# CSS简介

层叠样式表 (Cascading Style Sheets，缩写为 CSS），是一种 样式表 语言，用来描述 HTML 或 XML（包括如 SVG、MathML、XHTML 之类的 XML 分支语言）文档的呈现。CSS 描述了在屏幕、纸质、音频等其它媒体上的元素应该如何被渲染的问题。

CSS 是开放网络的核心语言之一，由 W3C 规范 实现跨浏览器的标准化。CSS节省了大量的工作。 样式可以通过定义保存在外部.css文件中，同时控制多个网页的布局，这意味着开发者不必经历在所有网页上编辑布局的麻烦。CSS 被分为不同等级：CSS1 现已废弃， CSS2.1 是推荐标准， CSS3 分成多个小模块且正在标准化中。

# CSS基础指南

## CSS是怎么工作的

### 什么是CSS

正如我们之前提到的，CSS是一种用于向用户指定文档如何呈现的语言 — 它们如何被指定样式、布局等。

文档通常是用标记语言结构化的文本文件 — HTML 是最常用的标记语言, 但你依然可以遇见一些其他的标记语言，比如 SVG 或者 XML。

呈现文档给用户意味着将其转换为用户可用的形式。浏览器，比如 Firefox, Chrome 或者 Internet Explorer，被设计用于可视化呈现文档，例如，在计算机屏幕，投影仪或打印机上。

### CSS如何影响HTML

Web浏览器将CSS规则应用于文档以影响它们的显示方式。一个CSS规则由以下组成：

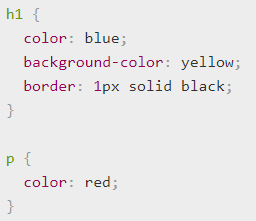
一组 属性：属性的值更新了 HTML 的内容的显示方式。比如，我想让元素的宽度是其父元素的50％，或者元素背景变为红色。

一个 选择器：它选择元素，这（些）元素是你想应用这些最新的属性值于其上的元素。比如，我想将我的CSS规则应用到我HTML文档中的所有段落上。

首先，我们使用一个简单的HTML文档，包含一个 <h1> 元素和一个 <p> 元素（(注意到通过使用一个 <link> 元素将样式表应用到了 HTML 上）：



现在让我们来看一个包含了两个规则的非常简单的CSS例子：



第一条规则从 h1 选择器开始，这意味着它将其属性值应用到 <h1> 元素上，它包含三个属性和属性各自的值（每个 属性/值 对称为一个声明）：

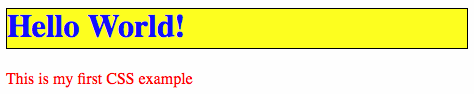
第一个声明将文本颜色设置为蓝色；

第二个声明将背景颜色设置为黄色；

第三个声明将标题（h1是标题元素）边框（border）设置为：1像素宽、实线（不是虚线、点线等）、颜色黑色。

第二个规则从 p 选择器开始，这意味着它将其属性值应用到 <p> 元素上。它包含一条声明，该声明设置字体颜色为红色。

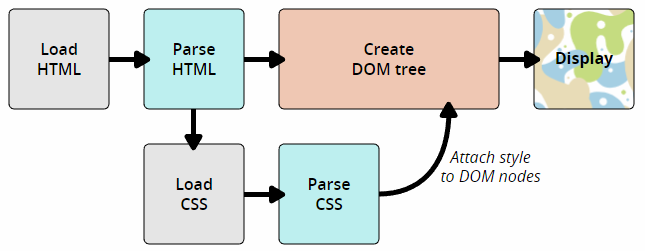
在Web浏览器中，上面的代码将产生以下输出：



### CSS 实际上如何工作

当浏览器显示文档时，它必须将文档的内容与其样式信息结合。它分两个阶段处理文档：

1. 浏览器将 HTML 和 CSS 转化成 DOM （文档对象模型）。DOM在计算机内存中表示文档。它把文档内容和其样式结合在一起。
2. 浏览器显示 DOM 的内容。



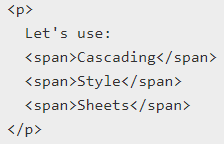
**关于 DOM**

DOM是一种树形结构. 标记语言中的每个元素,属性,文本片段都变为一个 DOM 节点。这些节点由它们与其它 DOM 节点的关系来定义。有的元素是某些子节点的父节点，且这些子节点有兄弟（节点）。

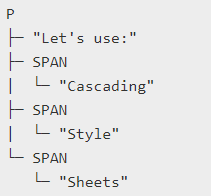
由于 DOM 是 CSS 与文档内容的相遇之处，理解 DOM 有助于设计，调试和维护你的 CSS 文件。

表示DOM：

让我们假定下面这段HTML代码：



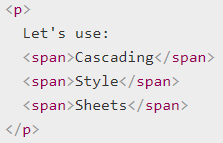
在该 DOM 中，我们的 <p> 元素所对应的节点是父节点。它的子节点是一个文本节点和我们的一些 <span> 元素对应的节点。这些 SPAN 结点也是父节点，它们各自的文本节点就是它们的子节点：



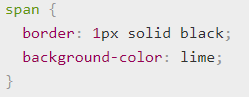
这就是浏览器解释先前的HTML片段的过程 —它渲染上述的DOM树，之后在浏览器中像这样输出它。

**应用 CSS 到 DOM**

假设我们要添加一些 CSS 到我们的文档上，使其更具风格。再一次，HTML 如下所示：



如果我们将下面 CSS 应用到它上：



浏览器会解析 HTML 并通过它创建 DOM，之后解析 CSS。由于 CSS 只有一个可用的规则，该规则有一个span选择器，它会将这个规则应用到这三个<span>的每一个上。更新后的输出如下所示：



### 如何将你的 CSS 应用到你的 HTML 上

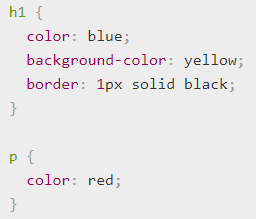
这有你常见的三种不同方式将 CSS 应用到 HTML 文档上，有的方式比其他方式更有用。在这里，我们将简要回顾一下每一种方式：

**外部样式表**

你已经在这篇文章看到了外部样式表，但是并不知道它的名字。外部样式表是指：当你将你的 CSS 保存在一个独立的扩展名为 .css 的文件中，并从HTML的 <link> 元素中引用它。此时 HTML 文件看起来像这样：



以及下面的 CSS 文件



这种方法可以说是最好的，因为你可以使用一个样式表来设置多个文档的样式，并且需要更新 CSS 的时候只要在一个地方更新。

**内部样式表**

内部样式表是指不使用外部 CSS 文件，而是将你的 CSS 放置在<style> 元素中，该元素包含在 HTML head 内。此时HTML看起来像这样：



这在某些情况下很有用（也许你正在使用一个内容管理系统，不能直接修改 CSS 文件），但它不如外部样式表高效 —— 在网站中，CSS 将需要在每个页面重复，并且需要更新时要更改的多个位置。

**内联样式**

内联样式是仅影响一个元素的CSS声明，被 style 属性包括着



除非有必要，否则不要这么做！这很难维护（你可能不得不在每份文档里更新多次同样的信息），并且它还混合了 CSS 表示的样式信息和 HTML 的结构信息，使 CSS 难以阅读和理解。保持不同类型代码的分离和纯净使处理该代码的任何人工作更为容易。

您唯一可能需要使用内联样式是当您的工作环境真的非常受限（也许您的CMS只允许您编辑 HTML 的 body）。

## CSS语法

### 语法结构

从最基本的层次来看，CSS是由两块内容组合而成的：

* 属性（Property）：一些人类可理解的标识符，这些标识符指出你想修改哪一些样式，例如：字体，宽度，背景颜色等。
* 属性值（Value）：每个指定的属性都需要给定一个值，这个值表示你想把那些样式特征修改成什么样，例如，你想把字体，宽度或背景颜色改成什么。

与值配对的属性被称为CSS声明。CSS声明会被放置在一个CSS声明块中。最后，CSS声明块与选择器相结合形成一个CSS规则集（或CSS规则）。

#### CSS 声明

给 CSS 属性设置特定的值是 CSS 语言的核心功能。CSS 引擎会通过计算，将对应的 CSS 声明应用到页面的每一个元素上，从而使得元素们以适当的方式布局，并展示出适当的样式。特别需要记住的是，CSS 的属性和属性值都是区分大小写的。属性和属性值之间，用英文半角冒号 (:) 隔离，如下图所示：

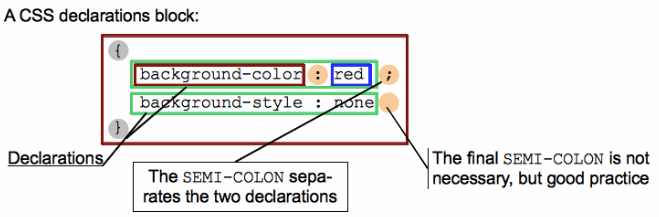


CSS 有超过300 个不同的属性以及几乎无穷无尽的属性值。属性和属性值不能任意组合：每个属性都有一个已经定义好的可用属性值范围。

#### CSS 声明块

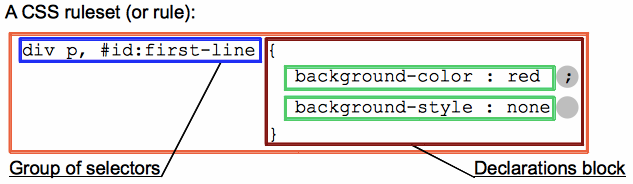
声明按块分组，每一组声明都用一对大括号包裹，用 ({) 开始，用 (}) 结束。

声明块里的每一个声明必须用半角分号（;）分隔，否则代码会不生效（至少不会按预期结果生效）。声明块里的最后一个声明结束的地方，不需要加分号，但是最后加分号是个好习惯，因为可以防止在后续增加声明时忘记加分号。



#### CSS 选择器和规则

我们的拼图还少了一块——我们需要讨论一下如何告知我们的声明块：哪些元素是它们需要应用的。通过在每个声明块前加上选择器（selector）来完成这一动作，选择器是一种模式，它能在页面上匹配一些元素。这将使相关的声明仅被应用到被选择的元素上。选择器加上声明块被称为规则集（ruleset），通常简称规则（rule）。



选择器可以很复杂 —— 你可以制作一个匹配多种元素的规则，这是通过把多个选择器囊括成用逗号分隔的选择器（一组,）来达成；选择器还可以被“链”在一起，例如我要选择所有 class 是 "blah" 的元素, 但是当且仅当它在 <article> 元素里、并且仅当鼠标指针悬停在这个元素上的时候。 别担心：随着你在CSS上的经验变的更丰富，事情会变的更清晰，并且我们将会在下一篇文章选择器里，详细地解释选择器。

一个元素可以被多个选择器所匹配，因此，一个给定的属性可能被多个规则设置多次。 CSS 定义了哪个规则比其它规则更具优先级，则更具优先级的规则必定被应用：这被称为层叠算法（cascade algorithm），关于层叠算法的更多内容和运作原理见层叠和继承。

#### CSS 语句（CSS statements）

CSS 规则是样式表的主要组成块 —— 是你在 CSS 中最常见的块。但这有一些其它类型的块，你以后偶尔会碰上 —— CSS 规则只是被称为 CSS 语句中的一种。其它类型如下：

**@-规则(At-rules)**

在CSS中被用来传递元数据、条件信息或其它描述性信息。它由（@）符号开始，紧跟着一个表明它是哪种规则的描述符，之后是这种规则的语法块，并最终由一个半角分号（;）结束。每种由描述符定义的@-规则，都有其特有的内部语法和语义。一些例子如下：

1. @charset 和 @import （元数据）
2. @media 或 @document （条件信息，又被称为嵌套语句，见下方。)
3. @font-face （描述性信息）

具体语法示例：



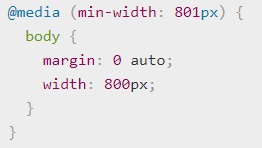
该@-规则向当前 CSS 导入其它 CSS 文件

**嵌套语句**

是@-规则中的一种，它的语法是 CSS 规则的嵌套块，只有在特定条件匹配时才会应用到文档上。特定条件如下：

1. @media 只有在运行浏览器的设备匹配其表达条件时才会应用该@-规则的内容；
2. @supports 只有浏览器确实支持被测功能时才会应用该@-规则的内容；
3. @document 只有当前页面匹配一些条件时才会应用该@-规则的内容。

具体语法示例



上述的嵌套语句只有在页面宽度超过801像素时才会应用。

|  |  |  |
| --- | --- | --- |
| 规则名称 | 当前状态 | 描述 |
| [@charset](#charset) | 稳定 | 指定样式表中使用的字符编码。它必须是样式表中的第一个元素，而前面不得有任何字符。 |
| [@import](#import) | 稳定 | 用于从其他样式表导入样式规则。 |
| [@namespace](#namespace) | 稳定 | 告诉 CSS 引擎必须考虑XML命名空间。 |
| [@media](#media) | 稳定 | 如果满足媒介查询的条件则条件规则组里的规则生效。 |
| [@page](#page) | 稳定 | 描述打印文档时布局的变化 |
| @font-face | 实验中 | 描述将下载的外部的字体。 |
| @keyframes | 实验中 | 描述 CSS 动画的中间步骤 |
| @supports | 实验中 | 如果满足给定条件则条件规则组里的规则生效。 |
| @document | 实验中 | 如果文档样式表满足给定条件则条件规则组里的规则生效。 |

