost always better to increase the specificity of the surrounding selector than to use `!important`.

One of those rare exceptions where `!important` is not frowned upon, is when implementing generic helper classes like a `.hidden` or `.background-yellow` class that are supposed to always override one or more properties wherever they are encountered. And even then, you need to know what you're doing. The last thing you want, when writing maintainable CSS, is to have `!important` flags throughout your CSS.

A final note

A common misconception about CSS specificity is that the Group *A*, *B* and *c* values should be combined with each other (a=1, b=5, c=1 => 151). This is **not** the case. If this were the case, having 20 of a Group *B* or *c* selector would be enough to override a single Group *A* or *B* selector respectively. The three groups should be regarded as individual levels of specificity. Specificity cannot be represented by a single value.

When creating your CSS style sheet, you should maintain the lowest specificity as possible. If you need to make the specificity a little higher to overwrite another method, make it higher but as low as possible to make it higher. You shouldn't need to have a selector like this:

```
body.page header.container nav div#main-nav li a {}
```

This makes future changes harder and pollutes that css page.

You can calculate the specificity of your selector here

Section 17.2: The !important declaration

The `!important` declaration is used to override the usual specificity in a style sheet by giving a higher priority to a rule. Its usage is: `property : value !important;`

```
#mydiv {
    font-weight: bold !important;    /* This property won't be overridden
                                        by the rule below */
}

#outerdiv #mydiv {
    font-weight: normal;             /* #mydiv font-weight won't be set to normal
                                     even if it has a higher specificity because
                                     of the !important declaration above */
}
```

Avoiding the usage of `!important` is strongly recommended (unless absolutely necessary), because it will disturb the natural flow of css rules which can bring uncertainty in your style sheet. Also it is important to note that when multiple `!important` declarations are applied to the same rule on a certain element, the one with the higher specificity will be the ona applied.

以下是一些使用!important声明可以被合理化的例子：

- 如果你的规则不应被写在 HTML 元素的style属性中的任何内联样式覆盖。
- 为了让用户更好地控制网页的无障碍性，比如通过覆盖作者样式使用!important来增加或减少字体大小。
- 用于测试和调试时使用检查元素功能。

另见：

- W3C - 6 属性值赋值、层叠和继承 -- 6.4.2 !important 规则

第17.3节：级联

层叠性和特异性共同用于确定CSS样式属性的最终值。它们还定义了CSS规则集冲突解决的机制。

CSS加载顺序

样式按以下来源顺序读取：

1. 用户代理样式表（浏览器供应商提供的样式）
2. 用户样式表（用户在其浏览器中设置的额外样式）
3. 作者样式表（这里的作者指网页/网站的创建者）
    - 可能包含一个或多个.css文件
    - 在HTML文档的**<style>**元素中
4. 内联样式（在HTML元素的style属性中）

浏览器在渲染元素时会查找相应的样式。

冲突如何解决？

当只有一个CSS规则集尝试为元素设置样式时，不存在冲突，直接使用该规则集。

当发现多个规则集存在冲突设置时，首先使用特异性规则，然后使用层叠规则来确定使用哪种样式。

示例 1 - 特异性规则

```
.mystyle { color: blue; }  /* 特异性：0, 0, 1, 0 */
div { color: red; }        /* 特异性：0, 0, 0, 1 */
```

```
<div class="mystyle">Hello World</div>
```

文本颜色会是什么？（悬停查看答案）

> 蓝色

首先应用特异性规则，特异性最高的规则“获胜”。

示例 2 - 具有相同选择器的层叠规则

外部 CSS 文件

```
.class {
```


Here are some examples where using `!important` declaration can be justified:

- If your rules shouldn't be overridden by any inline style of the element which is written inside `style` attribute of the html element.
- To give the user more control over the web accessibility, like increasing or decreasing size of the font-size, by overriding the author style using `!important`.
- For testing and debugging using inspect element.

See also:

- W3C - 6 Assigning property values, Cascading, and Inheritance -- 6.4.2 !important rules

Section 17.3: Cascading

Cascading and specificity are used together to determine the final value of a CSS styling property. They also define the mechanisms for resolving conflicts in CSS rule sets.

CSS Loading order

Styles are read from the following sources, in this order:

1. User Agent stylesheet (The styles supplied by the browser vendor)
2. User stylesheet (The additional styling a user has set on his/her browser)
3. Author stylesheet (Author here means the creator of the webpage/website)
    - Maybe one or more `.css` files
    - In the **<style>** element of the HTML document
4. Inline styles (In the `style` attribute on an HTML element)

The browser will lookup the corresponding style(s) when rendering an element.

How are conflicts resolved?

When only one CSS rule set is trying to set a style for an element, then there is no conflict, and that rule set is used.

When multiple rule sets are found with conflicting settings, first the Specificty rules, and then the Cascading rules are used to determine what style to use.

Example 1 - Specificity rules

```
.mystyle { color: blue; }  /* specificity: 0, 0, 1, 0 */
div { color: red; }        /* specificity: 0, 0, 0, 1 */
```

```
<div class="mystyle">Hello World</div>
```

What color will the text be? (hover to see the answer)

> blue

First the specificity rules are applied, and the one with the highest specificity "wins".

Example 2 - Cascade rules with identical selectors

External css file

```
.class {
```

```
  background: #FFF;
}
```

内部 CSS（在 HTML 文件中）

```
<style>
.class {
background: #000;
}
<style>
```

在这种情况下，当选择器相同时，层叠规则生效，决定最后加载的那个“获胜”。

示例3 - 特异性规则之后的层叠规则

```
body > .mystyle { background-color: blue; }   /* 特异性：0, 0, 1, 1 */
.otherstyle > div { background-color: red; }  /* 特异性：0, 0, 1, 1 */
```

```
<body class="otherstyle">
  <div class="mystyle">Hello World</div>
</body>
```

背景颜色会是什么？

> 红色

应用特异性规则后，蓝色和红色之间仍存在冲突，因此在特异性规则之上应用层叠规则。层叠规则会查看规则的加载顺序，无论是在同一个.css文件内还是在多个样式源的集合中。最后加载的规则会覆盖之前的规则。在本例中，.otherstyle > div规则“获胜”。

最后一点说明

- 选择器特异性始终优先。
- 样式表顺序决定平局。
- 内联样式优先级最高。

第17.4节：更复杂的特异性示例

```
div {
    字体大小: 7像素;
    边框: 3像素点状 粉色;
    背景色: 黄色;
    颜色: 紫色;
}

body.mystyle > div.myotherstyle {
    字体大小: 11像素;
    背景色: 绿色;
}

#elmnt1 {
    字体大小: 24像素;
    边框颜色: 红色;
}
```



```
  background: #FFF;
}
```

Internal css (in HTML file)

```
<style>
.class {
  background: #000;
}
<style>
```

In this case, where you have identical selectors, the cascade kicks in, and determines that the last one loaded "wins".

Example 3 - Cascade rules after Specificity rules

```
body > .mystyle { background-color: blue; }   /* specificity: 0, 0, 1, 1 */
.otherstyle > div { background-color: red; }  /* specificity: 0, 0, 1, 1 */
```

```
<body class="otherstyle">
  <div class="mystyle">Hello World</div>
</body>
```

What color will the background be?

> red

After applying the specificity rules, there's still a conflict between blue and red, so the cascading rules are applied on top of the specificity rules. Cascading looks at the load order of the rules, whether inside the same `.css` file or in the collection of style sources. The last one loaded overrides any earlier ones. In this case, the `.otherstyle > div` rule "wins".

A final note

- Selector specificity always take precedence.
- Stylesheet order break ties.
- Inline styles trump everything.

Section 17.4: More complex specificity example

```
div {
    font-size: 7px;
    border: 3px dotted pink;
    background-color: yellow;
    color: purple;
}

body.mystyle > div.myotherstyle {
    font-size: 11px;
    background-color: green;
}

#elmnt1 {
    font-size: 24px;
    border-color: red;
}
```

```
.mystyle .myotherstyle {
    字体大小: 16像素;
    背景色: 黑色;
    颜色: 红色;
}
```

```
<body class="mystyle">
    <div id="elmnt1" class="myotherstyle">
        你好，世界！
    </div>
</body>
```

文本的边框、颜色和字体大小将是什么？

字体大小：

> 字体大小: 24;由于#elmnt1规则集对所讨论的 具有最高的特异性，这里的每个属性都被设置了。

边框：

> 边框: 3px 点状 红色;边框颜色红色取自#elmnt1规则集，因为它具有最高的特异性。边框的其他属性，边框厚度和边框样式来自div规则集。

背景颜色：

> 背景颜色: 绿色;背景颜色在div、body.mystyle > div.myotherstyle和.mystyle .myotherstyle规则集中设置。特异性分别为(0, 0, 1)、(0, 2, 2)和(0, 2, 0)，因此中间那个“获胜”。

颜色：

> 颜色: 红色;颜色在div和.mystyle .myotherstyle规则集中设置。后者具有更高的特异性(0, 2, 0)，因此“获胜”。


```
.mystyle .myotherstyle {
    font-size: 16px;
    background-color: black;
    color: red;
}
```

```
<body class="mystyle">
    <div id="elmnt1" class="myotherstyle">
        Hello, world!
    </div>
</body>
```

What borders, colors, and font-sizes will the text be?

font-size:

> **font-size**: 24;, since #elmnt1 rule set has the highest specificity for the ** ** in question, every property here is set.

border:

> **border**: 3px dotted red;. The border-color red is taken from #elmnt1 rule set, since it has the highest specificity. The other properties of the border, border-thickness, and border-style are from the div rule set.

background-color:

> **background-color**: green;. The background-color is set in the div, body.mystyle > div.myotherstyle, and .mystyle .myotherstyle rule sets. The specificities are (0, 0, 1) vs. (0, 2, 2) vs. (0, 2, 0), so the middle one "wins".

color:

> **color**: red;. The color is set in both the div and .mystyle .myotherstyle rule sets. The latter has the higher specificity of (0, 2, 0) and "wins".

第18章：颜色

第18.1节：currentColor

currentColor 返回当前元素的计算颜色值。

在同一元素中使用

这里 currentColor 计算为红色，因为color属性被设置为red：

```
div {
    color: red;
    border: 5px 实线 currentColor;
    box-shadow: 0 0 5px currentColor;
}
```

在这种情况下，为边框指定 currentColor 很可能是多余的，因为省略它应该产生相同的结果。只有在同一元素内的边框属性中使用 currentColor，且否则会被更具体的选择器覆盖时，才使用它。

由于它是计算后的颜色，以下示例中边框将是绿色，因为第二条规则覆盖了第一条规则：

```
div {
    color: blue;
    border: 3px 实线 currentColor;
    color: 绿色;
}
```

继承自父元素

父元素的颜色被继承，这里 currentColor 计算为“蓝色”，使子元素的边框颜色为蓝色。

```
.parent-class {
    color: blue;
}

.parent-class .child-class {
    border-color: currentColor;
}
```

currentColor 也可以被其他通常不会继承 color 属性的规则使用，例如 background-color。下面的例子展示了子元素使用父元素设置的颜色作为背景色：

```
.parent-class {
    color: blue;
}

.parent-class .child-class {
    background-color: currentColor;
}
```

可能的结果：

Chapter 18: Colors

Section 18.1: currentColor

`currentColor` returns the computed color value of the current element.

Use in same element

Here currentColor evaluates to red since the `color` property is set to `red`:

```
div {
   color: red;
   border: 5px solid currentColor;
   box-shadow: 0 0 5px currentColor;
}
```

In this case, specifying currentColor for the border is most likely redundant because omitting it should produce identical results. Only use currentColor inside the border property within the same element if it would be overwritten otherwise due to a more specific selector.

Since it's the computed color, the border will be green in the following example due to the second rule overriding the first:

```
div {
   color: blue;
   border: 3px solid currentColor;
   color: green;
}
```

Inherited from parent element

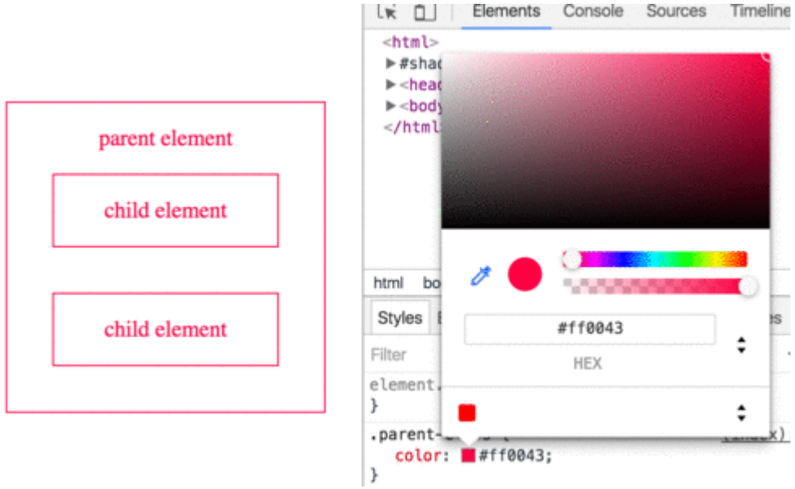
The parent's color is inherited, here currentColor evaluates to 'blue', making the child element's border-color blue.

```
.parent-class {
   color: blue;
}

.parent-class .child-class {
   border-color: currentColor;
}
```

currentColor can also be used by other rules which normally would not inherit from the color property, such as background-color. The example below shows the children using the color set in the parent as its background:

```
.parent-class {
   color: blue;
}

.parent-class .child-class {
   background-color: currentColor;
}
```

Possible Result:



第18.2节：颜色关键字

大多数浏览器支持使用颜色关键字来指定颜色。例如，要将元素的color设置为蓝色，使用blue关键字：

```
.some-class {
    color: 蓝色;
}
```

CSS 关键字不区分大小写—blue、Blue 和 BLUE 都会得到 #0000FF。

颜色关键字

颜色关键字

颜色名称	十六进制值	RGB 值	颜色
爱丽丝蓝	#F0F8FF	rgb(240,248,255)	
古董白	#FAEBD7	rgb(250,235,215)	
水	#00FFFF	rgb(0,255,255)	
碧绿色	#7FFFD4	rgb(127,255,212)	
蔚蓝色	#F0FFFF	rgb(240,255,255)	
米色	#F5F5DC	rgb(245,245,220)	
淡橙色	#FFE4C4	rgb(255,228,196)	
黑色	#000000	rgb(0,0,0)	
漂白杏仁色	#FFEBCD	rgb(255,235,205)	
蓝色	#0000FF	rgb(0,0,255)	
蓝紫色	#8A2BE2	rgb(138,43,226)	
布朗	#A52A2A	rgb(165,42,42)	

Section 18.2: Color Keywords

Most browsers support using color keywords to specify a color. For example, to set the color of an element to blue, use the blue keyword:

```
.some-class {
    color: blue;
}
```

CSS keywords are not case sensitive—blue, Blue and BLUE will all result in #0000FF.

Color Keywords

Color Keywords

Color name	Hex value	RGB values	Color
AliceBlue	#F0F8FF	rgb(240,248,255)	
AntiqueWhite	#FAEBD7	rgb(250,235,215)	
Aqua	#00FFFF	rgb(0,255,255)	
Aquamarine	#7FFFD4	rgb(127,255,212)	
Azure	#F0FFFF	rgb(240,255,255)	
Beige	#F5F5DC	rgb(245,245,220)	
Bisque	#FFE4C4	rgb(255,228,196)	
Black	#000000	rgb(0,0,0)	
BlanchedAlmond	#FFEBCD	rgb(255,235,205)	
Blue	#0000FF	rgb(0,0,255)	
BlueViolet	#8A2BE2	rgb(138,43,226)	
Brown	#A52A2A	rgb(165,42,42)	

硬木色	#DEB887	rgb(222,184,135)
军校蓝	#5F9EA0	rgb(95,158,160)
查特酒绿	#7FFF00	rgb(127,255,0)
巧克力色	#D2691E	rgb(210,105,30)
珊瑚色	#FF7F50	rgb(255,127,80)
矢车菊蓝	#6495ED	rgb(100,149,237)
玉米丝色	#FFF8DC	rgb(255,248,220)
深红色	#DC143C	rgb(220,20,60)
青色	#00FFFF	rgb(0,255,255)
深蓝色	#00008B	rgb(0,0,139)
深青色	#008B8B	rgb(0,139,139)
深金黄色	#B8860B	rgb(184,134,11)
深灰色	#A9A9A9	rgb(169,169,169)
深灰色	#A9A9A9	rgb(169,169,169)
深绿色	#006400	rgb(0,100,0)
深卡其布色	#BDB76B	rgb(189,183,107)
深洋红色	#8B008B	rgb(139,0,139)
深橄榄绿	#556B2F	rgb(85,107,47)
深橙色	#FF8C00	rgb(255,140,0)
深兰花紫	#9932CC	rgb(153,50,204)
深红色	#8B0000	rgb(139,0,0)
深鲑鱼色	#E9967A	rgb(233,150,122)
深海绿色	#8FBC8F	rgb(143,188,143)
深石板蓝	#483D8B	rgb(72,61,139)
深石板灰	#2F4F4F	rgb(47,79,79)
暗石板灰	#2F4F4F	rgb(47,79,79)
暗青色	#00CED1	rgb(0,206,209)
暗紫罗兰色	#9400D3	rgb(148,0,211)

BurlyWood	#DEB887	rgb(222,184,135)
CadetBlue	#5F9EA0	rgb(95,158,160)
Chartreuse	#7FFF00	rgb(127,255,0)
Chocolate	#D2691E	rgb(210,105,30)
Coral	#FF7F50	rgb(255,127,80)
CornflowerBlue	#6495ED	rgb(100,149,237)
Cornsilk	#FFF8DC	rgb(255,248,220)
Crimson	#DC143C	rgb(220,20,60)
Cyan	#00FFFF	rgb(0,255,255)
DarkBlue	#00008B	rgb(0,0,139)
DarkCyan	#008B8B	rgb(0,139,139)
DarkGoldenRod	#B8860B	rgb(184,134,11)
DarkGray	#A9A9A9	rgb(169,169,169)
DarkGrey	#A9A9A9	rgb(169,169,169)
DarkGreen	#006400	rgb(0,100,0)
DarkKhaki	#BDB76B	rgb(189,183,107)
DarkMagenta	#8B008B	rgb(139,0,139)
DarkOliveGreen	#556B2F	rgb(85,107,47)
DarkOrange	#FF8C00	rgb(255,140,0)
DarkOrchid	#9932CC	rgb(153,50,204)
DarkRed	#8B0000	rgb(139,0,0)
DarkSalmon	#E9967A	rgb(233,150,122)
DarkSeaGreen	#8FBC8F	rgb(143,188,143)
DarkSlateBlue	#483D8B	rgb(72,61,139)
DarkSlateGray	#2F4F4F	rgb(47,79,79)
DarkSlateGrey	#2F4F4F	rgb(47,79,79)
DarkTurquoise	#00CED1	rgb(0,206,209)
DarkViolet	#9400D3	rgb(148,0,211)

深粉色	#FF1493	rgb(255,20,147)
深天蓝色	#00BFFF	rgb(0,191,255)
暗灰色	#696969	rgb(105,105,105)
暗灰色	#696969	rgb(105,105,105)
道奇蓝	#1E90FF	rgb(30,144,255)
火砖红	#B22222	rgb(178,34,34)
花卉白	#FFFAF0	rgb(255,250,240)
森林绿	#228B22	rgb(34,139,34)
紫红色	#FF00FF	rgb(255,0,255)
淡灰色	#DCDCDC	rgb(220,220,220)
幽灵白	#F8F8FF	rgb(248,248,255)
黄金	#FFD700	rgb(255,215,0)
金棒色	#DAA520	rgb(218,165,32)
灰色	#808080	rgb(128,128,128)
灰色	#808080	rgb(128,128,128)
绿色	#008000	rgb(0,128,0)
黄绿色	#ADFF2F	rgb(173,255,47)
蜜露绿	#F0FFF0	rgb(240,255,240)
热粉色	#FF69B4	rgb(255,105,180)
印度红	#CD5C5C	rgb(205,92,92)
靛青色	#4B0082	rgb(75,0,130)
象牙色	#FFFFF0	rgb(255,255,240)
卡其色	#F0E68C	rgb(240,230,140)
薰衣草色	#E6E6FA	rgb(230,230,250)
薰衣草红	#FFF0F5	rgb(255,240,245)
草坪绿	#7CFC00	rgb(124,252,0)
柠檬绸	#FFFACD	rgb(255,250,205)
浅蓝色	#ADD8E6	rgb(173,216,230)

DeepPink	#FF1493	rgb(255,20,147)
DeepSkyBlue	#00BFFF	rgb(0,191,255)
DimGray	#696969	rgb(105,105,105)
DimGrey	#696969	rgb(105,105,105)
DodgerBlue	#1E90FF	rgb(30,144,255)
FireBrick	#B22222	rgb(178,34,34)
FloralWhite	#FFFAF0	rgb(255,250,240)
ForestGreen	#228B22	rgb(34,139,34)
Fuchsia	#FF00FF	rgb(255,0,255)
Gainsboro	#DCDCDC	rgb(220,220,220)
GhostWhite	#F8F8FF	rgb(248,248,255)
Gold	#FFD700	rgb(255,215,0)
GoldenRod	#DAA520	rgb(218,165,32)
Gray	#808080	rgb(128,128,128)
Grey	#808080	rgb(128,128,128)
Green	#008000	rgb(0,128,0)
GreenYellow	#ADFF2F	rgb(173,255,47)
HoneyDew	#F0FFF0	rgb(240,255,240)
HotPink	#FF69B4	rgb(255,105,180)
IndianRed	#CD5C5C	rgb(205,92,92)
Indigo	#4B0082	rgb(75,0,130)
Ivory	#FFFFF0	rgb(255,255,240)
Khaki	#F0E68C	rgb(240,230,140)
Lavender	#E6E6FA	rgb(230,230,250)
LavenderBlush	#FFF0F5	rgb(255,240,245)
LawnGreen	#7CFC00	rgb(124,252,0)
LemonChiffon	#FFFACD	rgb(255,250,205)
LightBlue	#ADD8E6	rgb(173,216,230)

浅珊瑚色	#F08080	rgb(240,128,128)
浅青色	#E0FFFF	rgb(224,255,255)
浅金菊黄	#FAFAD2	rgb(250,250,210)
浅灰色	#D3D3D3	rgb(211,211,211)
浅灰色	#D3D3D3	rgb(211,211,211)
浅绿色	#90EE90	rgb(144,238,144)
浅粉色	#FFB6C1	rgb(255,182,193)
浅鲑鱼色	#FFA07A	rgb(255,160,122)
浅海绿色	#20B2AA	rgb(32,178,170)
浅天蓝	#87CEFA	rgb(135,206,250)
浅石板灰	#778899	rgb(119,136,153)
浅石板灰	#778899	rgb(119,136,153)
浅钢蓝	#B0C4DE	rgb(176,196,222)
浅黄色	#FFFFE0	rgb(255,255,224)
石灰	#00FF00	rgb(0,255,0)
酸橙绿	#32CD32	rgb(50,205,50)
亚麻色	#FAF0E6	rgb(250,240,230)
品红色	#FF00FF	rgb(255,0,255)
栗色	#800000	rgb(128,0,0)
中等碧绿色	#66CDAA	rgb(102,205,170)
中蓝色	#0000CD	rgb(0,0,205)
中兰花紫	#BA55D3	rgb(186,85,211)
中紫色	#9370DB	rgb(147,112,219)
中海洋绿	#3CB371	rgb(60,179,113)
中岩蓝	#7B68EE	rgb(123,104,238)
中春绿色	#00FA9A	rgb(0,250,154)
中绿松石色	#48D1CC	rgb(72,209,204)
中紫罗兰红	#C71585	rgb(199,21,133)

LightCoral	#F08080	rgb(240,128,128)
LightCyan	#E0FFFF	rgb(224,255,255)
LightGoldenRodYellow	#FAFAD2	rgb(250,250,210)
LightGray	#D3D3D3	rgb(211,211,211)
LightGrey	#D3D3D3	rgb(211,211,211)
LightGreen	#90EE90	rgb(144,238,144)
LightPink	#FFB6C1	rgb(255,182,193)
LightSalmon	#FFA07A	rgb(255,160,122)
LightSeaGreen	#20B2AA	rgb(32,178,170)
LightSkyBlue	#87CEFA	rgb(135,206,250)
LightSlateGray	#778899	rgb(119,136,153)
LightSlateGrey	#778899	rgb(119,136,153)
LightSteelBlue	#B0C4DE	rgb(176,196,222)
LightYellow	#FFFFE0	rgb(255,255,224)
Lime	#00FF00	rgb(0,255,0)
LimeGreen	#32CD32	rgb(50,205,50)
Linen	#FAF0E6	rgb(250,240,230)
Magenta	#FF00FF	rgb(255,0,255)
Maroon	#800000	rgb(128,0,0)
MediumAquaMarine	#66CDAA	rgb(102,205,170)
MediumBlue	#0000CD	rgb(0,0,205)
MediumOrchid	#BA55D3	rgb(186,85,211)
MediumPurple	#9370DB	rgb(147,112,219)
MediumSeaGreen	#3CB371	rgb(60,179,113)
MediumSlateBlue	#7B68EE	rgb(123,104,238)
MediumSpringGreen	#00FA9A	rgb(0,250,154)
MediumTurquoise	#48D1CC	rgb(72,209,204)
MediumVioletRed	#C71585	rgb(199,21,133)

午夜蓝	#191970	rgb(25,25,112)
薄荷奶油色	#F5FFFA	rgb(245,255,250)
薄雾玫瑰	#FFE4E1	rgb(255,228,225)
鹿皮色	#FFE4B5	rgb(255,228,181)
纳瓦霍白	#FFDEAD	rgb(255,222,173)
海军蓝	#000080	rgb(0,0,128)
老花边色	#FDF5E6	rgb(253,245,230)
橄榄色	#808000	rgb(128,128,0)
橄榄褐	#6B8E23	rgb(107,142,35)
橙色	#FFA500	rgb(255,165,0)
橙红色	#FF4500	rgb(255,69,0)
兰花紫	#DA70D6	rgb(218,112,214)
苍金麒麟	#EEE8AA	rgb(238,232,170)
浅绿色	#98FB98	rgb(152,251,152)
浅绿松石色	#AFEEEE	rgb(175,238,238)
浅紫罗兰红	#DB7093	rgb(219,112,147)
番木瓜奶油色	#FFEFD5	rgb(255,239,213)
桃色泡芙	#FFDAB9	rgb(255,218,185)
秘鲁	#CD853F	rgb(205,133,63)
粉红色	#FFC0CB	rgb(255,192,203)
李子色	#DDA0DD	rgb(221,160,221)
粉蓝色	#B0E0E6	rgb(176,224,230)
紫色	#800080	rgb(128,0,128)
丽贝卡紫	#663399	rgb(102,51,153)
红色	#FF0000	rgb(255,0,0)
玫瑰棕	#BC8F8F	rgb(188,143,143)
皇家蓝	#4169E1	rgb(65,105,225)
马鞍棕	#8B4513	rgb(139,69,19)

MidnightBlue	#191970	rgb(25,25,112)
MintCream	#F5FFFA	rgb(245,255,250)
MistyRose	#FFE4E1	rgb(255,228,225)
Moccasin	#FFE4B5	rgb(255,228,181)
NavajoWhite	#FFDEAD	rgb(255,222,173)
Navy	#000080	rgb(0,0,128)
OldLace	#FDF5E6	rgb(253,245,230)
Olive	#808000	rgb(128,128,0)
OliveDrab	#6B8E23	rgb(107,142,35)
Orange	#FFA500	rgb(255,165,0)
OrangeRed	#FF4500	rgb(255,69,0)
Orchid	#DA70D6	rgb(218,112,214)
PaleGoldenRod	#EEE8AA	rgb(238,232,170)
PaleGreen	#98FB98	rgb(152,251,152)
PaleTurquoise	#AFEEEE	rgb(175,238,238)
PaleVioletRed	#DB7093	rgb(219,112,147)
PapayaWhip	#FFEFD5	rgb(255,239,213)
PeachPuff	#FFDAB9	rgb(255,218,185)
Peru	#CD853F	rgb(205,133,63)
Pink	#FFC0CB	rgb(255,192,203)
Plum	#DDA0DD	rgb(221,160,221)
PowderBlue	#B0E0E6	rgb(176,224,230)
Purple	#800080	rgb(128,0,128)
RebeccaPurple	#663399	rgb(102,51,153)
Red	#FF0000	rgb(255,0,0)
RosyBrown	#BC8F8F	rgb(188,143,143)
RoyalBlue	#4169E1	rgb(65,105,225)
SaddleBrown	#8B4513	rgb(139,69,19)

鲑鱼	#FA8072	rgb(250,128,114)
沙棕色	#F4A460	rgb(244,164,96)
海洋绿	#2E8B57	rgb(46,139,87)
海贝壳色	#FFF5EE	rgb(255,245,238)
赭色	#A0522D	rgb(160,82,45)
银色	#C0C0C0	rgb(192,192,192)
天蓝色	#87CEEB	rgb(135,206,235)
石板蓝	#6A5ACD	rgb(106,90,205)
石板灰	#708090	rgb(112,128,144)
石板灰	#708090	rgb(112,128,144)
雪	#FFFAFA	rgb(255,250,250)
春绿色	#00FF7F	rgb(0,255,127)
钢蓝色	#4682B4	rgb(70,130,180)
茶色	#D2B48C	rgb(210,180,140)
水鸭色	#008080	rgb(0,128,128)
蓟色	#D8BFD8	rgb(216,191,216)
番茄红	#FF6347	rgb(255,99,71)
绿松石色	#40E0D0	rgb(64,224,208)
紫罗兰色	#EE82EE	rgb(238,130,238)
小麦色	#F5DEB3	rgb(245,222,179)
白色	#FFFFFF	rgb(255,255,255)
白烟色	#F5F5F5	rgb(245,245,245)
黄色	#FFFF00	rgb(255,255,0)
黄绿色	#9ACD32	rgb(154,205,50)

除了命名颜色外，还有关键字transparent，表示完全透明的黑色：

```
rgba(0,0,0,0)
```


Salmon	#FA8072	rgb(250,128,114)
SandyBrown	#F4A460	rgb(244,164,96)
SeaGreen	#2E8B57	rgb(46,139,87)
SeaShell	#FFF5EE	rgb(255,245,238)
Sienna	#A0522D	rgb(160,82,45)
Silver	#C0C0C0	rgb(192,192,192)
SkyBlue	#87CEEB	rgb(135,206,235)
SlateBlue	#6A5ACD	rgb(106,90,205)
SlateGray	#708090	rgb(112,128,144)
SlateGrey	#708090	rgb(112,128,144)
Snow	#FFFAFA	rgb(255,250,250)
SpringGreen	#00FF7F	rgb(0,255,127)
SteelBlue	#4682B4	rgb(70,130,180)
Tan	#D2B48C	rgb(210,180,140)
Teal	#008080	rgb(0,128,128)
Thistle	#D8BFD8	rgb(216,191,216)
Tomato	#FF6347	rgb(255,99,71)
Turquoise	#40E0D0	rgb(64,224,208)
Violet	#EE82EE	rgb(238,130,238)
Wheat	#F5DEB3	rgb(245,222,179)
White	#FFFFFF	rgb(255,255,255)
WhiteSmoke	#F5F5F5	rgb(245,245,245)
Yellow	#FFFF00	rgb(255,255,0)
YellowGreen	#9ACD32	rgb(154,205,50)

In addition to the named colors, there is also the keyword `transparent`, which represents a fully-transparent black:

```
rgba(0,0,0,0)
```

第18.3节：十六进制值

背景

CSS颜色也可以表示为十六进制三元组，其中各成员代表颜色的红色、绿色和蓝色分量。每个值表示一个范围在`00`到`FF`之间的数字，或十进制表示的`0`到`255`。十六进制值可以使用大写和/或小写（例如`#3fc`= `#3FC`= `#33ffCC`）。浏览器会将`#369`解释为`#336699`。如果这不是你想要的，而是想要`#306090`，

你需要明确指定。

使用十六进制表示的颜色总数为256的三次方，即16,777,216种。

语法

```
color: #rrggbb;
color: #rgb
```

值	描述
`rr`	`00`- FF表示红色的量
`gg`	`00`- FF表示绿色的量
`bb`	`00`- FF 表示蓝色的量

```
.some-class {
    /* 这相当于使用颜色关键字 'blue' */
    color: #0000FF;
}

.also-blue {
    /* 如果你想用单个数字指定每个范围值，也是可以的！
这相当于 '#0000FF'（和 'blue'）*/
    color: #00F;
}
```

十六进制表示法 用于按照 W3C 的“数值颜色

指“指定RGB颜色格式的颜色值。

互联网上有很多工具可以用来查找十六进制（或简称hex）颜色值。

使用您喜欢的网页浏览器搜索“十六进制颜色调色板”或“十六进制颜色选择器”，可以找到很多选项！

十六进制值总是以井号（#）开头，最多有六个“数字”，且不区分大小写：也就是说，它们不在乎大小写。#FFC125和#ffc125是相同的颜色。

第18.4节：rgb() 表示法

RGB是一种加色模型，用红色、绿色和蓝色光的混合来表示颜色。本质上，RGB表示法是十六进制表示法的十进制等价物。在十六进制中，每个数字范围是00-FF，相当于十进制的0-255以及百分比的0%-100%。

```
.some-class {
    /* 标量RGB，等同于“蓝色” */
    color: rgb(0, 0, 255);
}

.also-blue {
    /* 百分比RGB值 */
    color: rgb(0%, 0%, 100%);
}
```


Section 18.3: Hexadecimal Value

Background

CSS colors may also be represented as a hex triplet, where the members represent the red, green and blue components of a color. Each of these values represents a number in the range of `00` to `FF`, or `0` to `255` in decimal notation. Uppercase and/or lowercase Hexadecimal values may be used (i.e. `#3fc` = `#3FC` = `#33ffCC`). The browser interprets `#369` as `#336699`. If that is not what you intended but rather wanted `#306090`, you need to specify that explicitly.

The total number of colors that can be represented with hex notation is 256 ^ 3 or 16,777,216.

Syntax

```
color: #rrggbb;
color: #rgb
```

Value	Description
`rr`	`00` - FF for the amount of red
`gg`	`00` - FF for the amount of green
`bb`	`00` - FF for the amount of blue

```
.some-class {
    /* This is equivalent to using the color keyword 'blue' */
    color: #0000FF;
}

.also-blue {
    /* If you want to specify each range value with a single number, you can!
       This is equivalent to '#0000FF' (and 'blue') */
    color: #00F;
}
```

Hexadecimal notation is used to specify color values in the RGB color format, per the W3C's 'Numerical color values'.

There are a lot of tools available on the Internet for looking up hexadecimal (or simply hex) color values.

Search for "**hex color palette**" or "**hex color picker**" with your favorite web browser to find a bunch of options!

Hex values always start with a pound sign (#), are up to six "digits" long, and are case-insensitive: that is, they don't care about capitalization. `#FFC125` and `#ffc125` are the same color.

Section 18.4: rgb() Notation

RGB is an additive color model which represents colors as mixtures of red, green, and blue light. In essence, the RGB representation is the decimal equivalent of the Hexadecimal Notation. In Hexadecimal each number ranges from 00-FF which is equivalent to 0-255 in decimal and 0%-100% in percentages.

```
.some-class {
    /* Scalar RGB, equivalent to 'blue'*/
    color: rgb(0, 0, 255);
}

.also-blue {
    /* Percentile RGB values*/
    color: rgb(0%, 0%, 100%);
}
```

语法

```
rgb(<red>, <green>, <blue>)
```

值	描述
`<red>`	0 - 255之间的整数或0 - 100%的百分比
`<green>`	一个从0到255的整数或从0到100%的百分比
`<blue>`	一个从0到255的整数或从0到100%的百分比

第18.5节：rgba() 表示法

类似于 rgb() 表示法，但增加了一个 alpha（不透明度）值。

```
.red {
    /* 不透明的红色 */
    color: rgba(255, 0, 0, 1);
}

.red-50p {
    /* 半透明的红色。 */
    color: rgba(255, 0, 0, .5);
}
```

语法

```
rgba(<red>, <green>, <blue>, <alpha>);
```

值	描述
`<red>`	0 - 255之间的整数或0 - 100%的百分比
`<green>`	取值为0-255的整数或0-100%的百分比
`<blue>`	取值为0-255的整数或0-100%的百分比
`<alpha>`	取值为0-1的数字，其中0.0表示完全透明，1.0表示完全不透明

第18.6节：hsl() 表示法

HSL代表色相（hue，意为“哪种颜色”）、饱和度（saturation，意为“颜色的浓度”）和亮度（lightness，意为“白色的多少”）。

色相用0°到360°的角度表示（无单位），而饱和度和亮度用百分比表示。

```
p {
    color: hsl(240, 100%, 50%); /* 蓝色 */
}
```


Syntax

```
rgb(<red>, <green>, <blue>)
```

Value	Description
`<red>`	an integer from 0 - 255 or percentage from 0 - 100%
`<green>`	an integer from 0 - 255 or percentage from 0 - 100%
`<blue>`	an integer from 0 - 255 or percentage from 0 - 100%

Section 18.5: rgba() Notation

Similar to rgb() notation, but with an additional alpha (opacity) value.

```
.red {
    /* Opaque red */
    color: rgba(255, 0, 0, 1);
}

.red-50p {
    /* Half-translucent red. */
    color: rgba(255, 0, 0, .5);
}
```

Syntax

```
rgba(<red>, <green>, <blue>, <alpha>);
```

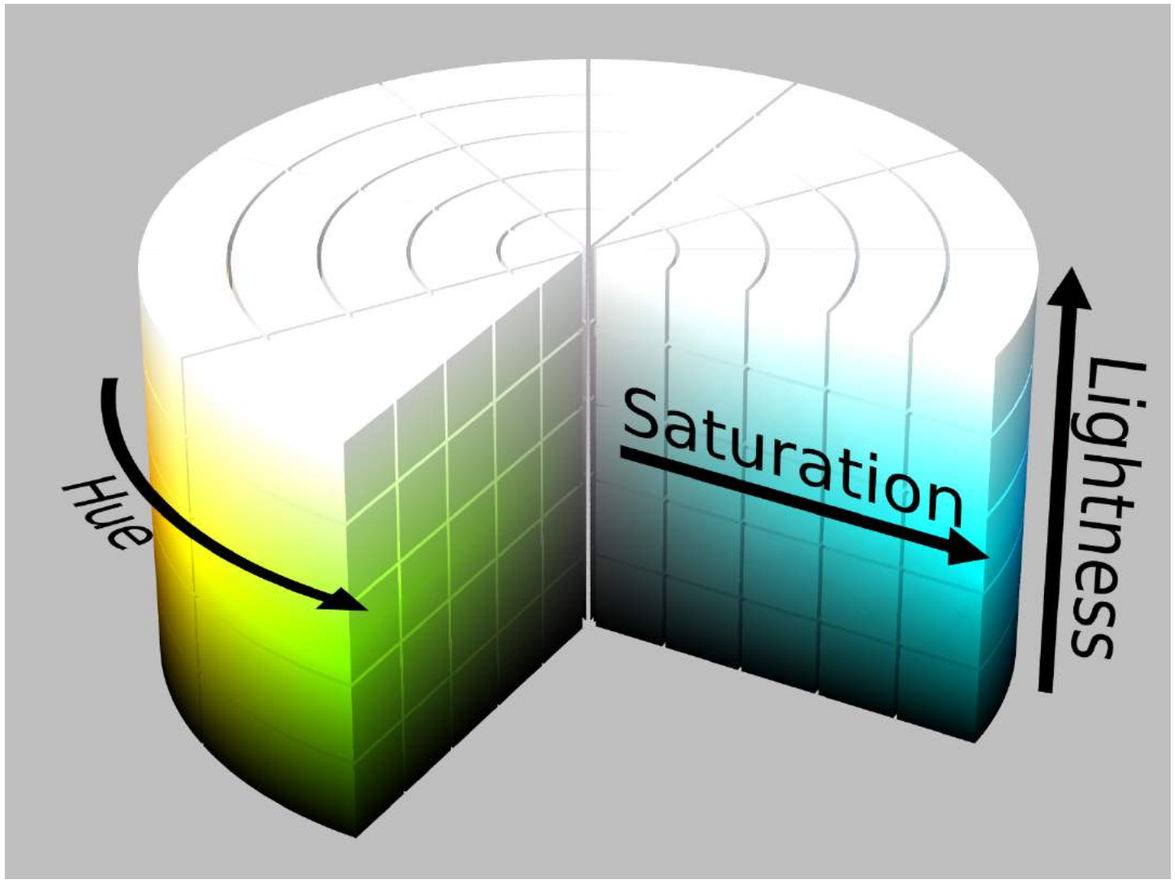
Value	Description
`<red>`	an integer from 0 - 255 or percentage from 0 - 100%
`<green>`	an integer from 0 - 255 or percentage from 0 - 100%
`<blue>`	an integer from 0 - 255 or percentage from 0 - 100%
`<alpha>`	a number from 0 - 1, where 0.0 is fully transparent and 1.0 is fully opaque

Section 18.6: hsl() Notation

HSL stands for **hue** ("which color"), **saturation** ("how much color") and **lightness** ("how much white").