* **@****charset**

@charset CSS @规则 指定样式表中使用的字符编码。它必须是样式表中的第一个元素，而前面不得有任何字符。因为它不是一个嵌套语句，所以不能在@规则条件组中使用。如果有多个 @charset @规则被声明，只有第一个会被使用，而且不能在HTML元素或HTML页面的字符集相关 <style> 元素内的样式属性内使用。

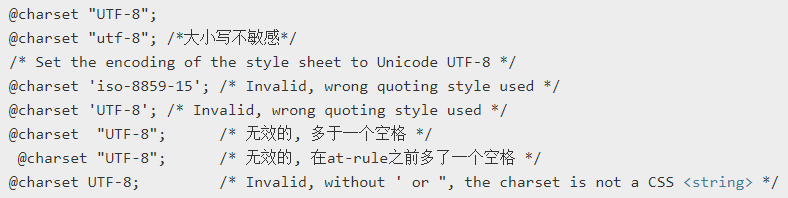
此 @规则 在某些 CSS 属性中使用非 ASCII 字符时非常有用，例如 content。

在样式表中有多种方法去声明字符编码，浏览器会按照以下顺序尝试下边的方法（一旦找到就停止并得出结果）：

* 文件的开头的 Unicode byte-order 字符值。
* 由Content-Type：HTTP header 中的 charset 属性给出的值或用于提供样式表的协议中的等效值。
* CSS @规则 @charset。
* 使用参考文档定义的字符编码： <link> 元素的 charset 属性。 该方法在 HTML5 标准中已废除，无法使用。
* 假设文档是 UTF-8。



它是一个 <string> 表示字符编码被使用。它必须是在被 IANA-registry 声明过的 web-safe 字符编码中的一个, 还必须被双引号包围, 遵循一个空格字符 (U+0020)，并且立即以分号结束。 如果有多个相关的编码名字，只有被标记为 preferred 的那个才会被使用。



* **@****import**

@import CSS@规则，用于从其他样式表导入样式规则。这些规则必须先于所有其他类型的规则，@charset 规则除外; 因为它不是一个嵌套语句，@import不能在条件组的规则中使用。

因此，用户代理可以避免为不支持的媒体类型检索资源，作者可以指定依赖媒体的@import规则。这些条件导入在URI之后指定逗号分隔的媒体查询。在没有任何媒体查询的情况下，导入是无条件的。指定所有的媒体具有相同的效果。

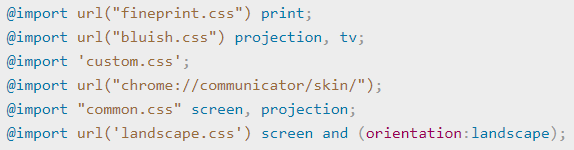


**url**

是一个表示要引入资源位置的 <string> 或者 <uri> 。 这个 URL 可以是绝对路径或者相对路径。 要注意的是这个 URL 不需要指明一个文件； 可以只指明包名，然后合适的文件会被自动选择 (e.g. chrome://communicator/skin/). See here 了解更多。

**list-of-media-queries**

是一个逗号分隔的 媒体查询 条件列表，决定通过URL引入的 CSS 规则 在什么条件下应用。如果浏览器不支持列表中的任何一条媒体查询条件，就不会引入URL指明的CSS文件。



* **@****namespace**

@namespace 是用来定义使用在CSS样式表中的XML命名空间的@规则。定义的命名空间可以把通配、元素和属性选择器限制在指定命名空间里的元素。@namespace规则通常在处理包含多个namespaces的文档时才有用，比如HTML5里内联的SVG、MathML或者混合多个词汇表的XML。

任何 @namespace 规则都必须在所有的 @charset 和 @import 规则之后, 并且在样式表中，位于其他任何 style declarations 之前。

@namespace 可以用来定义默认命名空间。当定义过默认命名空间后, 所有的通配选择器和类型选择器（但不包括属性选择器，详情看下面的note）都只应用在这个命名空间的元素中。

@namespace 规则也可以用于定义命名空间前缀。当一个通配、类型、属性选择器前面有命名空间前缀修饰时，这个选择器将只匹配那些命名空间与 元素名或属性匹配 的元素。

在 HTML5 中，已知的 foreign elements(“外语元素，比如SVG和math命名空间中的元素”)将被自动加上命名空间。这意味着即使文档中没有任何xmlns属性，HTML元素也将表现的如同他们在XHTML命名空间内 (http://www.w3.org/1999/xhtml)一样，而 <svg> 和 <math> 元素则将被加上对应的命名空间 (http://www.w3.org/2000/svg 和http://www.w3.org/1998/Math/MathML).

Note: 在 XML中，除非前缀直接定义在了一个属性上（例如：xlink:href），这个属性就不会有命名空间。换句话说，属性将不会继承其所在元素的命名空间，为了符合这个行为，CSS中的默认命名空间不会加到属性选择器中



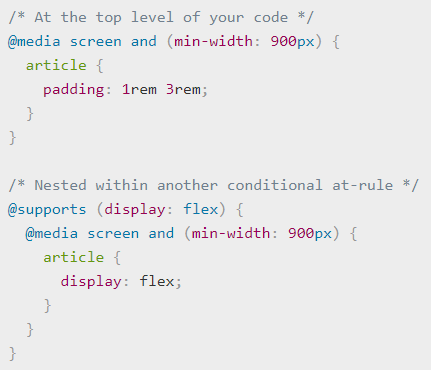


* **@****media**

@media CSS @规则 可根据一个或多个 媒体查询 的结果应用样式表的一部分。

注： 在JavaScript中， @media 可通过CSSMediaRule CSS 对象模型接口访问。

@media 规则可置于您代码的顶层或位于其它任何@条件规则组内。



* **@****page**

@page 规则用于在打印文档时修改某些CSS属性。你不能用@page规则来修改所有的CSS属性，而是只能修改margin,orphans,widow 和 page breaks of the document。对其他属性的修改是无效的。

### 使 CSS 更具可读性

正如你所见的，CSS 语法并不难写：如果你写了一些错误的规则，它将仅被忽略。但是，即使有这样的机制，这也有一些值得了解的最佳实践，使您的CSS代码更易于使用和维护。

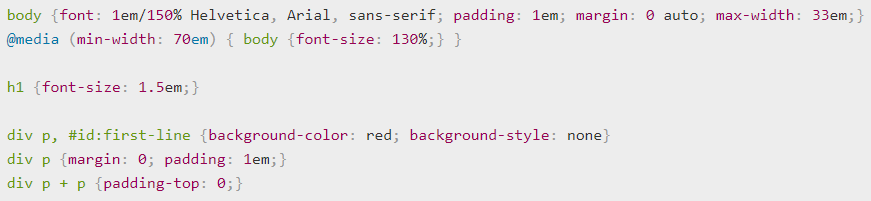
**空格**

空格实际上意味着一些空格，一些制表符以及一些新行。你可以通过添加空格使你的样式表更具可读性。

用与处理 HTML 的方式类似，浏览器会倾向于忽略你 CSS 中的许多空格；许多空格在那的意义仅仅是增加了可读性。在下面的第一个例子中，我们的每个声明（以及规则的开始 / 结束）都在自己各自的行上 —— 这可以说是编写 CSS 的好方法，因为它很容易维护和理解：



你可以编写跟这完全一样的CSS，并移除大部分空格，它在功能上和第一个例子完全相同，但我确信你会同意这样有些难读：



你选择的代码布局通常依据个人喜好，但当你开始在团队中工作时，你可能会发现团队已有自己的风格指南，约定了代码编写规范。

您在CSS中需要注意的空格是属性和属性值边上的空格。例如，以下是有效的CSS：



但以下是无效的：



因为 0auto 不被解析为 margin 属性的有效值（0和auto是两个单独的值），并且浏览器不会将 padding- 解析为有效属性。因此，你应该始终确保至少用一个空格分隔不同的值，且保持属性名/值为一个连续的字符串。

**注释**

与HTML一样，我们鼓励您在CSS中使用注释，以帮助您在几个月后遇上代码时了解代码如何工作，并帮助他人理解该代码。它同样对暂时注释某些部分的代码进行测试很有用，例如当您尝试查找是代码的哪一部分导致错误时。

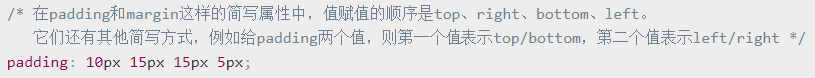
CSS中的注释以 /\* 开始并以 \*/ 结束。



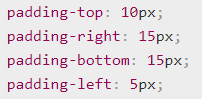
**简写**

一些属性比如 font，background，padding，border，和 margin 被称为简写属性 —— 这是由于它们允许你在一行设置多个属性，从而节省时间并使代码更整洁。

例如，像这行代码：



和以下的所有代码做了相同的工作：



而这一行：



和以下这些做了相同的事：

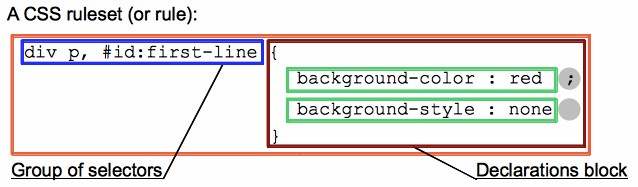


## 选择器

在CSS中，选择器用于定位我们想要样式化的网页HTML 元素。各种各样可用的CSS选择器允许我们精确选择要样式化的元素。

### 基础

在上篇文章中我们已经详细介绍了常规的 CSS 语法和术语 。简而言之，选择器是 CSS 规则的一部分且位于 CSS 声明块前。



**不同种类的CSS选择器**

选择器可以被分为以下类别：

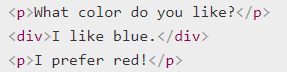
1. 简单选择器（Simple selectors）：通过元素类型、class 或 id 匹配一个或多个元素。
2. 属性选择器（Attribute selectors）：通过 属性 / 属性值 匹配一个或多个元素。
3. 伪类（Pseudo-classes）：匹配处于确定状态的一个或多个元素，比如被鼠标指针悬停的元素，或当前被选中或未选中的复选框，或元素是DOM树中一父节点的第一个子节点。
4. 伪元素（Pseudo-elements）:匹配处于相关的确定位置的一个或多个元素，例如每个段落的第一个字，或者某个元素之前生成的内容。
5. 组合器（Combinators）：这里不仅仅是选择器本身，还有以有效的方式组合两个或更多的选择器用于非常特定的选择的方法。例如，你可以只选择divs的直系子节点的段落，或者直接跟在headings后面的段落。
6. 多重选择器（Multiple selectors）：这些也不是单独的选择器；这个思路是将以逗号分隔开的多个选择器放在一个CSS规则下面， 以将一组声明应用于由这些选择器选择的所有元素。

### 简单选择器

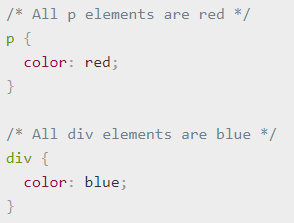
1. **类型选择器（又名元素选择器）**

此选择器只是一个选择器名和指定的HTML元素名的不区分大小写的匹配。这是选择所有指定类型的最简单方式。让我们一起看看下面这个例子:

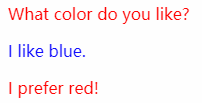
这是HTML:



这是样式表:



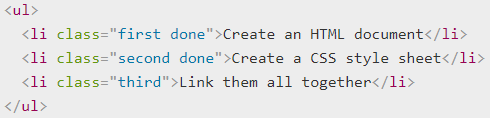
我们将得到这个：



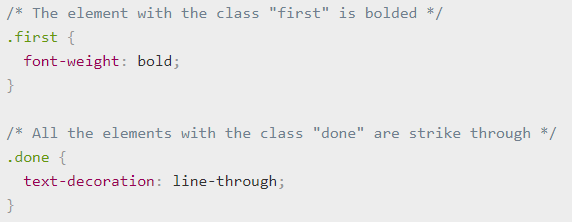
1. **类选择器**

类选择器由一个点“.”以及类后面的类名组成。类名是在HTML class文档元素属性中没有空格的任何值。由你自己选择一个名字。同样值得一提的是，文档中的多个元素可以具有相同的类名，而单个元素可以有多个类名(以空格分开多个类名的形式书写)。以下是一个简单的例子：

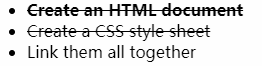
这是一些HTML：



这是一些CSS样式:



我们得到这样一个结果:



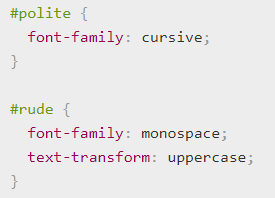
1. ID 选择器

ID选择器是由一个哈希/磅符号 (#)，后面跟着给定元素的ID名称组成的。 任何元素都可以使用id属性设置唯一的ID名称。 由你自己决定选择一个ID名称是什么。 这是选择单个元素的最有效的方式。

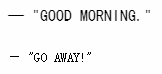
这是HTML：



这是样式表：



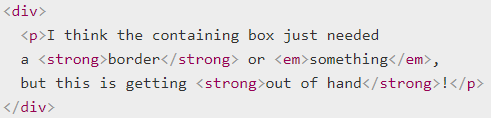
而我们得到这个作为输出：



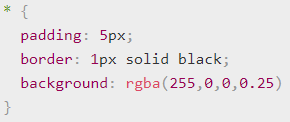
1. 通用选择器

通用选择（\*）是最终的王牌。它允许选择在一个页面中的所有元素。由于给每个元素应用同样的规则几乎没有什么实际价值，更常见的做法是与其他选择器结合使用(参考下面 组合 .)

这是HTML：



这是样式表：



组合在一起，会得到这样的结果：



### 属性选择器

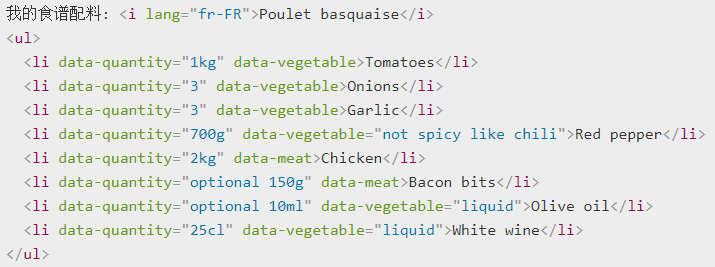
属性选择器是一种特殊类型的选择器，它根据元素的 属性 和属性值来匹配元素。它们的通用语法由方括号 ([]) 组成，其中包含属性名称，后跟可选条件以匹配属性的值。 属性选择器可以根据其匹配属性值的方式分为两类： 存在和值属性选择器和子串值属性选择器。

#### 存在和值（Presence and value）属性选择器

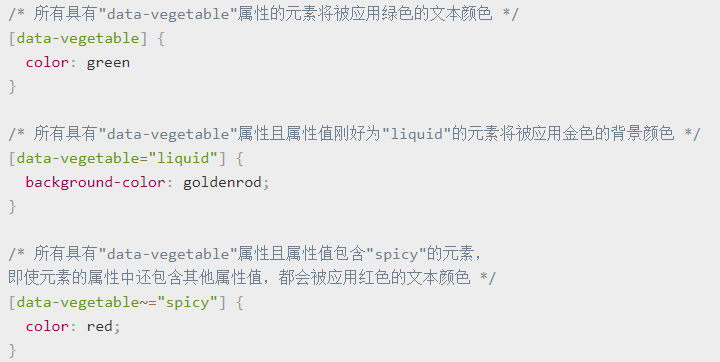
这些属性选择器尝试匹配精确的属性值：

* + [attr]：该选择器选择包含 attr 属性的所有元素，不论 attr 的值为何。
  + [attr=val]：该选择器仅选择 attr 属性被赋值为 val 的所有元素。
  + [attr~=val]：该选择器仅选择具有 attr 属性的元素，而且要求 val 值是 attr 值包含的被空格分隔的取值列表里中的一个。

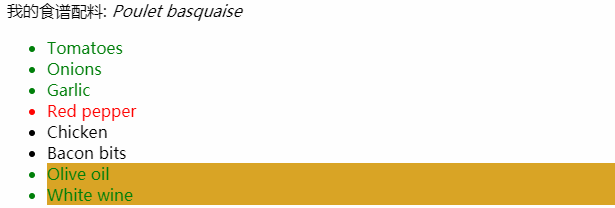
让我们看一个特别的例子，下面是它的的HTML代码：



和一个简单的样式表：



结果如下：



#### 子串值（Substring value）属性选择器

这种情况的属性选择器也被称为“伪正则选择器”，因为它们提供类似 regular expression 的灵活匹配方式（但请注意，这些选择器并不是真正的正则表达式）：

* [attr|=val] : 选择attr属性的值是 val 或值以 val- 开头的元素（注意，这里的 “-” 不是一个错误，这是用来处理语言编码的）。
* [attr^=val] : 选择attr属性的值以 val 开头（包括 val）的元素。
* [attr$=val] : 选择attr属性的值以 val 结尾（包括 val）的元素。
* [attr\*=val] : 选择attr属性的值中包含子字符串 val 的元素（一个子字符串就是一个字符串的一部分而已，例如，”cat“ 是 字符串 ”caterpillar“ 的子字符串）。

让我们继续我们前面的例子，并添加以下CSS规则：



有了这些新的规则，我们会得到这个：



### 伪类和伪元素

在选择器系列文章的第三篇中，我们讨论伪选择器 。该选择器不是选择元素，而是元素的某些部分，或仅在某些特定上下文中存在的元素。它们有两种主要类型 ： 伪类和伪元素。

#### 伪类（Pseudo-class）

一个 CSS 伪类（pseudo-class） 是一个以冒号(:)作为前缀，被添加到一个选择器末尾的关键字，当你希望样式在特定状态下才被呈现到指定的元素时，你可以往元素的选择器后面加上对应的伪类（pseudo-class）。你可能希望某个元素在处于某种状态下呈现另一种样式，例如当鼠标悬停在元素上面时，或者当一个复选框被禁用或被勾选时，又或者当一个元素是它在 DOM 树中父元素的第一个子元素时。

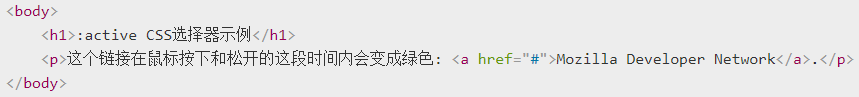
|  |  |  |
| --- | --- | --- |
| **伪类名称** | **当前状态** | **功能说明** |
| [:active](#active) | 稳定 | 匹配被用户激活的元素。它让页面能在浏览器监测到激活时给出反馈。 |
| :any-link | 试验中 | 代表一个有链接锚点的元素，而不管它是否被访问过，也就是说，它会匹配每一个有 href 属性的 <a>、<area> 或 <link> 元素。因此，它会匹配到所有的 :link 或 :visited |
| [:checked](#checked) | 稳定 | 表示任何处于选中状态的radio(<input type="radio">), checkbox (<input type="checkbox">) 或("select") 元素中的option HTML元素("option")。 |
| :blank | 实验中 | 用于匹配如下节点：没有子节点、仅有空的文本节点、仅有空白符的文本节点。 |
| [:default](#default) | 稳定 | 表示一组相关元素中的默认表单元素。该选择器可以在 <button>, <input type="checkbox">, <input type="radio">, 以及 <option> 上使用。 |
| [:defined](#defined) | 稳定 | 表示任何已定义的元素。这包括任何浏览器内置的标准元素以及已成功定义的自定义元素 |
| :dir() | 实验中 | 匹配特定文字书写方向的元素。在HTML中, 文字方向由dir属性决定。其他的文档类型可能有其他定义文字方向的方法。 |
| [:disabled](#disabled) | 稳定 | 表示任何被禁用的元素。如果一个元素不能被激活（如选择、点击或接受文本输入）或获取焦点，则该元素处于被禁用状态。 |
| [:empty](#empty) | 稳定 | 代表没有子元素的元素。子元素只可以是元素节点或文本（包括空格）。注释或处理指令都不会产生影响。 |
| [:enabled](#enabled) | 稳定 | 表示任何启用的（enabled）元素。如果一个元素能够被激活（如选择、点击或接受文本输入）或获取焦点，则该元素是启用的。 |
| [:first](#first) | 稳定 | :first @page CSS 伪类选择器 描述的是：打印文档的时候，第一页的样式。 |
| [:first-child](#firstChild) | 稳定 | 表示在一组兄弟元素中的第一个元素。 |
| [:first-of-type](#firstOfType) | 稳定 | 表示一组兄弟元素中其类型的第一个元素。 |
| [:fullscreen](#fullscreen) | 稳定 | 应用于当前处于全屏显示模式的元素。 它不仅仅选择顶级元素，还包括所有已显示的栈内元素。 |
| [:focus](#focus) | 稳定 | 表示获得焦点的元素（如表单输入）。当用户点击或触摸元素或通过键盘的 “tab” 键选择它时会被触发。 |
| :focus-visible | 实验中 | 这个选择器可以有效地根据用户的输入方式(鼠标 vs 键盘)展示不同形式的焦点。 |
| [:focus-within](#focusWithin) | 稳定 | 表示一个元素获得焦点，或，该元素的后代元素获得焦点。换句话说，元素自身或者它的某个后代匹配:focus伪类。 |
| :has | 实验中 | :has() 伪类选择器 能够选出具有满足参数选择器特性的元素. 目前而言 :has() 还不能被应用到行内样式表, 但是可以使用js的方法 document.querySelector() 选择. |
| [:host()](#host) | 实验中 | CSS 伪类函数 :host() 选择包含使用这段 CSS 的 Shadow DOM 的影子宿主（这样你就可以从 Shadow DOM 中选择包括它的自定义元素）——但前提是该函数的参数与选择的阴影宿主相匹配。 |
| :host-context() | 实验中 | CSS伪类:host-context()函数选择包含CSS的影子的影子DOM是内部使用(所以你可以选择一个自定义的元素从阴影中DOM)——但只有在选择器作为函数的参数匹配的影子主机的祖先(s)将它坐落在DOM层次结构。 |
| [:hover](#hover) | 稳定 | :hover CSS伪类在用户与指向设备的元素交互时匹配，但不一定激活它。它通常在用户用鼠标(鼠标指针)悬停在元素上时触发。 |
| [:indeterminate](#indeterminate) | 稳定 | 表示状态不确定的表单元素 |
| [:in-range](#inRange) | 稳定 | 代表一个 <input> 元素，其当前值处于属性min 和max 限定的范围之内。该伪类用于给用户一个可视化的提示，表示输入域的当前值处于允许范围内。 |
| [:invalid](#invalid) | 稳定 | 表示任意内容未通过验证的 <input> 或其他 <form> 元素 |
| :matches() (:any()) | 稳定 | :matches() CSS 伪类 函数将选择器列表作为参数，并选择该列表中任意一个选择器可以选择的元素。这对于以更紧凑的形式编写大型选择器非常有用。 |
| [:lang](#lang) | 稳定 | 基于元素语言来匹配页面元素。 |
| [:last-child](#lastChild) | 稳定 | 代表父元素的最后一个子元素。 |
| [:last-of-type](#lastOfType) | 稳定 | 表示在（它父元素的）子元素列表中，最后一个给定类型的元素。 |
| [:left](#left) | 稳定 | :left CSS 伪类, 需要和@规则 @page 配套使用, 对打印文档的左侧页设置CSS样式. |
| [:link](#link) | 稳定 | :link伪类选择器是用来选中元素当中的链接。它将会选中所有尚未访问的链接，包括那些已经给定了其他伪类选择器的链接（例如:hover选择器，:active选择器，:visited选择器）。 |
| [:not()](#not) | 稳定 | CSS 否定伪类，:not(X)，是以一个简单的以选择器X为参数的功能性标记函数。它匹配不符合参数选择器X描述的元素。X不能包含另外一个否定选择器。 |
| [:nth-child](#nthChild) | 稳定 | :nth-child(an+b) 这个 CSS 伪类首先找到所有当前元素的兄弟元素，然后按照位置先后顺序从1开始排序，选择的结果为第（an+b）个元素的集合（n=0，1，2，3...）。 |
| [:nth-last-child](#nthLastChild) | 稳定 | 匹配文档树中在其之后具有 an+b-1 个兄弟节点的元素，其中 n 为正值或零值。 |
| [:nth-last-of-type](#nthLastOfType) | 稳定 | 匹配那些在它之后有 an+b-1 个相同类型兄弟节点的元素，其中 n 为正值或零值。 |
| [:nth-of-type](#nthOfType) | 稳定 | 匹配文档树中在其之前具有 an+b-1 个相同兄弟节点的元素，其中 n 为正值或零值。 |
| [:only-child](#onlyChild) | 稳定 | 代表了属于某个父元素的唯一一个子元素. |
| [:only-of-type](#onlyOfType) | 稳定 | 代表了任意一个元素，这个元素没有其他相同类型的兄弟元素。 |
| [:optional](#optional) | 稳定 | 表示任意没有required属性的 <input>，<select> 或 <textarea> 元素使用它。 |
| [:out-of-range](#outOfRange) | 稳定 | 表示一个 <input> 元素，其当前值处于属性 min 和 max 限定的范围外。 |
| :placeholder-shown | 实验中 | 在 <input> 或 <textarea> 元素显示 placeholder text 时生效. |
| [:read-only](#readOnly) | 稳定 | 表示元素不可被用户编辑的状态 |
| [:read-write](#readWrite) | 稳定 | 代表一个元素（例如可输入文本的 input元素）可以被用户编辑。 |
| [:required](#required) | 稳定 | 表示 任意 <input> 元素表示任意拥有required属性的 <input> 或 <textarea> 元素使用它. 它允许表单在提交之前容易的展示必填字段并且渲染其外观. |
| [:right](#right) | 稳定 | :right CSS 伪类必须与@规则 @page 一起配套使用，表示打印文档的所有右页。 |
| [:root](#root) | 稳定 | 匹配文档树的根元素。对于 HTML 来说，:root 表示 <html> 元素，除了优先级更高之外，与 html 选择器相同。 |
| [:scope](#scope) | 实验中 | 匹配作为选择符匹配元素的参考点(css的作用域或作用点)。在HTML中，可以使用<style>的scoped属性来重新定义新的参考点。如果HTML中没有使用这个属性，那么默认的参考点(css的作用域或作用点)是<html>。 |
| [:target](#target) | 稳定 | 代表一个唯一的页面元素(目标元素)，其id 与当前URL片段匹配 . |
| [:valid](#valid) | 稳定 | 表示内容验证正确的<input> 或其他 <form> 元素。这能简单地将校验字段展示为一种能让用户辨别出其输入数据的正确性的样式。 |
| [:visited](#visited) | 稳定 | 表示用户已访问过的链接。 |
| :where() | 实验中 | :where() CSS 伪类 函数接受选择器列表作为它的参数，将会选择所有能被该选择器列表中任何一条规则选中的元素。 |