Hue is represented as an angle from 0° to 360° (without units), while saturation and lightness are represented as percentages.

```
p {
    color: hsl(240, 100%, 50%); /* Blue */
}
```



语法

```
color: hsl(<hue>, <saturation>%, <lightness>%);
```

值	描述
`<hue>`	以色轮上的度数表示（无单位），其中0°为红色，60°为黄色，120°为绿色，180°为青色，240°为蓝色，300°为品红，360°为红色
	<saturation>以百分比表示，0%表示完全去饱和（灰度），100%表示完全饱和（鲜艳的颜色）
`<lightness>`	以百分比表示，0%表示完全黑色，100%表示完全白色

注意

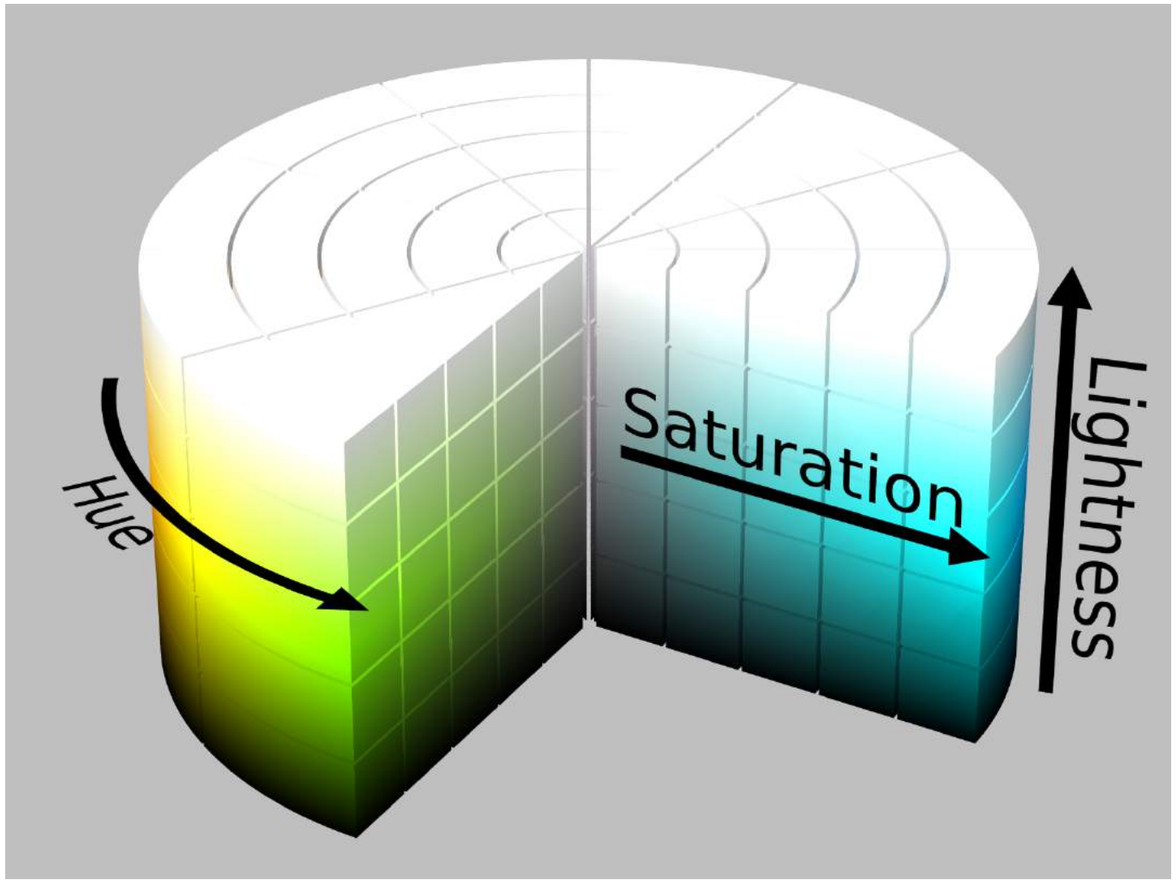
- 饱和度为0%时总是产生灰度颜色；改变色相不会有影响。
- 亮度为0%时总是产生黑色，亮度为100%时总是产生白色；改变色相或饱和度没有影响。

第18.7节：hsla() 表示法

类似于 hsl() 表示法，但增加了 alpha（不透明度）值。

```
hsla(240, 100%, 50%, 0)     /* 透明 */
hsla(240, 100%, 50%, 0.5)   /* 半透明蓝色 */
hsla(240, 100%, 50%, 1)     /* 完全不透明的蓝色 */
```

语法



Syntax

```
color: hsl(<hue>, <saturation>%, <lightness>%);
```

Value	Description
`<hue>`	specified in degrees around the color wheel (without units), where 0° is red, 60° is yellow, 120° is green, 180° is cyan, 240° is blue, 300° is magenta, and 360° is red
`<saturation>`	specified in percentage where 0% is fully desaturated (grayscale) and 100% is fully saturated (vividly colored)
`<lightness>`	specified in percentage where 0% is fully black and 100% is fully white

Notes

- A saturation of 0% always produces a grayscale color; changing the hue has no effect.
- A lightness of 0% always produces black, and 100% always produces white; changing the hue or saturation has no effect.

Section 18.7: hsla() Notation

Similar to hsl() notation, but with an added alpha (opacity) value.

```
hsla(240, 100%, 50%, 0)     /* transparent */
hsla(240, 100%, 50%, 0.5)   /* half-translucent blue */
hsla(240, 100%, 50%, 1)     /* fully opaque blue */
```

Syntax

```
hsla(<色相>, <饱和度>%, <亮度>%, <alpha>);
```

值	描述
<hue>	以色轮上的度数表示（无单位），其中0°为红色，60°为黄色，120°为绿色，180°为青色，240°为蓝色，300°为品红，360°为红色
<饱和度>	百分比，0% 表示完全去饱和（灰度），100% 表示完全饱和（鲜艳的颜色）
<亮度>	百分比，0% 表示完全黑色，100% 表示完全白色
<alpha>	一个从 0 到 1 的数字，0 表示完全透明，1 表示完全不透明


```
hsla(<hue>, <saturation>%, <lightness>%, <alpha>);
```

Value	Description
<hue>	specified in degrees around the color wheel (without units), where 0° is red, 60° is yellow, 120° is green, 180° is cyan, 240° is blue, 300° is magenta, and 360° is red
<saturation>	percentage where 0% is fully desaturated (grayscale) and 100% is fully saturated (vividly colored)
<lightness>	percentage where 0% is fully black and 100% is fully white
<alpha>	a number from 0 - 1 where 0 is fully transparent and 1 is fully opaque

第19章：不透明度

第19.1节：不透明度属性

元素的不透明度可以使用opacity属性设置。数值范围可以从0.0（透明）到1.0（不透明）。

示例用法

```
<div style="opacity:0.8;">
    这是一个部分透明的元素
</div>
```

属性值	透明度
`opacity: 1.0;`	不透明
`opacity: 0.75;`	25% 透明（75% 不透明）
`opacity: 0.5;`	50% 透明（50% 不透明）
`opacity: 0.25;`	75% 透明（25% 不透明）
`opacity: 0.0;`	透明

第19.2节：IE对`opacity`的兼容性

在所有版本的IE中使用opacity，顺序如下：

```
.transparent-element {
  /* 适用于IE 8和9 */
-ms-filter:"progid:DXImageTransform.Microsoft.Alpha(Opacity=60)"; // IE8
  /* 也适用于 IE 8 和 9，但同样适用于 5、6、7 */
  filter: alpha(opacity=60); // IE 5-7
  /* 现代浏览器 */
  opacity: 0.6;
}
```


Chapter 19: Opacity

Section 19.1: Opacity Property

An element's opacity can be set using the `opacity` property. Values can be anywhere from `0.0` (transparent) to `1.0` (opaque).

Example Usage

```
<div style="opacity:0.8;">
    This is a partially transparent element
</div>
```

Property Value	Transparency
`opacity: 1.0;`	Opaque
`opacity: 0.75;`	25% transparent (75% Opaque)
`opacity: 0.5;`	50% transparent (50% Opaque)
`opacity: 0.25;`	75% transparent (25% Opaque)
`opacity: 0.0;`	Transparent

Section 19.2: IE Compatibility for `opacity`

To use `opacity` in all versions of IE, the order is:

```
.transparent-element {
  /* for IE 8 & 9 */
  -ms-filter:"progid:DXImageTransform.Microsoft.Alpha(Opacity=60)"; // IE8
  /* works in IE 8 & 9 too, but also 5, 6, 7 */
  filter: alpha(opacity=60); // IE 5-7
  /* Modern Browsers */
  opacity: 0.6;
}
```

第20章：长度单位

单位	描述
%	根据属性定义相对于父对象或当前对象的尺寸
em	相对于元素的字体大小（2em 表示当前字体大小的两倍）
rem	相对于根元素的字体大小
vw	相对于视口宽度的1%
vh	相对于视口高度的1%
vmin	相对于视口较小边的1%
vmax	相对于视口较大边的1%
cm	厘米
mm	毫米
in	英寸（1英寸 = 96像素 = 2.54厘米）
px	像素（1像素 = 1/96英寸）
pt	磅（1磅 = 1/72英寸）
pc	派卡（1派卡 = 12 磅）
s	秒（用于动画和过渡）
ms	毫秒（用于动画和过渡）
ex	相对于当前字体的x高度
ch	基于数字零（0）字符的宽度
fr	分数单位（用于CSS网格布局）

CSS 距离测量是一个紧跟长度单位（px、em、pc、in、…）后的数字

CSS 支持多种长度测量单位。它们分为绝对单位或相对单位。

第20.1节：使用 rem 和 em 创建可缩放元素

版本 ≥ 3

你可以使用由 html 标签的 font-size 定义的 rem 来设置元素的 font-size 为 rem 值，并在元素内部使用 em 来创建随全局 font-size 缩放的元素。

HTML：

```
<input type="button" value="Button">
<input type="range">
<input type="text" value="Text">
```

相关 CSS：

```
html {
  font-size: 16px;
}

input[type="button"] {
  font-size: 1rem;
  padding: 0.5em 2em;
}

input[类型="range"] {
  font-size: 1rem;
```


Chapter 20: Length Units

Unit	Description
%	Define sizes in terms of parent objects or current object dependent on property
em	Relative to the font-size of the element (2em means 2 times the size of the current font)
rem	Relative to font-size of the root element
vw	Relative to 1% of the width of the viewport*
vh	Relative to 1% of the height of the viewport*
vmin	Relative to 1% of viewport's* smaller dimension
vmax	Relative to 1% of viewport's* larger dimension
cm	centimeters
mm	millimeters
in	inches (1in = 96px = 2.54cm)
px	pixels (1px = 1/96th of 1in)
pt	points (1pt = 1/72 of 1in)
pc	picas (1pc = 12 pt)
s	seconds (used for animations and transitions)
ms	milliseconds (used for animations and transitions)
ex	Relative to the x-height of the current font
ch	Based on the width of the zero (0) character
fr	fractional unit (used for CSS Grid Layout)

A CSS distance measurement is a number immediately followed by a length unit (px, em, pc, in, …)

CSS supports a number of length measurements units. They are absolute or relative.

Section 20.1: Creating scalable elements using rems and ems

Version ≥ 3

You can use `rem` defined by the `font-size` of your `html` tag to style elements by setting their `font-size` to a value of `rem` and use `em` inside the element to create elements that scale with your global `font-size`.

HTML:

```
<input type="button" value="Button">
<input type="range">
<input type="text" value="Text">
```

Relevant CSS:

```
html {
  font-size: 16px;
}

input[type="button"] {
  font-size: 1rem;
  padding: 0.5em 2em;
}

input[type="range"] {
  font-size: 1rem;
```

```
  width: 10em;
}

input[type=text] {
  font-size: 1rem;
  padding: 0.5em;
}
```

可能的结果：

第20.2节：使用rem的字体大小

CSS3引入了一些新的单位，包括rem单位，代表“根em”。让我们看看rem是如何工作的。

首先，让我们看看em和rem之间的区别。

- em：相对于父元素的字体大小。这会导致累积放大问题
- rem：相对于根元素或<html>元素的字体大小。这意味着可以为html元素声明一个统一的字体大小，并将所有rem单位定义为该大小的百分比。

使用rem进行字体大小设置的主要问题是这些值有点难以使用。以下是一些常见字体大小以rem单位表示的示例，假设基础大小为16px：

- 10px = 0.625rem
- 12px = 0.75rem
- 14px = 0.875rem
- 16px = 1rem（基础）
- 18px = 1.125rem
- 20px = 1.25rem
- 24px = 1.5rem
- 30px = 1.875rem
- 32px = 2rem

代码：

版本 ≥ 3

```
html {
  font-size: 16px;
}
```



```
  width: 10em;
}

input[type=text] {
  font-size: 1rem;
  padding: 0.5em;
}
```

Possible Result:

Section 20.2: Font size with rem

CSS3 introduces a few new units, including the `rem` unit, which stands for "root em". Let's look at how `rem` works.

First, let's look at the differences between `em` and `rem`.

- **em**: Relative to the font size of the parent. This causes the compounding issue
- **rem**: Relative to the font size of the root or **`<html>`** element. This means it's possible to declare a single font size for the html element and define all `rem` units to be a percentage of that.

The main issue with using `rem` for font sizing is that the values are somewhat difficult to use. Here is an example of some common font sizes expressed in `rem` units, assuming that the base size is 16px :

- 10px = 0.625rem
- 12px = 0.75rem
- 14px = 0.875rem
- 16px = 1rem (base)
- 18px = 1.125rem
- 20px = 1.25rem
- 24px = 1.5rem
- 30px = 1.875rem
- 32px = 2rem

CODE:

Version ≥ 3

```
html {
  font-size: 16px;
}
```

```css
h1 {
  font-size: 2rem;          /* 32px */
}

p {
  font-size: 1rem;          /* 16px */
}

li {
  font-size: 1.5em;         /* 24px */
}
```

第20.3节：vmin 和 vmax

- vmin：相对于视口较小尺寸的1%
- vmax：相对于视口较大尺寸的1%

换句话说，1 vmin 等于 1 vh 和 1 vw 中较小的那个

1 vmax 等于 1 vh 和 1 vw 中较大的那个

注意：vmax 在以下环境中 不支持：

- 任何版本的Internet Explorer
- 6.1版本之前的Safari

第20.4节：vh 和 vw

CSS3 引入了两种表示尺寸的单位。

- vh，代表视口高度，相当于视口高度的1%
- vw，代表视口宽度，相当于视口宽度的1%

版本 ≥ 3

```css
div {
  width: 20vw;
  height: 20vh;
}
```

上面，div 的尺寸占视口宽度和高度的20%

第20.5节：使用百分比 %

创建响应式应用时的有用单位之一。

其尺寸取决于其父容器。

公式：

> (父容器的宽度) * (百分比(%)) = 输出

例如：

父元素宽度为100px，而子元素宽度为50%。


```css
h1 {
  font-size: 2rem;          /* 32px */
}

p {
  font-size: 1rem;          /* 16px */
}

li {
  font-size: 1.5em;         /* 24px */
}
```

Section 20.3: vmin and vmax

- **vmin**: Relative to 1 percent of the viewport's smaller dimension
- **vmax**: Relative to 1 percent of the viewport's larger dimension

In other words, 1 vmin is equal to the smaller of *1 vh* and *1 vw*

1 vmax is equal to the larger of *1 vh* and *1 vw*

Note: vmax is not supported in:

- any version of Internet Explorer
- Safari before version 6.1

Section 20.4: vh and vw

CSS3 introduced two units for representing size.

- vh, which stands for viewport height is relative to 1% of the viewport height
- vw, which stands for viewport width is relative to 1% of the viewport width

Version ≥ 3

```css
div {
  width: 20vw;
  height: 20vh;
}
```

Above, the size for the div takes up 20% of the width and height of the viewport

Section 20.5: using percent %

One of the useful unit when creating a responsive application.

Its size depends on its parent container.

Equation:

> (Parent Container`s width) * (Percentage(%)) = Output

For Example:

Parent has **100px** width while the *Child* has **50%**.

输出时，子元素的宽度将是父元素的一半（50%），即50px。

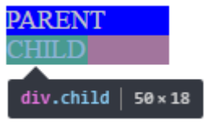
HTML

```
<div class="parent">
    父元素
     <div class="child">
       子元素
     </div>
</div>
```

CSS

```
<style>

*{
color: #CCC;
}

.parent{
background-color: blue;
   width: 100px;
}

.child{
background-color: green;
   width: 50%;
}

</style>
```

输出



On the output, the *Child*'s width will be half(50%) of the *Parent*'s, which is **50px**.

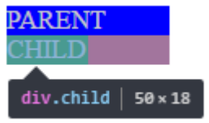
HTML

```
<div class="parent">
   PARENT
   <div class="child">
     CHILD
   </div>
</div>
```

CSS

```
<style>

*{
  color: #CCC;
}

.parent{
  background-color: blue;
  width: 100px;
}

.child{
  background-color: green;
  width: 50%;
}

</style>
```

OUTPUT



第21章：伪元素

伪元素	描述
::after	在元素内容之后插入内容
::before	在元素内容之前插入内容
::first-letter	选择每个元素的第一个字母
::first-line	选择每个元素的第一行
::selection	匹配用户选中的元素部分
::backdrop	用于创建一个背景，隐藏顶层堆栈中元素下方的文档内容
::placeholder	允许你为表单元素的占位符文本设置样式（实验性）
::marker	用于对指定元素应用列表样式属性（实验性）
::拼写错误	表示浏览器标记为拼写错误的文本段（实验性）
::语法错误	表示浏览器标记为语法错误的文本段（实验性）

伪元素，就像伪类一样，添加到CSS选择器中，但它们不是描述特殊状态，而是允许你限定并样式化HTML元素的某些部分。

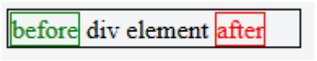
例如，::first-letter伪元素只针对选择器指定的块级元素的第一个字母。

第21.1节：伪元素

伪元素添加到选择器中，但不是描述特殊状态，而是允许你样式化文档的某些部分。

content属性是伪元素渲染所必需的；但是，该属性可以为空值（例如content: ""）。

```
div::after {
  content: 'after';
  color: red;
  border: 1px 实线 红色;
}

div {
  color: black;
  border: 1px solid black;
  padding: 1px;
}

div::before {
  content: 'before';
  color: green;
  border: 1px solid green;
}
```



第21.2节：列表中的伪元素

伪元素通常用于改变列表的外观（主要是无序列表，ul）。

Chapter 21: Pseudo-Elements

pseudo-element	Description
::after	Insert content after the content of an element
::before	Insert content before the content of an element
::first-letter	Selects the first letter of each element
::first-line	Selects the first line of each element
::selection	Matches the portion of an element that is selected by a user
::backdrop	Used to create a backdrop that hides the underlying document for an element in the top layer's stack
::placeholder	Allows you to style the placeholder text of a form element (Experimental)
::marker	For applying list-style attributes on a given element (Experimental)
::spelling-error	Represents a text segment which the browser has flagged as incorrectly spelled (Experimental)
::grammar-error	Represents a text segment which the browser has flagged as grammatically incorrect (Experimental)

Pseudo-elements, just like pseudo-classes, are added to a CSS selectors but instead of describing a special state, they allow you to scope and style certain parts of an html element.

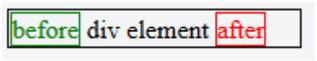
For example, the ::first-letter pseudo-element targets only the first letter of a block element specified by the selector.

Section 21.1: Pseudo-Elements

Pseudo-elements are added to selectors but instead of describing a special state, they allow you to style certain parts of a document.

The content attribute is required for pseudo-elements to render; however, the attribute can have an empty value (e.g. **content**: "").

```
div::after {
  content: 'after';
  color: red;
  border: 1px solid red;
}

div {
  color: black;
  border: 1px solid black;
  padding: 1px;
}

div::before {
  content: 'before';
  color: green;
  border: 1px solid green;
}
```



Section 21.2: Pseudo-Elements in Lists

Pseudo-elements are often used to change the look of lists (mostly for unordered lists, ul).

第一步是去除默认的列表符号：

```
ul {
  list-style-type: none;
}
```

然后添加自定义样式。在本例中，我们将为符号创建渐变方块。

```
li:before {
  content: "";
  display: inline-block;
  margin-right: 10px;
  height: 10px;
  width: 10px;
  background: linear-gradient(red, blue);
}
```

HTML

```
<ul>
   <li>测试 I</li>
   <li>测试 II</li>
</ul>
```

结果

The first step is to remove the default list bullets:

```
ul {
  list-style-type: none;
}
```

Then you add the custom styling. In this example, we will create gradient boxes for bullets.

```
li:before {
  content: "";
  display: inline-block;
  margin-right: 10px;
  height: 10px;
  width: 10px;
  background: linear-gradient(red, blue);
}
```

HTML

```
<ul>
   <li>Test I</li>
   <li>Test II</li>
</ul>
```

Result

第22章：定位

参数	详情
static	默认值。元素按文档流中的顺序渲染。top、right、bottom、left 和 z-index 属性不适用。
relative	元素相对于其正常位置定位，因此`left:20px`会在元素的左侧位置上增加20像素
fixed	该元素相对于浏览器窗口定位
绝对定位	该元素相对于其第一个已定位（非静态）祖先元素定位
初始值	将此属性设置为默认值。
继承	从其父元素继承此属性。
粘性定位	实验性功能。它在父元素内表现得像`position: static`，直到达到给定的偏移阈值，然后表现得像`position: fixed`。
取消设置	initial和inherit的组合。更多信息请见here。

第22.1节：带有z-index的重叠元素

要更改默认的堆叠顺序定位元素（position属性设置为relative、absolute或fixed），请使用z-index属性。

z-index 越高，在堆叠上下文（z 轴上）中位置越靠上。

示例

在下面的示例中，z-index 值为 3 时绿色位于最上层，z-index 为 2 时红色位于其下方，z-index 为 1 时蓝色位于最底层。

HTML

```
<div id="div1"></div>
<div id="div2"></div>
<div id="div3"></div>
```

CSS

```
div {
    position: absolute;
    height: 200px;
    width: 200px;
}
div#div1 {
    z-index: 1;
    left: 0px;
    top: 0px;
    background-color: blue;
}
div#div2 {
    z-index: 3;
    left: 100像素;
    top: 100像素;
    background-color: 绿色;
}
div#div3 {
    z-index: 2;
    left: 50像素;
```


Chapter 22: Positioning

Parameter	Details
static	Default value. Elements render in order, as they appear in the document flow. The top, right, bottom, left and z-index properties do not apply.
relative	The element is positioned relative to its normal position, so `left:20px` adds 20 pixels to the element's LEFT position
fixed	The element is positioned relative to the browser window
absolute	The element is positioned relative to its first positioned (not static) ancestor element
initial	Sets this property to its default value.
inherit	Inherits this property from its parent element.
sticky	Experimental feature. It behaves like `position: static` within its parent until a given offset threshold is reached, then it acts as `position: fixed`.
unset	Combination of initial and inherit. More info here.

Section 22.1: Overlapping Elements with z-index

To change the default stack order positioned elements (`position` property set to `relative`, `absolute` or `fixed`), use the `z-index` property.

The higher the z-index, the higher up in the stacking context (on the z-axis) it is placed.

Example

In the example below, a z-index value of 3 puts green on top, a z-index of 2 puts red just under it, and a z-index of 1 puts blue under that.

HTML

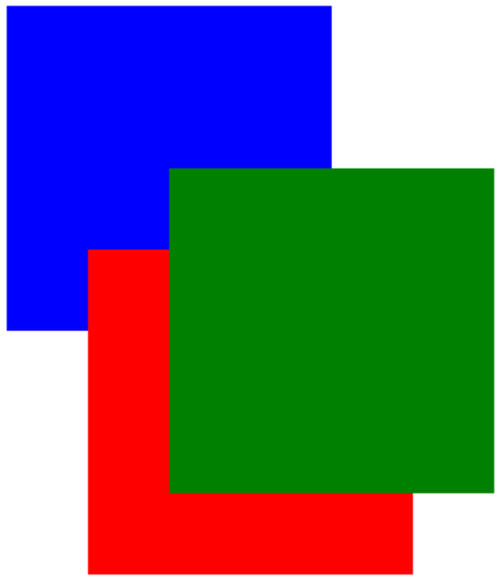
```
<div id="div1"></div>
<div id="div2"></div>
<div id="div3"></div>
```

CSS

```
div {
    position: absolute;
    height: 200px;
    width: 200px;
}
div#div1 {
    z-index: 1;
    left: 0px;
    top: 0px;
    background-color: blue;
}
div#div2 {
    z-index: 3;
    left: 100px;
    top: 100px;
    background-color: green;
}
div#div3 {
    z-index: 2;
    left: 50px;
```

```
    top: 150像素;
    background-color: red;
}
```

这会产生以下效果：



请参见 JSFiddle 上的示例。

语法

```
z-index: [ 数字 ] | auto;
```

参数	详情
数字	整数值。数字越大，越位于z-index堆栈的上层。0是默认值。允许负值。
自动	使元素具有与其父元素相同的堆叠上下文。（默认）

备注

所有元素在CSS中都按照三维轴进行布局，包括由z-index属性测量的深度轴。 z-index仅对定位元素有效：（参见：为什么z-index需要定义定位才能生效？）。唯一被忽略的值是默认值static。

请阅读关于z-index属性和堆叠上下文的内容，详见CSS规范中关于分层展示的部分，以及Mozilla开发者网络。

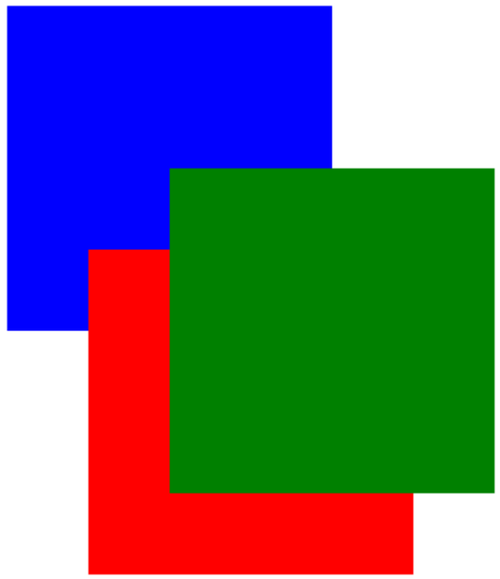
第22.2节：绝对定位

当使用绝对定位时，目标元素的盒子会脱离正常流，不再影响页面上其他元素的位置。偏移属性：

1.top
2.left
3.right


```
    top: 150px;
    background-color: red;
}
```

This creates the following effect:



See a working example at JSFiddle.

Syntax

```
z-index: [ number ] | auto;
```

Parameter	Details
number	An integer value. A higher number is higher on the z-index stack. 0 is the default value. Negative values are allowed.
auto	Gives the element the same stacking context as its parent. (**Default**)

Remarks

All elements are laid out in a 3D axis in CSS, including a depth axis, measured by the z-index property. z-index only works on positioned elements: (see: Why does z-index need a defined position to work?). The only value where it is ignored is the default value, static.

Read about the z-index property and Stacking Contexts in the CSS Specification on layered presentation and at the Mozilla Developer Network.

Section 22.2: Absolute Position

When absolute positioning is used the box of the desired element is taken out of the *Normal Flow* and it no longer affects the position of the other elements on the page. Offset properties:

1. top
2. left
3. right

4. bottom

指定元素应相对于其最近的非静态包含元素出现。

```
.abspos{
    position:absolute;
    top:0px;
    left:500px;
}
```

此代码将包含属性class="abspos"的元素的盒子相对于其包含元素向下移动0像素，向右移动500像素。

第22.3节：固定定位

将position定义为fixed，我们可以将元素从文档流中移除，并相对于浏览器窗口设置其位置。一个明显的用途是当我们希望在滚动到长页面底部时某个元素仍然可见。

```
#stickyDiv {
    position:fixed;
    top:10px;
    left:10px;
}
```

第22.4节：相对定位

相对定位是相对于元素在正常流中的位置移动元素。偏移属性：

1.上
2.左
3.右
4.下

用于指示元素相对于其在正常流中的位置移动的距离。

```
.relpos{
    position:relative;
    top:20px;
    left:30px;
}
```

这段代码会将包含属性 class="relpos" 的元素的盒子从其在正常流中的位置向下移动20像素，向右移动30像素。

第22.5节：静态定位

元素的默认定位是static。引用MDN的说法：

> 该关键字让元素使用正常行为，即在流中的当前位置进行布局。top、right、bottom、left 和 z-index 属性不适用。

```
.element{
  position:static;
```


4. bottom

specify the element should appear in relation to its next non-static containing element.

```
.abspos{
    position:absolute;
    top:0px;
    left:500px;
}
```

This code will move the box containing element with attribute `class="abspos"` down 0px and right 500px relative to its containing element.

Section 22.3: Fixed position

Defining position as fixed we can remove an element from the document flow and set its position relatively to the browser window. One obvious use is when we want something to be visible when we scroll to the bottom of a long page.

```
#stickyDiv {
    position:fixed;
    top:10px;
    left:10px;
}
```

Section 22.4: Relative Position

Relative positioning moves the element in relation to where it would have been in *normal flow* .Offset properties:

1. top
2. left
3. right
4. bottom

are used to indicate how far to move the element from where it would have been in normal flow.

```
.relpos{
    position:relative;
    top:20px;
    left:30px;
}
```

This code will move the box containing element with attribute class="relpos" 20px down and 30px to the right from where it would have been in normal flow.

Section 22.5: Static positioning

The default position of an element is `static`. To quote MDN:

> This keyword lets the element use the normal behavior, that is it is laid out in its current position in the flow. The top, right, bottom, left and z-index properties do not apply.

```
.element{
  position:static;
```

```
}
```


}

第23章：布局控制

值	效果
none	隐藏元素且不占用空间。
block	块级元素，占据可用宽度的100%，元素后换行。
inline	行内元素，不占宽度，元素后不换行。
inline-block	结合行内元素和块级元素的特殊属性，不换行，但可以设置宽度。
inline-flex	将元素显示为行内级弹性容器。
inline-table	元素显示为行内级表格。
grid	表现如块级元素，根据网格模型布局其内容。
flex	表现如块级元素，根据弹性盒模型布局其内容。
inherit	从父元素继承值。
initial	将值重置为HTML规范中描述的行为或浏览器/用户默认样式表中的默认值。
table	表现得像HTML的table元素。
table-cell	让元素表现得像**<td>**元素
table-column	让元素表现得像**<col>**元素
table-row	让元素表现得像**<tr>**元素
list-item	让元素表现得像** **元素。

第23.1节：display属性

CSS属性display是控制HTML文档布局和流动的基础。大多数元素的默认display值为block或inline（尽管有些元素有其他默认值）。

内联

内联元素只占用必要的宽度。它与同类型的其他元素水平排列，并且可能不包含其他非内联元素。

```
<span>这是一些<b>加粗</b>的文本！</span>
```

This is some **bolded** text!

如上所示，两个inline元素，和<b>，都是内联的（因此得名），不会中断文本的流动。

块级元素

块级元素占据其父元素的最大可用宽度。它从新行开始，与内联元素不同，它不限制所包含元素的类型。

```
<div>你好，世界！</div><div>这是一个示例！</div>
```

Hello world!
This is an example!

div元素默认是块级的，如上所示，两个块级元素垂直堆叠，且与内联元素不同，文本流会中断。

Chapter 23: Layout Control

Value	Effect
none	Hide the element and prevent it from occupying space.
block	Block element, occupy 100% of the available width, break after element.
inline	Inline element, occupy no width, no break after element.
inline-block	Taking special properties from both inline and block elements, no break, but can have width.
inline-flex	Displays an element as an inline-level flex container.
inline-table	The element is displayed as an inline-level table.
grid	Behaves like a block element and lays out its content according to the grid model.
flex	Behaves like a block element and lays out its content according to the flexbox model.
inherit	Inherit the value from the parent element.
initial	Reset the value to the default value taken from behaviors described in the HTML specifications or from the browser/user default stylesheet.
table	Behaves like the HTML `table` element.
table-cell	Let the element behave like a **<td>** element
table-column	Let the element behave like a **<col>** element
table-row	Let the element behave like a **<tr>** element
list-item	Let the element behave like a ** ** element.

Section 23.1: The display property

The `display` CSS property is fundamental for controlling the layout and flow of an HTML document. Most elements have a default `display` value of either `block` or `inline` (though some elements have other default values).

Inline

An `inline` element occupies only as much width as necessary. It stacks horizontally with other elements of the same type and may not contain other non-inline elements.

```
<span>This is some <b>bolded</b> text!</span>
```

This is some **bolded** text!

As demonstrated above, two `inline` elements, **** and **<b>**, are in-line (hence the name) and do not break the flow of the text.

Block

A `block` element occupies the maximum available width of its' parent element. It starts with a new line and, in contrast to `inline` elements, it does not restrict the type of elements it may contain.

```
<div>Hello world!</div><div>This is an example!</div>
```

Hello world!
This is an example!

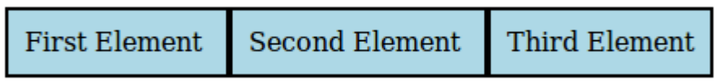
The `div` element is block-level by default, and as shown above, the two `block` elements are vertically stacked and, unlike the `inline` elements, the flow of the text breaks.

内联块

inline-block值结合了两者的优点：它使元素融入文本本流，同时允许我们使用padding、margin、高度等对内联元素无明显效果的属性。

具有此显示值的元素表现得像普通文本，因此会受到控制文本流的规则影响，如text-align。默认情况下，它们也会缩小到适合其内容的最小尺寸。

```
<!--内联：无序列表-->
<style>
li {
display : inline;
    background : lightblue;
    padding:10px;

border-width:2px;
    border-color:black;
    border-style:solid;
    }
</style>

<ul>
<li>第一个元素 </li>
<li>第二个元素 </li>
<li>第三个元素 </li>
</ul>
```



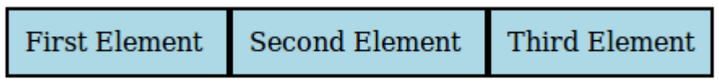

```
<!--block: unordered list-->
<style>
li {
display : block;
    background : lightblue;
    padding:10px;

border-width:2px;
    border-color:black;
    border-style:solid;
    }
</style>

<ul>
<li>第一个元素 </li>
<li>第二个元素 </li>
<li>第三个元素 </li>
</ul>
```

Inline Block

The `inline-block` value gives us the best of both worlds: it blends the element in with the flow of the text while allowing us to use `padding`, `margin`, `height` and similar properties which have no visible effect on `inline` elements.

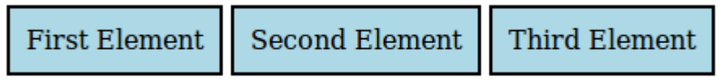
Elements with this display value act as if they were regular text and as a result are affected by rules controlling the flow of text such as `text-align`. By default they are also shrunk to the the smallest size possible to accommodate their content.

```
<!--Inline: unordered list-->
<style>
li {
    display : inline;
    background : lightblue;
    padding:10px;

    border-width:2px;
    border-color:black;
    border-style:solid;
    }
</style>

<ul>
<li>First Element </li>
<li>Second Element </li>
<li>Third Element </li>
</ul>
```




```
<!--block: unordered list-->
<style>
li {
    display : block;
    background : lightblue;
    padding:10px;

    border-width:2px;
    border-color:black;
    border-style:solid;
    }
</style>

<ul>
<li>First Element </li>
<li>Second Element </li>
<li>Third Element </li>
</ul>
```

```
<!--Inline-block: unordered list-->
<style>
li {
display : inline-block;
    background : lightblue;
    padding:10px;

border-width:2px;
    border-color:black;
    border-style:solid;
     }

</style>

<ul>
<li>第一个元素 </li>
<li>第二个元素 </li>
<li>第三个元素 </li>
</ul>
```



无

一个被赋予 display 属性值为 none 的元素将完全不显示。

例如，我们创建一个 id 为 myDiv 的 div 元素：

```
<div id="myDiv"></div>
```

现在可以通过以下CSS规则将其标记为不显示：

```
#myDiv {
    display: none;
}
```

当一个元素被设置为display:none;时，浏览器会忽略该特定元素的所有其他布局属性（包括position和float）。该元素不会渲染任何盒子，其在HTML中的存在也不会影响后续元素的位置。

请注意，这与将visibility属性设置为hidden不同。将元素的visibility设置为hidden；元素不会显示在页面上，但在渲染过程中仍会占据空间，就像它是可见的一样。因此，这会影响后续元素在页面上的显示方式。

display属性的none值通常与JavaScript一起使用，以便随意显示或隐藏元素，无需实际删除和重新创建它们。

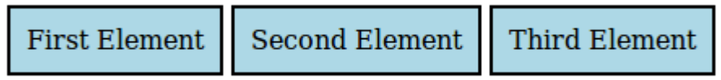
第23.2节：使用div获取旧的表格结构

这是普通的HTML表格结构

```
<style>
table {
        width: 100%;
    }
```



```
<!--Inline-block: unordered list-->
<style>
li {
    display : inline-block;
    background : lightblue;
    padding:10px;

    border-width:2px;
    border-color:black;
    border-style:solid;
    }

</style>

<ul>
<li>First Element </li>
<li>Second Element </li>
<li>Third Element </li>
</ul>
```



none

An element that is given the none value to its display property will not be displayed at all.

For example let's create a div-element that has an id of `myDiv`:

```
<div id="myDiv"></div>
```

This can now be marked as not being displayed by the following CSS rule:

```
#myDiv {
    display: none;
}
```

When an element has been set to be **`display`**`:none;` the browser ignores every other layout property for that specific element (both `position` and `float`). No box will be rendered for that element and its existence in html does not affect the position of following elements.

Note that this is different from setting the `visibility` property to `hidden`. Setting **`visibility`**`: hidden;` for an element would not display the element on the page but the element would still take up the space in the rendering process as if it would be visible. This will therefore affect how following elements are displayed on the page.

The `none` value for the display property is commonly used along with JavaScript to show or hide elements at will, eliminating the need to actually delete and re-create them.

Section 23.2: To get old table structure using div

This is the normal HTML table structure

```
<style>
    table {
        width: 100%;
    }
```

```
 </style>

 <table>
    <tr>
      <td>
我是一个表格
      </td>
    </tr>
</table>
```

你可以像这样实现相同的功能

```
<style>
.table-div {
        display: table;
    }
.table-row-div {
        display: table-row;
    }
.table-cell-div {
        display: table-cell;
    }
</style>

<div class="table-div">
   <div class="table-row-div">
      <div class="table-cell-div">
        我现在表现得像一个表格
      </div>
   </div>
</div>
```



```
</style>

<table>
  <tr>
    <td>
        I'm a table
    </td>
  </tr>
</table>
```

You can do same implementation like this

```
<style>
    .table-div {
        display: table;
    }
    .table-row-div {
        display: table-row;
    }
    .table-cell-div {
        display: table-cell;
    }
</style>

<div class="table-div">
  <div class="table-row-div">
    <div class="table-cell-div">
      I behave like a table now
    </div>
  </div>
</div>
```

第24章：网格布局

网格布局是一种新的强大CSS布局系统，可以轻松地将网页内容划分为行和列。

第24.1节：基本示例

属性 可能的取值
display grid / inline-grid

CSS 网格被定义为一种 display 属性。它只适用于父元素及其直接子元素。

考虑以下标记：

```
<section class="container">
  <div class="item1">item1</div>
  <div class="item2">item2</div>
  <div class="item3">item3</div>
  <div class="item4">item4</div>
</section>
```

将上述标记结构定义为网格的最简单方法是将其 display 属性设置为 grid：

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

然而，这样做会导致所有子元素重叠在一起。这是因为子元素目前不知道如何在网格中定位自己。但我们可以明确告诉它们。

首先，我们需要告诉网格元素 .container 它的结构由多少行和列组成，我们可以使用 grid-columns 和 grid-rows 属性（注意复数形式）来实现：

```
.container {
  display: grid;
  grid-columns: 50px 50px 50px;
  grid-rows: 50px 50px;
}
```

然而，这仍然帮不了我们太多，因为我们需要为每个子元素指定顺序。我们可以通过指定grid-row和grid-column的值来告诉它在网格中的位置：

```
.container .item1 {
  grid-column: 1;
  grid-row: 1;
}
.container .item2 {
  grid-column: 2;
  grid-row: 1;
}
.container .item3 {
  grid-column: 1;
  grid-row: 2;
}
.container .item4 {
  grid-column: 2;
```


Chapter 24: Grid

Grid layout is a new and powerful CSS layout system that allows to divide a web page content into rows and columns in an easy way.

Section 24.1: Basic Example

Property Possible Values
display grid / inline-grid

The CSS Grid is defined as a display property. It applies to a parent element and its immediate children only.

Consider the following markup:

```
<section class="container">
  <div class="item1">item1</div>
  <div class="item2">item2</div>
  <div class="item3">item3</div>
  <div class="item4">item4</div>
</section>
```

The easiest way to define the markup structure above as a grid is to simply set its `display` property to `grid`:

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

However, doing this will invariably cause all the child elements to collapse on top of one another. This is because the children do not currently know how to position themselves within the grid. But we can explicitly tell them.

First we need to tell the grid element `.container` how many rows and columns will make up its structure and we can do this using the `grid-columns` and `grid-rows` properties (note the pluralisation):

```
.container {
  display: grid;
  grid-columns: 50px 50px 50px;
  grid-rows: 50px 50px;
}
```

However, that still doesn't help us much because we need to give an order to each child element. We can do this by specifying the `grid-row` and `grid-column` values which will tell it where it sits in the grid:

```
.container .item1 {
  grid-column: 1;
  grid-row: 1;
}
.container .item2 {
  grid-column: 2;
  grid-row: 1;
}
.container .item3 {
  grid-column: 1;
  grid-row: 2;
}
.container .item4 {
  grid-column: 2;
```

```
  grid-row: 2;
}
```

通过为每个项目指定列和值，确定项目在容器中的顺序。

在JSFiddle上查看一个工作示例。您需要在IE10、IE11或Edge中查看此示例才能正常工作，因为这些浏览器目前是唯一支持带有供应商前缀-ms-的网格布局（Grid Layout）的浏览器，或者根据caniuse的说明，在Chrome、Opera和Firefox中启用一个标志以进行测试。


```
  grid-row: 2;
}
```

By giving each item a column and row value it identifies the items order within the container.

View a working example on JSFiddle. You'll need to view this in IE10, IE11 or Edge for it to work as these are currently the only browsers supporting Grid Layout (with vendor prefix `-ms-`) or enable a flag in Chrome, Opera and Firefox according to caniuse in order to test with them.