* **:active**

:active CSS伪类匹配被用户激活的元素。它让页面能在浏览器监测到激活时给出反馈。当用鼠标交互时，它代表的是用户按下按键和松开按键之间的时间。 :active 伪类通常用来匹配tab键交互。通常用于但并不限于 <a> 和 <button> HTML元素。

这个样式可能会被后声明的其他链接相关的伪类覆盖，这些伪类包括 :link，:hover和 :visited。为了正常加上样式，需要把 :active的样式放在所有链接相关的样式后，这种链接伪类先后顺序被称为LVHA顺序: :link — :visited — :hover — :active。

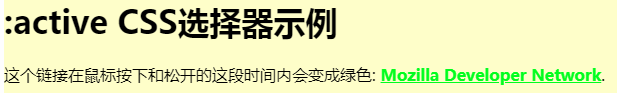
**HTML：**



**CSS：**



**结果：**



* **:****checked**

:checked CSS 伪类选择器表示任何处于选中状态的radio(<input type="radio">), checkbox (<input type="checkbox">) 或("select") 元素中的option HTML元素("option")。

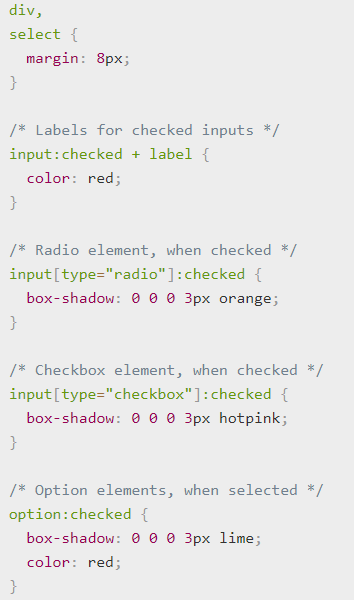


用户通过勾选/选中元素或取消勾选/取消选中，来改变该元素的 :checked 状态。

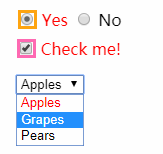
**HTML**



**CSS**



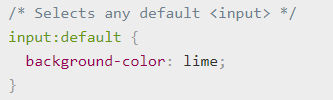
**结果**



* **:****default**

:default CSS pseudo-class 表示一组相关元素中的默认表单元素。

该选择器可以在 <button>, <input type="checkbox">, <input type="radio">, 以及 <option> 上使用。

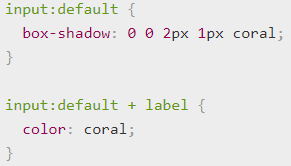


允许多个选择的分组元素也可以具有多个默认值，即，它们可以具有最初选择的多个项目。在这种情况下，所有默认值都使用 :default 伪类表示。例如，您可以在一组复选框之间设置默认复选框。

**HTML**



**CSS**



**结果**



* **:****defined**

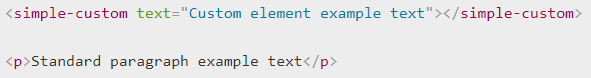
:defined CSS 伪类 表示任何已定义的元素。这包括任何浏览器内置的标准元素以及已成功定义的自定义元素 (例如通过 CustomElementRegistry.define() 方法)。



在这个 demo 中我们定义了一个非常简单的自定义元素:



然后在文档中插入一个该元素的副本，以及一个标准的 <p> 标签:



在 CSS 中我们首先包含以下规则:



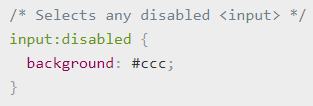
然后提供以下两个规则来隐藏未定义的自定义元素的所有实例，并显示被定义为块级元素的实例：



这在你有一个复杂的自定义元素需要一段时间才能加载到页面中时非常有用 —— 你可能想要隐藏元素的实例直到定义完成为止，这样你就不会在页面上出现一些难看的元素。

* **:****disabled**

:disabled CSS 伪类表示任何被禁用的元素。如果一个元素不能被激活（如选择、点击或接受文本输入）或获取焦点，则该元素处于被禁用状态。元素还有一个启用状态（enabled state），在启用状态下，元素可以被激活或获取焦点。

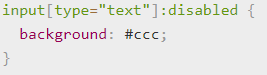


这个例子显示基本的购物表单。通过使用 JavaScript change 事件让用户启用/禁用付款字段。

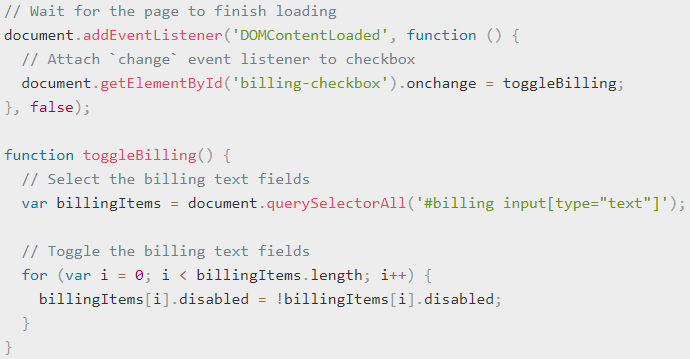
**HTML**



**CSS**



**JavaScript**

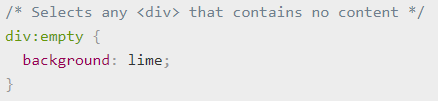


**结果**

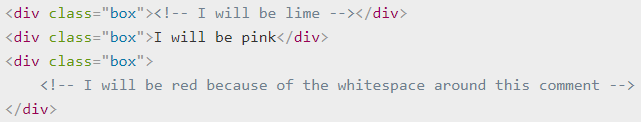


* **:****empty**

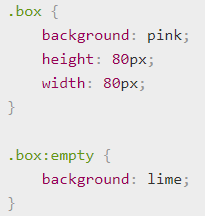
:empty CSS 伪类 代表没有子元素的元素。子元素只可以是元素节点或文本（包括空格）。注释或处理指令都不会产生影响。



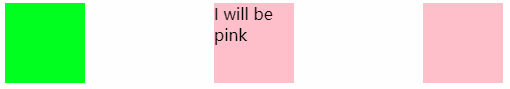
**HTML**



**CSS**



**结果**

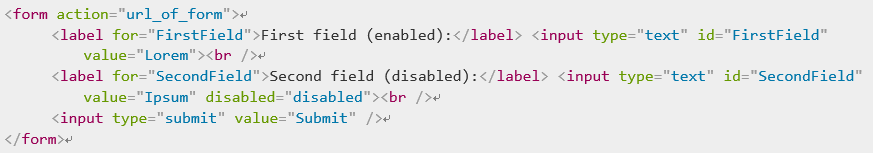


* **:****enabled**

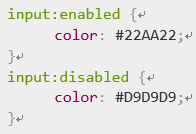
:enabled 表示任何启用的（enabled）元素。如果一个元素能够被激活（如选择、点击或接受文本输入）或获取焦点，则该元素是启用的。元素还有一个禁用的状态（disabled state），在被禁用时，元素不能被激活或获取焦点。

下面的代码，当文本输入框处于启用状态时，输入框的文本是中绿色（medium green），当处于禁用状态时，输入框的文本是灰色。这样可以把元素是否可用反馈给用户。

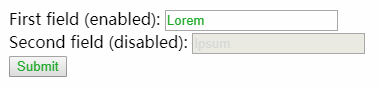
**HTML**



**CSS**



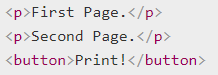
**结果**



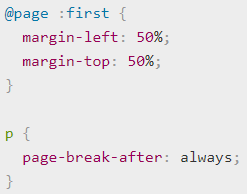
* **:****first**

:first @page CSS 伪类选择器 描述的是：打印文档的时候，第一页的样式。

**HTML**



**CSS**

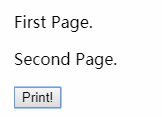


**JavaScript**



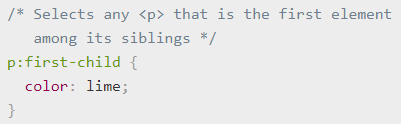
**结果**

点击下面的 print 按钮来打印这个例子。第一页的文字会在中间显示， 其他页的内容在默认位置显示。

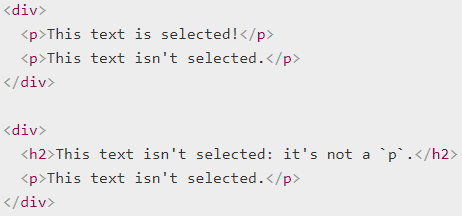


* **:****first-child**

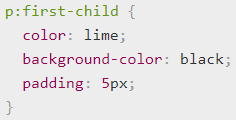
:first-child CSS pseudo-class 表示在一组兄弟元素中的第一个元素。



**HTML**



**CSS**

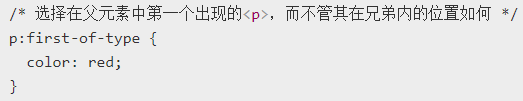


**结果**

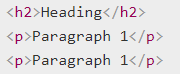


* **:****first-of-type**

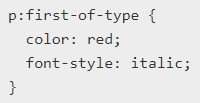
:first-of-type表示一组兄弟元素中其类型的第一个元素。



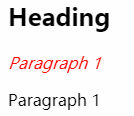
**HTML**



**CSS**



**结果**



* **:****fullscreen**

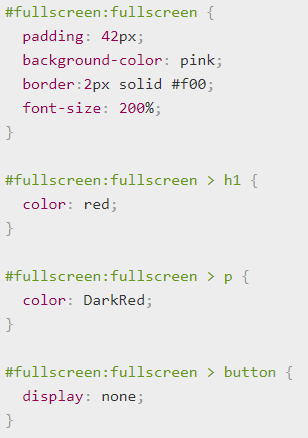
css伪类:fullscreen应用于当前处于全屏显示模式的元素。 它不仅仅选择顶级元素，还包括所有已显示的栈内元素。

W3C标准使用不带破折号的单词:fullscreen，但Webkit和Gecko应用接口各自使用前缀带有破折号的变量：:-webkit-full-screen 和:-moz-full-screen。微软的Edge和Internet Explorer各自使用标准语法：:fullscreen和:-ms-fullscreen。

**HTML**



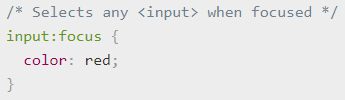
**CSS**



**结果**

* **:****focus**

:focus表示获得焦点的元素（如表单输入）。当用户点击或触摸元素或通过键盘的 “tab” 键选择它时会被触发。

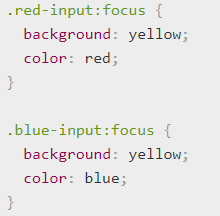


此伪类仅适用于焦点元素本身。如果要选择包含焦点元素的元素，请使用:focus-within 。

**HTML**



**CSS**



**结果**

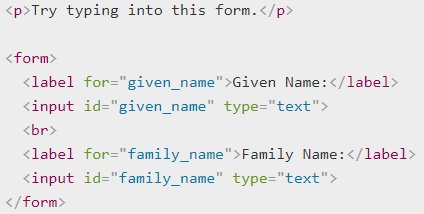


* **:****focus-within**

表示一个元素获得焦点，或，该元素的后代元素获得焦点。换句话说，元素自身或者它的某个后代匹配:focus伪类。



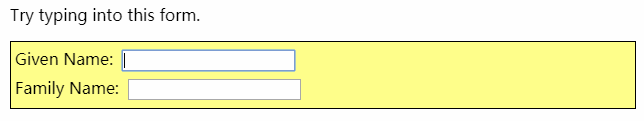
**HTML**



**CSS**



**结果**

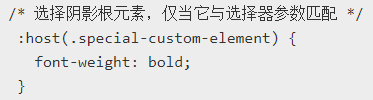


* **:****host()**

的参数与选择的阴影宿主相匹配。

最明显的用法是仅将类名放在某些自定义元素实例上，然后将相关的类选择器作为函数参数包含在内。您不能将它与后代选择器表达式一起使用，以仅选择特定祖先内部的自定义元素的实例。这是 :host-context() 的工作。

注意：在shadow DOM之外使用时，这没有任何效果。



在这个例子中，我们有一个简单的自定义元素 <context-span> 你可以用它包裹文本：



在元素的构造函数中，我们创建style和span元素，填充span自定义元素的内容，并style使用一些 CSS 规则填充元素：



:host(.footer) { color : red; } 规则为文档中所有在其上设置了 footer 类的<context-span> 元素实例（此实例中的影子宿主）设置样式——我们使用它来为 <footer> 中的元素提供实例一种特殊的颜色。

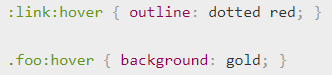
* **:****hover**

:hover CSS伪类适用于用户使用指示设备虚指一个元素（没有激活它）的情况。这个样式会被任何与链接相关的伪类重写，像:link, :visited, 和 :active等。为了确保生效，:hover规则需要放在:link和:visited规则之后，但是在:active规则之前，按照LVHA的循顺序声明:link－:visited－:hover－:active。

:hover伪类可以任何伪元素上使用。

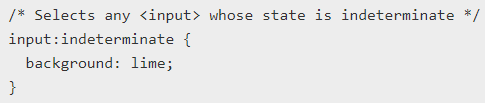
用户的可视客户端比如Firefox, Internet Explorer, Safari, Opera or Chrome, 会在光标（鼠标指针）悬停在元素上时提供关联的样式

注意: 在触摸屏上 :hover 有问题，基本不可用。不同的浏览器上:hover 伪类表现不同。 可能从不会触发；或者在触摸某元素后触发了一小会儿；或者总是触发即使用户不在触摸了，直到用户触摸别的元素。 触摸屏非常普遍，所以网页开发人员不要让任何内容只能通过悬停才能展示出来，不然这些内容对于触摸屏使用者来说是很难或者说不可能看到。

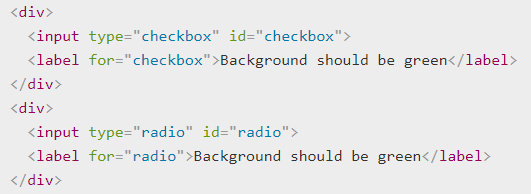


* **:****indeterminate**

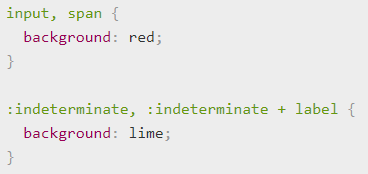
:indeterminate CSS 伪类 表示状态不确定的表单元素:



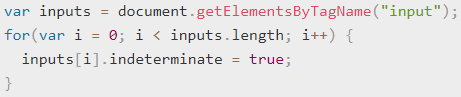
**HTML**



**CSS**



**JavaScript**

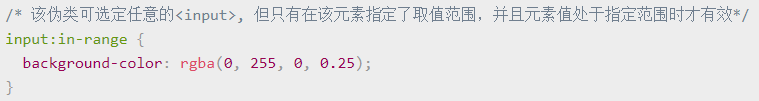


**结果**



* **:****in-range**

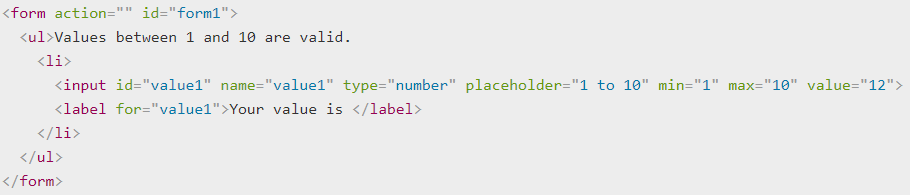
in-range CSS 伪类 代表一个 <input> 元素，其当前值处于属性min 和max 限定的范围之内.



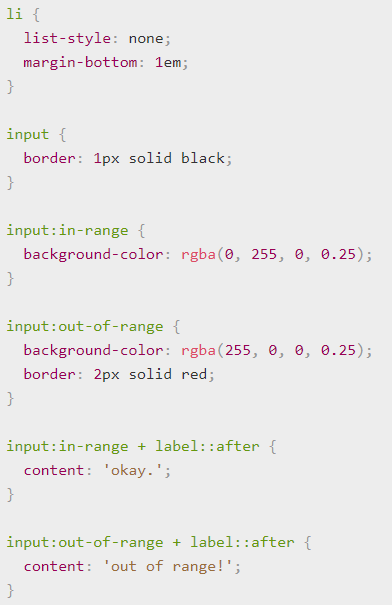
该伪类用于给用户一个可视化的提示，表示输入域的当前值处于允许范围内。

注意：该伪类仅适用于那些拥有（或可以接受）取值范围设定的元素。若缺少此类设定，元素值就无所谓“in-range”和“out-range”。

**HTML**



**CSS**

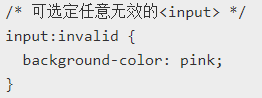


**结果**



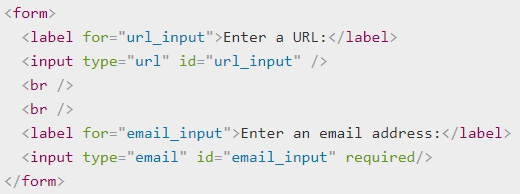
* **:****invalid**

:invalid CSS 伪类 表示任意内容未通过验证的 <input> 或其他 <form> 元素 .

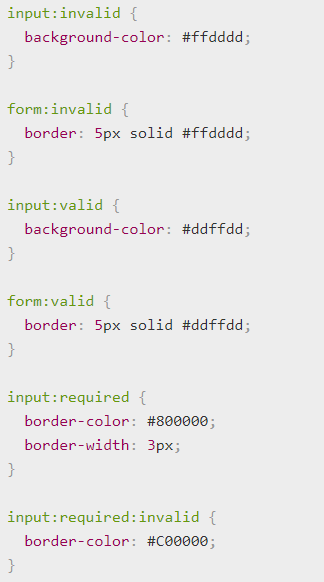


这个伪类对于突出显示用户的字段错误非常有用。

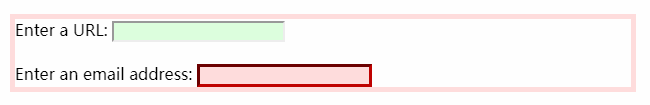
**HTML**



**CSS**



**结果**



注意

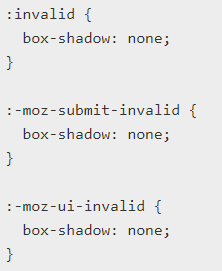
单选钮（Radio buttons）

若一组单选钮被设定为必须选定一个，在该组中没有按钮被选中的情况，:invalid伪类被应用到该组中的所有按钮。（单选钮组按照name属性共享相同值。）

Gecko 默认

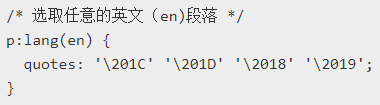
默认情况下，Gecko不应用:invalid伪类设置的样式。但是，Gecko可以使用:-moz-ui-invalid伪类的样式（使用box-shadow 属性设置红色发光边框）来达到类似的效果，它可以被应用在:invalid伪类的子集中。

你可以使用以下CSS禁用红色发光边框,或完全重写它以改变无效输入域的外观：



* **:****lang**

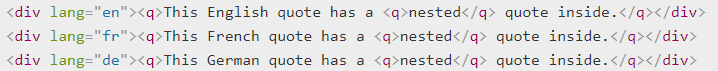
:lang() CSS 伪类基于元素语言来匹配页面元素。



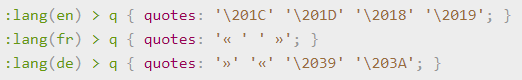
注意: 在HTML中, 语言是通过lang 属性,和 <meta> 元素的组合来决定的, 也可能是通过协议的信息来确定（例如HTTP头）. 对于其他文档类型，也可能存在其他用于确定语言的方法。

在这个例子中, :lang() 伪类使用子选择器来匹配引用元素(<q>)的父元素. 需要注意的是，此处演示的方法并不是唯一的，最好的方法需要依据文档类型来确定。还需要注意的是， Unicode 值用于指定一些特殊字符的引用.

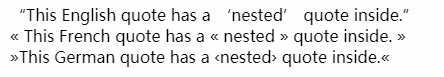
**HTML**



**CSS**



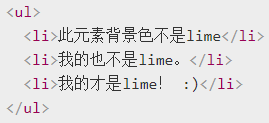
**结果**



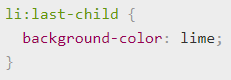
* **:****last-child**

:last-child CSS 伪类 代表父元素的最后一个子元素。

**HTML**



**CSS**



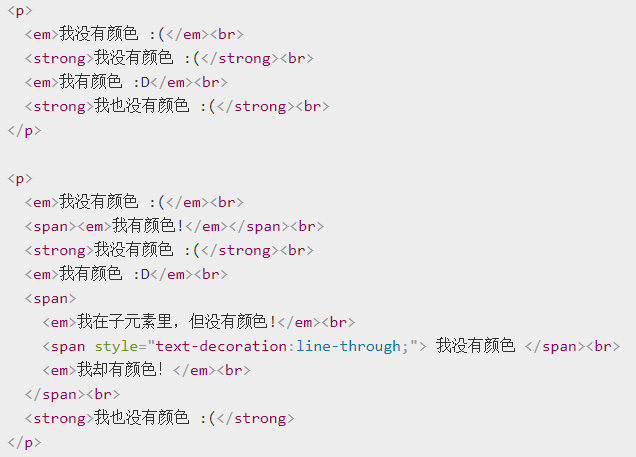
**结果**



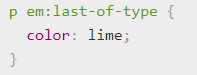
* **:****last-of-type**

:last-of-type CSS 伪类 表示了在（它父元素的）子元素列表中，最后一个给定类型的元素。当代码类似Parent tagName:last-of-type的作用区域包含父元素的所有子元素中的最后一个选定元素，也包括子元素的最后一个子元素并以此类推。

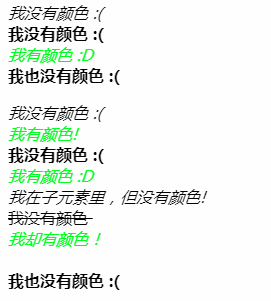
**HTML**



**CSS**



**结果**



* **:****left**
* **:****link**
* **:****not()**
* **:****nth-child**
* **:****nth-last-child**
* **:****nth-last-of-type**
* **:****nth-of-type**
* **:****only-child**
* **:****only-of-type**
* **:****optional**
* **:****out-of-range**
* **:****read-only**
* **:****read-write**
* **:****required**
* **:****right**
* **:****root**
* **:****scope**
* **:****target**
* **:****valid**
* **:****visited**

#### 伪元素（Pseudo-element）

伪元素（Pseudo-element）跟伪类很像，但它们又有不同的地方。它们都是关键字，但这次伪元素前缀是两个冒号 (::) ， 同样是添加到选择器后面去选择某个元素的某个部分。

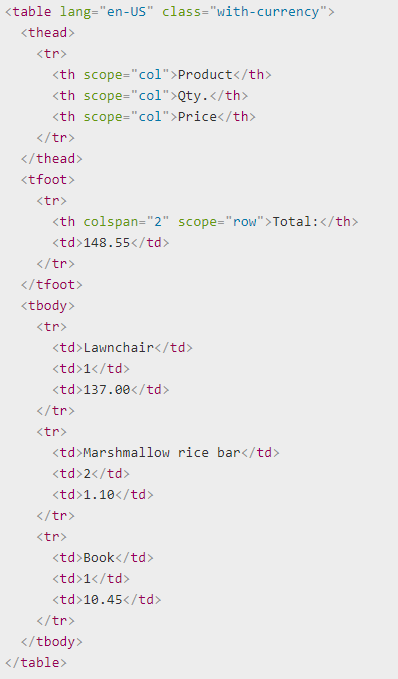
|  |  |  |
| --- | --- | --- |
| 伪元素名称 | 当前状态 | 功能说明 |
| ::after(:after) | 稳定 | 作为已选中元素的最后一个子元素。通常会配合content属性来为该元素添加装饰内容。这个虚拟元素默认是行内元素。 |
| ::backdrop | 实验中 | 是在任何处于全屏模式的元素下的即刻渲染的盒子（并且在所有其他在堆中的层级更低的元素之上）。 |
| ::before (:before) | 稳定 | 将成为匹配选中的元素的第一个子元素。常通过 content 属性来为一个元素添加修饰性的内容。此元素默认为行内元素。 |
| ::cue (:cue) | 稳定 | 匹配所选元素中的WebVTT提示。这可以用于在VTT轨道的媒体中使用字幕和其他线索。 |
| ::first-letter (:first-letter) | 稳定 | 选中某 block-level element（块级元素）第一行的第一个字母，并且文字所处的行之前没有其他内容（如图片和内联的表格） 。 |
| ::first-line (:first-line) | 稳定 | 在某 block-level element （块级元素）的第一行应用样式。第一行的长度取决于很多因素，包括元素宽度，文档宽度和文本的文字大小。 |
| ::grammar-error | 实验中 | 用于浏览器标识为语法错误的文本段 |
| ::placeholder | 实验中 | 选择一个表单元素的占位文本，它允许开发者和设计师自定义占位文本的样式。 |
| ::selection | 稳定 | 应用于文档中被用户高亮的部分（比如使用鼠标或其他选择设备选中的部分）。 |
| ::slotted() | 稳定 | 用于选定那些被放在 HTML模板 中的元素 (更多请查看 使用模板和插槽). |
| ::spelling-error | 实验中 | 表示浏览器标记为不正确拼写的文本段。 |