第25章：表格

第25.1节：table-layout

table-layout属性改变用于表格布局的算法。

下面是两个表格的示例，均设置了width: 150px：

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

左侧的表格设置了table-layout: auto，而右侧的表格设置了table-layout: fixed。前者宽度超过指定宽度（210px而非150px），但内容适合。后者采用定义的150px宽度，无论内容是否溢出。

值	描述
auto	这是默认值。它定义表格的布局由其单元格内容决定。
fixed	该值将表格布局设置为由表格的宽度属性决定。如果单元格内容超过此宽度，单元格不会调整大小，而是让内容溢出。

如果单元格的内容超过此宽度，单元格不会调整大小，而是让内容溢出。

第25.2节：空单元格

empty-cells 属性决定是否显示没有内容的单元格。除非 border-collapse 设置为 separate，否则此属性无效。

下面是两个表格的示例，它们对 empty-cells 属性设置了不同的值：

First name	Last name	Homeworld
Luke		Tatooine
Leia	Organa	

First name	Last name	Homeworld
Luke		Tatooine
Leia	Organa	

左边的表格设置了 empty-cells: show，而右边的表格设置了 empty-cells: hide。前者显示空单元格，后者则不显示。

值	描述
show	*这是默认值。即使单元格为空，也会显示该单元格。*
hide	如果单元格内没有内容，则完全隐藏该单元格。

更多信息：

- https://www.w3.org/TR/CSS21/tables.html#empty-cells
- https://developer.mozilla.org/en-US/docs/Web/CSS/empty-cells
- http://codepen.io/SitePoint/pen/yfhtq
- https://css-tricks.com/almanac/properties/e/empty-cells/

第25.3节：border-collapse

border-collapse 属性决定表格的边框是分开显示还是合并显示。

Chapter 25: Tables

Section 25.1: table-layout

The `table-layout` property changes the algorithm that is used for the layout of a table.

Below an example of two tables both set to **`width`**: `150px`:

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

The table on the left has **`table-layout`**: `auto` while the one on the right has **`table-layout`**: `fixed`. The former is wider than the specified width (210px instead of 150px) but the contents fit. The latter takes the defined width of 150px, regardless if the contents overflow or not.

Value	Description
auto	This is the default value. It defines the layout of the table to be determined by the contents of its' cells.
fixed	This value sets the table layout to be determined by the width property provided to the table. If the content of a cell exceeds this width, the cell will not resize but instead, let the content overflow.

Section 25.2: empty-cells

The `empty-cells` property determines if cells with no content should be displayed or not. This has no effect unless `border-collapse` is set to `separate`.

Below an example with two tables with different values set to the `empty-cells` property:

First name	Last name	Homeworld
Luke		Tatooine
Leia	Organa	

First name	Last name	Homeworld
Luke		Tatooine
Leia	Organa	

The table on the left has **`empty-cells`**: `show` while the one on the right has **`empty-cells`**: `hide`. The former does display the empty cells whereas the latter does not.

Value	Description
show	This is the default value. It shows cells even if they are empty.
hide	This value hides a cell altogether if there are no contents in the cell.

More Information:

- https://www.w3.org/TR/CSS21/tables.html#empty-cells
- https://developer.mozilla.org/en-US/docs/Web/CSS/empty-cells
- http://codepen.io/SitePoint/pen/yfhtq
- https://css-tricks.com/almanac/properties/e/empty-cells/

Section 25.3: border-collapse

The `border-collapse` property determines if a tables' borders should be separated or merged.

下面是两个表格对border-collapse属性不同取值的示例：

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

左边的表格的border-collapse属性为separate，而右边的表格的border-collapse属性为collapse。

值	描述
separate	这是默认值。它使表格的边框彼此分开。
collapse	该值使表格的边框合并在一起，而不是分开的。

第25.4节：border-spacing

border-spacing属性决定单元格之间的间距。除非border-collapse设置为separate，否则该属性无效。

下面是两个表格对border-spacing属性不同取值的示例：

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

左边的表格的border-spacing属性为2px（默认值），而右边的表格的border-spacing属性为8px。

值	描述
<length>	这是默认行为，但具体数值可能因浏览器而异。
<length> <length>	该语法允许分别指定水平和垂直的数值。

第25.5节：caption-side

caption-side 属性决定了表格中 <caption> 元素的垂直位置。如果该元素不存在，则此属性无效。

下面是两个表格的示例，它们的 caption-side 属性设置了不同的值：

Star Wars figures

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

Star Wars figures

左侧的表格设置了 caption-side: top，而右侧的表格设置了 caption-side: bottom。

值	描述
top	这是默认值。它将标题放置在表格上方。
bottom	该值将标题放置在表格下方。

Below an example of two tables with different values to the `border-collapse` property:

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

The table on the left has **`border-collapse`**: `separate` while the one on the right has **`border-collapse`**: `collapse`.

Value	Description
separate	This is the default value. It makes the borders of the table separate from each other.
collapse	This value sets the borders of the table to merge together, rather than being distinct.

Section 25.4: border-spacing

The `border-spacing` property determines the spacing between cells. This has no effect unless `border-collapse` is set to `separate`.

Below an example of two tables with different values to the `border-spacing` property:

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

The table on the left has **`border-spacing`**: `2px` (default) while the one on the right has **`border-spacing`**: `8px`.

Value	Description
<length>	This is the default behavior, though the exact value can vary between browsers.
<length> <length>	This syntax allows specifying separate horizontal and vertical values respectively.

Section 25.5: caption-side

The `caption-side` property determines the vertical positioning of the **`<caption>`** element within a table. This has no effect if such element does not exist.

Below an example with two tables with different values set to the `caption-side` property:

Star Wars figures

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

First name	Last name	Homeworld
Luke	Skywalker	Tatooine
Leia	Organa	Alderaan

Star Wars figures

The table on the left has **`caption-side`**: `top` while the one on the right has **`caption-side`**: `bottom`.

Value	Description
top	This is the default value. It places the caption above the table.
bottom	This value places the caption below the table.

第26章：过渡

参数	详情
transition-property	需要过渡的具体CSS属性值变化，或者使用 all 表示需要过渡所有可过渡属性。
transition-duration	过渡必须发生的持续时间（或周期），以秒（s）或毫秒（ms）为单位。
transition-timing-function	描述过渡过程中中间值的函数是计算。常用的值有ease、ease-in、ease-out、ease-in-out、linear、cubic-bezier()、steps()。 关于各种时间函数的更多信息可以在W3C规范中找到。
transition-delay	必须经过的时间量，才能开始转换。可以用秒（s）或毫秒（ms）来指定

第26.1节：过渡速记

CSS

```
div{
    宽度: 150像素;
    height:150px;
    背景颜色: 红色;
    过渡: 背景颜色 1秒;
}
div:hover{
    background-color: green;
}
```

HTML

```
<div></div>
```

此示例将在鼠标悬停在 div 上时更改背景颜色，背景颜色的变化将持续1秒。

第26.2节：cubic-bezier

cubic-bezier 函数是一种过渡时间函数，常用于自定义和平滑的过渡效果。

transition-timing-function: cubic-bezier(0.1, 0.7, 1.0, 0.1);

该函数接受四个参数：

cubic-bezier(P1_x, P1_y, P2_x, P2_y)

Chapter 26: Transitions

Parameter	Details
transition-property	The specific CSS property whose value change needs to be transitioned (or) `all`, if all the transitionable properties need to be transitioned.
transition-duration	The duration (or period) in seconds (`s`) or milliseconds (`ms`) over which the transition must take place.
transition-timing-function	A function that describes how the intermediate values during the transition are calculated. Commonly used values are `ease`, `ease-in`, `ease-out`, `ease-in-out`, `linear`, `cubic-bezier()`, `steps()`. More information about the various timing functions can be found in the W3C specs.
transition-delay	The amount of time that must have elapsed before the transition can start. Can be specified in seconds (`s`) or milliseconds (`ms`)

Section 26.1: Transition shorthand

CSS

```
div{
    width: 150px;
    height:150px;
    background-color: red;
    transition: background-color 1s;
}
div:hover{
    background-color: green;
}
```

HTML

```
<div></div>
```

This example will change the background color when the div is hovered the background-color change will last 1 second.

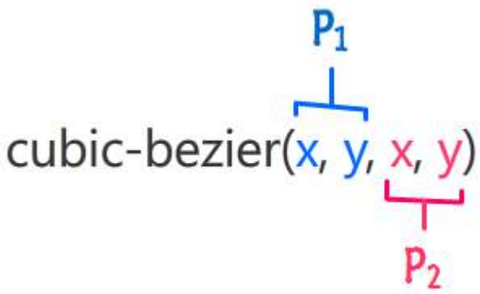
Section 26.2: cubic-bezier

The `cubic-bezier` function is a transition timing function which is often used for custom and smooth transitions.

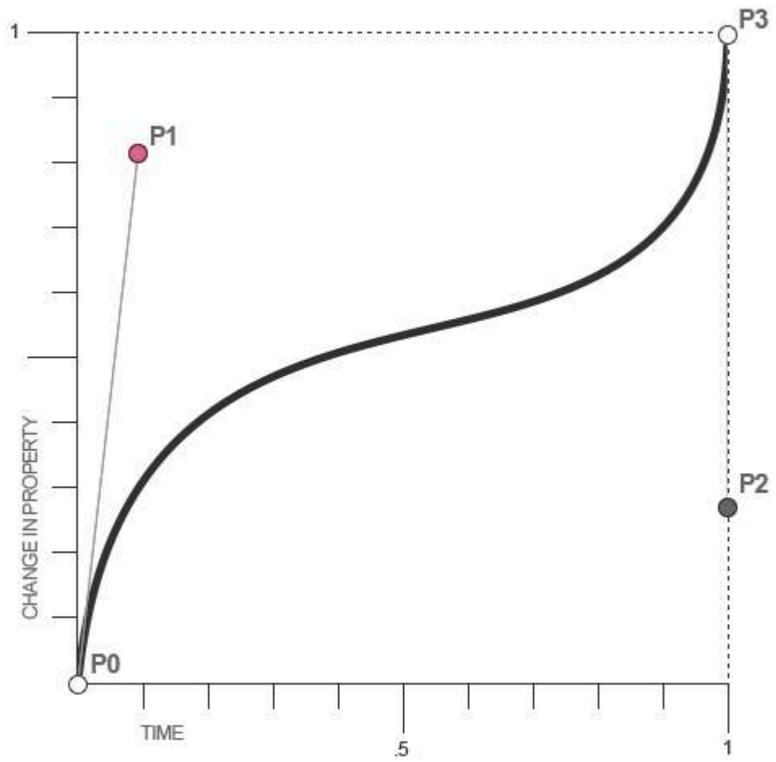
```
transition-timing-function: cubic-bezier(0.1, 0.7, 1.0, 0.1);
```

The function takes four parameters:

```
cubic-bezier(P1_x, P1_y, P2_x, P2_y)
```



这些参数将映射为贝塞尔曲线的一部分点Bézier 曲线：

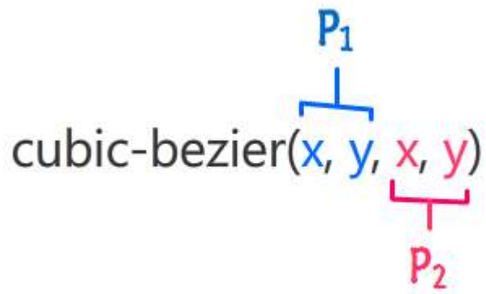


对于 CSS 贝塞尔曲线，P0 和 P3 总是在同一位置。P0 位于 (0,0)，P3 位于 (1,1)，这意味着传递给 cubic-bezier 函数的参数只能在0到1之间。

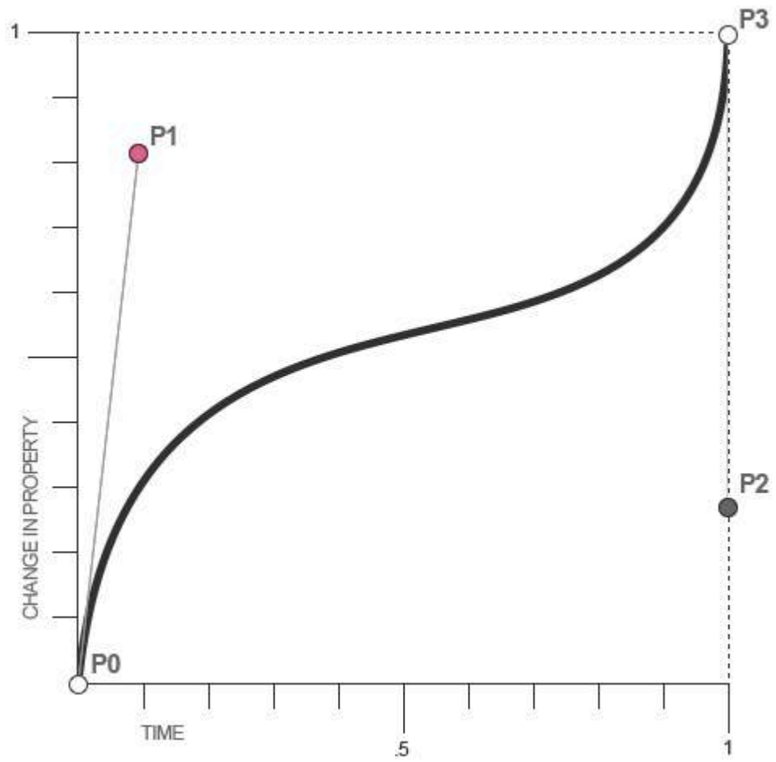
如果传入的参数不在此区间内，函数将默认使用线性过渡。

由于 cubic-bezier 是 CSS 中最灵活的过渡函数，你可以将所有其他过渡时间函数转换为 cubic-bezier 函数：

```
linear: cubic-bezier(0,0,1,1)

ease-in: cubic-bezier(0.42, 0.0, 1.0, 1.0)

ease-out: cubic-bezier(0.0, 0.0, 0.58, 1.0)

ease-in-out: cubic-bezier(0.42, 0.0, 0.58, 1.0)
```

These parameters will be mapped to points which are part of a Bézier curve:



For CSS Bézier Curves, P0 and P3 are always in the same spot. P0 is at (0,0) and P3 is at (1,1), which menas that the parameters passed to the cubic-bezier function can only be between 0 and 1.

If you pass parameters which aren't in this interval the function will default to a `linear` transition.

Since cubic-bezier is the most flexible transition in CSS, you can translate all other transition timing function to cubic-bezier functions:

```
linear: cubic-bezier(0,0,1,1)

ease-in: cubic-bezier(0.42, 0.0, 1.0, 1.0)

ease-out: cubic-bezier(0.0, 0.0, 0.58, 1.0)

ease-in-out: cubic-bezier(0.42, 0.0, 0.58, 1.0)
```

第26.3节：过渡（长写法）

CSS

```
div {
  height: 100px;
  宽度: 100像素;
  边框: 1像素 实线;
  transition-property: 高度, 宽度;
  transition-duration: 1秒, 500毫秒;
  transition-timing-function: 线性;
  transition-delay: 0秒, 1秒;
}
div:悬停 {
  高度: 200像素;
  宽度: 200像素;
}
```

HTML

```
<div></div>
```

- transition-property: 指定过渡效果作用的 CSS 属性。在本例中，div 在悬停时会在水平方向和垂直方向上同时扩展。
- transition-duration: 指定过渡完成所需的时间长度。在上述示例中，高度和宽度的过渡时间分别为1秒和500毫秒。
- transition-timing-function：指定过渡效果的速度曲线。线性（linear）值表示过渡从开始到结束速度保持不变。
- **transition-delay**：指定在过渡效果开始前需要等待的时间。在此情况下，高度将立即开始过渡，而宽度将等待1秒。

Section 26.3: Transition (longhand)

CSS

```
div {
  height: 100px;
  width: 100px;
  border: 1px solid;
  transition-property: height, width;
  transition-duration: 1s, 500ms;
  transition-timing-function: linear;
  transition-delay: 0s, 1s;
}
div:hover {
  height: 200px;
  width: 200px;
}
```

HTML

```
<div></div>
```

- **transition-property**: Specifies the CSS properties the transition effect is for. In this case, the div will expand both horizontally and vertically when hovered.
- **transition-duration**: Specifies the length of time a transition takes to complete. In the above example, the height and width transitions will take 1 second and 500 milliseconds respectively.
- **transition-timing-function**: Specifies the speed curve of the transition effect. A *linear* value indicates the transition will have the same speed from start to finish.
- **transition-delay**: Specifies the amount of time needed to wait before the transition effect starts. In this case, the height will start transitioning immediately, whereas the width will wait 1 second.

第27章：动画

过渡

参数	详情
属性	要过渡的CSS属性，或者`all`，表示所有可过渡的属性。
持续时间	过渡时间，可以是秒或毫秒。
时间函数	指定一个函数来定义属性中间值的计算方式。常见值有 ease、linear和step-end。更多内容请查看缓动函数速查表。
延迟	在播放动画之前等待的时间，单位为秒或毫秒。

@keyframes

[from | to | <百分比>]	你可以指定一个百分比值的时间点，或者两个百分比值，例如10%，20%，表示关键帧设置属性生效的时间段。
块	关键帧的任意数量的CSS属性。

第27.1节：带关键帧的动画

对于多阶段的CSS动画，你可以创建CSS `@keyframes`。关键帧允许你定义多个动画点，称为关键帧，以定义更复杂的动画。

基本示例

在本例中，我们将制作一个基本的背景动画，循环显示所有颜色。

```
@keyframes rainbow-background {
         0%      { background-color: #ff0000; }
     8.333%      { background-color: #ff8000; }
    16.667%      { background-color: #ffff00; }
    25.000%      { background-color: #80ff00; }
    33.333%      { background-color: #00ff00; }
    41.667%      { background-color: #00ff80; }
    50.000%      { background-color: #00ffff; }
    58.333%      { background-color: #0080ff; }
    66.667%      { background-color: #0000ff; }
    75.000%      { background-color: #8000ff; }
    83.333%      { background-color: #ff00ff; }
    91.667%      { background-color: #ff0080; }
    100.00%      { background-color: #ff0000; }
}

.RainbowBackground {
    animation: rainbow-background 5s infinite;
}
```

查看结果

这里有几个不同的点需要注意。首先是实际的@keyframes语法。

```
@keyframes rainbow-background{
```

这将动画的名称设置为 rainbow-background。

Chapter 27: Animations

Transition

Parameter	Details
property	Either the CSS property to transition on, or `all`, which specifies all transition-able properties.
duration	Transition time, either in seconds or milliseconds.
timing-function	Specifies a function to define how intermediate values for properties are computed. Common values are `ease`, `linear`, and `step-end`. Check out the easing function cheat-sheet for more.
delay	Amount of time, in seconds or milliseconds, to wait before playing the animation.

@keyframes

[from | to | **<percentage>**]	You can either specify a set time with a percentage value, or two percentage values, ie `10%, 20%`, for a period of time where the keyframe's set attributes are set.
`block`	Any amount of CSS attributes for the keyframe.

Section 27.1: Animations with keyframes

For multi-stage CSS animations, you can create CSS `@keyframes`. Keyframes allow you to define multiple animation points, called a keyframe, to define more complex animations.

Basic Example

In this example, we'll make a basic background animation that cycles between all colors.

```
@keyframes rainbow-background {
         0%      { background-color: #ff0000; }
     8.333%      { background-color: #ff8000; }
    16.667%      { background-color: #ffff00; }
    25.000%      { background-color: #80ff00; }
    33.333%      { background-color: #00ff00; }
    41.667%      { background-color: #00ff80; }
    50.000%      { background-color: #00ffff; }
    58.333%      { background-color: #0080ff; }
    66.667%      { background-color: #0000ff; }
    75.000%      { background-color: #8000ff; }
    83.333%      { background-color: #ff00ff; }
    91.667%      { background-color: #ff0080; }
    100.00%      { background-color: #ff0000; }
}

.RainbowBackground {
    animation: rainbow-background 5s infinite;
}
```

View Result

There's a few different things to note here. First, the actual `@keyframes` syntax.

```
@keyframes rainbow-background{
```

This sets the name of the animation to `rainbow-background`.

```
0%     { background-color: #ff0000; }
```

这是动画中关键帧的定义。第一部分，即这里的0%，定义了关键帧在动画中的位置。0%表示从开始算起动画总时间的0%。

动画会自动在关键帧之间过渡。因此，通过在8.333%设置下一个背景颜色，动画将在8.333%的时间内平滑地完成这两个关键帧之间的过渡。

```
.RainbowBackground {
    animation: rainbow-background 5s infinite;
}
```

这段代码将我们的动画应用到所有带有.RainbowBackground类的元素上。

实际的动画属性接受以下参数。

- **animation-name**：动画的名称。在本例中为 rainbow-background
- **animation-duration**：动画持续的时间，本例中为5秒。
- animation-iteration-count（可选）：动画循环的次数。本例中，动画将无限循环。默认情况下，动画只播放一次。
- animation-delay（可选）：指定动画开始前等待的时间。默认值为0秒，可以取负值。例如，-2s 表示动画从循环的第2秒开始。
- animation-timing-function（可选）：指定动画的速度曲线。默认值为 ease，动画开始时慢，加速，然后结束时慢。

在这个特定的例子中，0% 和 100% 关键帧都指定了 { background-color: #ff0000; }。当两个或多个关键帧共享同一状态时，可以用一条语句来指定它们。在本例中，两个 0% 和 100%行可以用以下单行替代：

```
0%, 100%     { background-color: #ff0000; }
```

跨浏览器兼容性

对于较旧的基于 WebKit 的浏览器，需要在 @keyframes 声明和 animation 属性上都使用厂商前缀，如下所示：

```
@-webkit-keyframes{}

-webkit-animation: ...
```

第27.2节：带有过渡属性的动画

对于简单动画非常有用，CSS的transition属性允许基于数值的CSS属性在状态之间进行动画过渡。

示例

```
.Example{
    height: 100px;
    background: #fff;
}

.Example:hover{
    height: 120px;
```



```
0%     { background-color: #ff0000; }
```

This is the definition for a keyframe within the animation. The first part, the `0%` in the case, defines where the keyframe is during the animation. The `0%` implies it is 0% of the total animation time from the beginning.

The animation will automatically transition between keyframes. So, by setting the next background color at `8.333%`, the animation will smoothly take 8.333% of the time to transition between those keyframes.

```
.RainbowBackground {
    animation: rainbow-background 5s infinite;
}
```

This code attaches our animation to all elements which have the `.RainbowBackground` class.

The actual animation property takes the following arguments.

- **animation-name**: The name of our animation. In this case, `rainbow-background`
- **animation-duration**: How long the animation will take, in this case 5 seconds.
- **animation-iteration-count (Optional)**: The number of times the animation will loop. In this case, the animation will go on indefinitely. By default, the animation will play once.
- **animation-delay (Optional)**: Specifies how long to wait before the animation starts. It defaults to 0 seconds, and can take negative values. For example, `-2s` would start the animation 2 seconds into its loop.
- **animation-timing-function (Optional)**: Specifies the speed curve of the animation. It defaults to `ease`, where the animation starts slow, gets faster and ends slow.

In this particular example, both the `0%` and `100%` keyframes specify `{ background-color: #ff0000; }`. Wherever two or more keyframes share a state, one may specify them in a single statement. In this case, the two `0%` and `100%` lines could be replaced with this single line:

```
0%, 100%     { background-color: #ff0000; }
```

Cross-browser compatibility

For older WebKit-based browsers, you'll need to use the vendor prefix on both the `@keyframes` declaration and the `animation` property, like so:

```
@-webkit-keyframes{}

-webkit-animation: ...
```

Section 27.2: Animations with the transition property

Useful for simple animations, the CSS `transition` property allows number-based CSS properties to animate between states.

Example

```
.Example{
    height: 100px;
    background: #fff;
}

.Example:hover{
    height: 120px;
```

```
    background: #ff0000;
}
```

查看结果

默认情况下，鼠标悬停在带有.Example类的元素上时，元素的高度会立即跳变到120px，背景颜色变为红色（#ff0000）。

通过添加transition属性，我们可以使这些变化随着时间发生：

```
.Example{
    ...
    transition: all 400ms ease;
}
```

查看结果

all值将过渡应用于所有兼容的（基于数值的）属性。任何兼容的属性名称（例如height或top）都可以替代该关键字。

400毫秒 指定过渡所需的时间。在此情况下，元素高度的变化将花费400毫秒完成。

最后，值 ease 是动画函数，决定动画的播放方式。 ease 意味着开始慢，逐渐加速，然后再次慢速结束。其他值有 linear、ease-out 和 ease-in。

跨浏览器兼容性

transition 属性在所有主流浏览器中通常都得到良好支持，IE 9 除外。对于早期版本的 Firefox 和基于 Webkit 的浏览器，请使用供应商前缀，如下所示：

```
.Example{
    transition:         all 400ms ease;
    -moz-transition:    all 400ms ease;
    -webkit-transition: all 400ms ease;
}
```

注意： transition 属性可以对任意两个数值之间的变化进行动画，无论单位如何。它也可以在单位之间过渡，例如从 100px 到 50vh。然而，它不能在数值和默认或自动值之间过渡，比如将元素高度从 100px 过渡到 auto。

第27.3节：语法示例

我们的第一个语法示例展示了使用所有可用属性/参数的动画简写属性：

```
animation: 3秒 ease-in 1秒 2 反向 双向 暂停
    slidein;
/*      持续时间 | 时间函数 | 延迟 | 迭代次数 | 方向 | 填充模式 | 播放状态 | 名称   */
```

我们的第二个例子稍微简单一些，展示了某些属性可以被省略：

```
animation: 3秒 线性 1秒 slidein;
/*      持续时间 | 时间函数 | 延迟 | 名称   */
```



```
    background: #ff0000;
}
```

View Result

By default, hovering over an element with the `.Example` class would immediately cause the element's height to jump to `120px` and its background color to red (`#ff0000`).

By adding the `transition` property, we can cause these changes to occur over time:

```
.Example{
    ...
    transition: all 400ms ease;
}
```

View Result

The `all` value applies the transition to all compatible (numbers-based) properties. Any compatible property name (such as `height` or `top`) can be substituted for this keyword.

`400ms` specifies the amount of time the transition takes. In this case, the element's change in height will take 400 milliseconds to complete.

Finally, the value `ease` is the animation function, which determines how the animation is played. `ease` means start slow, speed up, then end slow again. Other values are `linear`, `ease-out`, and `ease-in`.

Cross-Browser Compatibility

The `transition` property is generally well-supported across all major browsers, excepting IE 9. For earlier versions of Firefox and Webkit-based browsers, use vendor prefixes like so:

```
.Example{
    transition:         all 400ms ease;
    -moz-transition:    all 400ms ease;
    -webkit-transition: all 400ms ease;
}
```

Note: The `transition` property can animate changes between any two numerical values, regardless of unit. It can also transition between units, such as `100px` to `50vh`. However, it cannot transition between a number and a default or automatic value, such as transitioning an element's height from `100px` to `auto`.

Section 27.3: Syntax Examples

Our first syntax example shows the animation shorthand property using all of the available properties/parameters:

```
  animation: 3s ease-in 1s 2 reverse both paused
      slidein;
  /*      duration | timing-function | delay | iteration-count | direction | fill-mode | play-
state | name   */
```

Our second example is a little more simple, and shows that some properties can be omitted:

```
  animation: 3s linear 1s slidein;
  /*      duration | timing-function | delay | name   */
```

我们的第三个示例展示了最简洁的声明。请注意，必须声明 animation-name 和 animation-duration ：

```
animation: 3秒          slidein;
/*         持续时间 | 名称 */
```

同样值得一提的是，使用 animation 简写时，属性的顺序会产生影响。显然，浏览器可能会将你的持续时间和延迟混淆。

如果你不喜欢简写，也可以跳过简写属性，单独写出每个属性：

```
animation-duration: 3秒;
animation-timing-function: 缓入;
animation-delay: 1秒;
animation-iteration-count: 2;
animation-direction: 反向;
animation-fill-mode: 两者;
animation-play-state: 暂停;
animation-name: slidein;
```

第27.4节：使用 `will-change` 属性提升动画性能

在创建动画和其他GPU密集型操作时，理解 will-change 属性非常重要。

CSS关键帧动画和transition属性都使用GPU加速。通过将计算任务卸载到设备的GPU，性能得以提升。这是通过创建绘制层（页面中单独渲染的部分）来实现的，这些绘制层被卸载到GPU进行计算。 will-change属性告诉浏览器哪些内容将要动画，从而允许浏览器创建更小的绘制区域，进而提升性能。

will-change属性接受一个以逗号分隔的将要动画的属性列表。例如，如果你计划对一个对象进行变换并改变其不透明度，你可以指定：

```
.Example{
    …
    will-change: transform, opacity;
}
```

注意：请谨慎使用will-change。为页面上的每个元素设置will-change可能导致性能问题，因为浏览器可能会尝试为每个元素创建绘制层，显著增加GPU的处理负担。

Our third example shows the most minimal declaration. Note that the animation-name and animation-duration must be declared:

```
animation: 3s          slidein;
/*         duration | name */
```

It's also worth mentioning that when using the animation shorthand the order of the properties makes a difference. Obviously the browser may confuse your duration with your delay.

If brevity isn't your thing, you can also skip the shorthand property and write out each property individually:

```
animation-duration: 3s;
animation-timing-function: ease-in;
animation-delay: 1s;
animation-iteration-count: 2;
animation-direction: reverse;
animation-fill-mode: both;
animation-play-state: paused;
animation-name: slidein;
```

Section 27.4: Increasing Animation Performance Using the `will-change` Attribute

When creating animations and other GPU-heavy actions, it's important to understand the `will-change` attribute.

Both CSS keyframes and the `transition` property use GPU acceleration. Performance is increased by offloading calculations to the device's GPU. This is done by creating paint layers (parts of the page that are individually rendered) that are offloaded to the GPU to be calculated. The `will-change` property tells the browser what will animate, allowing the browser to create smaller paint areas, thus increasing performance.

The `will-change` property accepts a comma-separated list of properties to be animated. For example, if you plan on transforming an object and changing its opacity, you would specify:

```
.Example{
    ...
    will-change: transform, opacity;
}
```

Note: Use `will-change` sparingly. Setting `will-change` for every element on a page can cause performance problems, as the browser may attempt to create paint layers for every element, significantly increasing the amount of processing done by the GPU.

第28章：二维变换

函数/参数	详情
rotate(x)	定义绕Z轴固定点旋转元素的变换
translate(x,y)	移动元素在X轴和Y轴上的位置
translateX(x)	移动元素在X轴上的位置
translateY(y)	移动元素在Y轴上的位置
scale(x,y)	修改元素在X轴和Y轴上的大小
scaleX(x)	修改元素在X轴上的大小
scaleY(y)	修改元素在Y轴上的大小
skew(x,y)	剪切映射，或称平移变换，使元素的每个点在每个方向上按一定角度发生扭曲
skewX(x)	水平剪切映射，使元素的每个点在水平方向上按一定角度发生扭曲
skewY(y)	垂直剪切映射，使元素的每个点在垂直方向上按一定角度发生扭曲
matrix()	以变换矩阵的形式定义二维变换。
angle	元素应旋转或剪切的角度（取决于所使用的函数）。角度可以用度（deg）、梯度（grad）、弧度（rad）或圈数（turn）表示。在 skew()函数中，第二个角度是可选的。如果未提供，则Y轴方向无（0）剪切。
length-or-percentage	该距离以长度或百分比表示，表示元素应移动的距离 已翻译。在translate()函数中，第二个长度或百分比是可选的。如果未提供，则Y轴不会有（0）平移。
缩放因子	一个数字，定义元素在指定轴上应缩放多少倍。 在scale()函数中，第二个缩放因子是可选的。如果未提供，第一个缩放因子也将应用于Y轴。

第28.1节：旋转

HTML

```
<div class="rotate"></div>
```

CSS

```
.rotate {
    宽度: 100px;
    高度: 100px;
    背景色: 青色;
    变换: rotate(45deg);
}
```

此示例将使div顺时针旋转45度。旋转中心位于div的中心，距离左侧50%，距离顶部50%。您可以通过设置transform-origin属性来更改旋转中心。

```
transform-origin: 100% 50%;
```

上述示例将旋转中心设置为右侧末端的中间位置。

Chapter 28: 2D Transforms

Function/Parameter	Details
rotate(x)	Defines a transformation that moves the element around a fixed point on the Z axis
translate(x,y)	Moves the position of the element on the X and Y axis
translateX(x)	Moves the position of the element on the X axis
translateY(y)	Moves the position of the element on the Y axis
scale(x,y)	Modifies the size of the element on the X and Y axis
scaleX(x)	Modifies the size of the element on the X axis
scaleY(y)	Modifies the size of the element on the Y axis
skew(x,y)	Shear mapping, or transvection, distorting each point of an element by a certain angle in each direction
skewX(x)	Horizontal shear mapping distorting each point of an element by a certain angle in the horizontal direction
skewY(y)	Vertical shear mapping distorting each point of an element by a certain angle in the vertical direction
matrix()	Defines a 2D transformation in the form of a transformation matrix.
angle	The angle by which the element should be rotated or skewed (depending on the function with which it is used). Angle can be provided in degrees (deg), gradians (grad), radians (rad) or turns (turn). In skew() function, the second angle is optional. If not provided, there will be no (0) skew in Y-axis.
length-or-percentage	The distance expressed as a length or a percentage by which the element should be translated. In translate() function, the second length-or-percentage is optional. If not provided, then there would be no (0) translation in Y-axis.
scale-factor	A number which defines how many times the element should be scaled in the specified axis. In scale() function, the second scale-factor is optional. If not provided, the first scale-factor will be applied for Y-axis also.

Section 28.1: Rotate

HTML

```
<div class="rotate"></div>
```

CSS

```
.rotate {
    width: 100px;
    height: 100px;
    background: teal;
    transform: rotate(45deg);
}
```

This example will rotate the div by 45 degrees clockwise. The center of rotation is in the center of the div, 50% from left and 50% from top. You can change the center of rotation by setting the transform-origin property.

```
transform-origin: 100% 50%;
```

The above example will set the center of rotation to the middle of the right side end.

第28.2节：缩放

HTML

```
<div class="scale"></div>
```

CSS

```
.scale {
    宽度: 100像素;
    height: 100px;
    background: teal;
    transform: scale(0.5, 1.3);
}
```

此示例将在X轴上将div缩放为100px * 0.5 = 50px，在Y轴上缩放为100px * 1.3 = 130px。

变换的中心位于div的中心，距离左侧50%，距离顶部50%。

第28.3节：倾斜

HTML

```
<div class="skew"></div>
```

CSS

```
.skew {
    宽度: 100像素;
    height: 100px;
    背景: 青绿色;
    变换: 倾斜(20度, -30度);
}
```

此示例将在X轴上将div倾斜20度，在Y轴上倾斜-30度。
变换的中心位于div的中心，距离左侧50%，距离顶部50%。

请看结果 here。

第28.4节：多重变换

可以在一个属性中对元素应用多个变换，如下所示：

```
transform: rotate(15deg) translateX(200px);
```

这将使元素顺时针旋转15度，然后向右平移200像素。

在链式变换中，**坐标系随元素移动**。这意味着平移不会是
水平的，而是在顺时针旋转15度的轴上，如下图所示：

Section 28.2: Scale

HTML

```
<div class="scale"></div>
```

CSS

```
.scale {
    width: 100px;
    height: 100px;
    background: teal;
    transform: scale(0.5, 1.3);
}
```

This example will scale the div to `100px * 0.5 = 50px` on the X axis and to `100px * 1.3 = 130px` on the Y axis.

The center of the transform is in the center of the div, `50%` from left and `50%` from top.

Section 28.3: Skew

HTML

```
<div class="skew"></div>
```

CSS

```
.skew {
    width: 100px;
    height: 100px;
    background: teal;
    transform: skew(20deg, -30deg);
}
```

This example will skew the div by 20 degrees on the X axis and by - 30 degrees on the Y axis.
The center of the transform is in the center of the div, `50%` from left and `50%` from top.

See the result here.

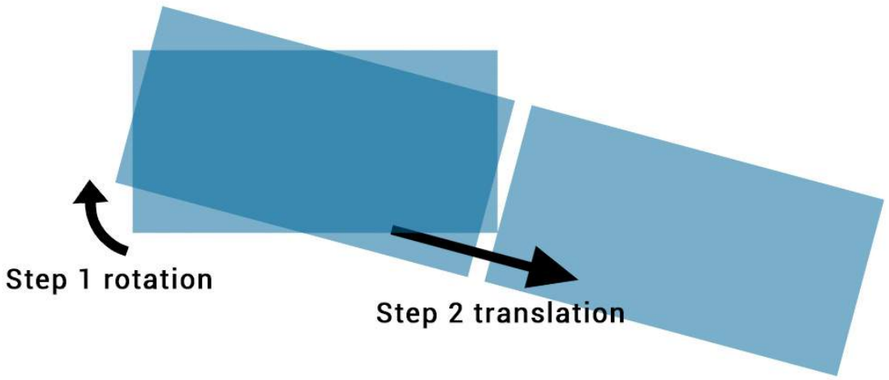
Section 28.4: Multiple transforms

Multiple transforms can be applied to an element in one property like this:

```
transform: rotate(15deg) translateX(200px);
```

This will rotate the element 15 degrees clockwise and then translate it 200px to the right.

In chained transforms, **the coordinate system moves with the element**. This means that the translation won't be horizontal but on an axis rotate 15 degrees clockwise as shown in the following image:



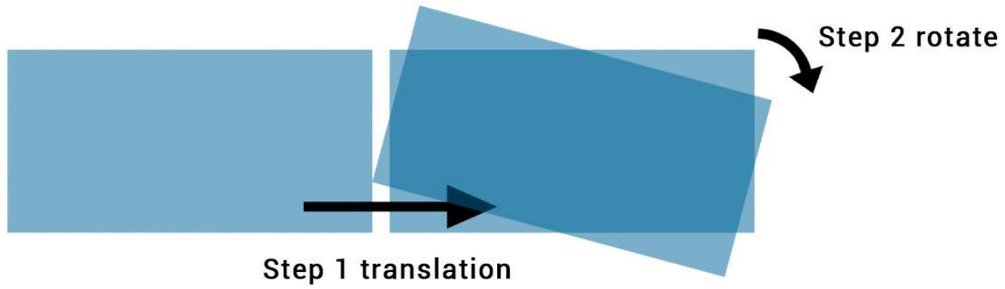
改变变换的顺序会改变输出。第一个示例将不同于

```
transform: translateX(200px) rotate(15deg);
```

```
<div class="transform"></div>
```

```
.transform {
  transform: rotate(15deg) translateX(200px);
}
```

如图所示：



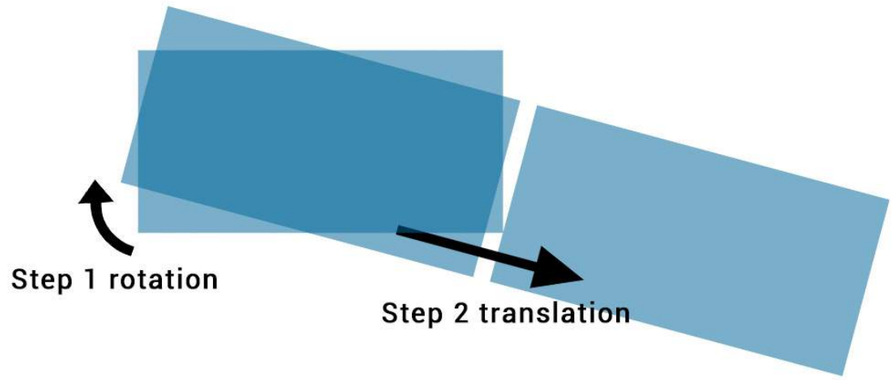
第28.5节：平移

HTML

```
<div class="translate"></div>
```

CSS

```
.translate {
    宽度: 100像素;
    height: 100px;
    background: teal;
    transform: translate(200px, 50%);
```

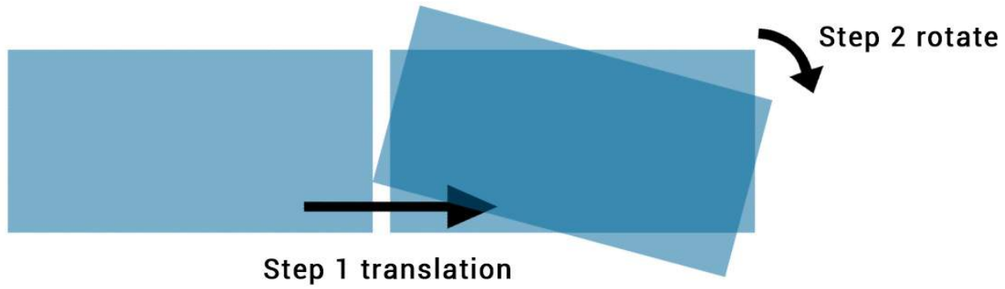
Changing the order of the transforms will change the output. The first example will be different to

```
transform: translateX(200px) rotate(15deg);
```

```
<div class="transform"></div>
```

```
.transform {
  transform: rotate(15deg) translateX(200px);
}
```

As shown in this image:



Section 28.5: Translate

HTML

```
<div class="translate"></div>
```

CSS

```
.translate {
    width: 100px;
    height: 100px;
    background: teal;
    transform: translate(200px, 50%);
```

```
}
```

此示例将在X轴上移动div 200px，在Y轴上移动100px * 50% = 50px。

你也可以只指定单轴的平移。

在X轴上：

```
.translate {
    transform: translateX(200px);
}
```

在Y轴上：

```
.translate {
    transform: translateY(50%);
}
```

第28.6节：变换原点

变换是相对于由transform-origin属性定义的点进行的。

该属性接受两个值：transform-origin: X Y;

在下面的示例中，第一个div（.tl）围绕左上角旋转，transform-origin: 0 0；第二个div（.tr）围绕其右上角变换，transform-origin: 100% 0。旋转在悬停时应用：

HTML：

```
<div class="transform originl"></div>
<div class="transform origin2"></div>
```

CSS：

```
.transform {
    display: inline-block;
    width: 200px;
    height: 100px;
    background: teal;
    transition: transform 1s;
}

.origin1 {
    transform-origin: 0 0;
}

.origin2 {
    transform-origin: 100% 0;
}

.transform:hover {
    transform: rotate(30deg);
}
```

transform-origin 属性的默认值是 50% 50%，即元素的中心。


```
}
```

This example will move the div by 200px on the X axis and by `100px * 50% = 50px` on the Y axis.

You can also specify translations on a single axis.

On the X axis:

```
.translate {
    transform: translateX(200px);
}
```

On the Y axis:

```
.translate {
    transform: translateY(50%);
}
```

Section 28.6: Transform Origin

Transformations are done with respect to a point which is defined by the `transform-origin` property.

The property takes 2 values : **`transform-origin`**: X Y;

In the following example the first div (`.tl`) is rotate around the top left corner with **`transform-origin`**: `0 0`; and the second (`.tr`)is transformed around it's top right corner with **`transform-origin`**: `100% 0`. The rotation is applied **on hover** :

HTML:

```
<div class="transform originl"></div>
<div class="transform origin2"></div>
```

CSS:

```
.transform {
    display: inline-block;
    width: 200px;
    height: 100px;
    background: teal;
    transition: transform 1s;
}

.origin1 {
    transform-origin: 0 0;
}

.origin2 {
    transform-origin: 100% 0;
}

.transform:hover {
    transform: rotate(30deg);
}
```

The default value for the transform-origin property is `50% 50%` which is the center of the element.

第29章：3D变换

第29.1节：使用3D变换的指南针指针或针形

CSS

```
div.needle {
   margin: 100px;
   height: 150px;
   宽度: 150像素;
   transform: rotateY(85deg) rotateZ(45deg);
   /* 仅用于展示 */
   background-image: 线性渐变(向左上方, #555 0%, #555 40%, #444 50%, #333 97%);
   box-shadow: 内阴影 6px 6px 22px 8px #272727;
}
```

HTML

```
<div class='needle'></div>
```

在上述示例中，使用3D变换创建了一个针形或指南针指针形状。通常，当我们对一个元素应用rotate变换时，旋转仅发生在Z轴上，最多我们只能得到菱形。但当在其上叠加rotateY变换时，元素在Y轴方向被压缩，因此看起来像一根针。Y轴旋转越多，元素看起来越被压缩。

上述示例的输出将是一根针尖朝下的针。若要创建一根针底部朝下的针，旋转应沿X轴而非Y轴进行。因此，transform属性的值应类似于rotateX(85deg) rotateZ(45deg);。

这个笔使用了类似的方法来创建一个类似Safari标志或指南针表盘的形状。

无变换元素的截图：

仅有二维变换元素的截图：

Chapter 29: 3D Transforms

Section 29.1: Compass pointer or needle shape using 3D transforms

CSS

```
div.needle {
  margin: 100px;
  height: 150px;
  width: 150px;
  transform: rotateY(85deg) rotateZ(45deg);
  /* presentational */
  background-image: linear-gradient(to top left, #555 0%, #555 40%, #444 50%, #333 97%);
  box-shadow: inset 6px 6px 22px 8px #272727;
}
```

HTML

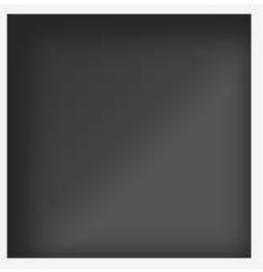
```
<div class='needle'></div>
```

In the above example, a needle or compass pointer shape is created using 3D transforms. Generally when we apply the `rotate` transform on an element, the rotation happens only in the Z-axis and at best we will end up with diamond shapes only. But when a `rotateY` transform is added on top of it, the element gets squeezed in the Y-axis and thus ends up looking like a needle. The more the rotation of the Y-axis the more squeezed the element looks.

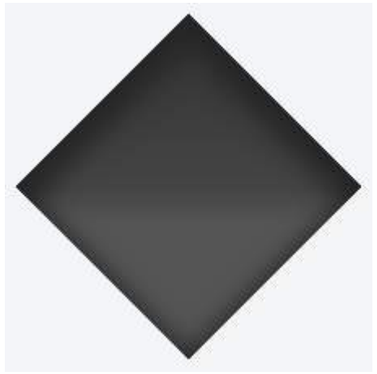
The output of the above example would be a needle resting on its tip. For creating a needle that is resting on its base, the rotation should be along the X-axis instead of along Y-axis. So the `transform` property's value would have to be something like `rotateX(85deg) rotateZ(45deg);`.

This pen uses a similar approach to create something that resembles the Safari logo or a compass dial.

Screenshot of element with no transform:



Screenshot of element with only 2D transform:



具有三维变换元素的截图：

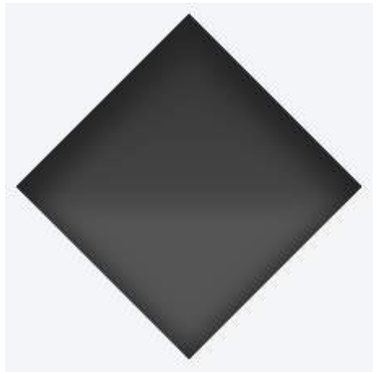
第29.2节：带阴影的3D文本效果

HTML：

```
<div id="title">
  <h1 data-content="HOVER">HOVER</h1>
</div>
```

CSS：

```
*{margin:0;padding:0;}
html,body{height:100%;width:100%;overflow:hidden;background:#0099CC;}
#title{
  position:absolute;
  top:50%; left:50%;
  transform:translate(-50%,-50%);
  perspective-origin:50% 50%;
  perspective:300px;
}
h1{
  text-align:center;
  font-size:12vmin;
  font-family: 'Open Sans', sans-serif;
  color:rgba(0,0,0,0.8);
  line-height:1em;
  transform:rotateY(50deg);
  perspective:150px;
  perspective-origin:0% 50%;
```

Screenshot of element with 3D transform:

Section 29.2: 3D text effect with shadow

HTML:

```
<div id="title">
  <h1 data-content="HOVER">HOVER</h1>
</div>
```

CSS:

```
*{margin:0;padding:0;}
html,body{height:100%;width:100%;overflow:hidden;background:#0099CC;}
#title{
  position:absolute;
  top:50%; left:50%;
  transform:translate(-50%,-50%);
  perspective-origin:50% 50%;
  perspective:300px;
}
h1{
  text-align:center;
  font-size:12vmin;
  font-family: 'Open Sans', sans-serif;
  color:rgba(0,0,0,0.8);
  line-height:1em;
  transform:rotateY(50deg);
  perspective:150px;
  perspective-origin:0% 50%;
```

```
}
h1:after{
  content:attr(data-content);
  position:absolute;
  left:0;top:0;
  transform-origin:50% 100%;
  transform:rotateX(-90deg);
  color:#0099CC;
}
#title:before{
  content:'';
  position:absolute;
  top:-150%; left:-25%;
  width:180%; height:328%;
  background:rgba(255,255,255,0.7);
  transform-origin: 0 100%;
  transform: translatez(-200px) rotate(40deg) skewX(35deg);
  border-radius:0 0 100% 0;
}
```

查看带有额外悬停效果的示例

在此示例中，文本经过变换，看起来像是远离用户进入屏幕内部。

阴影也相应地进行了变换，以跟随文本。由于它是用伪元素和`data`
属性制作的，因此继承了其父元素（H1标签）的变换。

白色的“光”是用#title元素上的伪元素制作的。它被倾斜并使用了圆角边框（border-radius）来实现圆角效果。

第29.3节：背面可见性

backface-visibility 属性与3D变换相关。

通过3D变换和backface-visibility属性，你可以旋转一个元素，使得元素的原始正面不再朝向屏幕。


```
}
h1:after{
  content:attr(data-content);
  position:absolute;
  left:0;top:0;
  transform-origin:50% 100%;
  transform:rotateX(-90deg);
  color:#0099CC;
}
#title:before{
  content:'';
  position:absolute;
  top:-150%; left:-25%;
  width:180%; height:328%;
  background:rgba(255,255,255,0.7);
  transform-origin: 0 100%;
  transform: translatez(-200px) rotate(40deg) skewX(35deg);
  border-radius:0 0 100% 0;
}
```

View example with additional hover effect

In this example, the text is transformed to make it look like it is going into the screen away from the user.

The shadow is transformed accordingly so it follows the text. As it is made with a pseudo element and the `data` attribute, it inherits the transforms form it's parent (the H1 tag).

The white "light" is made with a pseudo element on the `#title` element. It is skewed and uses border-radius for the rounded corner.

Section 29.3: backface-visibility

The `backface-visibility` property relates to 3D transforms.

With 3D transforms and the `backface-visibility` property, you're able to rotate an element such that the original front of an element no longer faces the screen.

例如，这将使元素翻转远离屏幕：

JSFIDDLE

```
<div class="flip">Loren ipsum</div>
<div class="flip back">Lorem ipsum</div>
```

```
.flip {
-webkit-transform: rotateY(180deg);
-moz-transform: rotateY(180deg);
-ms-transform: rotateY(180deg);
-webkit-backface-visibility: visible;
-moz-backface-visibility:    visible;
-ms-backface-visibility:     visible;
}

.flip.back {
-webkit-backface-visibility: hidden;
  -moz-backface-visibility:    hidden;
  -ms-backface-visibility:     hidden;
}
```

Firefox 10 及以上版本和 IE 10 及以上版本支持无前缀的backface-visibility。Opera、Chrome、Safari、iOS 和 Android 都需要-webkit-backface-visibility。

它有4个取值：

1. visible（默认）- 元素即使背对屏幕也始终可见。
2. hidden - 元素背对屏幕时不可见。
3. inherit - 该属性将从其父元素继承值。
4. initial - 将属性设置为其默认值，即可见

第29.4节：3D立方体

3D变换可用于创建多种3D形状。以下是一个简单的3D CSS立方体示例：

HTML：

```
<div class="cube">
  <div class="cubeFace"></div>
  <div class="cubeFace face2"></div>
</div>
```

CSS：

```
body {
  perspective-origin: 50% 100%;
  perspective: 1500px;
  overflow: hidden;
}
.cube {
  position: relative;
  padding-bottom: 20%;
  transform-style: preserve-3d;
  transform-origin: 50% 100%;
  transform: rotateY(45deg) rotateX(0);
}
.cubeFace {
```


For example, this would flip an element away from the screen:

JSFIDDLE

```
<div class="flip">Loren ipsum</div>
<div class="flip back">Lorem ipsum</div>
```

```
.flip {
  -webkit-transform: rotateY(180deg);
  -moz-transform: rotateY(180deg);
  -ms-transform: rotateY(180deg);
  -webkit-backface-visibility: visible;
  -moz-backface-visibility:    visible;
  -ms-backface-visibility:     visible;
}

.flip.back {
  -webkit-backface-visibility: hidden;
  -moz-backface-visibility:    hidden;
  -ms-backface-visibility:     hidden;
}
```

Firefox 10+ and IE 10+ support `backface-visibility` without a prefix. Opera, Chrome, Safari, iOS, and Android all need `-webkit-backface-visibility`.

It has 4 values:

1. **visible** (default) - the element will always be visible even when not facing the screen.
2. **hidden** - the element is not visible when not facing the screen.
3. **inherit** - the property will gets its value from the its parent element
4. **initial** - sets the property to its default, which is visible

Section 29.4: 3D cube

3D transforms can be use to create many 3D shapes. Here is a simple 3D CSS cube example:

HTML:

```
<div class="cube">
  <div class="cubeFace"></div>
  <div class="cubeFace face2"></div>
</div>
```

CSS:

```
body {
  perspective-origin: 50% 100%;
  perspective: 1500px;
  overflow: hidden;
}
.cube {
  position: relative;
  padding-bottom: 20%;
  transform-style: preserve-3d;
  transform-origin: 50% 100%;
  transform: rotateY(45deg) rotateX(0);
}
.cubeFace {
```

```
  position: absolute;
  top: 0;
  left: 40%;
  width: 20%;
  height: 100%;
  margin: 0 auto;
  transform-style: inherit;
  background: #C52329;
  box-shadow: inset 0 0 0 5px #333;
  transform-origin: 50% 50%;
  transform: rotateX(90deg);
  backface-visibility: hidden;
}
.face2 {
  transform-origin: 50% 50%;
  transform: rotatez(90deg) translateX(100%) rotateY(90deg);
}
.cubeFace:before, .cubeFace:after {
  content: '';
  position: absolute;
  width: 100%;
  height: 100%;
  transform-origin: 0 0;
  background: inherit;
  box-shadow: inherit;
  backface-visibility: inherit;
}
.cubeFace:before {
  top: 100%;
  left: 0;
  transform: rotateX(-90deg);
}
.cubeFace:after {
  top: 0;
  left: 100%;
  transform: rotateY(90deg);
}
```

查看此示例

演示中添加了额外的样式，并在悬停时应用变换以查看立方体的6个面。

应注意：

- 4个面由伪元素制作
- 应用了链式变换


```
  position: absolute;
  top: 0;
  left: 40%;
  width: 20%;
  height: 100%;
  margin: 0 auto;
  transform-style: inherit;
  background: #C52329;
  box-shadow: inset 0 0 0 5px #333;
  transform-origin: 50% 50%;
  transform: rotateX(90deg);
  backface-visibility: hidden;
}
.face2 {
  transform-origin: 50% 50%;
  transform: rotatez(90deg) translateX(100%) rotateY(90deg);
}
.cubeFace:before, .cubeFace:after {
  content: '';
  position: absolute;
  width: 100%;
  height: 100%;
  transform-origin: 0 0;
  background: inherit;
  box-shadow: inherit;
  backface-visibility: inherit;
}
.cubeFace:before {
  top: 100%;
  left: 0;
  transform: rotateX(-90deg);
}
.cubeFace:after {
  top: 0;
  left: 100%;
  transform: rotateY(90deg);
}
```

View this example

Additional styling is added in the demo and a transform is applied on hover to view the 6 faces of the cube.

Should be noted that:

- 4 faces are made with pseudo elements
- chained transforms are applied

第30章：滤镜属性

值	描述
模糊(x)	将图像模糊x像素。
亮度(x)	当值高于1.0或100%时，图像会变亮。低于该值时，图像会变暗。
对比度(x)	在任何高于1.0或100%的数值下，提供更多的图像对比度。低于该值时，图像的饱和度会降低。
drop-shadow(h, v, x, y, z)	为图像添加投影。h和v可以为负值。x、y和z为可选参数。
greyscale(x)	以灰度显示图像，最大值为1.0或100%。
hue-rotate(x)	对图像应用色相旋转。
invert(x)	反转图像颜色，最大值为1.0或100%。
opacity(x)	设置图像的不透明度/透明度，最大值为1.0或100%。
饱和(saturate)(x)	使图像在任何高于1.0或100%的值时饱和。低于该值时，图像将开始去饱和。
sepia(x)	将图像转换为最大值为1.0或100%的棕褐色调。

第30.1节：模糊

HTML

```
<img src='donald-duck.png' alt='唐老鸭' title='唐老鸭' />
```

CSS

```
img {
-webkit-filter: 模糊(1px);
        filter: 模糊(1px);
}
```

结果

让你想揉揉眼镜。

第30.2节：投影（如果可能，改用box-shadow）

Chapter 30: Filter Property

Value	Description
blur(x)	Blurs the image by x pixels.
brightness(x)	Brightens the image at any value above 1.0 or 100%. Below that, the image will be darkened.
contrast(x)	Provides more contrast to the image at any value above 1.0 or 100%. Below that, the image will get less saturated.
drop-shadow(h, v, x, y, z)	Gives the image a drop-shadow. h and v can have negative values. x, y, and z are optional.
greyscale(x)	Shows the image in greyscale, with a maximum value of 1.0 or 100%.
hue-rotate(x)	Applies a hue-rotation to the image.
invert(x)	Inverts the color of the image with a maximum value of 1.0 or 100%.
opacity(x)	Sets how opaque/transparent the image is with a maximum value of 1.0 or 100%.
saturate(x)	Saturates the image at any value above 1.0 or 100%. Below that, the image will start to de-saturate.
sepia(x)	Converts the image to sepia with a maximum value of 1.0 or 100%.

Section 30.1: Blur

HTML

```
<img src='donald-duck.png' alt='Donald Duck' title='Donald Duck' />
```

CSS

```
img {
    -webkit-filter: blur(1px);
    filter: blur(1px);
}
```

Result

Makes you wanna rub your glasses.

Section 30.2: Drop Shadow (use box-shadow instead if

可能的话）

HTML

```
<p>我的影子总是跟着我。</p>
```

CSS

```
p {
-webkit-filter: 投影滤镜(10px 10px 1px 绿色);
      filter: 投影滤镜(10px 10px 1px 绿色);
}
```

结果

My shadow always follows me.

第30.3节：色相旋转

HTML

```
<img src='donald-duck.png' alt='唐老鸭' title='唐老鸭' />
```

CSS

```
img {
-webkit-filter: 色相旋转(120度);
      filter: 色相旋转(120度);
}
```

结果

第30.4节：多重滤镜值

要使用多个滤镜，请用空格分隔每个值。

HTML

possible)

HTML

```
<p>My shadow always follows me.</p>
```

CSS

```
p {
    -webkit-filter: drop-shadow(10px 10px 1px green);
    filter: drop-shadow(10px 10px 1px green);
}
```

Result

My shadow always follows me.

Section 30.3: Hue Rotate

HTML

```
<img src='donald-duck.png' alt='Donald Duck' title='Donald Duck' />
```

CSS

```
img {
    -webkit-filter: hue-rotate(120deg);
    filter: hue-rotate(120deg);
}
```

Result

Section 30.4: Multiple Filter Values

To use multiple filters, separate each value with a space.

HTML

```
<img src='donald-duck.png' alt='唐老鸭' title='唐老鸭' />
```

CSS

```
img {
-webkit-filter: 亮度(200%) 灰度(100%) 深褐色(100%) 反转(100%);
      filter: 亮度(200%) 灰度(100%) 深褐色(100%) 反转(100%);
}
```

结果

第30.5节：反转颜色

HTML

```
<div></div>
```

CSS

```
div {
      宽度: 100像素;
      height: 100px;
      background-color: white;
      -webkit-filter: invert(100%);
      filter: invert(100%);
}
```

结果

从白色变为黑色。


```
<img src='donald-duck.png' alt='Donald Duck' title='Donald Duck' />
```

CSS

```
img {
    -webkit-filter: brightness(200%) grayscale(100%) sepia(100%) invert(100%);
    filter: brightness(200%) grayscale(100%) sepia(100%) invert(100%);
}
```

Result

Section 30.5: Invert Color

HTML

```
<div></div>
```

CSS

```
div {
    width: 100px;
    height: 100px;
    background-color: white;
    -webkit-filter: invert(100%);
    filter: invert(100%);
}
```

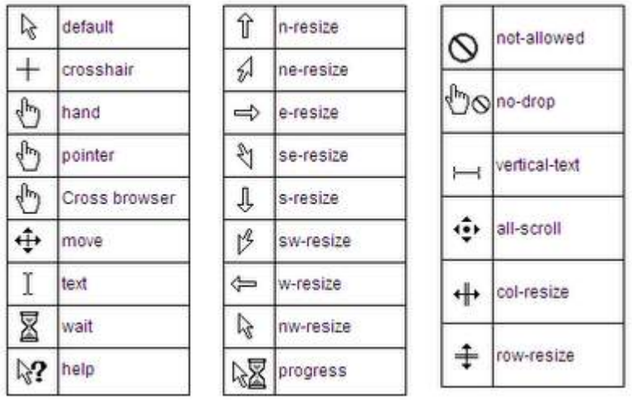
Result

Turns from white to black.

第31章：光标样式

第31.1节：更改光标类型

```
cursor: value;
```



示例：

值	描述
无	该元素不显示光标
自动	默认。浏览器设置光标
帮助	光标表示有帮助信息可用
等待	光标表示程序正在忙碌
移动光标表示某物将被移动	
指针	光标是指针，表示一个链接

第31.2节：pointer-events

pointer-events属性允许控制HTML元素如何响应鼠标/触摸事件。

```
.disabled {
  pointer-events: none;
}
```

在此示例中，

> ‘none’阻止指定HTML元素上的所有点击、状态和光标选项[[1]]

HTML元素的其他有效值有：

- 自动；
- 继承。

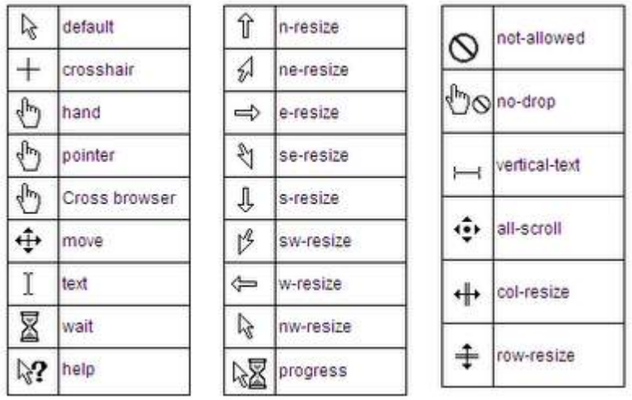
1.https://css-tricks.com/almanac/properties/p/pointer-events/

其他资源：

Chapter 31: Cursor Styling

Section 31.1: Changing cursor type

```
cursor: value;
```



Examples:

Value	Description
none	No cursor is rendered for the element
auto	Default. The browser sets a cursor
help	The cursor indicates that help is available
wait	The cursor indicates that the program is busy
move	The cursor indicates something is to be moved
pointer	The cursor is a pointer and indicates a link

Section 31.2: pointer-events

The pointer-events property allows for control over how HTML elements respond to mouse/touch events.

```
.disabled {
  pointer-events: none;
}
```

In this example,

> 'none' prevents all click, state and cursor options on the specified HTML element [[1]]

Other valid values for HTMl elements are:

- auto;
- inherit.

1. https://css-tricks.com/almanac/properties/p/pointer-events/

Other resources:

- https://developer.mozilla.org/en-US/docs/Web/CSS/pointer-events
- https://davidwalsh.name/pointer-events

第31.3节：caret-color

caret-color CSS属性指定插入符的颜色，插入符是文本和其他内容由用户输入或编辑时在元素中插入点的可见指示器。

HTML

```
<input id="example" />
```

CSS

```
#example {
  caret-color: red;
}
```

资源：

- https://developer.mozilla.org/en-US/docs/Web/CSS/caret-color

- https://developer.mozilla.org/en-US/docs/Web/CSS/pointer-events
- https://davidwalsh.name/pointer-events

Section 31.3: caret-color

The caret-color CSS property specifies the color of the caret, the visible indicator of the insertion point in an element where text and other content is inserted by the user's typing or editing.

HTML

```
<input id="example" />
```

CSS

```
#example {
  caret-color: red;
}
```

Resources:

- https://developer.mozilla.org/en-US/docs/Web/CSS/caret-color

第32章：box-shadow

参数	详细信息
inset	默认情况下，阴影被视为投影阴影。inset关键字将阴影绘制在框架/边框内部。
水平偏移量	水平距离
垂直偏移量	垂直距离
模糊半径	默认值为0。数值不能为负。数值越大，阴影越大且越浅。
缩小。	默认扩散半径为0。正值会使阴影扩展，负值会使阴影收缩。
颜色	可以有多种表示法：颜色关键字、十六进制、rgb()、rgba()、hsl()、hsla()

第32.1节：使用伪元素的仅底部投影

JSFiddle: https://jsfiddle.net/UnsungHero97/80qod7aL/2/

HTML

```
<div class="box_shadow"></div>
```

CSS

```
.box_shadow {
  背景色: #1C90F3;
  宽度: 200px;
  高度: 100px;
  外边距: 50px;
}

.box_shadow:after {
  内容: "";
  宽度: 190px;
  高度: 1px;
  上外边距: 98px;
  左外边距: 5px;
  显示: 块级;
  定位: 绝对;
  z轴索引: -1;
-webkit-box-shadow: 0px 0px 8px 2px 4444444;
-moz-box-shadow: 0px 0px 8px 2px 4444444;
          box-shadow: 0px 0px 8px 2px #444444;
}
```


Chapter 32: box-shadow

Parameters	Details
inset	by default, the shadow is treated as a drop shadow. the inset keyword draws the shadow inside the frame/border.
offset-x	the horizontal distance
offset-y	the vertical distance
blur-radius	0 by default. value cannot be negative. the bigger the value, the bigger and lighter the shadow becomes.
spread-radius	0 by default. positive values will cause the shadow to expand. negative values will cause the shadow to shrink.
color	can be of various notations: a color keyword, hexadecimal, `rgb()`, `rgba()`, `hsl()`, `hsla()`

Section 32.1: bottom-only drop shadow using a pseudo-element

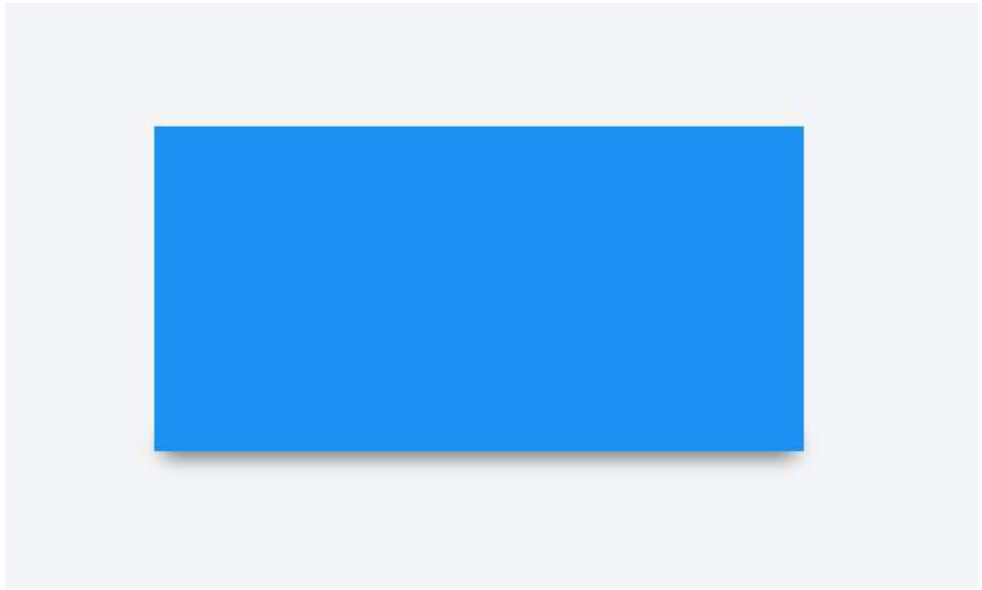
JSFiddle: https://jsfiddle.net/UnsungHero97/80qod7aL/2/

HTML

```
<div class="box_shadow"></div>
```

CSS

```
.box_shadow {
  background-color: #1C90F3;
  width: 200px;
  height: 100px;
  margin: 50px;
}

.box_shadow:after {
  content: "";
  width: 190px;
  height: 1px;
  margin-top: 98px;
  margin-left: 5px;
  display: block;
  position: absolute;
  z-index: -1;
  -webkit-box-shadow: 0px 0px 8px 2px #444444;
     -moz-box-shadow: 0px 0px 8px 2px #444444;
          box-shadow: 0px 0px 8px 2px #444444;
}
```



第32.2节：投影阴影

JSFiddle: https://jsfiddle.net/UnsungHero97/80qod7aL/

HTML

```
<div class="box_shadow"></div>
```

CSS

```
.box_shadow {
-webkit-box-shadow: 0px 0px 10px -1px #444444;
     -moz-box-shadow: 0px 0px 10px -1px #444444;
          box-shadow: 0px 0px 10px -1px #444444;
}
```

第32.3节：内投影阴影

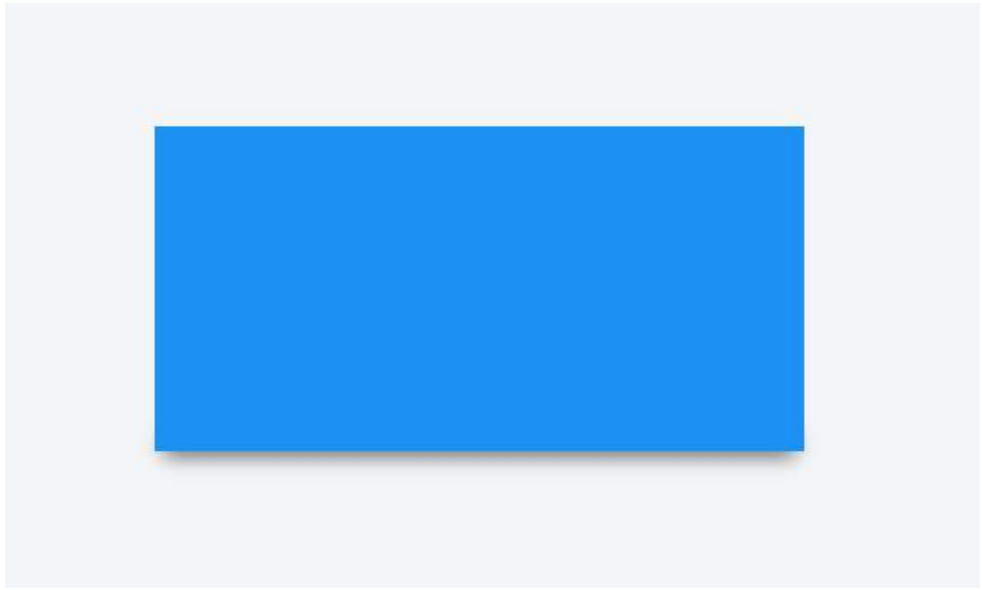
HTML

```
<div class="box_shadow"></div>
```

CSS

```
.box_shadow {
   背景色: #1C90F3;
   宽度: 200px;
   height: 100px;
   margin: 50px;
-webkit-box-shadow: inset 0px 0px 10px 0px #444444;
     -moz-box-shadow: inset 0px 0px 10px 0px #444444;
          box-shadow: inset 0px 0px 10px 0px #444444;
}
```

结果：



Section 32.2: drop shadow

JSFiddle: https://jsfiddle.net/UnsungHero97/80qod7aL/

HTML

```
<div class="box_shadow"></div>
```

CSS

```
.box_shadow {
  -webkit-box-shadow: 0px 0px 10px -1px #444444;
     -moz-box-shadow: 0px 0px 10px -1px #444444;
          box-shadow: 0px 0px 10px -1px #444444;
}
```

Section 32.3: inner drop shadow

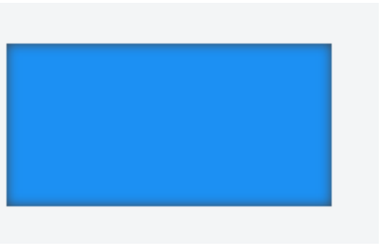
HTML

```
<div class="box_shadow"></div>
```

CSS

```
.box_shadow {
  background-color: #1C90F3;
  width: 200px;
  height: 100px;
  margin: 50px;
  -webkit-box-shadow: inset 0px 0px 10px 0px #444444;
     -moz-box-shadow: inset 0px 0px 10px 0px #444444;
          box-shadow: inset 0px 0px 10px 0px #444444;
}
```

Result:



JSFiddle: https://jsfiddle.net/UnsungHero97/80qod7aL/1/

第32.4节：多重阴影

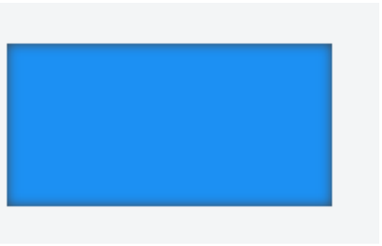
JSFiddle: https://jsfiddle.net/UnsungHero97/80qod7aL/5/

HTML

```
<div class="box_shadow"></div>
```

CSS

```
.box_shadow {
  宽度: 100像素;
  高度: 100像素;
  外边距: 100像素;
  盒子阴影:
    -52像素 -52像素 0像素 0像素 #f65314,
    52像素 -52像素 0像素 0像素 #7cbb00,
    -52像素 52像素 0像素 0像素 #00a1f1,
    52像素 52像素 0像素 0像素 #ffbb00;
}
```

JSFiddle: https://jsfiddle.net/UnsungHero97/80qod7aL/1/

Section 32.4: multiple shadows

JSFiddle: https://jsfiddle.net/UnsungHero97/80qod7aL/5/

HTML

```
<div class="box_shadow"></div>
```

CSS

```
.box_shadow {
  width: 100px;
  height: 100px;
  margin: 100px;
  box-shadow:
    -52px -52px 0px 0px #f65314,
    52px -52px 0px 0px #7cbb00,
    -52px 52px 0px 0px #00a1f1,
    52px 52px 0px 0px #ffbb00;
}
```

第33章：浮动的形状

参数	详情
无	值为 none 表示浮动区域（用于环绕浮动元素内容的区域）不受影响。这是默认/初始值。
basic-shape	指的是 inset()、circle()、ellipse() 或 polygon() 中的一个。使用这些函数之一及其定义形状的值。
形状盒	指margin-box、border-box、padding-box、content-box中的一个。当仅有<shape-box>时提供的（不含 <basic-shape>）此框*是*形状。当与 <basic-shape> 一起使用时，它作为参考框。
图片	当图像作为值提供时，形状是基于指定图像的 alpha 通道计算的。

第33.1节：带有基本形状的外部形状 – circle()

使用shape-outside CSS属性，可以为浮动区域定义形状值，使内联内容绕着形状而不是浮动框进行环绕。

CSS

```
img:nth-of-type(1) {
  shape-outside: circle(80px at 50% 50%);
  float: left;
  width: 200px;
}
img:nth-of-type(2) {
  shape-outside: circle(80px at 50% 50%);
  float: right;
  width: 200px;
}
p {
  text-align: center;
  line-height: 30px; /* 仅用于演示 */
}
```

HTML

```
<img src="http://images.clipartpanda.com/circle-clip-art-circlergb.jpg">
<img src="http://images.clipartpanda.com/circle-clip-art-circlergb.jpg">
<p>某段文字内容需要环绕，使其沿着圆的曲线两侧排列。然后有一些填充文本以使文字足够长。

Lorem Ipsum Dolor Sit Amet....</p>
```

在上述示例中，两个图像实际上都是正方形图像，当文本放置时如果没有shape-outside属性，文本不会环绕圆的两侧。它只会环绕图像的包含框。使用shape-outside后，浮动区域被重新定义为一个圆形，内容会围绕这个通过shape-outside创建的虚拟圆形流动。

用于重新定义浮动区域的虚拟圆形是一个半径为80像素的圆，绘制于图像参考框的中心点。

以下是几张截图，用于说明在使用shape-outside和不使用时内容如何环绕排布。

使用shape-outside的输出效果

Chapter 33: Shapes for Floats

Parameter	Details
none	A value of `none` means that the float area (the area that is used for wrapping content around a float element) is unaffected. This is the default/initial value.
basic-shape	Refers to one among `inset()`, `circle()`, `ellipse()` or `polygon()`. Using one of these functions and its values the shape is defined.
shape-box	Refers to one among `margin-box`, `border-box`, `padding-box`, `content-box`. When only <shape-box> is provided (without <basic-shape>) this box *is the* shape. When it is used along with <basic-shape>, this acts as the reference box.
image	When an image is provided as value, the shape is computed based on the alpha channel of the image specified.

Section 33.1: Shape Outside with Basic Shape – circle()

With the `shape-outside` CSS property one can define shape values for the float area so that the inline content wraps around the shape instead of the float's box.

CSS

```
img:nth-of-type(1) {
  shape-outside: circle(80px at 50% 50%);
  float: left;
  width: 200px;
}
img:nth-of-type(2) {
  shape-outside: circle(80px at 50% 50%);
  float: right;
  width: 200px;
}
p {
  text-align: center;
  line-height: 30px; /* purely for demo */
}
```

HTML

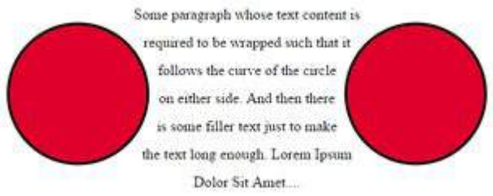
```
<img src="http://images.clipartpanda.com/circle-clip-art-circlergb.jpg">
<img src="http://images.clipartpanda.com/circle-clip-art-circlergb.jpg">
<p>Some paragraph whose text content is required to be wrapped such that it follows the curve of
the circle on either side. And then there is some filler text just to make the text long enough.
Lorem Ipsum Dolor Sit Amet....</p>
```

In the above example, both the images are actually square images and when the text is placed without the `shape-outside` property, it will not flow around the circle on either side. It will flow around the containing box of the image only. With `shape-outside` the float area is re-defined as a *circle* and the content is made to flow around this *imaginary circle* that is created using `shape-outside`.

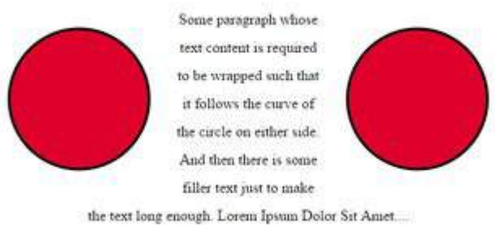
The *imaginary circle* that is used to re-define the float area is a circle with radius of 80px drawn from the center-mid point of the image's reference box.

Below are a couple of screenshots to illustrate how the content would be wrapped around when `shape-outside` is used and when it is not used.

Output with `shape-outside`



不使用shape-outside的输出效果



第33.2节：形状边距

shape-margin CSS属性为shape-outside添加了一个margin。

CSS

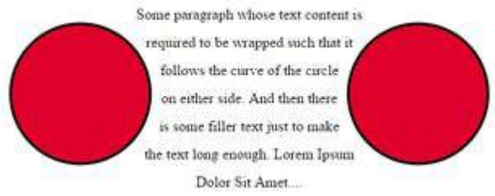
```
img:nth-of-type(1) {
  shape-outside: circle(80px 位于 50% 50%);
  shape-margin: 10px;
  float: left;
  width: 200px;
}
img:nth-of-type(2) {
  shape-outside: circle(80px 位于 50% 50%);
  shape-margin: 10px;
  float: right;
  width: 200px;
}
p {
  text-align: center;
  line-height: 30px; /* 仅用于演示 */
}
```

HTML

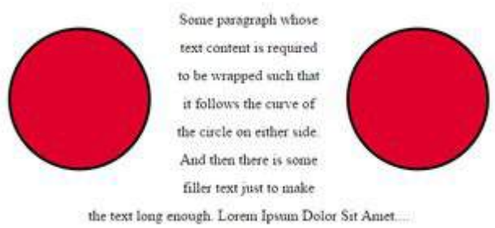
```
<img src="http://images.clipartpanda.com/circle-clip-art-circlergb.jpg">
<img src="http://images.clipartpanda.com/circle-clip-art-circlergb.jpg">
<p>某段文字内容需要环绕，使其沿着圆的曲线两侧排列。然后有一些填充文本以使文字足够长。

Lorem Ipsum Dolor Sit Amet....</p>
```

在此示例中，使用shape-margin在shape周围添加了10像素的边距。这在定义浮动区域的假想圆形与实际环绕内容之间创建了更多的空间。

输出：



Output without `shape-outside`



Section 33.2: Shape margin

The `shape-margin` CSS property adds a *margin* to `shape-outside`.

CSS

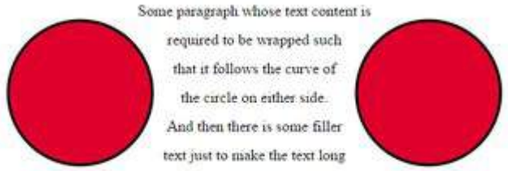
```
img:nth-of-type(1) {
  shape-outside: circle(80px at 50% 50%);
  shape-margin: 10px;
  float: left;
  width: 200px;
}
img:nth-of-type(2) {
  shape-outside: circle(80px at 50% 50%);
  shape-margin: 10px;
  float: right;
  width: 200px;
}
p {
  text-align: center;
  line-height: 30px; /* purely for demo */
}
```

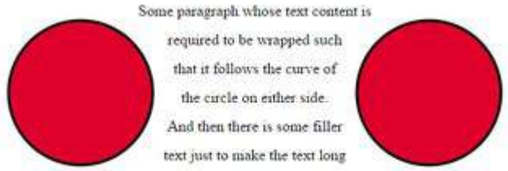
HTML

```
<img src="http://images.clipartpanda.com/circle-clip-art-circlergb.jpg">
<img src="http://images.clipartpanda.com/circle-clip-art-circlergb.jpg">
<p>Some paragraph whose text content is required to be wrapped such that it follows the curve of
the circle on either side. And then there is some filler text just to make the text long enough.
Lorem Ipsum Dolor Sit Amet....</p>
```

In this example, a 10px margin is added around the **shape** using `shape-margin`. This creates a bit more space between the *imaginary circle* that defines the float area and the actual content that is flowing around.

Output:


Some paragraph whose text content is
required to be wrapped such
that it follows the curve of
the circle on either side.
And then there is some filler
text just to make the text long
enough. Lorem Ipsum Dolor Sit Amet....


Some paragraph whose text content is
required to be wrapped such
that it follows the curve of
the circle on either side.
And then there is some filler
text just to make the text long
enough. Lorem Ipsum Dolor Sit Amet....