### 组合器和多用选择器

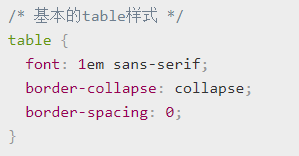
虽然一次使用一个选择器就很有用，但在某些情形中却可能效率低下。 CSS选择器在你开始组合他们进行细粒度选择的时候变得更具使用价值。基于元素之间的相互关联关系，CSS提供了几种方法来对元素进行选择。下表使用连接符展示了这些关联关系(A和B代表前文所述的任意选择器):

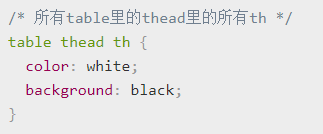
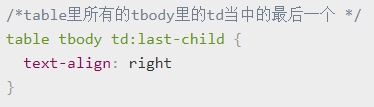
|  |  |  |
| --- | --- | --- |
| 名称 | 组合器 | 选择 |
| 选择器组 | A,B | 匹配满足A（和/或）B的任意元素 |
| 后代选择器 | A B | 匹配B元素，满足条件：B是A的后代结点（B是A的子节点，或者A的子节点的子节点） |
| 子选择器 | A > B | 匹配B元素，满足条件：B是A的直接子节点 |
| 相邻兄弟选择器 | A + B | 匹配B元素，满足条件：B是A的下一个兄弟节点（AB有相同的父结点，并且B紧跟在A的后面） |
| 通用兄弟选择器 | A ~ B | 匹配B元素，满足条件：B是A之后的任意一个兄弟节点（AB有相同的父节点，B在A之后，但不一定是紧挨着A） |

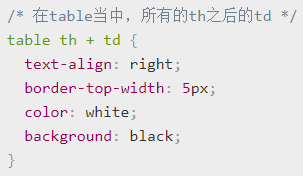
**HTML**

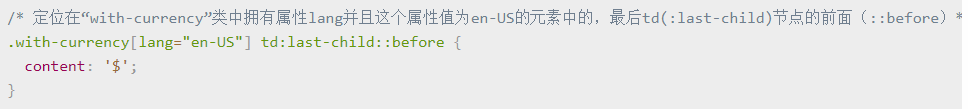


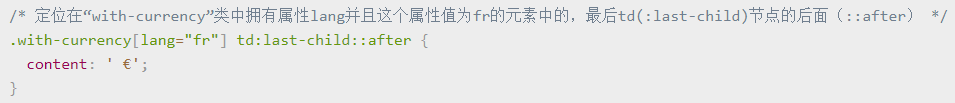
**CSS**

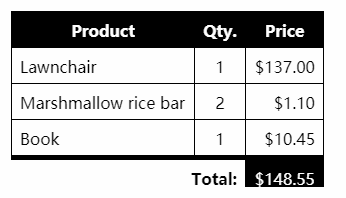
   





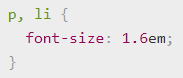


**结果**

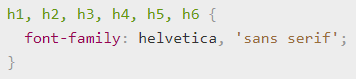


**应用同一规则的选择器组**

你已经遇见过这种做法的许多例子，但还是让我们来把它进一步阐释清楚。通过相互间用逗号分隔的多个选择器所形成的组，可以一次性将同一规则同时应用到多组选定元素。例如：



或者这样:



## CSS数值和单位

CSS的属性值可以有很多种类，可以是普通类型的数值，也可以是有特定作用的颜色和函数（比如内置的背景图片，或者旋转一个元素）。有些值通过特定的单位来指定与之相对应的值。

* 数值: 长度值，用于指定例如元素宽度、边框（border）宽度或字体大小；以及无单位整数。用于指定例如相对线宽或运行动画的次数。
* 百分比: 可以用于指定尺寸或长度——例如取决于父容器的长度或高度，或默认的字体大小。
* 颜色: 用于指定背景颜色，字体颜色等。
* 坐标位置: 例如，以屏幕的左上角为坐标原点定位元素的位置。
* 函数: 例如，用于指定背景图片或背景图片渐变。

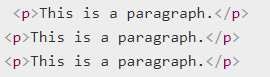
### 数值

你会在很多地方看到CSS单位中使用到数值。这一部分将会讨论数值的最常用类别。

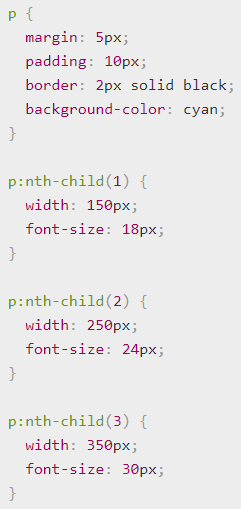
#### 长度和尺寸

在CSS布局、排版等等的所有时候，你都会使用到长度/尺寸单位。

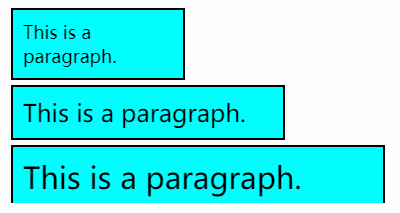
**HTML**



**CSS**



**结果**



在这个例子中我们做了以下事情：

* 分别将每个段落的 margin，padding 和 border-width 设置为5 pixels, 10 pixels 和 2 pixels。一个单独的margin/padding值表示所有的4个面都被设置成同样的值。边框也被设置成了 border 的缩写值。
* 为三个不同的段落设置越来越大的宽度（width ）像素值，也就是意味着越往下盒子越大。
* 为三个不同的段落设置越来越大字号（ font-size）像素值，也就是意味着越往下文本越大。font-size代表字体/字形的高度。

像素 (px) 是一种绝对单位（absolute units）， 因为无论其他相关的设置怎么变化，像素指定的值是不会变化的。其他的绝对单位如下：

* mm, cm, in: 毫米（Millimeters），厘米（centimeters），英寸（inches）
* pt, pc: 点（Points (1/72 of an inch)）， 十二点活字（ picas (12 points.)）

除了px之外，你很可能都不怎么使用其他的单位。

也有相对单位，他们是相对于当前元素的字号（ font-size ）或者视口（viewport ）尺寸。

* em:1em与当前元素的字体大小相同（更具体地说，一个大写字母M的宽度）。CSS样式被应用之前，浏览器给网页设置的默认基础字体大小是16像素，这意味着对一个元素来说1em的计算值默认为16像素。但是要小心—em单位是会继承父元素的字体大小，所以如果在父元素上设置了不同的字体大小，em的像素值就会变得复杂。现在不要过于担心这个问题，我们将在后面的文章和模块中更详细地介绍继承和字体大小设置。em是Web开发中最常用的相对单位。
* ex, ch: 分别是小写x的高度和数字0的宽度。这些并不像em那样被普遍使用或很好地被支持。
* rem: REM（root em）和em以同样的方式工作，但它总是等于默认基础字体大小的尺寸；继承的字体大小将不起作用，所以这听起来像一个比em更好的选择，虽然在旧版本的IE上不被支持（查看关于跨浏览器支持 Debugging CSS.)
* vw, vh: 分别是视口宽度的1/100和视口高度的1/100，其次，它不像rem那样被广泛支持。

使用相对单位是非常有用的-你可以相对于你的字体或视口大小来调整HTML元素的大小，这意味着，假设整个网站上的文本大小被视力障碍用户调整为原来的两倍，而网站的布局仍将保持正确。

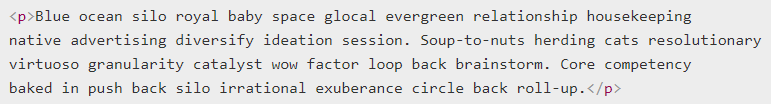
#### 无单位的值

在CSS中，你有时会遇到一些无单位的数值——这并不总是意味着错误，在某些情况下，使用无单位的数值是完全允许的。例如，如果你想让一个元素完全去除外边框和内边框，你可以只使用无单位的0——因为0就是0，不管单位是什么！

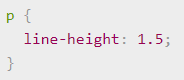


**无单位的行高**

另一个例子是 line-height，设置元素中每行文本的高度。你可以使用单位设置特定的行的高度，但使用一个无单位的值往往更容易，它就像一个简单的乘法因子。例如，使用下面的HTML：



CSS



这里font-size的值为16px; 行高为font-size值的1.5倍，也就是24px。

#### 动画的数值

CSS动画能够让页面上的HTML元素动起来。我们来看一个例子，当我们把鼠标浮动到一个段落上的时候，它能够旋转起来。这个例子的HTML代码很简单：



**CSS**



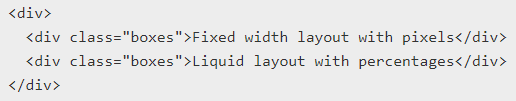
这里你可可以看到一些我们我们之前没有明确提到的有趣单位 (<angle>s、 <time>s、 <timing-function>s、 <string>s...)，但是我们感兴趣的是这一行 animation-iteration-count: 5; ——此行控制着动画启动（这里是指光标浮动至段落上时）后会执行多少次，而且这是一个简短无单位纯数字（计算机中称之为整型）。



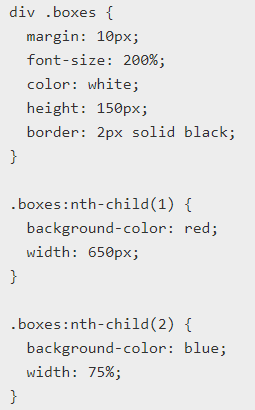
### 百分比

大部分使用特定数值指定的内容同样可以使用百分比来指定。这使的我们可以创建那些，例如，其宽度总是会被调整到其父容器宽度一定百分比的盒子。反观那些宽度被设置为某个固定单位值（如px或em）的盒子，它们总是保持固定的尺寸，即使它们父容器的宽度发生变化。

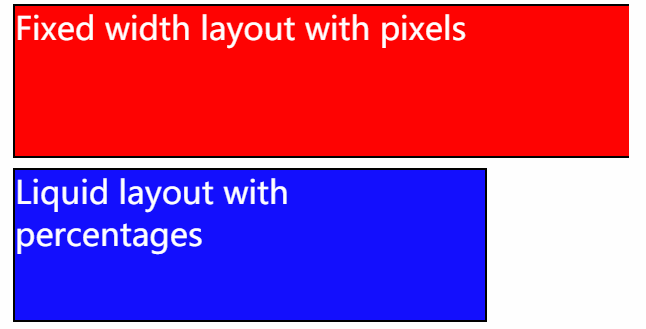
首先，使用HTML标记创建两个相似的盒子：



然后是一些CSS来装饰这些盒子：



这给了我们以下结果：



在这里，我们给两个div设置了一些 margin, height, font-size, border and color 。 然后我们给第一个div和第二个div不同的 background-color，所以我们可以很容易地区分它们。 我们还将第一个div的 width 设置为650px，将第二个div的宽度设置为75％。 这样做的效果是，第一个div始终具有相同的宽度，即使视口大小被调整（当视口变得比屏幕更窄时，它将从屏幕上消失），而第二个div的宽度随着视口（viewport ）的变化而变化，使其始终保持其父元素宽度的75％。 在这个例子中，div的父元素是<body>元素，默认情况下宽度是视口宽度的100％。

我们也可以设置font-size的百分比为200%。它将和你预计的工作方式有一点不同，但这是讲得通的，它的新大小是相对于父容器的字体大小的，就像em一样。在这种情况下，父容器的字体大小为16px——页面的默认值，所以计算的值为16px的200%，即32px。这和em的风格确实很类似——200%基本上和2em一样。

这两种不同的框布局类型通常被称为动态（流体）布局（跟随浏览器视口大小的变化）和固定宽度布局（不管怎样都保持不变），两种布局方式有着不同的应用场景：

* 可以使用动态布局来确保标准文档始终适合于屏幕，并且可以在不同大小的移动设备屏幕上表现良好。
* 一个固定宽度的布局可以用来保持在线地图的大小相同；浏览器视口可以在地图上滚动，只在一个时间内查看一定数量的地图。您可以立即看到的量取决于您的视口有多大。

### 颜色

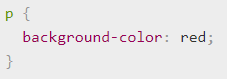
CSS中指定颜色的方法有很多，其中一些是最近实现的。CSS中到处都可以使用相同的颜色值，无论是指定文本颜色、背景颜色，还是其他任何颜色。

现代计算机中可用的标准颜色系统是24位，通过不同的红、绿、蓝通道，每个通道有256种不同的值，从而显示出大约1670万种不同的颜色。 (256 x 256 x 256 = 16,777,216.)