第34章：列表样式

值	描述
列表样式类型	列表项标记的类型。
list-style-position	指定标记的位置
list-style-image	指定列表项标记的类型
initial	将此属性设置为默认值
继承	从其父元素继承此属性

第34.1节：项目符号位置

列表由包含元素（ 或 ）内的 <li> 元素组成。列表项和容器都可以有影响列表项内容在文档中确切位置的外边距和内边距。不同浏览器的外边距和内边距默认值可能不同。为了实现跨浏览器的统一布局，需要专门设置这些值。

每个列表项都有一个“标记框”，其中包含项目符号标记。该框可以放置在列表项框的内部或外部。

```
list-style-position: inside;
```

将项目符号放置在 <li> 元素内，必要时将内容向右推移。

```
list-style-position: outside;
```

将项目符号放置在`<li>`元素的左侧。如果包含元素的内边距空间不足，
标记框将向左延伸，即使它会超出页面范围。

显示inside和outside定位的结果：jsfiddle

第34.2节：移除项目符号/数字

有时，列表不应显示任何项目符号或数字。在这种情况下，记得指定外边距
和内边距。

```
<ul>
    <li>第一项</li>
    <li>第二项</li>
</ul>
```

CSS

```
ul {
    list-style-type: none;
}
li {
    margin: 0;
    padding: 0;
}
```

第34.3节：项目符号或编号的类型

专用于无序列表（<ul>）中的<li>标签：

Chapter 34: List Styles

Value	Description
list-style-type	the type of list-item marker.
list-style-position	specifies where to place the marker
list-style-image	specifies the type of list-item marker
initial	sets this property to its default value
inherit	inherits this property from its parent element

Section 34.1: Bullet Position

A list consists of **`<li>`** elements inside a containing element (**`<ul>`** or **`<ol>`**). Both the list items and the container can have margins and paddings which influence the exact position of the list item content in the document. The default values for the margin and padding may be different for each browser. In order to get the same layout cross-browser, these need to be set specifically.

Each list item gets a 'marker box', which contains the bullet marker. This box can either be placed inside or outside of the list item box.

```
list-style-position: inside;
```

places the bullet within the **`<li>`** element, pushing the content to the right as needed.

```
list-style-position: outside;
```

places the bullet left of the **`<li>`** element. If there is not enough space in the padding of the containing element, the marker box will extend to the left even if it would fall off the page.

Showing the result of `inside` and `outside` positioning: jsfiddle

Section 34.2: Removing Bullets / Numbers

Sometimes, a list should just not display any bullet points or numbers. In that case, remember to specify margin and padding.

```
<ul>
    <li>first item</li>
    <li>second item</li>
</ul>
```

CSS

```
ul {
    list-style-type: none;
}
li {
    margin: 0;
    padding: 0;
}
```

Section 34.3: Type of Bullet or Numbering

Specific for **`<li>`** tags within an unordered list (**`<ul>`**):

```
list-style: disc;               /* 实心圆（默认） */
list-style: circle;             /* 空心圆 */
list-style: square;             /* 实心方块 */
list-style: '-';                /* 任意字符串 */
```

专用于有序列表（<ol>）中的<li>标签：

```
list-style: decimal;            /* 从1开始的十进制数字（默认） */
list-style: decimal-leading-zero;/* 前面补零的十进制数字（01, 02, 03, … 10） */
list-style: lower-roman;        /* 小写罗马数字（i., ii., iii., iv., …） */
list-style: upper-roman;        /* 大写罗马数字（I., II., III., IV., …） */
list-style-type: lower-greek;   /* 小写希腊字母（α., β., γ., δ., …） */
list-style-type: lower-alpha;   /* 小写字母（a., b., c., d., …） */
list-style-type: lower-latin;   /* 小写字母（a., b., c., d., …） */
list-style-type: upper-alpha;   /* 大写字母（A., B., C., D., …） */
list-style-type: upper-latin;   /* 大写字母（A., B., C., D., …） */
```

非特定：

```
list-style: none;               /* 无可见列表标记 */
list-style: inherit;            /* 继承自父元素 */
```



```
list-style: disc;               /* A filled circle (default) */
list-style: circle;             /* A hollow circle */
list-style: square;             /* A filled square */
list-style: '-';                /* any string */
```

Specific for **<li>** tags within an ordered list (**<ol>**):

```
list-style: decimal;            /* Decimal numbers beginning with 1 (default) */
list-style: decimal-leading-zero;/* Decimal numbers padded by initial zeros (01, 02, 03, … 10) */
list-style: lower-roman;        /* Lowercase roman numerals (i., ii., iii., iv., ...) */
list-style: upper-roman;        /* Uppercase roman numerals (I., II., III., IV., ...) */
list-style-type: lower-greek;   /* Lowercase roman letters (α., β., γ., δ., ...) */
list-style-type: lower-alpha;   /* Lowercase letters (a., b., c., d., ...) */
list-style-type: lower-latin;   /* Lowercase letters (a., b., c., d., ...) */
list-style-type: upper-alpha;   /* Uppercase letters (A., B., C., D., ...) */
list-style-type: upper-latin;   /* Uppercase letters (A., B., C., D., ...) */
```

Non-specific:

```
list-style: none;               /* No visible list marker */
list-style: inherit;            /* Inherits from parent */
```

第35章：计数器

参数	详情
counter-name	这是需要创建、递增或打印的计数器名称。它可以是开发者希望的任何自定义名称。
整数	这个整数是一个可选值，当与计数器名称一起提供时，将表示计数器的初始值（在counter-set、counter-reset属性中）或计数器应递增的值（在counter-increment中）。
无	这是所有3个counter-*属性的初始值。当该值用于counter-increment时，不影响任何计数器的值。当用于另外两个属性时，不会创建任何计数器。
计数器样式	指定计数器值需要显示的样式。它支持list-style-type属性支持的所有值。如果使用none，则计数器值完全不打印。
连接字符串	表示必须放置在两个不同计数器级别值之间的字符串（如“2.1.1”中的“.”）。

第35.1节：将罗马数字样式应用于计数器输出

CSS

```
body {
  counter-reset: item-counter;
}

.item {
  counter-increment: item-counter;
}

.item:before {
  content: counter(item-counter, upper-roman) ". "; /* 通过指定 upper-roman 作为样式，输出将为罗马数字 */

}
```

HTML

```
<div class='item'>项目编号: 1</div>
<div class='item'>项目编号: 2</div>
<div class='item'>项目编号: 3</div>
```

在上述示例中，计数器的输出将显示为 I、II、III（罗马数字），而不是通常的 1、2、3，因为开发者明确指定了计数器的样式。

第35.2节：使用 CSS 计数器为每个项目编号

CSS

```
body {
  counter-reset: item-counter; /* 创建计数器 */
}
.item {
  counter-increment: item-counter; /* 每次遇到类名为 "item" 的元素时，计数器递增 */

}
.item-header:before {
  content: counter(item-counter) ". "; /* 在标题前打印计数器的值并
在其后添加一个"." */
}
```


Chapter 35: Counters

Parameter	Details
counter-name	This is the name of the counter that needs to be created or incremented or printed. It can be any custom name as the developer wishes.
integer	This integer is an optional value that when provided next to the counter name will represent the initial value of the counter (in `counter-set`, `counter-reset` properties) or the value by which the counter should be incremented (in `counter-increment`).
none	This is the initial value for all 3 `counter-*` properties. When this value is used for `counter-increment`, the value of none of the counters are affected. When this is used for the other two, no counter is created.
counter-style	This specifies the style in which the counter value needs to be displayed. It supports all values supported by the `list-style-type` property. If none is used then the counter value is not printed at all.
connector-string	This represents the string that must be placed between the values of two different counter levels (like the "." in "2.1.1").

Section 35.1: Applying roman numerals styling to the counter output

CSS

```
body {
  counter-reset: item-counter;
}

.item {
  counter-increment: item-counter;
}

.item:before {
  content: counter(item-counter, upper-roman) ". "; /* by specifying the upper-roman as style the
output would be in roman numbers */
}
```

HTML

```
<div class='item'>Item No: 1</div>
<div class='item'>Item No: 2</div>
<div class='item'>Item No: 3</div>
```

In the above example, the counter's output would be displayed as I, II, III (roman numbers) instead of the usual 1, 2, 3 as the developer has explicitly specified the counter's style.

Section 35.2: Number each item using CSS Counter

CSS

```
body {
  counter-reset: item-counter; /* create the counter */
}
.item {
  counter-increment: item-counter; /* increment the counter every time an element with class "item"
is encountered */
}
.item-header:before {
  content: counter(item-counter) ". "; /* print the value of the counter before the header and
append a "." to it */
}
```

```
/* 仅用于演示 */

.item {
  border: 1px 实线;
  height: 100px;
  margin-bottom: 10px;
}
.item-header {
  border-bottom: 1px 实线;
  height: 40px;
  line-height: 40px;
  padding: 5px;
}
.item-content {
  padding: 8px;
}
```

HTML

```
<div class='item'>
  <div class='item-header'>项目 1 标题</div>
  <div class='item-content'>Lorem Ipsum Dolor Sit Amet....</div>
</div>
<div class='item'>
  <div class='item-header'>项目2标题</div>
  <div class='item-content'>Lorem Ipsum Dolor Sit Amet....</div>
</div>
<div class='item'>
  <div class='item-header'>项目3标题</div>
  <div class='item-content'>Lorem Ipsum Dolor Sit Amet....</div>
</div>
```

上面的示例为页面中的每个“项目”编号，并在其标题前添加项目编号（使用.item-header元素的:before伪元素的content属性）。该代码的在线演示可在此处获得。

第35.3节：使用CSS计数器实现多级编号

CSS

```
ul {
  list-style: none;
  counter-reset: list-item-number; /* 自嵌套计数器，因所有级别名称相同 */
}
li {
  counter-increment: list-item-number;
}
li:before {
  content: counters(list-item-number, ".") " "; /* 使用counters()函数表示打印前合并所有更高级别的计数器值 */

}
```

HTML

```
<ul>
  <li>第1级
    <ul>
      <li>第1.1级
        <ul>
          <li>第1.1.1级</li>
        </ul>
      </li>
    </ul>
```



```
/* just for demo */

.item {
  border: 1px solid;
  height: 100px;
  margin-bottom: 10px;
}
.item-header {
  border-bottom: 1px solid;
  height: 40px;
  line-height: 40px;
  padding: 5px;
}
.item-content {
  padding: 8px;
}
```

HTML

```
<div class='item'>
  <div class='item-header'>Item 1 Header</div>
  <div class='item-content'>Lorem Ipsum Dolor Sit Amet....</div>
</div>
<div class='item'>
  <div class='item-header'>Item 2 Header</div>
  <div class='item-content'>Lorem Ipsum Dolor Sit Amet....</div>
</div>
<div class='item'>
  <div class='item-header'>Item 3 Header</div>
  <div class='item-content'>Lorem Ipsum Dolor Sit Amet....</div>
</div>
```

The above example numbers every "item" in the page and adds the item's number before its header (using `content` property of `.item-header` element's `:before` pseudo). A live demo of this code is available here.

Section 35.3: Implementing multi-level numbering using CSS counters

CSS

```
ul {
  list-style: none;
  counter-reset: list-item-number; /* self nesting counter as name is same for all levels */
}
li {
  counter-increment: list-item-number;
}
li:before {
  content: counters(list-item-number, ".") " "; /* usage of counters() function means value of
counters at all higher levels are combined before printing */
}
```

HTML

```
<ul>
  <li>Level 1
    <ul>
      <li>Level 1.1
        <ul>
          <li>Level 1.1.1</li>
        </ul>
      </li>
    </ul>
```

```
  </li>
  <li>第二级
    <ul>
      <li>第二级 2.1
        <ul>
          <li>第二级 2.1.1</li>
          <li>第二级 2.1.2</li>
        </ul>
      </li>
    </ul>
  </li>
  <li>第三级</li>
</ul>
```

以上是使用 CSS 计数器实现多级编号的示例。它利用了计数器的自嵌套概念。自嵌套是指如果一个元素已经有了给定名称的计数器，但又需要创建另一个计数器时，则将其作为现有计数器的子计数器创建。在这里，第二级的ul已经继承了其父元素的list-item-number计数器，但又需要为其子元素li创建自己的list-item-number计数器，因此创建了list-item-number[1]（第二级计数器），并将其嵌套在list-item-number[0]（第一级计数器）下。这样就实现了多级编号。

输出使用了counters()函数而非counter()函数，因为counters()函数设计用于在打印输出时，前缀所有更高级别计数器（父级）的值。


```
  </li>
  <li>Level 2
    <ul>
      <li>Level 2.1
        <ul>
          <li>Level 2.1.1</li>
          <li>Level 2.1.2</li>
        </ul>
      </li>
    </ul>
  </li>
  <li>Level 3</li>
</ul>
```

The above is an example of multi-level numbering using CSS counters. It makes use of the ***self-nesting*** concept of counters. Self nesting is a concept where if an element already has a counter with the given name but is having to create another then it creates it as a child of the existing counter. Here, the second level `ul` already inherits the `list-item-number` counter from its parent but then has to create its own `list-item-number` (for its children `li`) and so creates `list-item-number[1]` (counter for second level) and nests it under `list-item-number[0]` (counter for first level). Thus it achieves the multi-level numbering.

The output is printed using the `counters()` function instead of the `counter()` function because the `counters()` function is designed to prefix the value of all higher level counters (parent) when printing the output.

第36章：函数

第36.1节：calc() 函数

接受一个数学表达式并返回一个数值。

当处理不同类型的单位（例如从百分比中减去像素值）以计算属性值时，它尤其有用。

可以使用 +、-、/ 和 * 运算符，并且如果需要，可以添加括号来指定运算顺序。

使用 calc() 来计算 div 元素的宽度：

```
#div1 {
    position: absolute;
    left: 50px;
    width: calc(100% - 100px);
    border: 1px solid black;
    background-color: yellow;
    padding: 5px;
    text-align: center;
}
```

使用 calc() 来确定背景图像的位置：

```
background-position: calc(50% + 17px) calc(50% + 10px), 50% 50%;
```

使用 calc() 来确定元素的高度：

```
height: calc(100% - 20px);
```

第36.2节：attr() 函数

返回所选元素的属性值。

下面是一个blockquote元素，其中包含一个字符，位于data-*属性中，CSS可以使用该函数在（例如在::before和::after伪元素中）使用该字符。

```
<blockquote data-mark=""></blockquote>
```

在以下CSS代码块中，字符被添加到元素内文本的前后：

```
blockquote[data-mark]::before,
blockquote[data-mark]::after {
    content: attr(data-mark);
}
```

第36.3节：var() 函数

var() 函数允许访问CSS变量。

```
/* 设置一个变量 */
:root {
```


Chapter 36: Functions

Section 36.1: calc() function

Accepts a mathematical expression and returns a numerical value.

It is especially useful when working with different types of units (e.g. subtracting a px value from a percentage) to calculate the value of an attribute.

+, -, /, and * operators can all be used, and parentheses can be added to specify the order of operations if necessary.

Use `calc()` to calculate the width of a div element:

```
#div1 {
    position: absolute;
    left: 50px;
    width: calc(100% - 100px);
    border: 1px solid black;
    background-color: yellow;
    padding: 5px;
    text-align: center;
}
```

Use `calc()` to determine the position of a background-image:

```
background-position: calc(50% + 17px) calc(50% + 10px), 50% 50%;
```

Use `calc()` to determine the height of an element:

```
height: calc(100% - 20px);
```

Section 36.2: attr() function

Returns the value of an attribute of the selected element.

Below is a blockquote element which contains a character inside a `data-*` attribute which CSS can use (e.g. inside the `::before` and `::after` pseudo-element) using this function.

```
<blockquote data-mark=""></blockquote>
```

In the following CSS block, the character is appended before and after the text inside the element:

```
blockquote[data-mark]::before,
blockquote[data-mark]::after {
    content: attr(data-mark);
}
```

Section 36.3: var() function

The var() function allows CSS variables to be accessed.

```
/* set a variable */
:root {
```

```
    --primary-color: blue;
}

/* 访问变量 */
selector {
    color: var(--primary-color);
}
```

此功能目前正在开发中。请查看 caniuse.com 以获取最新的浏览器支持情况。

第36.4节：radial-gradient() 函数

创建一个表示从渐变中心向外辐射颜色渐变的图像

```
radial-gradient(red, orange, yellow) /* 从渐变中心向外发散的渐变，中心为红色，然后是橙色，最后边缘为
黄色 */
```

第36.5节：linear-gradient() 函数

创建一个表示线性渐变颜色的图像。

```
linear-gradient( 0deg, 红色, 黄色 50%, 蓝色);
```

这会创建一个从下到上的渐变，颜色从红色开始，然后在50%处为黄色，最后以蓝色结束。


```
    --primary-color: blue;
}

/* access variable */
selector {
    color: var(--primary-color);
}
```

This feature is currently under development. Check caniuse.com for the latest browser support.

Section 36.4: radial-gradient() function

Creates an image representing a gradient of colors radiating from the center of the gradient

```
radial-gradient(red, orange, yellow) /*A gradient coming out from the middle of the
gradient, red at the center, then orange, until it is finally yellow at the edges*/
```

Section 36.5: linear-gradient() function

Creates a image representing a linear gradient of colors.

```
linear-gradient( 0deg, red, yellow 50%, blue);
```

This creates a gradient going from bottom to top, with colors starting at red, then yellow at 50%, and finishing in blue.

第37章：自定义属性（变量）

CSS变量允许作者创建可重用的值，这些值可以在整个CSS文档中使用。

例如，在CSS中常常在整个文档中重复使用同一种颜色。在CSS变量出现之前，这意味着需要在文档中多次重复相同的颜色值。使用CSS变量后，可以将颜色值赋给一个变量，并在多个地方引用。这使得更改值更容易，也比使用传统的CSS值更具语义性。

第37.1节：变量颜色

```
:root {
  --red: #b00;
  --blue: #4679bd;
  --grey: #ddd;
}
.Bx1 {
  color: var(--red);
  background: var(--grey);
  border: 1px solid var(--red);
}
```

第37.2节：变量维度

```
:root {
  --W200: 200像素;
  --W10: 10像素;
}
.Bx2 {
  宽度: var(--W200);
  高度: var(--W200);
  外边距: var(--W10);
}
```

第37.3节：变量级联

CSS变量的级联方式与其他属性类似，并且可以安全地重新声明。

你可以多次定义变量，只有具有最高特异性的定义会应用于所选元素
。

假设有以下HTML：

```
<a class="button">绿色按钮</a>
<a class="button button_red">红色按钮</a>
<a class="button">悬停按钮</a>
```

我们可以这样写CSS：

```
.button {
  --color: 绿色;
  padding: .5rem;
  border: 1px 实线 var(--color);
  color: var(--color);
}
```


Chapter 37: Custom Properties (Variables)

CSS Variables allow authors to create reusable values which can be used throughout a CSS document.

For example, it's common in CSS to reuse a single color throughout a document. Prior to CSS Variables this would mean reusing the same color value many times throughout a document. With CSS Variables the color value can be assigned to a variable and referenced in multiple places. This makes changing values easier and is more semantic than using traditional CSS values.

Section 37.1: Variable Color

```
:root {
  --red: #b00;
  --blue: #4679bd;
  --grey: #ddd;
}
.Bx1 {
  color: var(--red);
  background: var(--grey);
  border: 1px solid var(--red);
}
```

Section 37.2: Variable Dimensions

```
:root {
  --W200: 200px;
  --W10: 10px;
}
.Bx2 {
  width: var(--W200);
  height: var(--W200);
  margin: var(--W10);
}
```

Section 37.3: Variable Cascading

CSS variables cascade in much the same way as other properties, and can be restated safely.

You can define variables multiple times and only the definition with the highest specificity will apply to the element selected.

Assuming this HTML:

```
<a class="button">Button Green</a>
<a class="button button_red">Button Red</a>
<a class="button">Button Hovered On</a>
```

We can write this CSS:

```
.button {
  --color: green;
  padding: .5rem;
  border: 1px solid var(--color);
  color: var(--color);
}
```

```
.button:hover {
  --color: blue;
}

.button_red {
  --color: red;
}
```

并得到以下结果：

第37.4节：有效/无效

命名 在命名CSS变量时，只包含字母和连字符，就像其他CSS属性一样（例如：line-height，-moz-box-sizing），但应以双连字符（--）开头

```
//这些是无效的变量名
--123color: blue;
--#color: red;
--bg_color: yellow
--$width: 100px;

//有效的变量名
--color: red;
--bg-color: yellow
--width: 100px;
```

CSS 变量区分大小写。

```
/* 下面的变量名都是不同的变量 */
--pcolor: ;
--Pcolor: ;
--pColor: ;
```

空值与空格

```
/* 无效 */
    --color:;

/* 有效 */
  --color: ; /* 分配了空格 */
```

连接

```
    /* 无效 - CSS 不支持连接 */
.logo{
        --logo-url: 'logo';
        background: url('assets/img/' var(--logo-url) '.png');
    }

    /* 无效 - CSS 错误 */
.logo{
        --logo-url: 'assets/img/logo.png';
        background: url(var(--logo-url));
    }
```



```
.button:hover {
  --color: blue;
}

.button_red {
  --color: red;
}
```

And get this result:

Section 37.4: Valid/Invalids

Naming When naming CSS variables, it contains only letters and dashes just like other CSS properties (eg: line-height, -moz-box-sizing) but it should start with double dashes (--)

```
//These are Invalids variable names
--123color: blue;
--#color: red;
--bg_color: yellow
--$width: 100px;

//Valid variable names
--color: red;
--bg-color: yellow
--width: 100px;
```

CSS Variables are case sensitive.

```
/* The variable names below are all different variables */
--pcolor: ;
--Pcolor: ;
--pColor: ;
```

Empty Vs Space

```
/* Invalid */
    --color:;

/* Valid */
  --color: ; /* space is assigned */
```

Concatenations

```
    /* Invalid - CSS doesn't support concatenation*/
    .logo{
        --logo-url: 'logo';
        background: url('assets/img/' var(--logo-url) '.png');
    }

    /* Invalid - CSS bug */
    .logo{
        --logo-url: 'assets/img/logo.png';
        background: url(var(--logo-url));
    }
```

```
    /* 有效 */
.logo{
        --logo-url: url('assets/img/logo.png');
        background: var(--logo-url);
    }
```

使用单位时请小心

```
    /* 无效 */
    --width: 10;
    width: var(--width)px;

    /* 有效 */
    --width: 10px;
    width: var(--width);

    /* 有效 */
    --width: 10;
    width: calc(1px * var(--width)); /* 乘以1单位以转换 */
    width: calc(1em * var(--width));
```

第37.5节：使用媒体查询

你可以在媒体查询中重新设置变量，并让这些新值在使用它们的地方级联，
这是预处理器变量无法实现的。

这里，一个媒体查询改变了用于设置一个非常简单网格的变量：

HTML

```
<div></div>
<div></div>
<div></div>
<div></div>
```

CSS

```
:root{
    --width: 25%;
    --content: '这是桌面端';
}
@media only screen and (max-width: 767px){
    :root{
        --width:50%;
        --content: '这是移动端';
    }
}
@media only screen and (max-width: 480px){
    :root{
        --width:100%;
    }
}

div{
    width: calc(var(--width) - 20px);
    height: 100px;
}
div:before{
    content: var(--content);
```



```
    /* Valid */
    .logo{
        --logo-url: url('assets/img/logo.png');
        background: var(--logo-url);
    }
```

Careful when using Units

```
    /* Invalid */
    --width: 10;
    width: var(--width)px;

    /* Valid */
    --width: 10px;
    width: var(--width);

    /* Valid */
    --width: 10;
    width: calc(1px * var(--width)); /* multiply by 1 unit to convert */
    width: calc(1em * var(--width));
```

Section 37.5: With media queries

You can re-set variables within media queries and have those new values cascade wherever they are used, something that isn't possible with pre-processor variables.

Here, a media query changes the variables used to set up a very simple grid:

HTML

```
<div></div>
<div></div>
<div></div>
<div></div>
```

CSS

```
:root{
    --width: 25%;
    --content: 'This is desktop';
}
@media only screen and (max-width: 767px){
    :root{
        --width:50%;
        --content: 'This is mobile';
    }
}
@media only screen and (max-width: 480px){
    :root{
        --width:100%;
    }
}

div{
    width: calc(var(--width) - 20px);
    height: 100px;
}
div:before{
    content: var(--content);
```

```
}

/* 其他样式 */
body {
    padding: 10px;
}

div{
    display: flex;
    align-items: center;
    justify-content: center;
    font-weight:bold;
    float:left;
    margin: 10px;
    border: 4px solid black;
    background: red;
}
```

你可以尝试在这个CodePen演示中调整窗口大小

下面是调整大小时的动画截图：


```
}

/* Other Styles */
body {
    padding: 10px;
}

div{
    display: flex;
    align-items: center;
    justify-content: center;
    font-weight:bold;
    float:left;
    margin: 10px;
    border: 4px solid black;
    background: red;
}
```

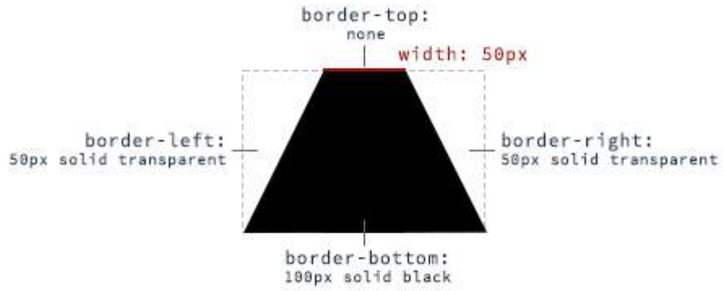
You can try resizing the window in this CodePen Demo

Here's an animated screenshot of the resizing in action:

第38章：单元素形状

第38.1节：梯形

梯形可以通过一个高度为零（高度为0px）、宽度大于零且带有边框的块级元素制作，边框除了一个边是透明的外：



HTML：

```
<div class="trapezoid"></div>
```

CSS：

```
.trapezoid {
    width: 50px;
    height: 0;
    border-left: 50px solid transparent;
    border-right: 50px solid transparent;
    border-bottom: 100px solid black;
}
```

通过改变边框的边，可以调整梯形的方向。

第38.2节：三角形

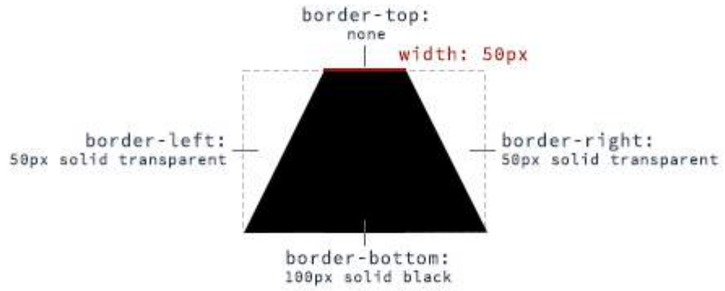
要创建一个CSS三角形，定义一个宽度和高度均为0像素的元素。三角形形状将通过边框属性形成。对于一个高度和宽度均为0的元素，四个边框（上、右、下、左）各自形成一个三角形。下面是一个高度/宽度为0且四个边框颜色不同的元素。

通过将某些边框设置为透明，其他边框设置为颜色，我们可以创建各种三角形。例如，在向上三角形中，我们将底部边框设置为所需颜色，然后将左边和右边边框设置为透明。下面的图片中，左边和右边的边框稍微着色，以显示三角形的形成方式。

Chapter 38: Single Element Shapes

Section 38.1: Trapezoid

A trapezoid can be made by a block element with zero height (height of `0px`), a width greater than zero and a border, that is transparent except for one side:



HTML:

```
<div class="trapezoid"></div>
```

CSS:

```
.trapezoid {
    width: 50px;
    height: 0;
    border-left: 50px solid transparent;
    border-right: 50px solid transparent;
    border-bottom: 100px solid black;
}
```

With changing the border sides, the orientation of the trapezoid can be adjusted.

Section 38.2: Triangles

To create a CSS triangle define an element with a width and height of 0 pixels. The triangle shape will be formed using border properties. For an element with 0 height and width the 4 borders (top, right, bottom, left) each form a triangle. Here's an element with 0 height/width and 4 different colored borders.

By setting some borders to transparent, and others to a color we can create various triangles. For example, in the Up triangle, we set the bottom border to the desired color, then set the left and right borders to transparent. Here's an image with the left and right borders shaded slightly to show how the triangle is being formed.

通过改变不同边框的宽度，可以调整三角形的尺寸——更高、更矮、不对称等。
下面的示例均显示一个50x50像素的三角形。

三角形 - 向上指

```
<div class="triangle-up"></div>
```

```
.triangle-up {
  宽度: 0;
  height: 0;
  border-left: 25px solid transparent;
  border-right: 25px solid transparent;
  border-bottom: 50px solid rgb(246, 156, 85);
}
```

三角形 - 向下指

```
<div class="triangle-down"></div>
```

```
.triangle-down {
  宽度: 0;
  height: 0;
  border-left: 25px solid transparent;
  border-right: 25px solid transparent;
  border-top: 50px solid rgb(246, 156, 85);
}
```

三角形 - 向右指

The dimensions of the triangle can be altered by changing the different border widths - taller, shorter, lopsided, etc. The examples below all show a 50x50 pixel triangle.

Triangle - Pointing Up

```
<div class="triangle-up"></div>
```

```
.triangle-up {
  width: 0;
  height: 0;
  border-left: 25px solid transparent;
  border-right: 25px solid transparent;
  border-bottom: 50px solid rgb(246, 156, 85);
}
```

Triangle - Pointing Down

```
<div class="triangle-down"></div>
```

```
.triangle-down {
  width: 0;
  height: 0;
  border-left: 25px solid transparent;
  border-right: 25px solid transparent;
  border-top: 50px solid rgb(246, 156, 85);
}
```

Triangle - Pointing Right

```
<div class="triangle-right"></div>
```

```
.triangle-right {
  宽度: 0;
  height: 0;
  border-top: 25px 实心 透明;
  border-bottom: 25px 实心 透明;
  border-left: 50px 实心 rgb(246, 156, 85);
}
```

三角形 - 指向左侧

```
<div class="triangle-left"></div>
```

```
.triangle-left {
  宽度: 0;
  height: 0;
  border-top: 25px 实心 透明;
  border-bottom: 25px 实心 透明;
  border-right: 50px 实心 rgb(246, 156, 85);
}
```

三角形 - 指向上/右

```
<div class="triangle-up-right"></div>
```

```
.triangle-up-right {
  宽度: 0;
  height: 0;
  border-top: 50px 实心 rgb(246, 156, 85);
  border-left: 50px 实心 透明;
}
```

三角形 - 指向上/左

```
<div class="triangle-up-left"></div>
```

```
.triangle-up-left {
```



```
<div class="triangle-right"></div>
```

```
.triangle-right {
  width: 0;
  height: 0;
  border-top: 25px solid transparent;
  border-bottom: 25px solid transparent;
  border-left: 50px solid rgb(246, 156, 85);
}
```

Triangle - Pointing Left

```
<div class="triangle-left"></div>
```

```
.triangle-left {
  width: 0;
  height: 0;
  border-top: 25px solid transparent;
  border-bottom: 25px solid transparent;
  border-right: 50px solid rgb(246, 156, 85);
}
```

Triangle - Pointing Up/Right

```
<div class="triangle-up-right"></div>
```

```
.triangle-up-right {
  width: 0;
  height: 0;
  border-top: 50px solid rgb(246, 156, 85);
  border-left: 50px solid transparent;
}
```

Triangle - Pointing Up/Left

```
<div class="triangle-up-left"></div>
```

```
.triangle-up-left {
```

```
  宽度: 0;
  height: 0;
  border-top: 50px 实心 rgb(246, 156, 85);
  border-right: 50px 实心 透明;
}
```

三角形 - 向下/向右指

```
<div class="triangle-down-right"></div>
```

```
.triangle-down-right {
  宽度: 0;
  height: 0;
  border-bottom: 50px solid rgb(246, 156, 85);
  border-left: 50px solid transparent;
}
```

三角形 - 指向下/左

```
<div class="triangle-down-left"></div>
```

```
.triangle-down-left {
  宽度: 0;
  height: 0;
  border-bottom: 50px solid rgb(246, 156, 85);
  border-right: 50px solid transparent;
}
```

第38.3节：圆和椭圆

圆

要创建一个圆形，定义一个宽度和高度相等的元素（一个正方形），然后将该元素的border-radius属性设置为`50%`。

HTML


```
  width: 0;
  height: 0;
  border-top: 50px solid rgb(246, 156, 85);
  border-right: 50px solid transparent;
}
```

Triangle - Pointing Down/Right

```
<div class="triangle-down-right"></div>
```

```
.triangle-down-right {
  width: 0;
  height: 0;
  border-bottom: 50px solid rgb(246, 156, 85);
  border-left: 50px solid transparent;
}
```

Triangle - Pointing Down/Left

```
<div class="triangle-down-left"></div>
```

```
.triangle-down-left {
  width: 0;
  height: 0;
  border-bottom: 50px solid rgb(246, 156, 85);
  border-right: 50px solid transparent;
}
```

Section 38.3: Circles and Ellipses

Circle

To create a **circle**, define an element with an equal `width` and `height` (a *square*) and then set the `border-radius` property of this element to `50%`.

HTML

```
<div class="circle"></div>
```

CSS

```
.circle {
    width: 50px;
    height: 50px;
    background: rgb(246, 156, 85);
    border-radius: 50%;
}
```

椭圆

椭圆是类似于圆形的形状，但宽度和高度的数值不同。

HTML

```
<div class="oval"></div>
```

CSS

```
.oval {
    width: 50px;
    height: 80px;
    background: rgb(246, 156, 85);
    border-radius: 50%;
}
```

第38.4节：爆发形状

爆发形状类似于星形，但各点从主体延伸的距离较短。可以将爆发形状想象成一个正方形，上面叠加了稍微旋转的额外正方形。

这些额外的正方形是使用::before和::after伪元素创建的。

8点爆发形状

一个8点爆发由两层正方形组成。底部的正方形是元素本身，额外的正方形是使用:before伪元素创建的。底部旋转了20°，顶部正方形旋转了135°。

```
<div class="burst-8"></div>

.burst-8 {
background: rgb(246, 156, 85);
  width: 40px;
```



```
<div class="circle"></div>
```

CSS

```
.circle {
    width: 50px;
    height: 50px;
    background: rgb(246, 156, 85);
    border-radius: 50%;
}
```

Ellipse

An **ellipse** is similar to a circle, but with different values for `width` and `height`.

HTML

```
<div class="oval"></div>
```

CSS

```
.oval {
  width: 50px;
  height: 80px;
  background: rgb(246, 156, 85);
  border-radius: 50%;
}
```

Section 38.4: Bursts

A burst is similar to a star but with the points extending less distance from the body. Think of a burst shape as a square with additional, slightly rotated, squares layered on top.

The additional squares are created using the `::before` and `::after` psuedo-elements.

8 Point Burst

An 8 point burst are 2 layered squares. The bottom square is the element itself, the additional square is created using the `:before` pseudo-element. The bottom is rotated 20°, the top square is rotated 135°.

```
<div class="burst-8"></div>

.burst-8 {
  background: rgb(246, 156, 85);
  width: 40px;
```

```
  height: 40px;
position: relative;
  text-align: center;
  -ms-transform: rotate(20deg);
  transform: rotate(20eg);
}

.burst-8::before {
  content: "";
  position: absolute;
  top: 0;
  left: 0;
  height: 40px;
  width: 40px;
background: rgb(246, 156, 85);
  -ms-transform: rotate(135deg);
  transform: rotate(135deg);
}
```

12 点爆炸效果

一个12点爆发由3层方块组成。最底层的方块是元素本身，额外的方块是使用:before和:after伪元素创建的。最底层旋转0°，下一层方块旋转30°，最顶层旋转60°。

```
<div class="burst-12"></div>

.burst-12 {
  width: 40px;
  height: 40px;
position: relative;
  text-align: center;
  background: rgb(246, 156, 85);
}

.burst-12::before, .burst-12::after {
  content: "";
position: absolute;
  top: 0;
  left: 0;
  height: 40px;
  width: 40px;
background: rgb(246, 156, 85);
}

.burst-12::before {
  -ms-transform: rotate(30deg);
  transform: rotate(30deg);
}

.burst-12::after {
  -ms-transform: rotate(60deg);
  transform: rotate(60deg);
```



```
  height: 40px;
  position: relative;
  text-align: center;
  -ms-transform: rotate(20deg);
  transform: rotate(20eg);
}

.burst-8::before {
  content: "";
  position: absolute;
  top: 0;
  left: 0;
  height: 40px;
  width: 40px;
  background: rgb(246, 156, 85);
  -ms-transform: rotate(135deg);
  transform: rotate(135deg);
}
```

12 Point Burst

An 12 point burst are 3 layered squares. The bottom square is the element itself, the additional squares are created using the `:before` and `:after` pseudo-elements. The bottom is rotated 0°, the next square is rotated 30°, and the top is rotated 60°.

```
<div class="burst-12"></div>

.burst-12 {
  width: 40px;
  height: 40px;
  position: relative;
  text-align: center;
  background: rgb(246, 156, 85);
}

.burst-12::before, .burst-12::after {
  content: "";
  position: absolute;
  top: 0;
  left: 0;
  height: 40px;
  width: 40px;
  background: rgb(246, 156, 85);
}

.burst-12::before {
  -ms-transform: rotate(30deg);
  transform: rotate(30deg);
}

.burst-12::after {
  -ms-transform: rotate(60deg);
  transform: rotate(60deg);
```

```
}
```

第38.5节：方块

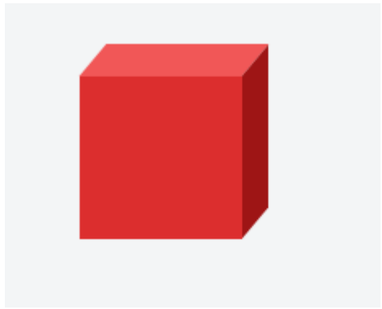
要创建一个方块，需要定义一个同时具有宽度和高度的元素。在下面的示例中，我们有一个宽度和高度均为100像素的元素。

```
<div class="square"></div>
```

```
.square {
    宽度: 100像素;
    height: 100px;
    background: rgb(246, 156, 85);
}
```

第38.6节：立方体

此示例展示了如何使用二维变换方法 skewX() 和 skewY() 在伪元素上创建立方体。



HTML：

```
<div class="cube"></div>
```

CSS：

```
.cube {
  background: #dc2e2e;
  width: 100px;
  height: 100px;
  position: relative;
  margin: 50px;
}

.cube::before {
  content: '';
```



```
}
```

Section 38.5: Square

To create a square, define an element with both a width and height. In the example below, we have an element with a `width` and `height` of 100 pixels each.

```
<div class="square"></div>
```

```
.square {
    width: 100px;
    height: 100px;
    background: rgb(246, 156, 85);
}
```

Section 38.6: Cube

This example shows how to create a cube using 2D transformation methods `skewX()` and `skewY()` on pseudo elements.

HTML:

```
<div class="cube"></div>
```

CSS:

```
.cube {
  background: #dc2e2e;
  width: 100px;
  height: 100px;
  position: relative;
  margin: 50px;
}

.cube::before {
  content: '';
```

```
  display: inline-block;
  background: #f15757;
  width: 100px;
  height: 20px;
  transform: skewX(-40deg);
  position: absolute;
  top: -20px;
  left: 8px;
}

.cube::after {
  content: '';
  display: inline-block;
  background: #9e1515;
  width: 16px;
  height: 100px;
  transform: skewY(-50deg);
  position: absolute;
  top: -10px;
  left: 100%;
}
```

查看演示

第38.7节：金字塔

此示例展示了如何使用边框和二维变换方法 skewY() 和
rotate() 在伪元素上创建一个金字塔。

HTML：

```
<div class="pyramid"></div>
```

CSS：

```
.pyramid {
  宽度: 100像素;
  高度: 200像素;
  位置: 相对;
  外边距: 50像素;
}

.pyramid::before,.pyramid::after {
  content: '';
  display: inline-block;
  width: 0;
  height: 0;
  border: 50px solid;
```



```
  display: inline-block;
  background: #f15757;
  width: 100px;
  height: 20px;
  transform: skewX(-40deg);
  position: absolute;
  top: -20px;
  left: 8px;
}

.cube::after {
  content: '';
  display: inline-block;
  background: #9e1515;
  width: 16px;
  height: 100px;
  transform: skewY(-50deg);
  position: absolute;
  top: -10px;
  left: 100%;
}
```

See demo

Section 38.7: Pyramid

This example shows how to create a **pyramid** using borders and 2D transformation methods `skewY()` and `rotate()` on pseudo elements.

HTML:

```
<div class="pyramid"></div>
```

CSS:

```
.pyramid {
  width: 100px;
  height: 200px;
  position: relative;
  margin: 50px;
}

.pyramid::before,.pyramid::after {
  content: '';
  display: inline-block;
  width: 0;
  height: 0;
  border: 50px solid;
```

```
  position: absolute;
}

.pyramid::before {
  border-color: transparent transparent #ff5656 transparent;
  transform: scaleY(2) skewY(-40deg) rotate(45deg);
}

.pyramid::after {
  border-color: transparent transparent #d64444 transparent;
  transform: scaleY(2) skewY(40deg) rotate(-45deg);
}
```



```
  position: absolute;
}

.pyramid::before {
  border-color: transparent transparent #ff5656 transparent;
  transform: scaleY(2) skewY(-40deg) rotate(45deg);
}

.pyramid::after {
  border-color: transparent transparent #d64444 transparent;
  transform: scaleY(2) skewY(40deg) rotate(-45deg);
}
```

第39章：列

第39.1节：简单示例（列数）

CSS多栏布局使创建多栏文本变得简单。

代码

```
<div id="multi-columns">Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est
laborum</div>
```

```
.multi-columns {
  -moz-column-count: 2;
  -webkit-column-count: 2;
  column-count: 2;
}
```

结果

Stack Overflow is a privately held website, the flagship site of the Stack Exchange Network,[4][5][6] created in 2008 by Jeff Atwood and Joel Spolsky.[7][8] It was created to be a more open alternative to earlier Q&A sites such as Experts-Exchange. The name for the website was chosen by voting in April 2008 by readers of Coding Horror, Atwood's popular programming blog.

The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers in a fashion similar to a wiki or Digg.[13] Users of Stack Overflow can earn reputation points and "badges"; for example, a person is awarded 10 reputation points for receiving an "up" vote on an answer given to a question, and can receive badges for their valued contributions,[14] which represents a kind of gamification of the traditional Q&A site or forum. All user-generated content is licensed under a Creative Commons Attribute-ShareAlike license.

第39.2节：列宽

column-width 属性设置最小列宽。如果未定义 column-count，浏览器将根据可用宽度创建尽可能多的列。

代码：

```
<div id="multi-columns">
洛雷姆·伊普苏姆（Lorem ipsum）是虚构的文本，用于排版和设计，表示“痛苦坐着，爱着，做着，时间流逝，工作和痛苦带来伟大的快乐。为了获得最小的好处，谁不愿意进行无节制的锻炼，除非从事某种便利的结果。痛苦的厌恶者在享乐中不会感到痛苦。例外的是，非故意的爱好者，在过失中，谁会放弃那些使灵魂愉快的活动。”
```


Chapter 39: Columns

Section 39.1: Simple Example (column-count)

The CSS multi-column layout makes it easy to create multiple columns of text.

Code

```
<div id="multi-columns">Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est
laborum</div>
```

```
.multi-columns {
  -moz-column-count: 2;
  -webkit-column-count: 2;
  column-count: 2;
}
```

Result

Stack Overflow is a privately held website, the flagship site of the Stack Exchange Network,[4][5][6] created in 2008 by Jeff Atwood and Joel Spolsky.[7][8] It was created to be a more open alternative to earlier Q&A sites such as Experts-Exchange. The name for the website was chosen by voting in April 2008 by readers of Coding Horror, Atwood's popular programming blog.

The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers in a fashion similar to a wiki or Digg.[13] Users of Stack Overflow can earn reputation points and "badges"; for example, a person is awarded 10 reputation points for receiving an "up" vote on an answer given to a question, and can receive badges for their valued contributions,[14] which represents a kind of gamification of the traditional Q&A site or forum. All user-generated content is licensed under a Creative Commons Attribute-ShareAlike license.

Section 39.2: Column Width

The `column-width` property sets the minimum column width. If `column-count` is not defined the browser will make as many columns as fit in the available width.

Code:

```
<div id="multi-columns">
    Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut
labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt
in culpa qui officia deserunt mollit anim id est laborum
```

```
</div>
```

```
.multi-columns {
  -moz-column-width: 100px;
  -webkit-column-width: 100px;
  column-width: 100px;
}
```

结果

Stack Overflow is a privately held website, the flagship site of the Stack Exchange Network,[4][5] [6] created in 2008 by Jeff Atwood and Joel Spolsky. [7][8] It was created to be a more open alternative to earlier Q&A sites such as Experts-Exchange. The name for the website was chosen by voting in April 2008 by readers of Coding Horror, Atwood's popular programming blog.

The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers in a fashion similar to a wiki or Digg.[13] Users of Stack Overflow can earn reputation points and "badges"; for example, a person is awarded 10 reputation points for receiving an "up" vote on an answer given to a question, and can receive badges for their valued contributions, [14] which represents a kind of gamification of the traditional Q&A site or forum. All user-generated content is licensed under a Creative Commons Attribute-ShareAlike license.


```
</div>
```

```
.multi-columns {
  -moz-column-width: 100px;
  -webkit-column-width: 100px;
  column-width: 100px;
}
```

Result

Stack Overflow is a privately held website, the flagship site of the Stack Exchange Network,[4][5] [6] created in 2008 by Jeff Atwood and Joel Spolsky. [7][8] It was created to be a more open alternative to earlier Q&A sites such as Experts-Exchange. The name for the website was chosen by voting in April 2008 by readers of Coding Horror, Atwood's popular programming blog.

The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers in a fashion similar to a wiki or Digg.[13] Users of Stack Overflow can earn reputation points and "badges"; for example, a person is awarded 10 reputation points for receiving an "up" vote on an answer given to a question, and can receive badges for their valued contributions, [14] which represents a kind of gamification of the traditional Q&A site or forum. All user-generated content is licensed under a Creative Commons Attribute-ShareAlike license.

第40章：多栏布局

CSS允许定义元素内容分成多栏，并在栏之间设置间隙和分隔线。

第40.1节：创建多栏

```
<div class="content">
洛雷姆·伊普苏姆（Lorem ipsum）是虚构的文本，用于排版和设计，表示“痛苦坐着，爱着，做着，时间流
逝，工作和痛苦带来伟大的快乐。为了获得最小的好处，谁不愿意进行无节制的锻炼，除非从事某种便利
的结果。痛苦的厌恶者在享乐中不会感到痛苦。谁愿意承担痛苦的责任，除非为了获得某种利益。”

在波动中承受痛苦是烦恼的结果，痛苦带来快乐，真正的痛苦是为了获得荣誉和正义的痛苦，轻视
那些带来痛苦的事情。自由的时间使我们能够选择无拘无束的选项，没有任何限制，主宰着我们所能做
的最愉快的事情。


</div>
```

CSS

```
.内容 {
-webkit-column-count: 3; /* Chrome，Safari，Opera */
-moz-column-count: 3; /* Firefox */
column-count: 3;
}
```

第40.2节：基本示例

请考虑以下HTML标记：

```
<section>
  <p>Lorem ipsum dolor sit amet，consetetur sadipscing elitr，sed diam nonumy eirmod tempor
invidunt ut labore et dolore magna aliquyam erat，sed diam voluptua。At vero eos et accusam et
justo duo dolores et ea rebum。</p>
  <p> Stet clita kasd gubergren，no sea takimata sanctus est Lorem ipsum dolor sit amet。Lorem
ipsum dolor sit amet，consetetur sadipscing elitr，sed diam nonumy eirmod tempor invidunt ut labore
et dolore magna aliquyam erat，sed diam voluptua。At vero eos et accusam et justo duo dolores et ea
rebum。Stet clita kasd gubergren，no sea takimata sanctus est Lorem ipsum dolor sit amet。</p>
  <p>Lorem ipsum dolor sit amet，consetetur sadipscing elitr，sed diam nonumy eirmod tempor
invidunt ut labore et dolore magna aliquyam erat，sed diam voluptua。At vero eos et accusam et
justo duo dolores et ea rebum。Stet clita kasd gubergren，no sea takimata sanctus est Lorem ipsum
dolor sit amet。</p>
</section>
```

应用以下CSS后，内容被分成三栏，中间由两像素的灰色栏线分隔。

```
section {
  columns: 3;
  column-gap: 40px;
  column-rule: 2px 实线 灰色;
}
```

请参见 JSFiddle 上的实时示例。

Chapter 40: Multiple columns

CSS allows to define that element contents wrap into multiple columns with gaps and rules between them.

Section 40.1: Create Multiple Columns

```
<div class="content">
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh
euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim
ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl
ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in
hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu
feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui
blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla
facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil
imperdiet doming id quod mazim placerat facer possim assum.
</div>
```

CSS

```
.content {
-webkit-column-count: 3; /* Chrome, Safari, Opera */
-moz-column-count: 3; /* Firefox */
column-count: 3;
}
```

Section 40.2: Basic example

Consider the following HTML markup:

```
<section>
  <p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor
invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et
justo duo dolores et ea rebum.</p>
  <p> Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem
ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore
et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.</p>
  <p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor
invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et
justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum
dolor sit amet.</p>
</section>
```

With the following CSS applied the content is split into three columns separated by a gray column rule of two pixels.

```
section {
  columns: 3;
  column-gap: 40px;
  column-rule: 2px solid gray;
}
```

See a live sample of this on JSFiddle.

第41章：内联块布局

第41.1节：两端对齐的导航栏

水平两端对齐的导航（菜单）栏包含若干个应当对齐的项目。第一个（左侧）项目在容器内无左边距，最后一个（右侧）项目在容器内无右边距。项目之间的距离相等，与各个项目的宽度无关。

HTML

```
<nav>
    <ul>
        <li>abc</li>
        <li>abcdefghijkl</li>
        <li>abcdef</li>
    </ul>
</nav>
```

CSS

```
导航 {
    width: 100%;
    行高: 1.4em;
}
ul {
    列表样式: 无;
    显示: 块级;
    宽度: 100%;
    margin: 0;
    padding: 0;
    文本对齐: 两端对齐;
    下边距: -1.4em;
}
无序列表:之后 {
    内容: "";
    显示: 行内块;
    宽度: 100%;
}
li {
    显示: 行内块;
}
```

注释

- 选择 nav、ul 和 li 标签是因为它们在语义上表示“导航（菜单）项列表”。当然，也可以使用其他标签。
- :after 伪元素在 ul 中造成了额外的“行”，从而使该块有额外的空高度，将其他内容向下推。这通过负的 margin-bottom 解决，负的 margin-bottom 必须与 line-height 的大小相同（但为负值）。
- 如果页面变得过窄，无法容纳所有项目，项目将换行（从右侧开始）并在该行两端对齐。菜单的总高度将根据需要增加。

Chapter 41: Inline-Block Layout

Section 41.1: Justified navigation bar

The horizontally justified navigation (menu) bar has some number of items that should be justified. The first (left) item has no left margin within the container, the last (right) item has no right margin within the container. The distance between items is equal, independent on the individual item width.

HTML

```
<nav>
    <ul>
        <li>abc</li>
        <li>abcdefghijkl</li>
        <li>abcdef</li>
    </ul>
</nav>
```

CSS

```
nav {
    width: 100%;
    line-height: 1.4em;
}
ul {
    list-style: none;
    display: block;
    width: 100%;
    margin: 0;
    padding: 0;
    text-align: justify;
    margin-bottom: -1.4em;
}
ul:after {
    content: "";
    display: inline-block;
    width: 100%;
}
li {
    display: inline-block;
}
```

Notes

- The `nav`, `ul` and `li` tags were chosen for their semantic meaning of 'a list of navigation (menu) items'. Other tags may also be used of course.
- The `:after` pseudo-element causes an extra 'line' in the `ul` and thus an extra, empty height of this block, pushing other content down. This is solved by the negative `margin-bottom`, which has to have the same magnitude as the `line-height` (but negative).
- If the page becomes too narrow for all the items to fit, the items will break to a new line (starting from the right) and be justified on this line. The total height of the menu will grow as needed.

第42章：继承

第42.1节：自动继承

继承是CSS的一个基本机制，通过该机制，某些属性的计算值会应用到元素的子元素上。当你想为元素设置全局样式，而不是为标记中的每个元素单独设置这些属性时，这尤其有用。

常见的自动继承属性有：字体（font）、颜色（color）、文本对齐（text-align）、行高（line-height）。

假设以下样式表：

```
#myContainer {
  color: red;
  padding: 5px;
}
```

这将把color: red应用到 元素，不仅如此，还会应用到<h3>和<p>元素上。然而，由于padding的特性，其值不会被继承到这些元素上。

```
<div id="myContainer">
  <h3>一些标题</h3>
  <p>一些段落</p>
</div>
```

第42.2节：强制继承

有些属性不会自动从元素继承到其子元素。这是因为这些属性通常希望对应用该属性的元素（或元素选择）保持唯一。常见的此类属性有margin、padding、background、display等。

然而，有时仍然希望继承。为此，我们可以将inherit值应用于应继承的属性。该inherit值可以应用于任何CSS属性和任何HTML元素。

假设以下样式表：

```
#myContainer {
  color: red;
  padding: 5px;
}
#myContainer p {
  padding: inherit;
}
```

由于color属性的继承特性，这将使<h3>和<p>元素都应用color: red。但是，<p>元素也会继承其父元素的padding值，因为这是指定的。

```
<div id="myContainer">
  <h3>一些标题</h3>
  <p>一些段落</p>
</div>
```


Chapter 42: Inheritance

Section 42.1: Automatic inheritance

Inheritance the a fundamental mechanism of CSS by which the computed values of some properties of an element are applied to its' children. This is particularly useful when you want to set a global style to your elements rather than having to set said properties to each and every element in your markup.

Common properties that are automatically inherited are: `font`, `color`, `text-align`, `line-height`.

Assume the following stylesheet:

```
#myContainer {
  color: red;
  padding: 5px;
}
```

This will apply **`color`**: `red` not only to the **`<div>`** element but also to the **`<h3>`** and **`<p>`** elements. However, due to the nature of `padding` its value will ***not*** be inherited to those elements.

```
<div id="myContainer">
  <h3>Some header</h3>
  <p>Some paragraph</p>
</div>
```

Section 42.2: Enforced inheritance

Some properties are not automatically inherited from an element down to its' children. This is because those properties are typically desired to be unique to the element (or selection of elements) to which the property is applied to. Common such properties are `margin`, `padding`, `background`, `display`, etc.

However, sometimes inheritance is desired anyway. To achieve this, we can apply the `inherit` value to the property that should be inherited. The `inherit` value can be appied to *any* CSS property and *any* HTML element.

Assume the following stylesheet:

```
#myContainer {
  color: red;
  padding: 5px;
}
#myContainer p {
  padding: inherit;
}
```

This will apply **`color`**: `red` to both the **`<h3>`** and **`<p>`** elements due to the inheritance nature of the `color` property. However, the **`<p>`** element will also inherit the `padding` value from its' parent because this was specified.

```
<div id="myContainer">
  <h3>Some header</h3>
  <p>Some paragraph</p>
</div>
```

第43章：CSS图像精灵

第43.1节：基本实现

什么是图像精灵？

图像精灵是位于图像精灵表中的单个资源。图像精灵表是一个图像文件，
其中包含多个可以从中提取的资源。

例如：

上图是一个图像精灵表（sprite sheet），其中的每颗星星都是精灵表中的一个精灵。这些精灵表很有用，因为它们通过减少浏览器可能需要发出的HTTP请求数量来提升性能。

那么如何实现一个呢？这里有一些示例代码。

HTML

```
<div class="icon icon1"></div>
<div class="icon icon2"></div>
<div class="icon icon3"></div>
```

CSS

```
.icon {
    background: url(“icons-sprite.png”);
    display: inline-block;
    height: 20px;
    width: 20px;
}
.icon1 {
      background-position: 0px 0px;
}
.icon2 {
      background-position: -20px 0px;
}
.icon3 {
      background-position: -40px 0px;
}
```

通过设置精灵的宽度和高度，并使用 CSS 中的 background-position 属性（带有 x 和 y 值），你可以轻松地使用 CSS 从精灵图中提取精灵。

Chapter 43: CSS Image Sprites

Section 43.1: A Basic Implementation

What's an image sprite?

An image sprite is a single asset located within an image sprite sheet. An image sprite sheet is an image file that contains more than one asset that can be extracted from it.

For example:

The image above is an image sprite sheet, and each one of those stars is a sprite within the sprite sheet. These sprite sheets are useful because they improve performance by reducing the number of HTTP requests a browser might have to make.

So how do you implement one? Here's some example code.

HTML

```
<div class="icon icon1"></div>
<div class="icon icon2"></div>
<div class="icon icon3"></div>
```

CSS

```
.icon {
    background: url(“icons-sprite.png”);
    display: inline-block;
    height: 20px;
    width: 20px;
}
.icon1 {
      background-position: 0px 0px;
}
.icon2 {
      background-position: -20px 0px;
}
.icon3 {
      background-position: -40px 0px;
}
```

By using setting the sprite's width and height and by using the background-position property in CSS (with an x and y value) you can easily extract sprites from a sprite sheet using CSS.

第44章：裁剪和遮罩

参数	详情
clip-source	一个 URL，可以指向内联 SVG 元素或包含裁剪路径定义的外部文件中的 SVG 元素。
basic-shape	指的是 inset()、circle()、ellipse() 或 polygon() 中的一个。使用这些函数之一定义裁剪路径。这些形状函数的工作方式与 Shapes for Floats 中完全相同。
clip-geometry-box	这可以是content-box、padding-box、border-box、margin-box、fill-box中的一种。stroke-box、view-box作为值。当未为<basic-shape>提供任何值时，使用相应盒子的边缘作为裁剪路径。与<basic-shape>一起使用时，它作为形状的参考盒。
mask-reference	这可以是none，或者是图像，或者是指向遮罩图像源的引用URL。
repeat-style	指定遮罩在X轴和Y轴上的重复或平铺方式。支持的值有repeat-x、repeat-y、repeat、space、round、no-repeat。
mask-mode	可以是alpha、luminance或auto，表示遮罩应被视为alpha遮罩还是亮度遮罩。如果未提供值且mask-reference是直接图像，则视为alpha遮罩；如果mask-reference是URL，则视为亮度遮罩。
position	指定每个遮罩层的位置，行为类似于background-position属性。值可以采用单值语法（如top、10%）或双值语法（如top right、50% 50%）。
几何盒	这指定了应将遮罩裁剪到的盒子（遮罩绘制区域）或应作为遮罩原点参考的盒子（遮罩定位区域），具体取决于属性。可能的值列表包括内容盒、内边距盒、边框盒、外边距盒、填充盒、描边盒、视图盒。关于这些值的详细说明可参见W3C规范。
背景大小	这表示每个遮罩图像层的大小，语法与background-size相同。值可以是长度、百分比、auto、cover或contain。长度、百分比和auto可以作为单个值提供，也可以为每个轴分别提供。
合成操作符	这可以是每层中的add、subtract、exclude、multiply中的任意一个，定义了该层与其下方层应使用的合成操作类型。关于每个值的详细说明可参见W3C规范。

第44.1节：裁剪与遮罩：概述与区别

通过裁剪和遮罩，您可以使元素的某些指定部分变为透明或不透明。两者都可以应用于任何HTML元素。

裁剪

剪辑是矢量路径。路径外的部分元素将是透明的，路径内则是不透明的。因此，你可以在元素上定义一个clip-path属性。所有在SVG中存在的图形元素都可以在这里用作定义路径的函数。示例包括circle()、polygon()或ellipse()。

Chapter 44: Clipping and Masking

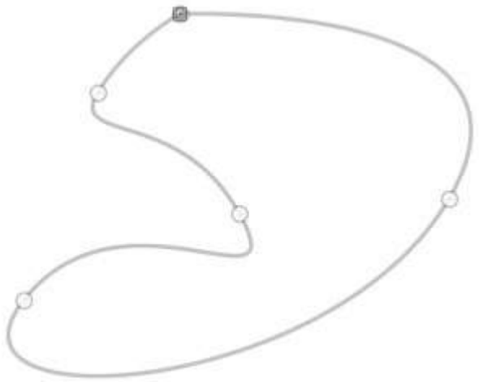
Parameter	Details
clip-source	A URL which can point to an inline SVG element (or) an SVG element in an external file that contains the clip path's definition.
basic-shape	Refers to one among `inset()`, `circle()`, `ellipse()` or `polygon()`. Using one of these functions the clipping path is defined. These shape functions work exactly the same way as they do in Shapes for Floats
clip-geometry-box	This can have one among `content-box`, `padding-box`, `border-box`, `margin-box`, `fill-box`, `stroke-box`, `view-box` as values. When this is provided without any value for <basic-shape>, the edges of the corresponding box is used as the path for clipping. When used with a <basic-shape>, this acts as the reference box for the shape.
mask-reference	This can be `none` or an image or a reference URL to a mask image source.
repeat-style	This specifies how the mask should be repeated or tiled in the X and Y axes. The supported values are `repeat-x`, `repeat-y`, `repeat`, `space`, `round`, `no-repeat`.
mask-mode	Can be `alpha` or `luminance` or `auto` and indicates whether the mask should be treated as a alpha mask or a luminance mask. If no value is provided and the mask-reference is a direct image then it would be considered as alpha mask (or) if the mask-reference is a URL then it would be considered as luminance mask.
position	This specifies the position of each mask layer and is similar in behavior to the `background-position` property. The value can be provided in 1 value syntax (like `top`, `10%`) or in 2 value syntax (like `top right`, `50% 50%`).
geometry-box	This specifies the box to which the mask should be clipped (*mask painting area*) or the box which should be used as reference for the mask's origin (*mask positioning area*) depending on the property. The list of possible values are `content-box`, `padding-box`, `border-box`, `margin-box`, `fill-box`, `stroke-box`, `view-box`. Detailed explanation of how each of those values work is available in the W3C Spec.
bg-size	This represents the size of each mask-image layer and has the same syntax as `background-size`. The value can be length or percentage or auto or cover or contain. Length, percentage and auto can either be provided as a single value or as one for each axis.
compositing-operator	This can be any one among `add`, `subtract`, `exclude`, `multiply` per layer and defines the type of compositing operation that should be used for this layer with those below it. Detailed explanation about each value is available in the W3C Specs.

Section 44.1: Clipping and Masking: Overview and Difference

With **Clipping** and **Masking** you can make some specified parts of elements transparent or opaque. Both can be applied to any HTML element.

Clipping

Clips are vector paths. Outside of this path the element will be transparent, inside it's opaque. Therefore you can define a `clip-path` property on elements. Every graphical element that also exists in SVG you can use here as a function to define the path. Examples are `circle()`, `polygon()` or `ellipse()`.



示例

```
clip-path: 圆形(100像素 在 中心);
```

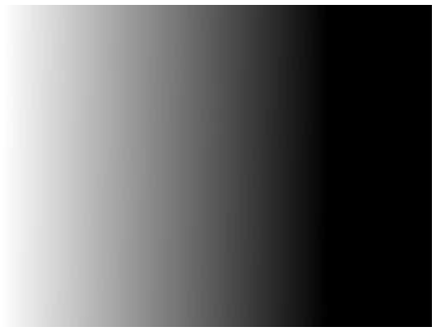
该元素仅在此圆形内可见，该圆形位于元素中心，半径为100像素。

遮罩

遮罩类似于裁剪，但不是定义路径，而是定义覆盖在元素上的遮罩。你可以将此遮罩想象成主要由黑白两色组成的图像。

亮度遮罩：黑色表示区域不透明，白色表示透明，但也有灰色区域是半透明的，因此你可以实现平滑过渡。

Alpha遮罩：只有遮罩的透明区域，元素才是不透明的。



例如，这张图像可以用作亮度遮罩，使元素从右到左实现非常平滑的不透明到透明的过渡。

mask 属性允许你指定遮罩类型和用作图层的图像。

示例

```
mask: url(masks.svg#rectangle) 亮度;
```

一个名为rectangle的元素定义在masks.svg中，将作为元素上的亮度遮罩使用。

Example

```
clip-path: circle(100px at center);
```

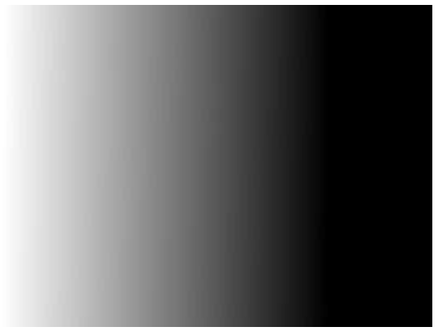
The element will be only visible inside of this circle, which is positioned at the center of the element and has a radius of 100px.

Masking

Masks are similar to Clips, but instead of defining a path you define a mask what layers over the element. You can imagine this mask as an image what consist of mainly two colors: black and white.

Luminance Mask: Black means the region is opaque, and white that it's transparent, but there is also a grey area which is semi-transparent, so you are able to make smooth transitions.

Alpha Mask: Only on the transparent areas of the mask the element will be opaque.



This image for example can be used as a luminance mask to make for an element a very smooth transition from right to left and from opaque to transparent.

The `mask` property let you specify the the mask type and an image to be used as layer.

Example

```
mask: url(masks.svg#rectangle) luminance;
```

An element called `rectangle` defined in `masks.svg` will be used as an **luminance mask** on the element.

第44.2节：简单遮罩，使图像从实心渐变为透明

CSS

```
div {
   高度: 200像素;
   宽度: 200px;
   背景: url(http://lorempixel.com/200/200/nature/1);
   遮罩图像: 线性渐变(向右, 白色, 透明);
}
```

HTML

```
<div></div>
```

在上述示例中，有一个元素以图像作为其背景。应用于图像上的遮罩（使用CSS）使其看起来像是从左到右逐渐消失。

遮罩是通过使用一个线性渐变作为遮罩实现的，该渐变从左侧的白色到右侧的透明。由于它是一个alpha遮罩，图像在遮罩透明的地方变得透明。

无遮罩输出：

有遮罩输出：

注意：如备注中所述，上述示例仅在Chrome、Safari和Opera中使用-webkit前缀时有效。该示例（使用线性渐变作为遮罩图像）尚不支持Firefox。

第44.3节：裁剪（圆形）

css :

```
div{
```


Section 44.2: Simple mask that fades an image from solid to transparent

CSS

```
div {
  height: 200px;
  width: 200px;
  background: url(http://lorempixel.com/200/200/nature/1);
  mask-image: linear-gradient(to right, white, transparent);
}
```

HTML

```
<div></div>
```

In the above example there is an element with an image as its background. The mask that is applied on the image (using CSS) makes it look as though it is fading out from left to right.

The masking is achieved by using a `linear-gradient` that goes from white (on the left) to transparent (on the right) as the mask. As it is an alpha mask, image becomes transparent where the mask is transparent.

Output without the mask:

Output with the mask:

Note: As mentioned in remarks, the above example would work in Chrome, Safari and Opera only when used with the `-webkit` prefix. This example (with a `linear-gradient` as mask image) is not yet supported in Firefox.

Section 44.3: Clipping (Circle)

CSS:

```
div{
```

```
  宽度: 200px;
  高度: 200像素;
  背景: 青绿色;
  clip-path: circle(30% at 50% 50%); /* 使用前请参阅备注 */
}
```

HTML

```
<div></div>
```

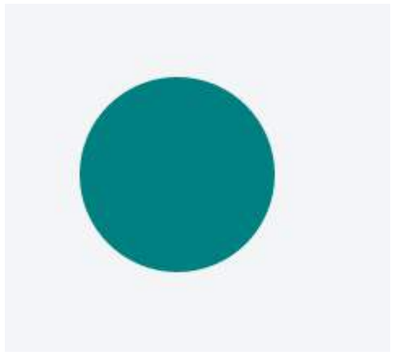
此示例展示了如何将一个div裁剪成圆形。该元素被裁剪成一个圆，其半径为参考框尺寸的30%，圆心位于参考框的中心点。这里由于未提供<clip-geometry-box>（换句话说，参考框），元素的border-box将被用作参考框。

圆形需要有一个半径和一个中心点，中心点的坐标为(x,y)：

```
circle(radius at x y)
```

查看示例

输出：



第44.4节：裁剪（多边形）

CSS：

```
div{
  width:200px;
  height:200px;
  background:teal;
  clip-path: polygon(0 0, 0 100%, 100% 50%); /* 使用前请参阅备注 */
}
```

HTML：

```
<div></div>
```

在上述示例中，使用了一个多边形裁剪路径将正方形（200 x 200）元素裁剪成三角形。输出形状是三角形，因为路径起点（即第一个坐标）位于0 0——这是盒子的左上角，接着路径到达0 100%——这是盒子的左下角，最后到达100% 50%，即盒子的右中点。这些路径是自闭合的（即起点也是终点），因此最终形状是三角形。

这也可以用于背景为图像或渐变的元素。

查看示例


```
  width: 200px;
  height: 200px;
  background: teal;
  clip-path: circle(30% at 50% 50%); /* refer remarks before usage */
}
```

HTML

```
<div></div>
```

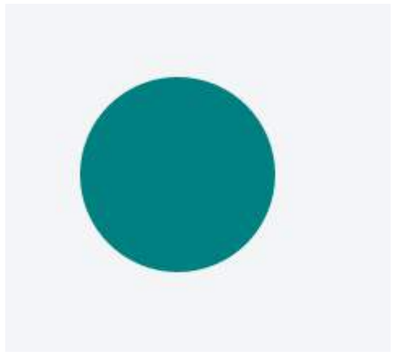
This example shows how to clip a div to a circle. The element is clipped into a circle whose radius is 30% based on the dimensions of the reference box with its center point at the center of the reference box. Here since no <clip-geometry-box> (in other words, reference box) is provided, the `border-box` of the element will be used as the reference box.

The circle shape needs to have a radius and a center with `(x,y)` coordinates:

```
circle(radius at x y)
```

View Example

Output:



Section 44.4: Clipping (Polygon)

CSS:

```
div{
  width:200px;
  height:200px;
  background:teal;
  clip-path: polygon(0 0, 0 100%, 100% 50%); /* refer remarks before usage */
}
```

HTML:

```
<div></div>
```

In the above example, a **polygonal** clipping path is used to clip the square (200 x 200) element into a triangle shape. The output shape is a triangle because the path starts at (that is, first coordinates are at) `0 0` - which is the top-left corner of the box, then goes to `0 100%` - which is bottom-left corner of the box and then finally to `100% 50%` which is nothing but the right-middle point of the box. These paths are self closing (that is, the starting point will be the ending point) and so the final shape is that of a triangle.

This can also be used on an element with an image or a gradient as background.

View Example

输出：

第44.5节：使用蒙版在图像中间挖孔

CSS

```
div {
    宽度: 200px;
    高度: 200像素;
    background: url(http://lorempixel.com/200/200/abstract/6);
    mask-image: radial-gradient(circle farthest-side at center, transparent 49%, white 50%); /* 使用前请查看备注 */

}
```

HTML

在上述示例中，使用径向渐变在中心创建了一个透明圆，然后将其用作蒙版，产生从图像中心挖出一个圆形的效果。

无蒙版的图像：

有蒙版的图像：

Output:

Section 44.5: Using masks to cut a hole in the middle of an image

CSS

```
div {
  width: 200px;
  height: 200px;
  background: url(http://lorempixel.com/200/200/abstract/6);
  mask-image: radial-gradient(circle farthest-side at center, transparent 49%, white 50%); /* check
remarks before using */
}
```

HTML

In the above example, a transparent circle is created at the center using `radial-gradient` and this is then used as a mask to produce the effect of a circle being cut out from the center of an image.

Image without mask:

Image with mask:

第44.6节：使用蒙版创建不规则形状的图像

CSS

```
div { /* 使用前检查备注 */
  height: 200px;
  宽度: 400像素;
  背景图像: url(http://lorempixel.com/400/200/自然/4);蒙版图像: 线性渐变(向右上
  角, 透明 49.5%, 白色 50.5%), 线性渐变(向左上角, 透明 49.5%, 白色 50.5%), 线性渐变(白色, 白色);

  蒙版大小: 75% 25%, 25% 25%, 100% 75%;
  蒙版位置: 左下, 右下, 左上;
  蒙版重复: 不重复;
}
```

HTML

```
<div></div>
```

在上述示例中，使用了三个线性渐变图像（当放置在适当位置时将覆盖容器大小的100% x 100%）作为蒙版，在图像底部产生一个透明的三角形切割效果。

无蒙版的图像：

带蒙版的图像：

Section 44.6: Using masks to create images with irregular shapes

CSS

```
div { /* check remarks before usage */
  height: 200px;
  width: 400px;
  background-image: url(http://lorempixel.com/400/200/nature/4);
  mask-image: linear-gradient(to top right, transparent 49.5%, white 50.5%), linear-gradient(to top
left, transparent 49.5%, white 50.5%), linear-gradient(white, white);
  mask-size: 75% 25%, 25% 25%, 100% 75%;
  mask-position: bottom left, bottom right, top left;
  mask-repeat: no-repeat;
}
```

HTML

```
<div></div>
```

In the above example, three `linear-gradient` images (which when placed in their appropriate positions would cover 100% x 100% of the container's size) are used as masks to produce a transparent triangular shaped cut at the bottom of the image.

Image without the mask:

Image with the mask:

第45章：碎片化

值	描述
autoDefault。自动分页	
always	总是插入分页符
avoid	尽量避免分页符
left	插入分页符，使下一页格式为左页
右页	插入分页符，使下一页格式为右页
initial	将此属性设置为其默认值。
inherit	从其父元素继承此属性。

第45.1节：媒体打印分页

```
@media print {
  p {
    page-break-inside: avoid;
  }
h1 {
    page-break-before: always;
  }
h2 {
    page-break-after: avoid;
  }
}
```

这段代码实现了三件事：

- 它防止在任何 p 标签内出现分页，这意味着如果可能，段落永远不会被分成两页。
- 它会在所有h1标题前强制分页，这意味着在每个h1出现之前都会有一个分页符。
- 它防止在任何h2之后立即分页

Chapter 45: Fragmentation

Value	Description
auto	Default. Automatic page breaks
always	Always insert a page break
avoid	Avoid page break (if possible)
left	Insert page breaks so that the next page is formatted as a left page
right	Insert page breaks so that the next page is formatted as a right page
initial	Sets this property to its default value.
inherit	Inherits this property from its parent element.

Section 45.1: Media print page-break

```
@media print {
  p {
    page-break-inside: avoid;
  }
  h1 {
    page-break-before: always;
  }
  h2 {
    page-break-after: avoid;
  }
}
```

This code does 3 things:

- it prevents a page break inside any p tags, meaning a paragraph will never be broken in two pages, if possible.
- it forces a page-break-before in all h1 headings, meaning that before every h1 occurrence, there will be a page break.
- it prevents page-breaks right after any h2

第46章：CSS对象模型（CSSOM）

第46.1节：通过CSSOM添加background-image规则

要通过CSSOM添加background-image规则，首先获取第一个样式表的规则引用：

```
var stylesheet = document.styleSheets[0].cssRules;
```

然后，获取样式表末尾的引用：

```
var end = stylesheet.length - 1;
```

最后，在样式表末尾为body元素插入background-image规则：

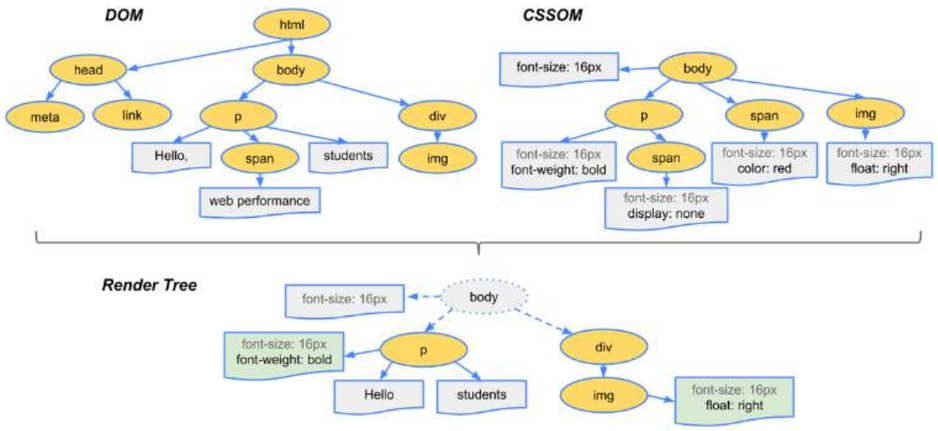
```
stylesheet.insertRule("body { background-image:
url('http://cdn.sstatic.net/Sites/stackoverflow/img/favicon.ico'); }", end);
```

第46.2节：介绍

浏览器从样式表中识别标记，并将其转换为节点，这些节点被链接成树状结构。
页面中所有节点及其关联样式的完整映射即为CSS对象模型（CSS Object Model）。

为了显示网页，网页浏览器执行以下步骤。

1. 网页浏览器检查你的HTML并构建DOM（文档对象模型）。
2. 网页浏览器检查你的CSS并构建CSSOM（CSS对象模型）。
3. 网页浏览器将DOM和CSSOM结合，创建渲染树。网页浏览器显示你的网页。



Chapter 46: CSS Object Model (CSSOM)

Section 46.1: Adding a background-image rule via the CSSOM

To add a background-image rule via the CSSOM, first get a reference to the rules of the first stylesheet:

```
var stylesheet = document.styleSheets[0].cssRules;
```

Then, get a reference to the end of the stylesheet:

```
var end = stylesheet.length - 1;
```

Finally, insert a background-image rule for the body element at the end of the stylesheet:

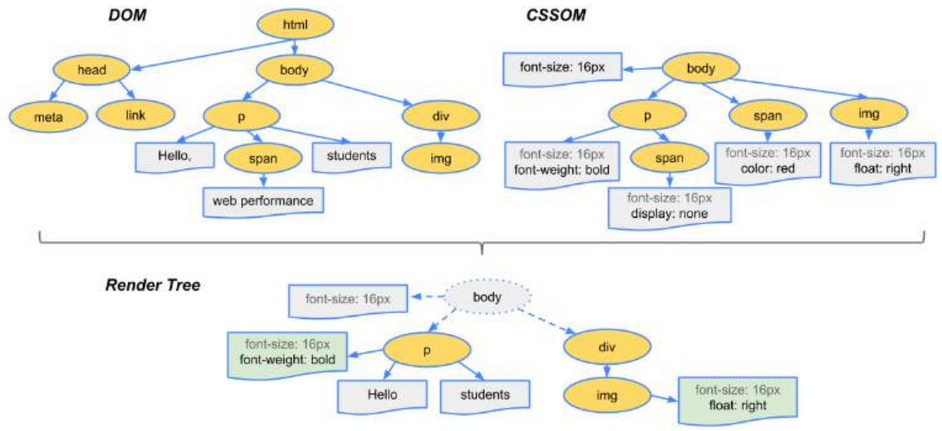
```
stylesheet.insertRule("body { background-image:
url('http://cdn.sstatic.net/Sites/stackoverflow/img/favicon.ico'); }", end);
```

Section 46.2: Introduction

The browser identifies tokens from stylesheet and coverts them into nodes which are linked into a tree structure.
The entire map of all the nodes with their associated styles of a page would be the CSS Object Model.

To display the webpage, a web browser takes following steps.

1. The web browser examines your HTML and builds the DOM (Document Object Model).
2. The web browser examines your CSS and builds the CSSOM (CSS Object Model).
3. The web browser combines the DOM and the CSSOM to create a render tree. The web browser displays your webpage.



第47章：特性查询

参数	详情
(property: value)	如果浏览器能处理该CSS规则，则评估为真。规则周围的括号是必需的。
和	仅当前一个和后一个条件都为真时返回真。
not	否定下一个条件
或	如果前一个或后一个条件为真，则返回真。
(...)	分组条件

第47.1节：基本的 @supports 用法

```
@supports (display: flex) {
  /* 支持 Flexbox，因此使用它 */
  .my-container {
    display: flex;
  }
}
```

在语法方面，@supports 与 @media 非常相似，但它不是检测屏幕大小和方向，@supports 会检测浏览器是否支持某个 CSS 规则。

与其写成 @supports (flex)，不如注意规则是 @supports (display: flex)。

第47.2节：链式特性检测

要同时检测多个特性，使用 and 运算符。

```
@supports (transform: translateZ(1px)) and (transform-style: preserve-3d) and (perspective: 1px) {
  /* 这里可能会做一些炫酷的3D效果 */
}
```

还有一个 or 运算符和一个 not 运算符：

```
@supports (display: flex) or (display: table-cell) {
  /* 如果浏览器支持flexbox或display: table-cell，则会使用这里的样式 */
}
@supports not (-webkit-transform: translate(0, 0, 0)) {
  /* 如果浏览器支持 -webkit-transform: translate(...)，则*不会*使用这里的样式 */
}
```

为了获得终极的 @supports 体验，尝试用括号对逻辑表达式进行分组：

```
@supports ((display: block) and (zoom: 1)) or ((display: flex) and (not (display: table-cell))) or
(transform: translateX(1px)) {
  /* ... */
}
```

如果浏览器支持，这将有效

1. 支持 display: block 和 zoom: 1，或者
2. 支持 display: flex 且不支持 display: table-cell，或者
3. 支持 transform: translateX(1px)。

Chapter 47: Feature Queries

Parameter	Details
(property: value)	Evaluates true if the browser can handle the CSS rule. The parenthesis around the rule are required.
and	Returns true only if both the previous and next conditions are true.
not	Negates the next condition
or	Returns true if either the previous or next condition is true.
(...)	Groups conditions

Section 47.1: Basic @supports usage

```
@supports (display: flex) {
  /* Flexbox is available, so use it */
  .my-container {
    display: flex;
  }
}
```

In terms of syntax, @supports is very similar to @media, but instead of detecting screen size and orientation, @supports will detect whether the browser can handle a given CSS rule.

Rather than doing something like @supports (flex), notice that the rule is @supports (display: flex).

Section 47.2: Chaining feature detections

To detect multiple features at once, use the and operator.

```
@supports (transform: translateZ(1px)) and (transform-style: preserve-3d) and (perspective: 1px) {
  /* Probably do some fancy 3d stuff here */
}
```

There is also an or operator and a not operator:

```
@supports (display: flex) or (display: table-cell) {
  /* Will be used if the browser supports flexbox or display: table-cell */
}
@supports not (-webkit-transform: translate(0, 0, 0)) {
  /* Will *not* be used if the browser supports -webkit-transform: translate(...) */
}
```

For the ultimate @supports experience, try grouping logical expressions with parenthesis:

```
@supports ((display: block) and (zoom: 1)) or ((display: flex) and (not (display: table-cell))) or
(transform: translateX(1px)) {
  /* ... */
}
```

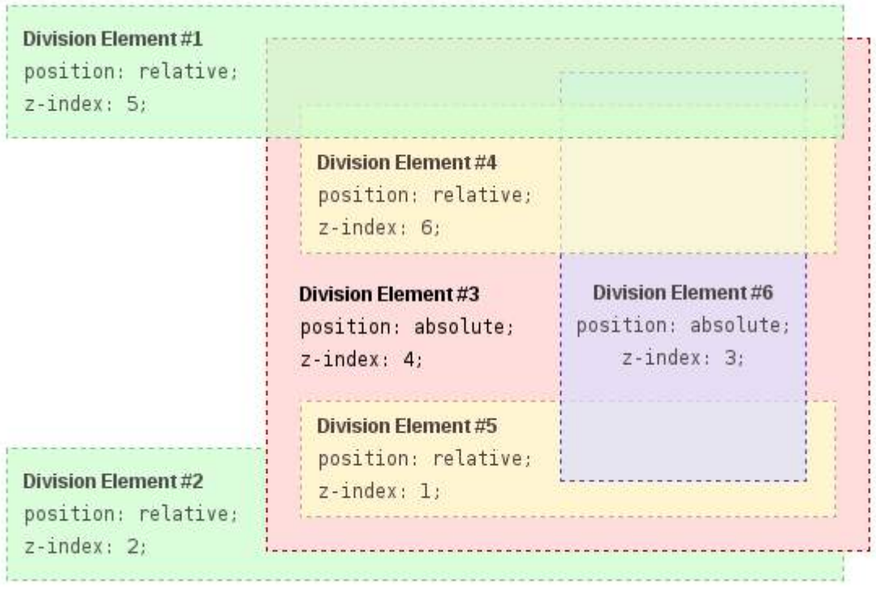
This will work if the browser

1. Supports display: block AND zoom: 1, or
2. Supports display: flex AND NOT display: table-cell, or
3. Supports transform: translateX(1px).