#### 关键词

CSS中最简单、最古老的颜色类型就是颜色关键词。这些都是代表特定颜色值的特定字符串。例如，下面的代码:

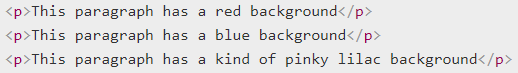


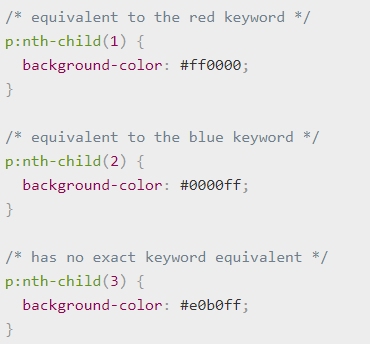


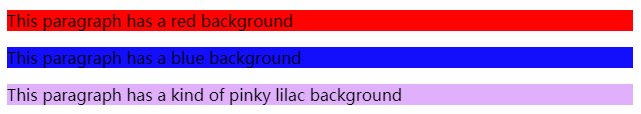


#### 十六进制值

下一个常用的颜色系统是十六进制颜色或十六进制代码。每个颜色包括一个哈希/磅符号（#）和其后面紧跟的六个十六进制数，其中每个十六进制数可以是0和F之间的一个值（一共16个），0123456789abcdef。每对十六进制数代表一个通道（红色、绿色或者蓝色）允许我们指定256个可用值。 (16 x 16 = 256.)



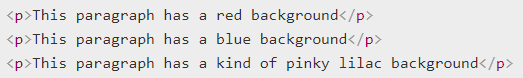


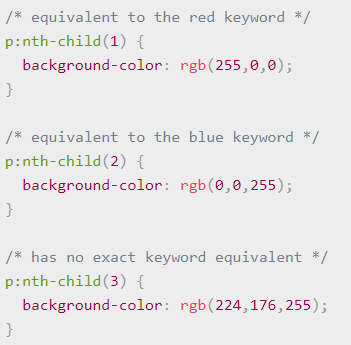


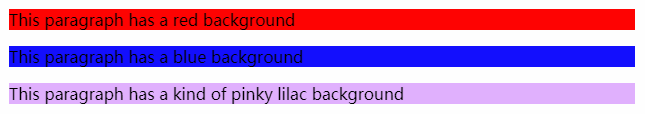
这些值比较复杂，不太容易理解，但是它们比关键字更灵活——您可以使用十六进制值来表示您想要在颜色方案中使用的任何颜色。

#### RGB

一个RGB值是一个函数——rgb() ——给定的三个参数，分别表示红色，绿色和蓝色通道的颜色值，这与十六进制值的工作方式大致相同。区别在于，RGB中每个通道不是由两个十六进制数字表示的，而是由0到255之间的十进制数表示的。



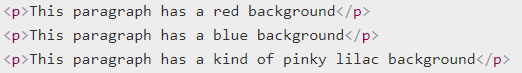


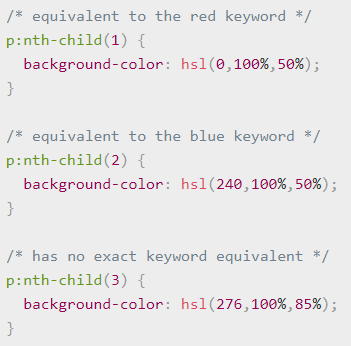


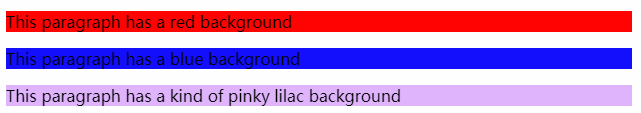
#### HSL

支持度比RGB稍微差一点的是HSL（旧版本的IE浏览器不支持），这是在众多开发者对其产生兴趣之后才实现的——hsl()函数接受三个表示色调、饱和度以及明度的参数，使用与上述三种不同的方式来区分大约1670万种颜色：

* 色调：颜色的底色调。这个值在0到360之间，表示色轮的角度。
* 饱和度：饱和度是多少？这需要一个从0-100%的值，其中0是没有颜色（它将显示为灰色），100%是全彩色饱和度。
* 明度：颜色有多亮或明亮？这需要一个从0-100%的值，其中0是无光（它会出现全黑的），100%是充满光的（它会出现全白）。





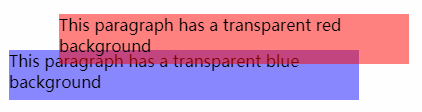


#### RGBA 和 HSLA

RGB和HSL都有相应的模式——RGBA和HSLA——不仅允许您设置想要显示的颜色,还有此颜色的透明度（ transparency ）。它们和与之相应的函数采用同样的参数,再加上第四个范围在0-1的值——设置透明度,或者说alpha通道。0是完全透明的,1是完全不透明的。





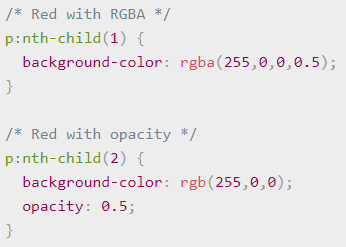


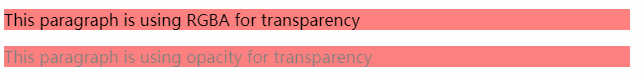
透明色对于产生更丰富的视觉效果非常有用——例如混合的背景,半透明的UI元素等等。

**不透明度（Opacity）**

还有另一种方法来指定透明度，通过CSS——opacity 属性。与设置某个特定颜色的透明度相比，这会设置所有选定元素以及它们的孩子节点的不透明度。为了看出他们的区别，我们来研究下面这个例子：







注意区别——第一个盒子使用RGBA颜色，只有一个半透明的背景。而一切在第二个盒子里的都是透明的，包括文本。您要仔细思考使用两者的时机——例如，当您想创建一个覆盖图片的标题，图片能透过标题显示，且标题的文本显示不受影响，此时应该使用RGBA修改标题背景色的透明度；另一方面，当您想要创建一个动画效果，让整个UI元素从完全隐藏到可见，此时应该使用不透明度（Opacity）。

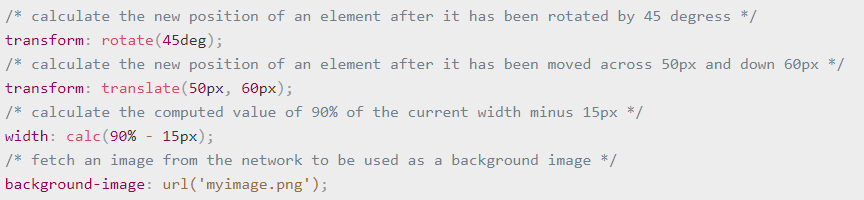
### 函数

在程序中， functions是代码中的可重复使用的部分，它可以多次运行，以便使开发人员和计算机以最小的代价完成重复的任务。函数通常存在于JavaScript，Python，C++等语言，但它也作为属性值存在于CSS中。我们已经在Colors部分中看到函数了，例如 rgb()， hsl()：



这些函数用来计算使用什么颜色。

但是你在其他地方也会看到函数——每当你看到一个名字后跟着括号,括号里包含用逗号分隔的一个或多个值,那么你所使用的就是一个函数。例如:



## 层叠和继承

在实际的工作中，我们可能还有些疑惑，当有多个选择器匹配到同一个元素上时，哪个选择器的 CSS 规则最终会应用到元素上？其实这是通过层叠机制来控制的，这也和样式继承(元素从其父元素那里获得属性值)有关.

元素的最终样式可以在多个地方定义，它们以复杂的形式相互影响。这些复杂的相互作用使CSS变得非常强大，但也使其非常难于调试和理解。

### 层叠（选择器的优先级）

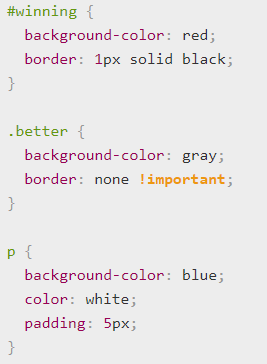
CSS 是 Cascading Style Sheets 的缩写，这暗示层叠（cascade）的概念是很重要的。在最基本的层面上，它表明CSS规则的顺序很重要，但它比那更复杂。什么选择器在层叠中胜出取决于三个因素（这些都是按重量级顺序排列的——前面的的一种会否决后一种）：

* 重要性（Importance）
* 专用性（Specificity）
* 源代码次序（Source order）

#### 重要性

在CSS中，有一个特别的语法可以让一条规则总是优先于其他规则：!important。把它加在属性值的后面可以使这条声明有无比强大的力量。







让我们一起来看看发生了什么。

* 你可以看到第三条规则 color 和 padding 被运用了, 但 background-color没有，为什么？实际上，这三种情况都应该应用，因为在源顺序后面的规则通常会覆盖较早的规则。
* 然而, 在前面的规则被运用了,因为 IDs/class 选择器优先于element选择器。
* 这两个元素都有 class并带有 better属性, 但是第二个元素有 id 值为winning 。 因为比起class而言id专用性更高(在一个页面上id是唯一的, 但很多元素可以拥有相同的class — ID 选择器在它们的目标中是非常优先的)，红色背景色和1pixel的黑色边框都应应用于第二元素，第一个元素获得灰色背景色，没有边框，如类所指定。
* 第二个元素获得红色背景色，但没有边框。为什么？因为 !important 在第二条规则中的声明——在 border: none之后写入它意味着尽管id具有更高的优先性，该声明也将优先于前面规则中的边界值声明。

知道 !important存在是很有用的，这样当你在别人的代码中遇到它时，你就知道它是什么了。但是！我们建议你千万不要使用它，除非你绝对必须使用它。您可能不得不使用它的一种情况是，当您在CMS中工作时，您不能编辑核心的CSS模块，并且您确实想要重写一种不能以其他方式覆盖的样式。 但是，如果你能避免的话，不要使用它。由于 !important 改变了层叠正常工作的方式，因此调试CSS问题，尤其是在大型样式表中，会变得非常困难。

要注意一个CSS声明的重要性取决于它被指定在什么样式表内——用户可以设置自定义样式表覆盖开发者的样式，例如用户可能有视力障碍，想设置字体大小对所有网页的访问是双倍的正常大小，以便更容易阅读。

相互冲突的声明将按以下顺序适用，后一种将覆盖先前的声明：

* 在用户代理样式表的声明 (例如：浏览器在没有其他声明的默认样式).
* 用户样式表中的普通声明（由用户设置的自定义样式）。
* 作者样式表中的普通声明（这是我们设置的样式，Web开发人员）。
* 作者样式表中的重要声明
* 用户样式表中的重要声明

Web开发者的样式表覆盖用户的样式表是合理的，所以设计可以保持预期，但是有时候用户有很好的理由来重写web开发人员样式，如上所述，这可以通过在用户的规则中使用!important。

#### 专用性

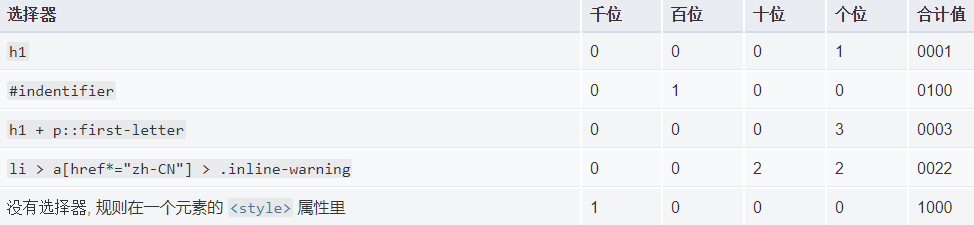
专用性基本上是衡量选择器的具体程度的一种方法——它能匹配多少元素。如上面所示的示例所示，元素选择器具有很低的专用性。类选择器具有更高的专用性，所以将战胜元素选择器。ID选择器有甚至更高的专用性, 所以将战胜类选择器. 战胜ID选择器的唯一方法是使用 !important。

一个选择器具有的专用性的量是用四种不同的值（或组件）来衡量的，它们可以被认为是千位，百位，十位和个位——在四个列中的四个简单数字：

* 千位：如果声明是在style 属性中该列加1分（这样的声明没有选择器，所以它们的专用性总是1000。）否则为0。
* 百位：在整个选择器中每包含一个ID选择器就在该列中加1分。
* 十位：在整个选择器中每包含一个类选择器、属性选择器、或者伪类就在该列中加1分。
* 个位：在整个选择器中每包含一个元素选择器或伪元素就在该列中加1分。

注意: 通用选择器 (\*), 复合选择器 (+, >, ~, ' ') 和否定伪类 (:not) 在专用性中无影响。

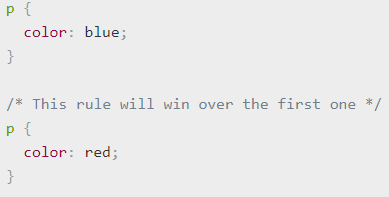
下表显示了几个示例。试着通过这些，并确保你理解他们为什么具有我们给予他们的专用性。



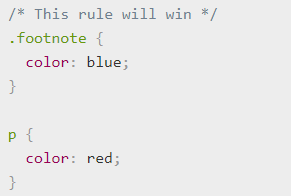
注意: 如果多个选择器具有相同的重要性和专用性，则选择哪一个选择器取决于资源顺序。