第48章：堆叠上下文

第48.1节：堆叠上下文

在此示例中，每个定位元素都会创建自己的堆叠上下文，因为它们的定位和 z-index 值。堆叠上下文的层级结构组织如下：



- 根元素
  - DIV #1
  - DIV #2
  - DIV #3
  - DIV #4
  - DIV #5
  - DIV #6

需要注意的是，DIV #4、DIV #5 和 DIV #6 是 DIV #3 的子元素，因此这些元素的堆叠完全在 DIV #3 内部解决。一旦 DIV #3 内部的堆叠和渲染完成，整个 DIV #3 元素将与其兄弟 DIV 一起传递到根元素进行堆叠。

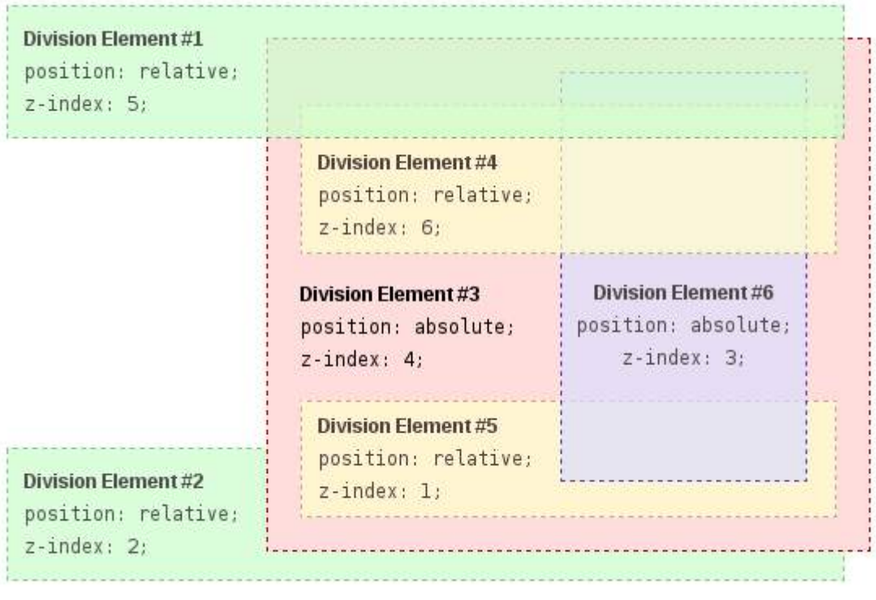
HTML：

```
<div id="div1">
   <h1>分区元素 #1</h1>
   <code>position: relative;<br/>
   z-index: 5;</code>
</div>
<div id="div2">
   <h1>分区元素 #2</h1>
   <code>position: relative;<br/>
   z-index: 2;</code>
</div>
<div id="div3">
   <div id="div4">
```


Chapter 48: Stacking Context

Section 48.1: Stacking Context

In this example every positioned element creates its own stacking context, because of their positioning and z-index values. The hierarchy of stacking contexts is organized as follows:



- Root
  - DIV #1
  - DIV #2
  - DIV #3
  - DIV #4
  - DIV #5
  - DIV #6

It is important to note that DIV #4, DIV #5 and DIV #6 are children of DIV #3, so stacking of those elements is completely resolved within DIV#3. Once stacking and rendering within DIV #3 is completed, the whole DIV #3 element is passed for stacking in the root element with respect to its sibling's DIV.

HTML:

```
<div id="div1">
  <h1>Division Element #1</h1>
  <code>position: relative;<br/>
  z-index: 5;</code>
</div>
<div id="div2">
  <h1>Division Element #2</h1>
  <code>position: relative;<br/>
  z-index: 2;</code>
</div>
<div id="div3">
  <div id="div4">
```

```
    <h1>分区元素 #4</h1>
    <code>position: relative;<br/>
   z-index: 6;</code>
  </div>
  <h1>分区元素 #3</h1>
  <code>position: absolute;<br/>
  z-index: 4;</code>
  <div id="div5">
    <h1>分区元素 #5</h1>
    <code>position: relative;<br/>
   z-index: 1;</code>
  </div>
  <div id="div6">
    <h1>分区元素 #6</h1>
    <code>position: absolute;<br/>
   z-index: 3;</code>
  </div>
</div>
```

css :

```
* {
    margin: 0;
}
html {
    padding: 20px ;
    字体: 12像素/20像素 Arial, 无衬线字体;
}
div {
    不透明度: 0.7;
    位置: 相对;
}
h1 {
    字体: 继承;
    字体粗细: 加粗;
}
#div1,
#div2 {
    边框: 1像素虚线 #696;
    内边距: 10像素;
    背景颜色: #cfc;
}
#div1 {
    层级: 5;
    下边距: 190像素;
}
#div2 {
    层级: 2;
}
#div3 {
    z-index: 4;
    opacity: 1;
    position: absolute;
    top: 40px;
    left: 180px;
    width: 330px;
    border: 1px dashed #900;
    background-color: #fdd;
    padding: 40px 20px 20px;
}
#div4,
```



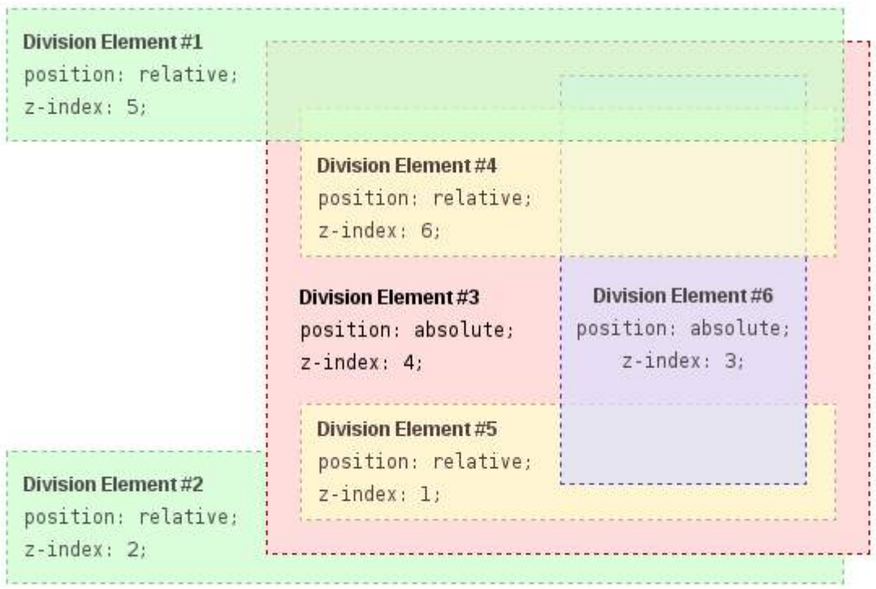
```
    <h1>Division Element #4</h1>
    <code>position: relative;<br/>
    z-index: 6;</code>
  </div>
  <h1>Division Element #3</h1>
  <code>position: absolute;<br/>
  z-index: 4;</code>
  <div id="div5">
    <h1>Division Element #5</h1>
    <code>position: relative;<br/>
    z-index: 1;</code>
  </div>
  <div id="div6">
    <h1>Division Element #6</h1>
    <code>position: absolute;<br/>
    z-index: 3;</code>
  </div>
</div>
```

CSS:

```
* {
    margin: 0;
}
html {
    padding: 20px;
    font: 12px/20px Arial, sans-serif;
}
div {
    opacity: 0.7;
    position: relative;
}
h1 {
    font: inherit;
    font-weight: bold;
}
#div1,
#div2 {
    border: 1px dashed #696;
    padding: 10px;
    background-color: #cfc;
}
#div1 {
    z-index: 5;
    margin-bottom: 190px;
}
#div2 {
    z-index: 2;
}
#div3 {
    z-index: 4;
    opacity: 1;
    position: absolute;
    top: 40px;
    left: 180px;
    width: 330px;
    border: 1px dashed #900;
    background-color: #fdd;
    padding: 40px 20px 20px;
}
#div4,
```

```css
#div5 {
    border: 1px dashed #996;
    background-color: #ffc;
}
#div4 {
    z-index: 6;
    margin-bottom: 15px;
    padding: 25px 10px 5px;
}
#div5 {
    z-index: 1;
    margin-top: 15px;
    padding: 5px 10px;
}
#div6 {
    z-index: 3;
    position: absolute;
    top: 20px;
    left: 180px;
    width: 150px;
    height: 125px;
    border: 1px dashed #009;
    padding-top: 125px;
    background-color: #ddf;
    text-align: center;
}
```

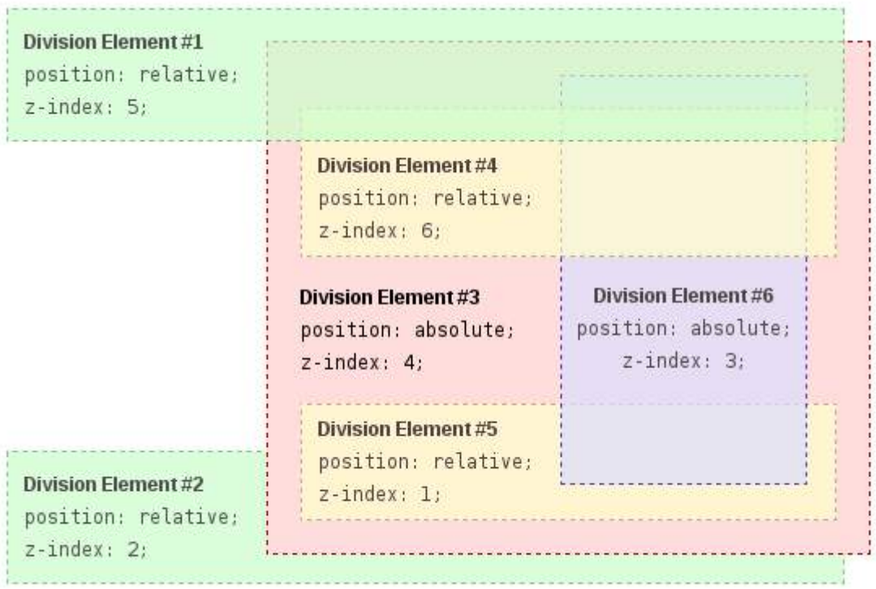
结果：



来源：
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Positioning/Understanding_z_index/The_stacking_context.


```css
#div5 {
    border: 1px dashed #996;
    background-color: #ffc;
}
#div4 {
    z-index: 6;
    margin-bottom: 15px;
    padding: 25px 10px 5px;
}
#div5 {
    z-index: 1;
    margin-top: 15px;
    padding: 5px 10px;
}
#div6 {
    z-index: 3;
    position: absolute;
    top: 20px;
    left: 180px;
    width: 150px;
    height: 125px;
    border: 1px dashed #009;
    padding-top: 125px;
    background-color: #ddf;
    text-align: center;
}
```

Result:

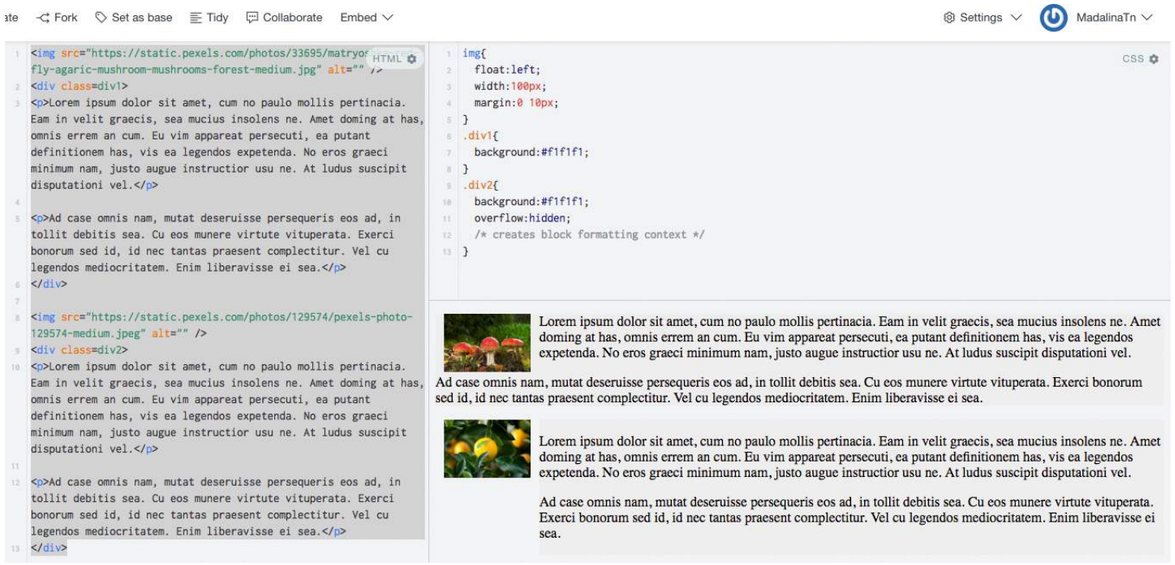


Source:
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Positioning/Understanding_z_index/The_stacking_context.

第49章：块格式化上下文

第49.1节：使用overflow属性且值不为visible

```
img{
   float:left;
   width:100px;
   margin:0 10px;
}
.div1{
   background:#f1f1f1;
   /* 不会创建块格式化上下文 */
}
.div2{
   background:#f1f1f1;
   overflow:hidden;
   /* 创建块格式化上下文 */
}
```



https://jsfiddle.net/MadalinaTn/qkwwmu6m/2/

> 使用 overflow 属性且值不为 visible（默认值）将创建一个新的块格式化上下文。这在技术上是必要的—如果浮动元素与滚动元素相交，它将强制重新换行内容。

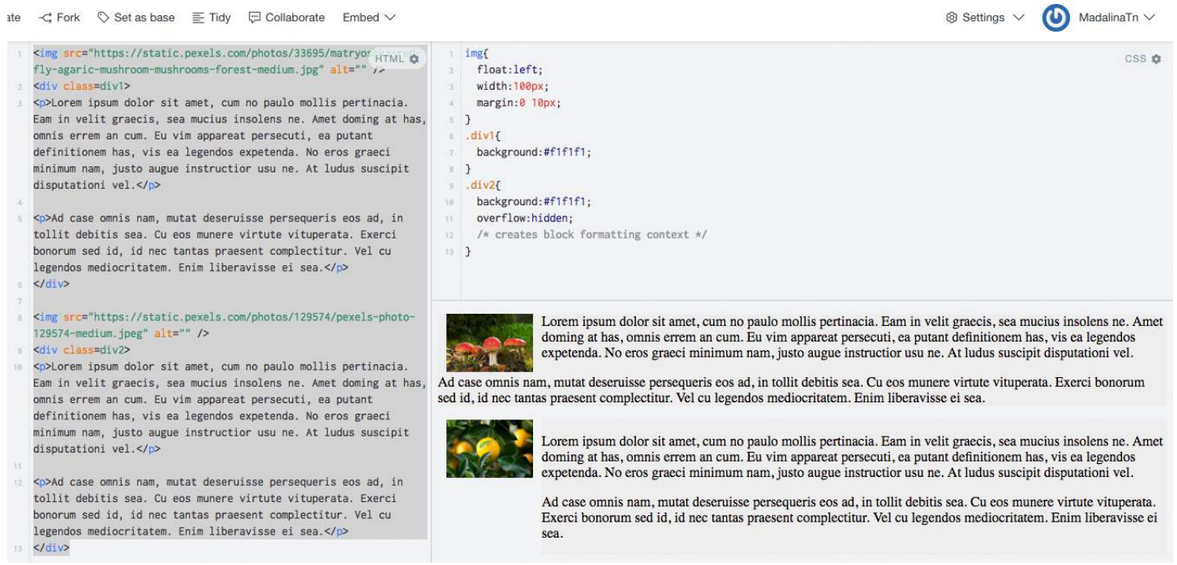
这个示例展示了多个段落如何与浮动图像交互，类似于这个示例，位于css-tricks.com。

2: https://developer.mozilla.org/en-US/docs/Web/CSS/overflow MDN

Chapter 49: Block Formatting Contexts

Section 49.1: Using the overflow property with a value different to visible

```
img{
   float:left;
   width:100px;
   margin:0 10px;
}
.div1{
   background:#f1f1f1;
   /* does not create block formatting context */
}
.div2{
   background:#f1f1f1;
   overflow:hidden;
   /* creates block formatting context */
}
```



https://jsfiddle.net/MadalinaTn/qkwwmu6m/2/

> Using the overflow property with a value different to visible (its default) will create a new block formatting context. This is technically necessary — if a float intersected with the scrolling element it would forcibly rewrap the content.

This example that show how a number of paragraphs will interact with a floated image is similar to this example, on css-tricks.com.

2: https://developer.mozilla.org/en-US/docs/Web/CSS/overflow MDN

第50章：垂直居中

第50.1节：使用display: table进行居中

HTML：

```
<div class="wrapper">
    <div class="outer">
        <div class="inner">
          居中
        </div>
    </div>
</div>
```

CSS：

```
.wrapper {
  高度: 600px;
  文本对齐: 居中;
}
.outer {
  显示: table;
  高度: 100%;
  宽度: 100%;
}
.outer .inner {
  显示: table-cell;
  文本对齐: 居中;
  垂直对齐: 居中;
}
```

第50.2节：使用Flexbox进行居中

HTML：

```
<div class="container">
    <div class="child"></div>
</div>
```

CSS：

```
.container {
  高度: 500px;
  width: 500px;
  display: flex;              // 使用Flexbox
  align-items: center;        // 这使子元素在父容器中垂直居中。
  justify-content: center;    // 这使子元素水平居中。
  背景: 白色;
}

.child {
  width: 100px;
  height: 100px;
  background: blue;
}
```


Chapter 50: Vertical Centering

Section 50.1: Centering with display: table

HTML:

```
<div class="wrapper">
    <div class="outer">
        <div class="inner">
            centered
        </div>
    </div>
</div>
```

CSS:

```
.wrapper {
  height: 600px;
  text-align: center;
}
.outer {
  display: table;
  height: 100%;
  width: 100%;
}
.outer .inner {
  display: table-cell;
  text-align: center;
  vertical-align: middle;
}
```

Section 50.2: Centering with Flexbox

HTML:

```
<div class="container">
    <div class="child"></div>
</div>
```

CSS:

```
.container {
  height: 500px;
  width: 500px;
  display: flex;              // Use Flexbox
  align-items: center;        // This centers children vertically in the parent.
  justify-content: center;    // This centers children horizontally.
  background: white;
}

.child {
  width: 100px;
  height: 100px;
  background: blue;
}
```

第50.3节：使用变换居中

HTML：

```
<div class="wrapper">
    <div class="centered">
       居中
    </div>
</div>
```

CSS：

```
.wrapper {
  position: relative;
  height: 600px;
}
.centered {
  position: absolute;
  z-index: 999;
  transform: translate(-50%, -50%);
  top: 50%;
  左: 50%;
}
```

第50.4节：使用行高居中文本

HTML：

```
<div class="container">
    <span>垂直居中</span>
</div>
```

CSS：

```
.container{
    height: 50px;           /* 设置高度 */
    line-height: 50px;      /* 设置行高等于高度 */
    vertical-align: middle; /* 没有这条规则也能生效，但明确设置更好 */
}
```

注意： 此方法仅能垂直居中单行文本。它不能正确居中块级元素，且如果文本换行，将会出现两行非常高的文本。

第50.5节：使用 position: absolute 居中

HTML：

```
<div class="wrapper">
  <img
       src="http://cdn.sstatic.net/Sites/stackoverflow/company/img/logos/so/so-icon.png?v=c78bd457575a">
</div>
```

CSS：

```
.wrapper{
    position:相对;
```


Section 50.3: Centering with Transform

HTML:

```
<div class="wrapper">
    <div class="centered">
        centered
    </div>
</div>
```

CSS:

```
.wrapper {
  position: relative;
  height: 600px;
}
.centered {
  position: absolute;
  z-index: 999;
  transform: translate(-50%, -50%);
  top: 50%;
  left: 50%;
}
```

Section 50.4: Centering Text with Line Height

HTML:

```
<div class="container">
    <span>vertically centered</span>
</div>
```

CSS:

```
.container{
    height: 50px;           /* set height */
    line-height: 50px;      /* set line-height equal to the height */
    vertical-align: middle; /* works without this rule, but it is good having it explicitly set */
}
```

Note: This method will only vertically center a *single line of text*. It will not center block elements correctly and if the text breaks onto a new line, you will have two very tall lines of text.

Section 50.5: Centering with Position: absolute

HTML:

```
<div class="wrapper">
  <img
src="http://cdn.sstatic.net/Sites/stackoverflow/company/img/logos/so/so-icon.png?v=c78bd457575a">
</div>
```

CSS:

```
.wrapper{
   position:relative;
```

```
    height: 600像素;
}
.wrapper img {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  margin: 自动;
}
```

如果你想让除图片以外的其他元素居中，则必须给该元素设置高度和宽度。

HTML：

```
<div class="wrapper">
  <div class="child">
    让我居中
  </div>
</div>
```

CSS：

```
.wrapper{
    position:相对;
    height: 600像素;
}
.wrapper .child {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  margin: auto;
  width: 200px;
  height: 30px;
  border: 1px solid #f00;
}
```

第50.6节：使用伪元素居中

HTML：

```
<div class="wrapper">
  <div class="content"></div>
</div>
```

CSS：

```
.wrapper{
    最小高度: 600px;
}

.wrapper:before{
  内容: "";
  显示: inline-block;
  高度: 100%;
  垂直对齐: 居中;
}
```



```
   height: 600px;
}
.wrapper img {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  margin: auto;
}
```

If you want to center other then images, then you must give height and width to that element.

HTML:

```
<div class="wrapper">
  <div class="child">
    make me center
  </div>
</div>
```

CSS:

```
.wrapper{
   position:relative;
   height: 600px;
}
.wrapper .child {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  margin: auto;
  width: 200px;
  height: 30px;
  border: 1px solid #f00;
}
```

Section 50.6: Centering with pseudo element

HTML:

```
<div class="wrapper">
  <div class="content"></div>
</div>
```

CSS:

```
.wrapper{
   min-height: 600px;
}

.wrapper:before{
  content: "";
  display: inline-block;
  height: 100%;
  vertical-align: middle;
}
```

```css
.内容 {
  显示: inline-block;
  高度: 80px;
  垂直对齐: 居中;
}
```

此方法最适用于在.wrapper内居中放置高度不一的.content的情况；并且当.content的高度超过.wrapper的最小高度时，您希望.wrapper的高度随之扩展。


```css
.content {
  display: inline-block;
  height: 80px;
  vertical-align: middle;
}
```

This method is best used in cases where you have a varied-height `.content` centered inside `.wrapper`; and you want `.wrapper`'s height to expand when `.content`'s height exceed `.wrapper`'s min-height.

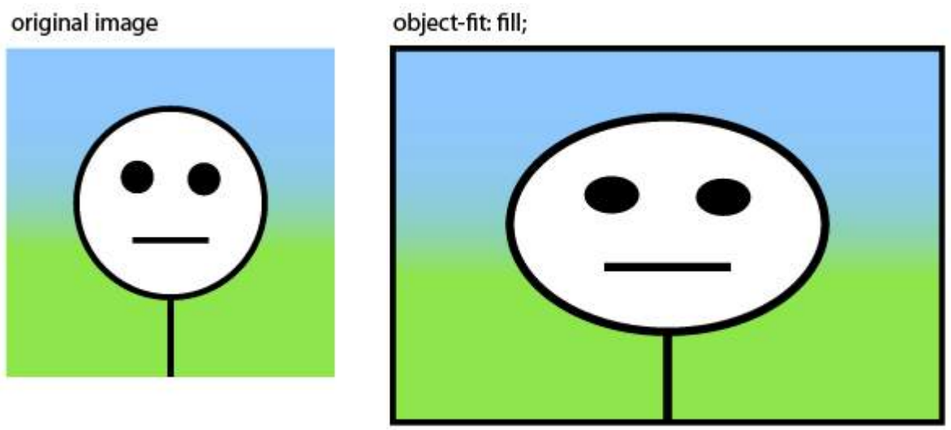
第51章：对象适应与定位

第51.1节：object-fit

object-fit 属性定义了元素如何适应具有固定高度和宽度的盒子。通常应用于图像或视频，object-fit 接受以下五个值：

填充

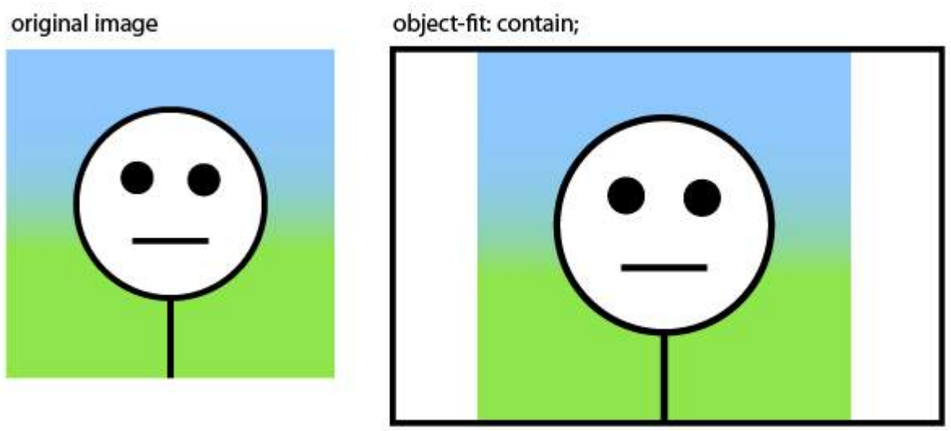
```
object-fit:fill;
```



填充会拉伸图像以适应内容框，而不考虑图像的原始宽高比。

包含

```
object-fit:contain;
```



包含会在保持图像宽高比的同时，使图像适应盒子的高度或宽度。

覆盖

```
object-fit:cover;
```

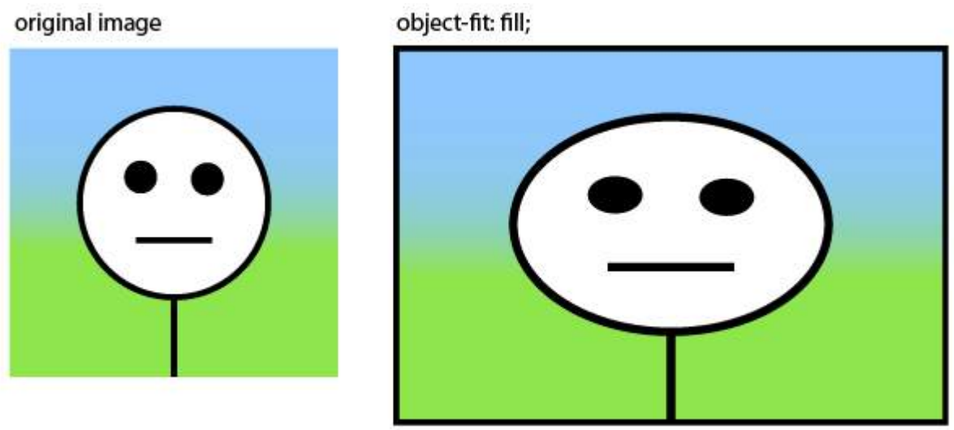

Chapter 51: Object Fit and Placement

Section 51.1: object-fit

The **object-fit** property will defines how an element will fit into a box with an established height and width. Usually applied to an image or video, Object-fit accepts the following five values:

FILL

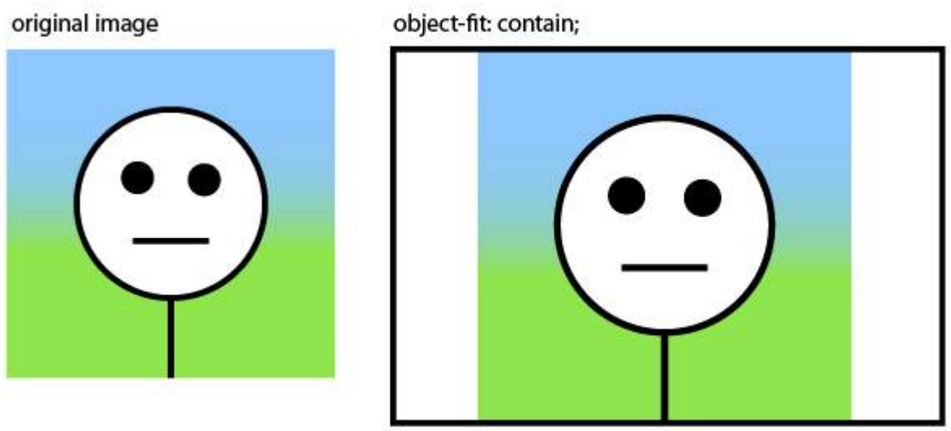
```
object-fit:fill;
```



Fill stretches the image to fit the content box without regard to the image's original aspect ratio.

CONTAIN

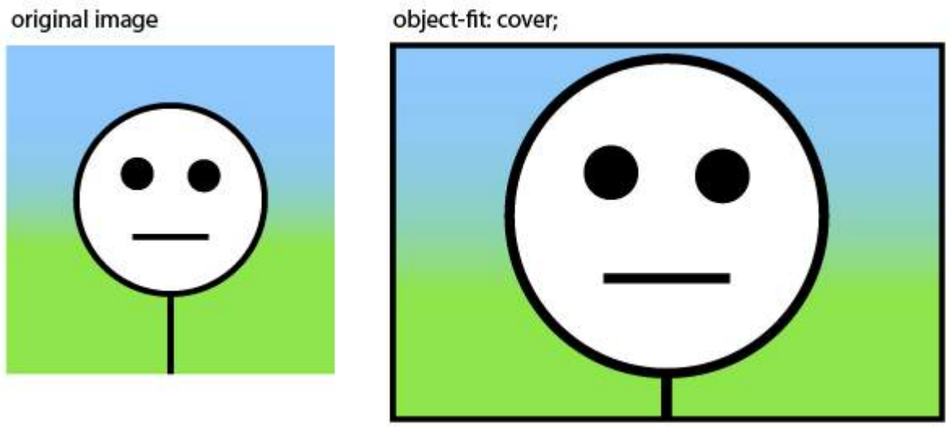
```
object-fit:contain;
```



Contain fits the image in the box's height or width while maintaining the image's aspect ratio.

COVER

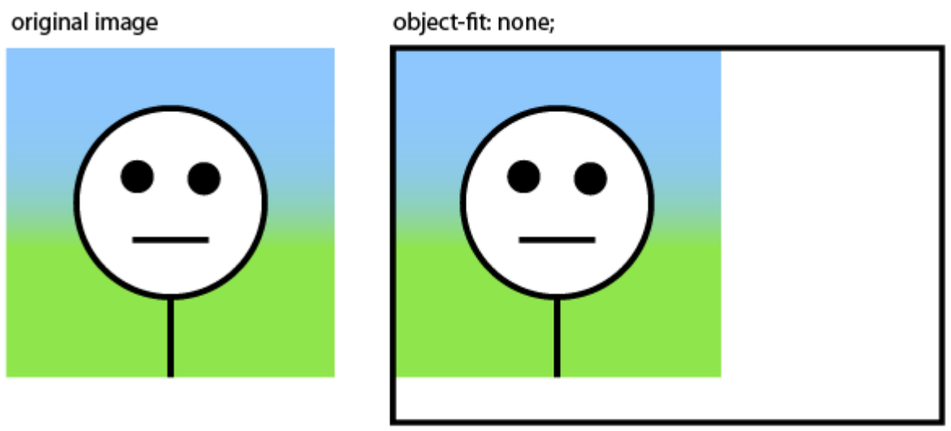
```
object-fit:cover;
```



Cover 会用图像填满整个盒子。图像的宽高比保持不变，但图像会被裁剪以适应盒子的尺寸。

无

```
object-fit:none;
```

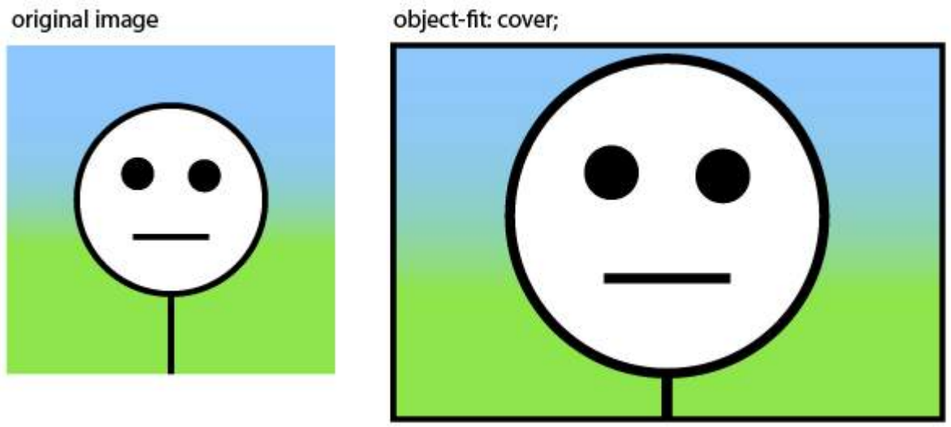


None 忽略盒子的大小，不进行缩放。

缩小

```
object-fit:scale-down;
```

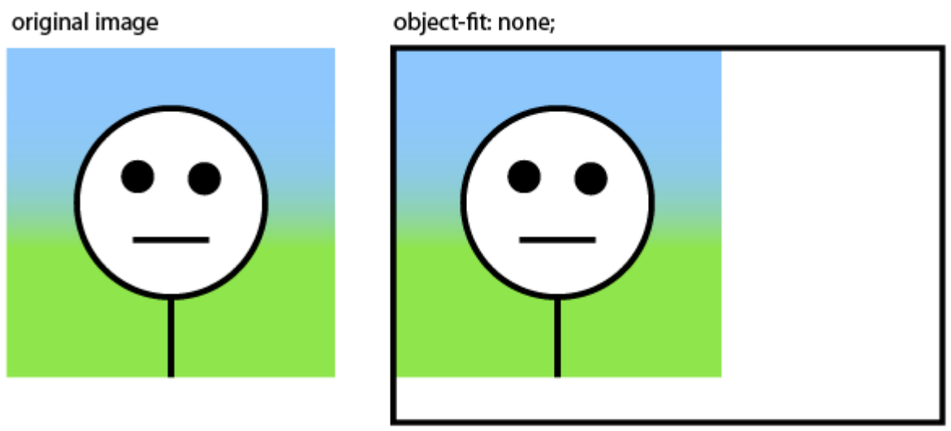
Scale-down 会将对象大小设置为none或contain。它显示导致图像尺寸更小的选项。



Cover fills the entire box with the image. The image aspect ratio is preserved, but the image is cropped to the dimensions of the box.

NONE

```
object-fit: none;
```



None ignores the size of the box and is not resized.

SCALE-DOWN

```
object-fit: scale-down;
```

Scale-down either sizes the object as `none` or as `contain`. It displays whichever option results in a smaller image size.

original image

object-fit: scale-down;

original image

object-fit: scale-down;

第52章：CSS设计模式

这些示例用于记录CSS特定的设计模式，如BEM、OOCSS和SMACSS。

这些示例不用于记录CSS框架，如Bootstrap或Foundation。

第52.1节：BEM

BEM代表块（Blocks）、元素（Elements）和修饰符（Modifiers）。这是一种最初由俄罗斯科技公司Yandex提出的方法论，但也在美国和西欧的网页开发者中获得了相当的关注。

顾名思义，BEM方法论就是将你的HTML和CSS代码组件化，分为三种类型的组件：

- 块（Blocks）：独立存在且有意义的实体

  示例有header、container、menu、checkbox和textbox

- 元素（Elements）：块的一部分，没有独立意义，语义上依附于其所属块。

  示例有menu item、list item、checkbox caption和header title

- 修饰符（Modifiers）：作用于块或元素的标记，用于改变外观或行为

  示例有disabled、highlighted、checked、fixed、size big和color yellow

BEM的目标是优化你的CSS代码的可读性、可维护性和灵活性。实现这一目标的方法是遵循以下规则。

- 块样式从不依赖页面上的其他元素块应有一个简单、简短的名称，避免使
- 用_或-字符在为元素设置样式时，使用格式为blockname__elementname的选
- 择器在为修饰符设置样式时，使用格式为blockname--modifiername和blockname__
- elementname--modifiername的选择器
- 具有修饰符的元素或块应继承其所修饰的块或元素的所有内容
  除了修饰符应修改的属性之外

代码示例

如果你将BEM应用于你的表单元素，CSS选择器应类似如下：

```
.form { }                    // 块
.form--theme-xmas { }          // 块 + 修饰符
.form--simple { }              // 块 + 修饰符
.form__input { }               // 块 > 元素
.form__submit { }              // 块 > 元素
.form__submit--disabled { }      // 块 > 元素 + 修饰符
```

对应的HTML应类似如下：

Chapter 52: CSS design patterns

These examples are for documenting CSS-specific design patterns like BEM, OOCSS and SMACSS.

These examples are NOT for documenting CSS frameworks like Bootstrap or Foundation.

Section 52.1: BEM

BEM stands for `Blocks, Elements and Modifiers`. It's a methodology initially conceived by Russian tech company Yandex, but which gained quite some traction among American & Western-European web developers as well.

As the same implies, BEM metholology is all about componentization of your HTML and CSS code into three types of components:

- **Blocks:** standalone entities that are meaningful on their own

  Examples are `header`, `container`, `menu`, `checkbox` & `textbox`

- **Elements:** Part of blocks that have no standalone meaning and are semantically tied to their blocks.

  Examples are `menu item`, `list item`, `checkbox caption` & `header title`

- **Modifiers:** Flags on a block or element, used to change appearance or behavior

  Examples are `disabled`, `highlighted`, `checked`, `fixed`, `size big` & `color yellow`

The goal of BEM is to keep optimize the readability, maintainability and flexibility of your CSS code. The way to achieve this, is to apply the following rules.

- Block styles are never dependent on other elements on a page
- Blocks should have a simple, short name and avoid _ or - characters
- When styling elements, use selectors of format `blockname__elementname`
- When styling modifiers, use selectors of format `blockname--modifiername` and `blockname__elementname--modifiername`
- Elements or blocks that have modifiers should inherit everything from the block or element it is modifying except the properties the modifier is supposed to modify

Code example

If you apply BEM to your form elements, your CSS selectors should look something like this:

```
.form { }                    // Block
.form--theme-xmas { }          // Block + modifier
.form--simple { }              // Block + modifier
.form__input { }               // Block > element
.form__submit { }              // Block > element
.form__submit--disabled { }      // Block > element + modifier
```

The corresponding HTML should look something like this:

```
<form class="form form--theme-xmas form--simple">
  <input class="form__input" type="text" />
  <input class="form__submit form__submit--disabled" type="submit" />
</form>
```



```
<form class="form form--theme-xmas form--simple">
  <input class="form__input" type="text" />
  <input class="form__submit form__submit--disabled" type="submit" />
</form>
```

第53章：浏览器支持与前缀

前缀	浏览器
-webkit-	谷歌浏览器、Safari、Opera 12及以上新版本、安卓、黑莓和UC浏览器
-moz-	Mozilla Firefox（火狐浏览器）
-ms-	Internet Explorer（互联网浏览器）、Edge（微软Edge浏览器）
-o-, -xv-	Opera 12及以前版本
-khtml-	Konquerer（Konquerer浏览器）

第53.1节：过渡

```
div {
-webkit-transition: all 4s ease;
       -moz-transition: all 4s ease;
         -o-transition: all 4s ease;
            transition: all 4s ease;
}
```

第53.2节：变换

```
div {
-webkit-transform: rotate(45deg);
       -moz-transform: rotate(45deg);
        -ms-transform: rotate(45deg);
         -o-transform: rotate(45deg);
            transform: rotate(45deg);
}
```


Chapter 53: Browser Support & Prefixes

Prefix	Browser(s)
-webkit-	Google Chrome, Safari, newer versions of Opera 12 and up, Android, Blackberry and UC browsers
-moz-	Mozilla Firefox
-ms-	Internet Explorer, Edge
-o-, -xv-	Opera until version 12
-khtml-	Konquerer

Section 53.1: Transitions

```
div {
    -webkit-transition: all 4s ease;
       -moz-transition: all 4s ease;
         -o-transition: all 4s ease;
            transition: all 4s ease;
}
```

Section 53.2: Transform

```
div {
    -webkit-transform: rotate(45deg);
       -moz-transform: rotate(45deg);
        -ms-transform: rotate(45deg);
         -o-transform: rotate(45deg);
            transform: rotate(45deg);
}
```

第54章：浏览器样式规范化

每个浏览器都有一套默认的CSS样式用于渲染元素。这些默认样式在不同浏览器之间可能不一致，原因包括：语言规范不明确，导致基础样式存在解释空间；浏览器可能不完全遵循规范；或者浏览器可能没有为较新的HTML元素设置默认样式。因此，人们可能希望尽可能在多种浏览器间规范化默认样式。

第54.1节：normalize.css

浏览器有一套默认的CSS样式用于渲染元素。其中一些样式甚至可以通过浏览器设置自定义，例如更改默认字体和字号。这些样式定义了哪些元素应为块级元素或内联元素等内容。

由于语言规范给予这些默认样式一定的自由度，且浏览器可能未能完全遵守规范，因此不同浏览器之间的默认样式可能存在差异。

这时，normalize.css就派上用场了。它覆盖了最常见的不一致问题并修复了已知的错误。

它的作用是什么

- 保留有用的默认样式，不同于许多 CSS 重置。
- 为各种元素规范化样式。
- 修正错误和常见的浏览器不一致问题。
- 通过细微的修改提升可用性。
- 通过详细注释解释代码的作用。

因此，通过在项目中包含 normalize.css，您的设计在不同浏览器中将看起来更加相似和一致。

与 reset.css 的区别

您可能听说过reset.css。两者有什么区别？

normalize.css 通过将不同属性设置为统一的默认值来提供一致性，而 reset.css 则通过移除浏览器可能应用的所有基本样式来实现一致性。虽然这听起来一开始是个好主意，但实际上这意味着您必须自己编写所有规则，这违背了拥有一个稳固标准的初衷。

第54.2节：方法与示例

CSS 重置采用了不同于浏览器默认设置的方法。Eric Meyer 的 Reset CSS 已经存在一段时间了。他的方法消除了许多已知会立即引发问题的浏览器元素。以下内容摘自他的版本（v2.0 | 20110126）CSS Reset。

```
html、body、div、span、applet、object、iframe、
h1、h2、h3、h4、h5、h6、p、blockquote、pre、
a、abbr、acronym、address、big、cite、code、
del、dfn、em、img、ins、kbd、q、s、samp、
small、strike、strong、sub、sup、tt、var、
b、u、i、center、
dl、dt、dd、ol、ul、li、
fieldset、form、label、legend、
table、caption、tbody、tfoot、thead、tr、th、td、
```


Chapter 54: Normalizing Browser Styles

Every browser has a default set of CSS styles that it uses for rendering elements. These default styles may not be consistent across browsers because: the language specifications are unclear so base styles are up for interpretation, browsers may not follow specifications that are given, or browsers may not have default styles for newer HTML elements. As a result, people may want to normalize default styles across as many browsers as possible.

Section 54.1: normalize.css

Browsers have a default set of CSS styles they use for rendering elements. Some of these styles can even be customised using the browser's settings to change default font face and size definitions, for example. The styles contain the definition of which elements are supposed to be block-level or inline, among other things.

Because these default styles are given some leeway by the language specifications and because browsers may not follow the specs properly they can differ from browser to browser.

This is where normalize.css comes into play. It overrides the most common inconsistencies and fixes known bugs.

What does it do

- Preserves useful defaults, unlike many CSS resets.
- Normalizes styles for a wide range of elements.
- Corrects bugs and common browser inconsistencies.
- Improves usability with subtle modifications.
- Explains what code does using detailed comments.

So, by including normalize.css in your project your design will look more alike and consistent across different browsers.

Difference to reset.css

You may have heard of `reset.css`. What's the difference between the two?

While normalize.css provides consistency by setting different properties to unified defaults, reset.css achieves consistency by **removing** all basic styling that a browser may apply. While this might sound like a good idea at first, this actually means you have to write **all** rules yourself, which goes against having a solid standard.

Section 54.2: Approaches and Examples

CSS resets take separate approaches to browser defaults. Eric Meyer's Reset CSS has been around for a while. His approach nullifies many of the browser elements that have been known to cause problems right off the back. The following is from his version (v2.0 | 20110126) CSS Reset.

```
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
```

```
article、aside、canvas、details、embed、
figure、figcaption、footer、header、hgroup、
menu、nav、output、ruby、section、summary、
time、mark、audio、video {
margin: 0;
padding: 0;
    border: 0;
font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}
```

Eric Meyer 的 Reset CSS

Normalize CSS 处理了其他方面，并分别处理了许多这些问题。以下是代码版本 (v4.2.0) 的示例。

```
/**
* 1. 更改所有浏览器中的默认字体系列（有主见的）。
* 2. 纠正所有浏览器中的行高。
* 3. 防止 IE 和 iOS 在方向变化后调整字体大小。
 */

/* 文档
   ========================================================================== */

html {
  font-family: 无衬线字体; /* 1 */
  line-height: 1.15; /* 2 */
-ms-text-size-adjust: 100%; /* 3 */
-webkit-text-size-adjust: 100%; /* 3 */
}

/* 区块
========================================================================== */

/**
* 在所有浏览器中移除外边距（有主见的做法）。
 */

body {
  margin: 0;
}

/**
* 在 IE 9 及更早版本中添加正确的显示方式。
 */

article,
aside,
footer,
header,
nav,
section {
  display: block;
}

/**
* 修正 Chrome、Firefox 和 Safari 中 `section` 和
 * `article` 环境下 `h1` 元素的字体大小和外边距。
 */
```



```
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}
```

Eric Meyer's Reset CSS

Normalize CSS on the other and deals with many of these separately. The following is a sample from the version (v4.2.0) of the code.

```
/**
 * 1. Change the default font family in all browsers (opinionated).
 * 2. Correct the line height in all browsers.
 * 3. Prevent adjustments of font size after orientation changes in IE and iOS.
 */

/* Document
   ========================================================================== */

html {
  font-family: sans-serif; /* 1 */
  line-height: 1.15; /* 2 */
  -ms-text-size-adjust: 100%; /* 3 */
  -webkit-text-size-adjust: 100%; /* 3 */
}

/* Sections
   ========================================================================== */

/**
 * Remove the margin in all browsers (opinionated).
 */

body {
  margin: 0;
}

/**
 * Add the correct display in IE 9-.
 */

article,
aside,
footer,
header,
nav,
section {
  display: block;
}

/**
 * Correct the font size and margin on `h1` elements within `section` and
 * `article` contexts in Chrome, Firefox, and Safari.
 */
```

```
h1 {
  font-size: 2em;
  margin: 0.67em 0;
}
```

规范化 CSS

h1 {
  font-size: 2em;
  margin: 0.67em 0;
}

Normalize CSS

第55章：Internet Explorer 技巧

第55.1节：为IE6和IE7添加inline-block支持

```
display: inline-block;
```

Internet Explorer 6和7不支持值为inline-block的display属性。一个解决方法是：

```
zoom: 1;
*display: inline;
```

zoom属性触发元素的hasLayout特性，该特性仅在Internet Explorer中可用。*display确保无效属性只在受影响的浏览器上执行，其他浏览器将忽略该规则。

第55.2节：Internet Explorer 10及以上版本的高对比度模式

在Internet Explorer 10及以上版本和Edge中，微软提供了-ms-high-contrast媒体选择器，用于暴露浏览器的"高对比度"设置，允许程序员相应地调整网站样式。

-ms-high-contrast选择器有3种状态：active、black-on-white和white-on-black。在IE10及以上版本中，它还有一个none状态，但该状态在Edge中已不再支持。

示例

```
@media screen and (-ms-high-contrast: active), (-ms-high-contrast: black-on-white) {
   .header{
      背景: #fff;
      颜色: #000;
   }
}
```

当高对比度模式激活且处于黑底白字模式时，这将把页眉背景改为白色，文字颜色改为黑色。

```
@media screen and (-ms-high-contrast: white-on-black) {
   .header{
      background: #000;
      color: #fff;
   }
}
```

与第一个示例类似，但此处专门选择白底黑字状态，并将页眉颜色反转为黑色背景白色文字。

更多信息：

Microsoft 文档 关于 -ms-high-contrast

Chapter 55: Internet Explorer Hacks

Section 55.1: Adding Inline Block support to IE6 and IE7

```
display: inline-block;
```

The display property with the value of inline-block is not supported by Internet Explorer 6 and 7. A work-around for this is:

```
zoom: 1;
*display: inline;
```

The zoom property triggers the hasLayout feature of elements, and it is available only in Internet Explorer. The *display makes sure that the invalid property executes only on the affected browsers. Other browsers will simply ignore the rule.

Section 55.2: High Contrast Mode in Internet Explorer 10 and greater

In Internet Explorer 10+ and Edge, Microsoft provides the -ms-high-contrast media selector to expose the "High Contrast" setting from the browser, which allows the programmer to adjust their site's styles accordingly.

The -ms-high-contrast selector has 3 states: active, black-on-white, and white-on-black. In IE10+ it also had a none state, but that is no longer supported in Edge going forward.

Examples

```
@media screen and (-ms-high-contrast: active), (-ms-high-contrast: black-on-white) {
   .header{
      background: #fff;
      color: #000;
   }
}
```

This will change the header background to white and the text color to black when high contrast mode is active *and* it is in black-on-white mode.

```
@media screen and (-ms-high-contrast: white-on-black) {
   .header{
      background: #000;
      color: #fff;
   }
}
```

Similar to the first example, but this specifically selects the white-on-black state only, and inverts the header colors to a black background with white text.

More Information:

Microsoft Documentation on -ms-high-contrast

第55.3节：仅适用于 Internet Explorer 6 和 Internet Explorer 7

要针对 Internet Explorer 6 和 Internet Explorer 7，请以*开头编写属性：

```
.hide-on-ie6-and-ie7 {
    *display：none; // 该行仅在 IE6 和 IE7 上处理
}
```

非字母数字前缀（除连字符和下划线外）在IE6和IE7中被忽略，因此此技巧适用于任何无前缀的属性：值对。

第55.4节：仅限Internet Explorer 8

要针对Internet Explorer 8，请将选择器包裹在@media \0 screen { }内：

```
@media \0 screen {
    .hide-on-ie8 {
        显示：无;
    }
}
```

所有位于@media \0 screen { }之间的内容仅由I处理

Section 55.3: Internet Explorer 6 & Internet Explorer 7 only

To target Internet Explorer 6 and Internet Explorer 7, start your properties with *:

```
.hide-on-ie6-and-ie7 {
    *display : none; // This line is processed only on IE6 and IE7
}
```

Non-alphanumeric prefixes (other than hyphens and underscores) are ignored in IE6 and IE7, so this hack works for any unprefixed `property: value` pair.

Section 55.4: Internet Explorer 8 only

To target Internet Explorer 8, wrap your selectors inside `@media \0 screen { }`:

```
@media \0 screen {
    .hide-on-ie8 {
        display : none;
    }
}
```

Everything between `@media \0 screen { }` is processed only by I

第56章：性能

第56.1节：使用transform和opacity避免触发布局计算

更改某些CSS属性会触发浏览器同步计算样式和布局，这在需要以60fps动画时是件坏事。

不要

使用left和top动画会触发布局计算。

```
#box {
  left: 0;
  top: 0;
  transition: left 0.5秒, top 0.5秒;
  position: absolute;
  width: 50像素;
  height: 50像素;
  background-color: 灰色;
}

#box.active {
  left: 100像素;
  top: 100像素;
}
```

演示耗时11.7毫秒进行渲染，9.8毫秒进行绘制

Chapter 56: Performance

Section 56.1: Use transform and opacity to avoid trigger layout

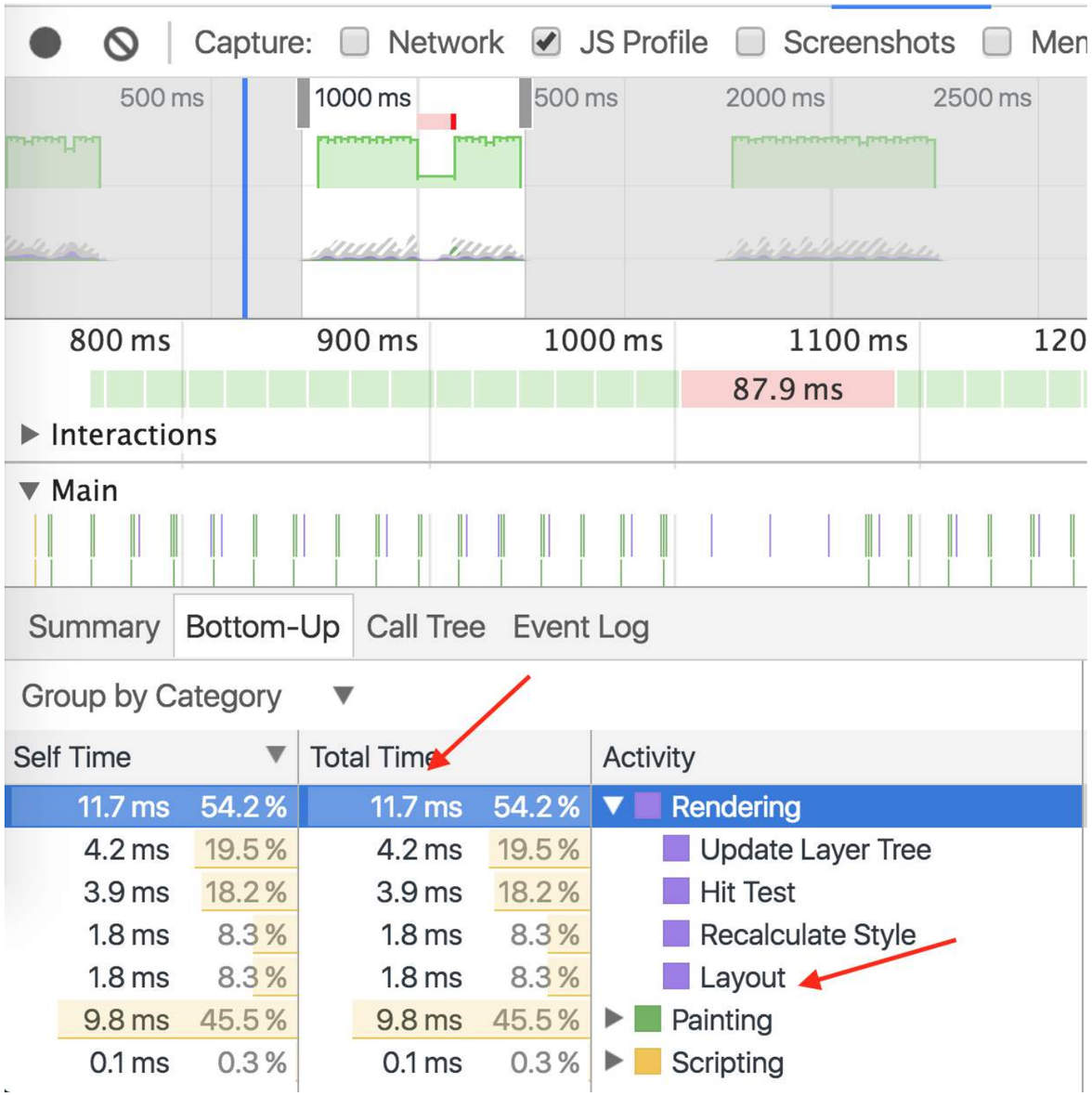
Changing some CSS attribute will trigger the browser to synchronously calculate the style and layout, which is a bad thing when you need to animate at 60fps.

DON'T

Animate with `left` and `top` trigger layout.

```
#box {
  left: 0;
  top: 0;
  transition: left 0.5s, top 0.5s;
  position: absolute;
  width: 50px;
  height: 50px;
  background-color: gray;
}

#box.active {
  left: 100px;
  top: 100px;
}
```

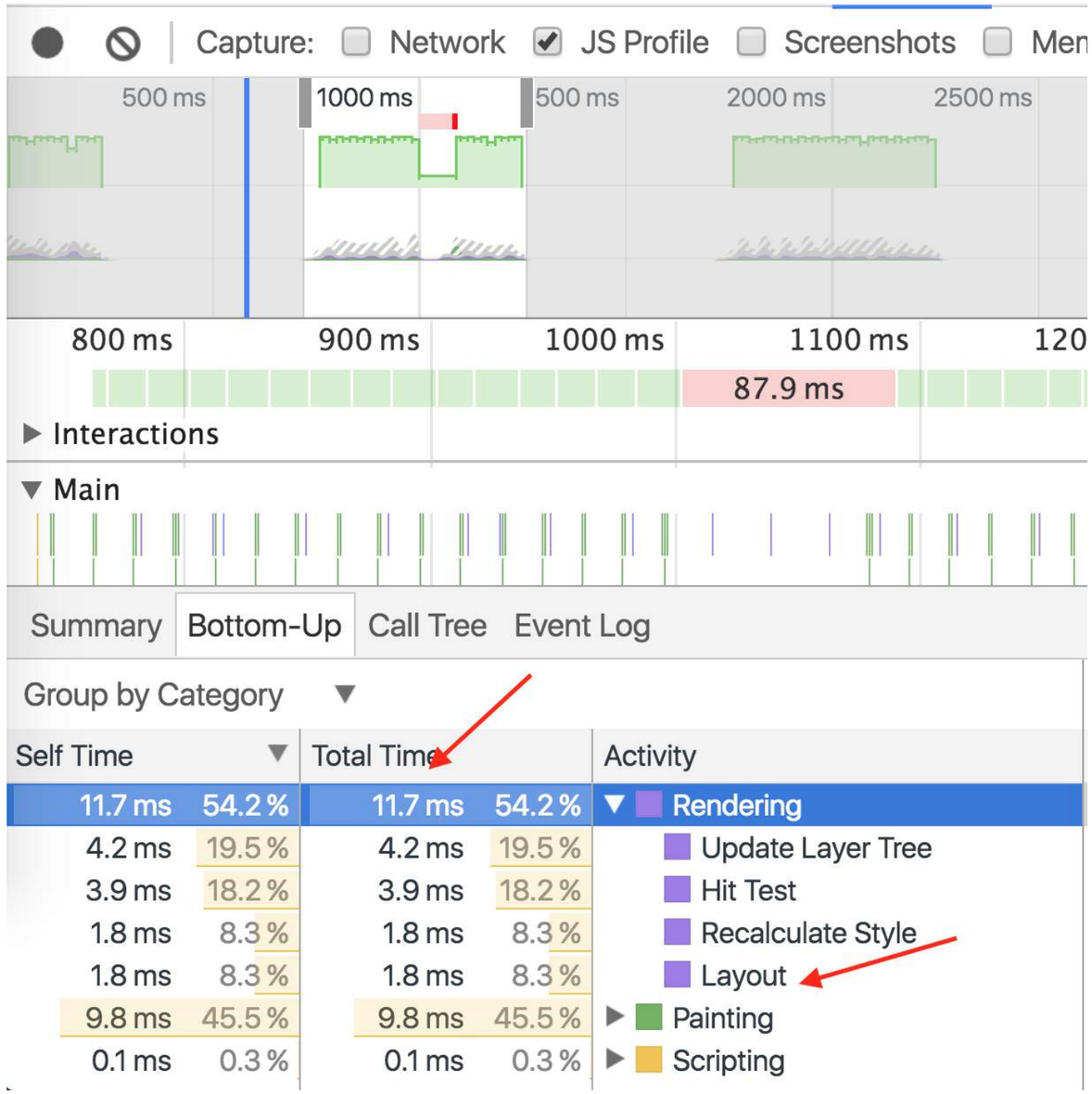
Demo took **11.7ms** for rendering, **9.8ms** for painting



DO

使用transform进行相同的动画。

```
#box {
  left: 0;
  top: 0;
  position: absolute;
  width: 50像素;
  height: 50px;
  background-color: 灰色;

  transition: transform 0.5秒;
  transform: translate3d(0, 0, 0);
}

#box.active {
```

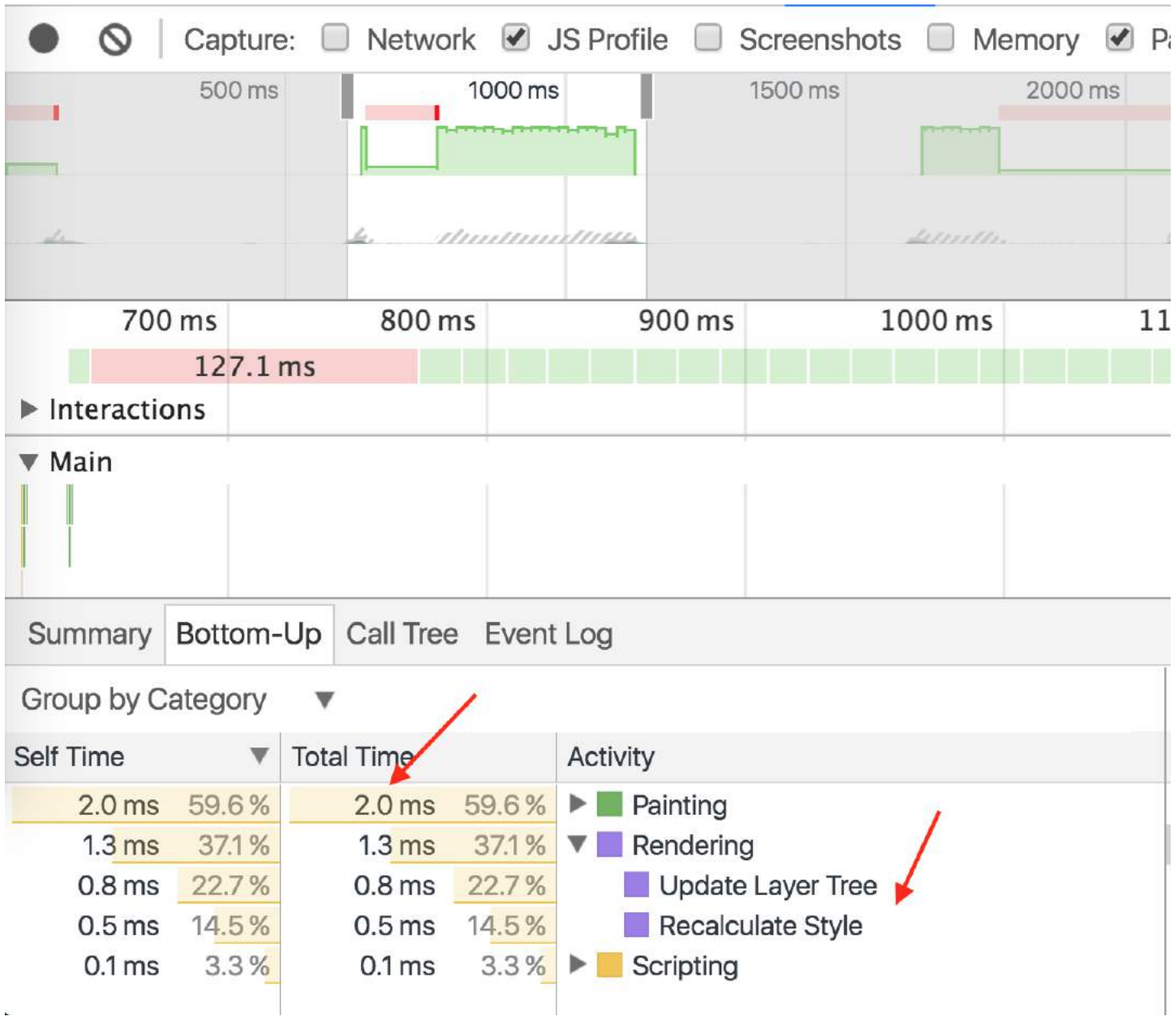
DO

Animate with `transform` with the same animation.

```
#box {
  left: 0;
  top: 0;
  position: absolute;
  width: 50px;
  height: 50px;
  background-color: gray;

  transition: transform 0.5s;
  transform: translate3d(0, 0, 0);
}

#box.active {
```

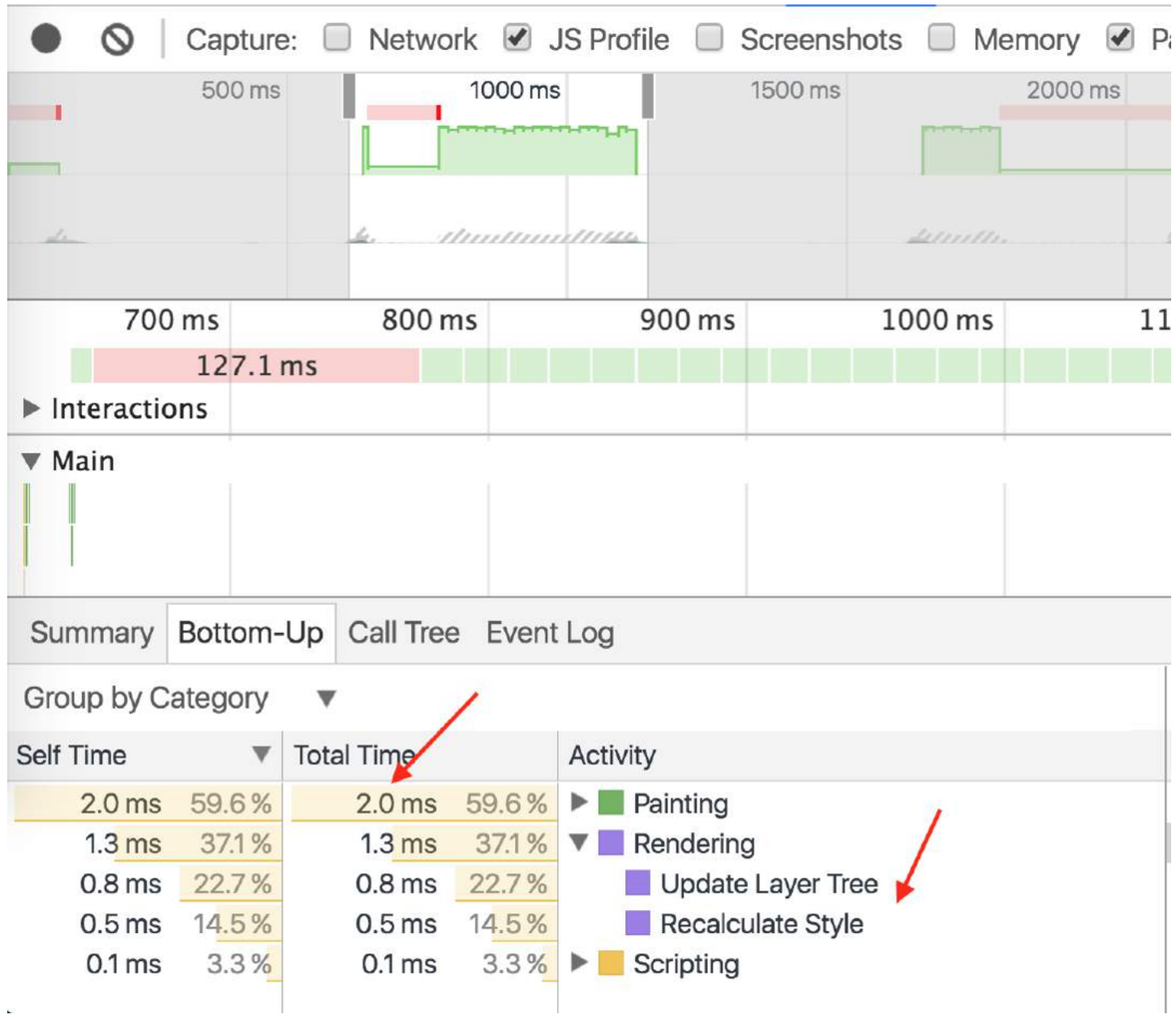
```
  transform: translate3d(100px, 100px, 0);
}
```

演示相同动画，渲染耗时1.3毫秒，绘制耗时2.0毫秒。




```
  transform: translate3d(100px, 100px, 0);
}
```

Demo same animation, took **1.3ms** for rendering, **2.0ms** for painting.



鸣谢

非常感谢所有来自Stack Overflow Documentation的人员帮助提供此内容，
更多更改可发送至web@petercv.com以发布或更新新内容

A B	第20章
A.J	第4章
亚伦	第4章
abaracedo	第4章
阿比谢克·辛格	第22章
adamboro	第1章
Aeolingamenfel	第27章和第55章
艾哈迈德·阿尔菲	第4章、第5章和第16章
Alohci	第15章
amflare	第13章和第17章
安德烈·洛佩斯	第44章
安德烈·麦格鲁德	第54章
安德烈亚斯	第15章和第38章
安德鲁	第12章、第19章和第53章
安德鲁·迈尔斯	第47章
阿尼尔	第4章
animuson	第4章、第50章和第53章
apaul	第6章和第27章
Araknid	第4章
阿里夫	第11章
阿尔扬·恩布	第4、7、8、15和17章
阿什温·拉马斯瓦米	第1和4章
阿西姆·K·T	第5和16章
AVAVT	第50章
awe	第1章
bdkopen	第3章
本·里斯	第5章
比蓬	第40章
饼干烘焙师	第7章
鲍里斯	第5章
博伊森莓	第1章
布兰达蒙	第17章
布雷特·德伍迪	第18章、第38章和第39章
CalvT	第5章和第9章
凯西	第11章
卡西迪·威廉姆斯	第10章和第22章
cdm	第5章和第8章
查理·H	第4章和第28章
查图兰加·贾亚纳特	第11章、第13章和第23章
冷水机	第38章
克里斯	第1章、第4章、第23章、第25章、第42章和第50章
克里斯·斯皮特尔斯	第8章和第24章
克里斯蒂安·马克斯	第28章
可可豆	第5章
coderfin	第3章
cone56	第31章和第36章
CPHPython	第4章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,
more changes can be sent to web@petercv.com for new content to be published or updated

A B	Chapter 20
A.J	Chapter 4
Aaron	Chapter 4
abaracedo	Chapter 4
Abhishek Singh	Chapter 22
adamboro	Chapter 1
Aeolingamenfel	Chapters 27 and 55
Ahmad Alfy	Chapters 4, 5 and 16
Alohci	Chapter 15
amflare	Chapters 13 and 17
Andre Lopes	Chapter 44
andre mcgruder	Chapter 54
andreas	Chapters 15 and 38
Andrew	Chapters 12, 19 and 53
Andrew Myers	Chapter 47
Anil	Chapter 4
animuson	Chapters 4, 50 and 53
apaul	Chapters 6 and 27
Araknid	Chapter 4
Arif	Chapter 11
Arjan Einbu	Chapters 4, 7, 8, 15 and 17
Ashwin Ramaswami	Chapters 1 and 4
Asim K T	Chapters 5 and 16
AVAVT	Chapter 50
awe	Chapter 1
bdkopen	Chapter 3
Ben Rhys	Chapter 5
Bipon	Chapter 40
BiscuitBaker	Chapter 7
Boris	Chapter 5
Boysenb3rry	Chapter 1
brandaemon	Chapter 17
Brett DeWoody	Chapters 18, 38 and 39
CalvT	Chapters 5 and 9
Casey	Chapter 11
Cassidy Williams	Chapters 10 and 22
cdm	Chapters 5 and 8
Charlie H	Chapters 4 and 28
Chathuranga Jayanath	Chapters 11, 13 and 23
Chiller	Chapter 38
Chris	Chapters 1, 4, 23, 25, 42 and 50
Chris Spittles	Chapters 8 and 24
Christiaan Maks	Chapter 28
CocoaBean	Chapter 5
coderfin	Chapter 3
cone56	Chapters 31 and 36
CPHPython	Chapter 4

csx.cc	第1章
cuervoo	第18章
丹尼尔·G·布拉斯克斯	第5章
丹尼尔·凯弗	第6章
丹尼尔·斯特拉多斯基	第5章
DarkAjax	第17章
darrylyeo	第2章、第13章和第18章
Darthstroke	第5章
戴夫·埃弗里特	第4章
大卫·富勒顿	第4章
德米特尔·迪米特里	第4章
demonofthemist	第14章
designcise	第4、5和18章
德维德·法里内利	第4和6章
德文·伯纳德	第4章
德克斯·斯塔尔	第27章
迭戈·V	第6章
迪尼杜·赫瓦格	第4章
dippas	第4章、第17章和第21章
doctorsherlock	第10章
dodopok	第13、36和45章
Elegant.Scripting	第43章
埃利兰·马尔卡	第6章
埃马努埃莱·帕里西奥	第6章
叶夫根尼	第15章
法尔扎德·YZ	第6章
fcalderan	第5章
feeela	第46章和第55章
费利佩·阿尔斯	第1、5、10、11、14、16、24和25章
费利克斯·A·J	第15章
费利克斯·埃德尔曼	第4章
费利克斯·舒茨	第4章
四十	第4章
fracz	第4章
fzzylogic	第16章
G	第1章和第17章
加布里埃尔·R.	第1章
gandreadis	第4章
geek1011	第21章
geeksal	第17章
杰拉达斯	第1章
格尼茨肖	第10章
GoatsWearHats	第1章
Gofilord	第21章
格兰特·帕林	第54章
H. 保维林	第4、18和36章
汉斯·Cz	第4章
哈里什·吉亚纳尼	第1章
哈里	第10、26、28、29、33、35和44章
亨利	第4章
霍斯特·雅恩斯	第5章
赫里斯托	第32章
雨果·巴夫	第4章

csx.cc	Chapter 1
cuervoo	Chapter 18
Daniel G. Blázquez	Chapter 5
Daniel Käfer	Chapter 6
Daniel Stradowski	Chapter 5
DarkAjax	Chapter 17
darrylyeo	Chapters 2, 13 and 18
Darthstroke	Chapter 5
Dave Everitt	Chapter 4
David Fullerton	Chapter 4
Demeter Dimitri	Chapter 4
demonofthemist	Chapter 14
designcise	Chapters 4, 5 and 18
Devid Farinelli	Chapters 4 and 6
Devon Bernard	Chapter 4
Dex Star	Chapter 27
Diego V	Chapter 6
Dinidu Hewage	Chapter 4
dippas	Chapters 4, 17 and 21
doctorsherlock	Chapter 10
dodopok	Chapters 13, 36 and 45
Elegant.Scripting	Chapter 43
Eliran Malka	Chapter 6
Emanuele Parisio	Chapter 6
Evgeny	Chapter 15
Farzad YZ	Chapter 6
fcalderan	Chapter 5
feeela	Chapters 46 and 55
FelipeAls	Chapters 1, 5, 10, 11, 14, 16, 24 and 25
Felix A J	Chapter 15
Felix Edelmann	Chapter 4
Felix Schütz	Chapter 4
Forty	Chapter 4
fracz	Chapter 4
fzzylogic	Chapter 16
G	Chapters 1 and 17
Gabriel R.	Chapter 1
gandreadis	Chapter 4
geek1011	Chapter 21
geeksal	Chapter 17
Gerardas	Chapter 1
Gnietschow	Chapter 10
GoatsWearHats	Chapter 1
Gofilord	Chapter 21
Grant Palin	Chapter 54
H. Pauwelyn	Chapters 4, 18 and 36
HansCz	Chapter 4
Harish Gyanani	Chapter 1
Harry	Chapters 10, 26, 28, 29, 33, 35 and 44
henry	Chapter 4
Horst Jahns	Chapter 5
Hristo	Chapter 32
Hugo Buff	Chapter 4

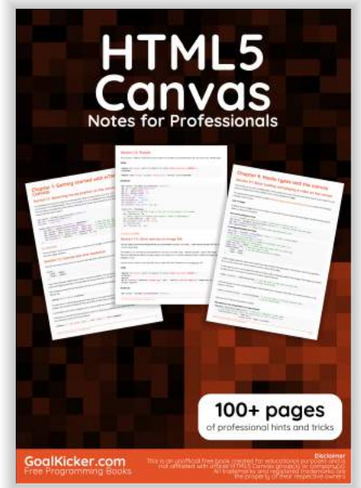
海恩斯	第4、5和15章
insertusernamehere	第15章
J·阿特金	第1和4章
J·F	第4和20章
雅各布·格雷	第4、5和22章
詹姆斯·唐纳利	第7和17章
詹姆斯·泰勒	第5章
jaredsk	第10、36和50章
JedaiCoder	第6章
Jef	第16章
杰弗里·唐	第30章
jehna1	第6章
jgh	第12章
JHS	第25章
Jmh2013	第13、23和43章
joejoe31b	第4和13章
JoelBonetR	第4章
joe_young	第1章和第15章
约翰·斯莱格斯	第4、5、6、13、17、18、28、52和55章
陈琼	第5章和第15章
乔纳森·阿根蒂耶罗	第6章
乔纳森·林	第1、6、7、16和22章
乔纳森·祖尼加	第5章
何塞·戈麦斯	第1章
只是一个学生	第1章
凯文·卡茨克	第23章
kingcobra1986	第17章
库汉	第18章
凯尔·拉特利夫	第6章
leo_ap	第50章
LiLacTac	第55章
卢卡·克尔	第29章
卢克·泰勒	第28章
马达琳娜·泰纳	第4、5、6、8、9、10、11、12、14、15、19、25、29、31、32、34、39、45和49章
马克	第20章
马克建筑	第21章
马乔里·皮卡德	第2章
马克·佩雷拉	第4章
马滕·科茨尔	第34章和第41章
马塔斯·瓦伊特凯维休斯	第4和13章
马蒂亚·阿斯托里诺	第22章
马克西米利安·劳迈斯特	第5章和第13章
马克苏赫尔	第6章
迈克尔·莫里亚蒂	第5章、第15章和第18章
Michael_B	第4和6章
Mifeet	第6章
迈克·麦考恩	第24章
迈尔斯	第12章和第51章
米罗	第18章
MMachinegun	第54章
mmativ	第50章
Mod代理	第6章
外星先生	第5章

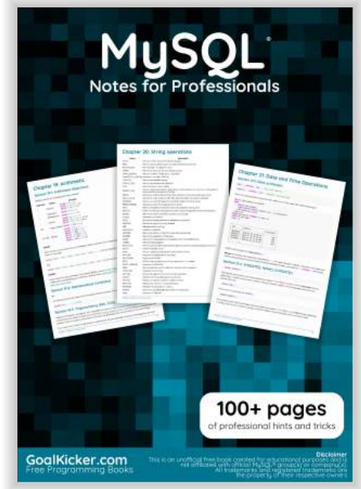
Hynes	Chapters 4, 5 and 15
insertusernamehere	Chapter 15
J Atkin	Chapters 1 and 4
J F	Chapters 4 and 20
Jacob Gray	Chapters 4, 5 and 22
James Donnelly	Chapters 7 and 17
James Taylor	Chapter 5
jaredsk	Chapters 10, 36 and 50
JedaiCoder	Chapter 6
Jef	Chapter 16
Jeffery Tang	Chapter 30
jehna1	Chapter 6
jgh	Chapter 12
JHS	Chapter 25
Jmh2013	Chapters 13, 23 and 43
joejoe31b	Chapters 4 and 13
JoelBonetR	Chapter 4
joe_young	Chapters 1 and 15
John Slegers	Chapters 4, 5, 6, 13, 17, 18, 28, 52 and 55
Jon Chan	Chapters 5 and 15
Jonathan Argentiero	Chapter 6
Jonathan Lam	Chapters 1, 6, 7, 16 and 22
Jonathan Zúñiga	Chapter 5
Jose Gomez	Chapter 1
Just a student	Chapter 1
Kevin Katzke	Chapter 23
kingcobra1986	Chapter 17
Kuhan	Chapter 18
Kyle Ratliff	Chapter 6
leo_ap	Chapter 50
LiLacTac	Chapter 55
Luka Kerr	Chapter 29
Luke Taylor	Chapter 28
Madalina Taina	Chapters 4, 5, 6, 8, 9, 10, 11, 12, 14, 15, 19, 25, 29, 31, 32, 34, 39, 45 and 49
Marc	Chapter 20
Marcatectura	Chapter 21
Marjorie Pickard	Chapter 2
Mark Perera	Chapter 4
Marten Koetsier	Chapters 34 and 41
Matas Vaitkevicius	Chapters 4 and 13
Mattia Astorino	Chapter 22
Maximillian Laumeister	Chapters 5 and 13
Maxouhell	Chapter 6
Michael Moriarty	Chapters 5, 15 and 18
Michael_B	Chapters 4 and 6
Mifeet	Chapter 6
Mike McCaughan	Chapter 24
Miles	Chapters 12 and 51
Miro	Chapter 18
MMachinegun	Chapter 54
mmativ	Chapter 50
Mod Proxy	Chapter 6
Mr. Alien	Chapter 5

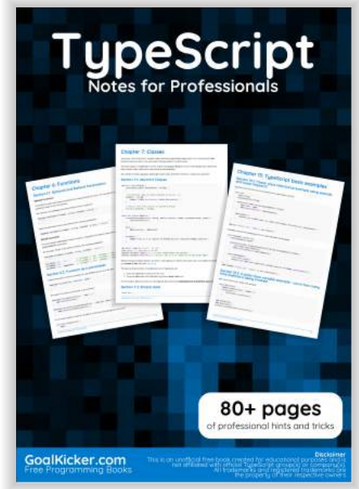
takeradi	第16章
泰勒	第6章
特德·戈阿斯	第12、15、34和43章
特奥·德拉戈维奇	第1和13章
ThatWeirdo	第4章
TheGenie OfTruth	第36章
西奥多·K.	第22章
think123	第5章
蒂莫西·米勒	第55章
托比	第15章和第20章
托德	第1章
托尼B	第15章
托特·扎姆	第8章
特雷弗·克拉克	第5、8、10和15章
TrungDQ	第56章
TylerH	第1章、第4章、第5章、第36章和第53章
乌尔里希·施瓦茨	第43章
user007	第18章
user2622348	第20章
vishak	第14章
vkopio	第7章
Vlusion	第15章
沃尔克·E.	第15章
web	第6、26、28、29和44章
威尔·迪弗鲁西奥	第9章
沃尔夫冈	第18章
X	第18章
李新阳	第1章
xpy	第4章
尤里·费多罗夫	第4章
扎克	第5章和第12章
扎菲	第4章
扎卡里亚·阿查尔基	第20章
Zaz	第4章
Ze Rubeus	第4章
zeel	第6章
zer00ne	第20章
Zeta	第5章
Zze	第5章

takeradi	Chapter 16
Taylor	Chapter 6
Ted Goas	Chapters 12, 15, 34 and 43
Teo Dragovic	Chapters 1 and 13
ThatWeirdo	Chapter 4
TheGenie OfTruth	Chapter 36
Theodore K.	Chapter 22
think123	Chapter 5
Timothy Miller	Chapter 55
Toby	Chapters 15 and 20
Todd	Chapter 1
ToniB	Chapter 15
Tot Zam	Chapter 8
Trevor Clarke	Chapters 5, 8, 10 and 15
TrungDQ	Chapter 56
TylerH	Chapters 1, 4, 5, 36 and 53
Ulrich Schwarz	Chapter 43
user007	Chapter 18
user2622348	Chapter 20
vishak	Chapter 14
vkopio	Chapter 7
Vlusion	Chapter 15
Volker E.	Chapter 15
web	Chapters 6, 26, 28, 29 and 44
Will DiFruscio	Chapter 9
Wolfgang	Chapter 18
X	Chapter 18
Xinyang Li	Chapter 1
xpy	Chapter 4
Yury Fedorov	Chapter 4
Zac	Chapters 5 and 12
Zaffy	Chapter 4
Zakaria Acharki	Chapter 20
Zaz	Chapter 4
Ze Rubeus	Chapter 4
zeel	Chapter 6
zer00ne	Chapter 20
Zeta	Chapter 5
Zze	Chapter 5

你可能也喜欢







You may also like