#### 源代码次序

如果多个相互竞争的选择器具有相同的重要性和专用性，那么第三个因素将帮助决定哪一个规则获胜——后面的规则将战胜先前的规则。



而在这个例子中的第一个规则将获胜，因为专用性高于源代码顺序：

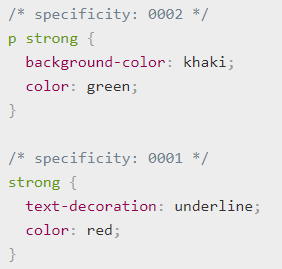


#### 关于规则混合的一点注记

在考虑所有这些层叠理论和什么样式优先于其他样式被应用时，你应该记住的一件事是，所有这些都发生在属性级别上——属性覆盖其他属性，但你不会让整个规则凌驾于其他规则之上。

当多个CSS规则匹配相同的元素时，它们都被应用到该元素中。只有在这之后，任何相互冲突的属性才会被评估，以确定哪种风格会战胜其他类型。







在这个例子中，因为专用性的关系，第一条规则中的color属性覆盖掉了第二条中的颜色值。但是，对于第一条中的 background-color和第二条中的text-decoration 的属性都在strong元素之中得到了体现。你也注意到了这个元素之中的字体是加粗的：这是浏览器默认的样式规则。

### 继承

CSS继承指的是应用于某个元素的一些属性值将由该元素的子元素继承，而有些则不会。

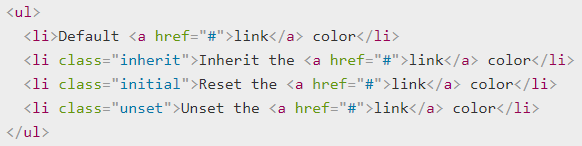
* 例如，对 font-family 和 color 进行继承是有意义的，因为这使得您可以很容易地设置一个站点范围的基本字体，方法是应用一个字体到 <html> 元素；然后，您可以在需要的地方覆盖单个元素的字体。如果要在每个元素上分别设置基本字体，那就太麻烦了。
* 再如，让 margin，padding，border 和 background-image 不被继承是有意义的。想象一下，如果在容器元素上设置这些属性并让它们由每个子元素继承，那么样式/布局会发生混乱，然后必须在每个单独的元素上取消它们！

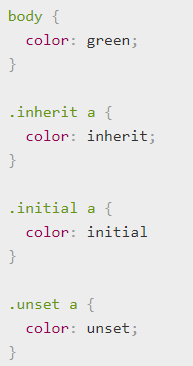
#### 控制继承

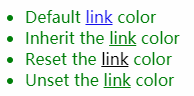
CSS为处理继承提供了四种特殊的通用属性值：

* inherit： 该值将应用到选定元素的属性值设置为与其父元素一样。
* initial ：该值将应用到选定元素的属性值设置为与浏览器默认样式表中该元素设置的值一样。如果浏览器默认样式表中没有设置值，并且该属性是自然继承的，那么该属性值就被设置为 inherit。
* unset ：该值将属性重置为其自然值，即如果属性是自然继承的，那么它就表现得像 inherit，否则就是表现得像 initial。
* revert ：如果当前的节点没有应用任何样式，则将该属性恢复到它所拥有的值。换句话说，属性值被设置成自定义样式所定义的属性（如果被设置）， 否则属性值被设置成用户代理的默认样式。

inherit 值是最有趣的——它允许我们显式地让一个元素从其父类继承一个属性值。







让我们解释这里发生了什么：

* 我们首先设置<body> 的color为绿色。
* 由于color属性是自然继承的，所有的body子元素都会有相同的绿色。需要注意的是默认情况下浏览器设置链接的颜色为蓝色，而不是自然继承color属性，因此在我们列表中的第一个链接是蓝色的。
* 第二个规则设置一个类 inherit 的元素内的链接，并从父类继承它的颜色。在这种情况下, 意思是说链接继承了父元素<li>的颜色，默认情况下<li>的颜色来自于它的父元素 <ul> , 最后<ul> 继承自 <body>元素，而<body>的color 根据第一条规则设置成了绿色。
* 第三个规则选择了在元素上使用类 initial 的任意链接然后设置他们的颜色为 initial 。通常， initial 的值被浏览器设置成了黑色，因此该链接被设置成了黑色。
* 最后一个规则选择了在元素上使用类 unset 的所有链接然后设置它们的颜色为 unset ——即我们不设置值。因为color属性是一个自然继承的属性，它实际上就像把值设置成 inherit 一样。结果是，该链接被设置成了与body一样的颜色——绿色。

#### 重新设置所有的属性值

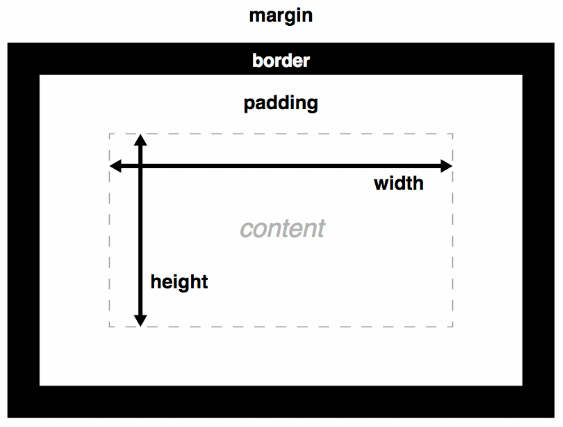
CSS速写属性 all 能够被应用到每一个继承属性，一次性应用所有的继承属性。这里的值可以是继承属性里的任何一个 (inherit, initial, unset, or revert)。对于撤销对样式的修改，这是非常方便的一种方式。然后你就可以在开始新的修改之前，返回到一个已知的开始点。

## 框模型

CSS框模型（译者注：也被称为“盒模型”）是网页布局的基础 ——每个元素被表示为一个矩形的方框，框的内容、内边距、边界和外边距像洋葱的膜那样，一层包着一层构建起来。浏览器渲染网页布局时，它会算出每个框的内容要用什么样式，周围的洋葱层有多大，以及框相对于其它框放在哪里。在理解如何创建 CSS 布局之前，你需要理解框模型。

### 框属性

文档的每个元素被构造成文档布局内的一个矩形框，框每层的大小都可以使用一些特定的CSS属性调整。相关属性如下:



**width 和 height**

width 和 height 设置内容框（content box）的宽度和高度。内容框是框内容显示的区域——包括框内的文本内容，以及表示嵌套子元素的其它框。

注意: 还有其他属性可以更巧妙地处理内容的大小——设置大小约束而不是绝对的大小。这些属性包括min-width、max-width、min-height 和 max-height。

**padding**

padding 表示一个 CSS 框的内边距 ——这一层位于内容框的外边缘与边界的内边缘之间。该层的大小可以通过简写属性padding 一次设置所有四个边，或用 padding-top、padding-right、padding-bottom 和 padding-left 属性一次设置一个边。

**border**

CSS 框的边界（border）是一个分隔层，位于内边距的外边缘以及外边距的内边缘之间。边界的默认大小为0——从而让它不可见——不过我们可以设置边界的厚度、风格和颜色让它出现。 border 简写属性可以让我们一次设置所有四个边，例如 border: 1px solid black; 但这个简写可以被各种普通书写的更详细的属性所覆盖：

* border-top, border-right, border-bottom, border-left: 分别设置某一边的边界厚度／风格／颜色。
* border-width, border-style, border-color: 分别仅设置边界的厚度／风格／颜色，并应用到全部四边边界。
* 你也可以单独设置某一个边的三个不同属性，如 border-top-width, border-top-style, border-top-color，等。

**margin**

外边距（margin）代表 CSS 框周围的外部区域，称为外边距，它在布局中推开其它 CSS 框。其表现与 padding 很相似；简写属性为 margin，单个属性分别为 margin-top、margin-right、margin-bottom 和 margin-left。

注意: 外边距有一个特别的行为被称作外边距塌陷（margin collapsing）：当两个框彼此接触时，它们的间距将取两个相邻外边界的最大值，而非两者的总和。

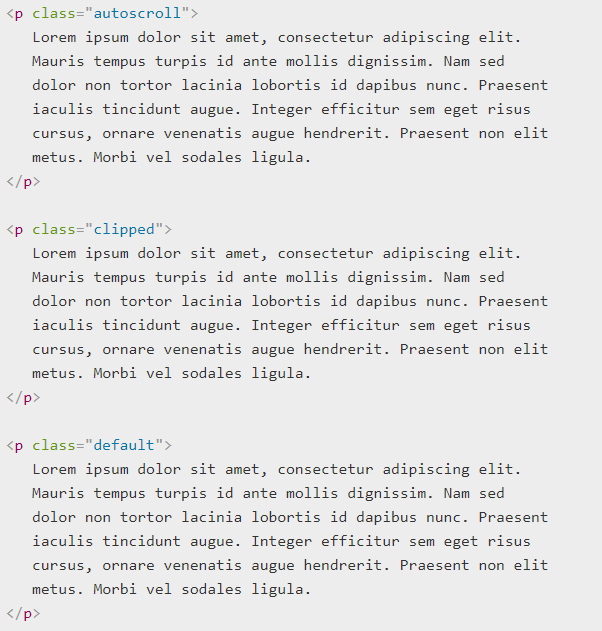
### 高级的框操作

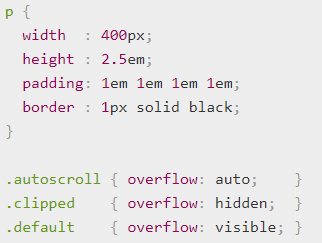
在设置框的width, height, border, padding 及margin之外， 还有一些其他的属性可以改变它们的行为。

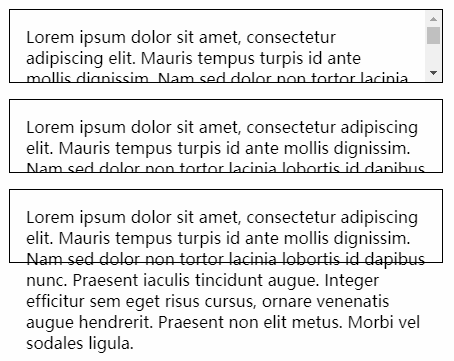
#### 溢流

当你使用绝对的值设置了一个框的大小（如，固定像素的宽/高），允许的大小可能不适合放置内容，这种情况下内容会从盒子溢流。我们使用overflow属性来控制这种情况的发生。它有一些可能的值，但是最常用的是：

* auto: 当内容过多，溢流的内容被隐藏，然后出现滚动条来让我们滚动查看所有的内容。
* hidden: 当内容过多，溢流的内容被隐藏。
* visible: 当内容过多，溢流的内容被显示在盒子的外边（这个是默认的行为）

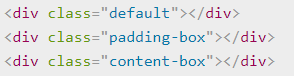






#### 背景裁剪（Background clip）

框的背景是由颜色和图片组成的，它们堆叠在一起（background-color, background-image）。 它们被应用到一个盒子里，然后被画在盒子的下面。默认情况下，背景延伸到了边界外沿。这通常是OK的，但是在一些情况下比较讨厌（假使你有一个平铺的背景图，你只想要它延伸到内容的边沿会怎么做？），该行为可以通过设置盒子的background-clip属性来调整。







#### 轮廓(Outline)

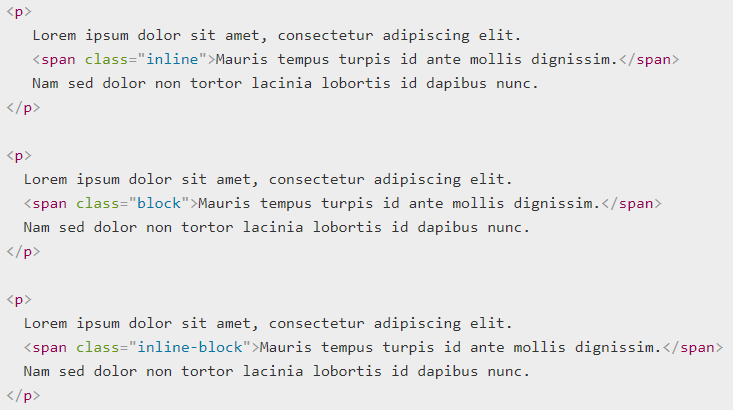
一个框的 outline 是一个看起来像是边界但又不属于框模型的东西。它的行为和边界差不多，但是并不改变框的尺寸（更准确的说，轮廓被勾画于在框边界之外，外边距区域之内）

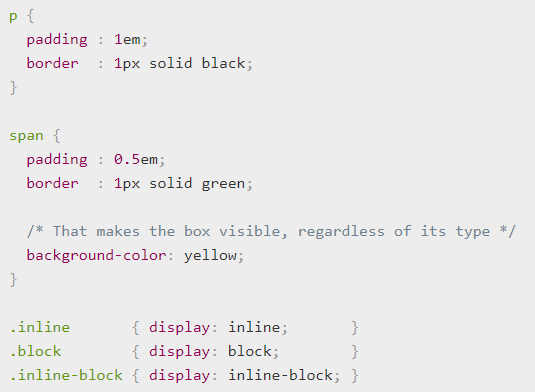
Note: 使用 outline 属性的时候要注意，它一般只在需要可用性的一些情况才被使用，例如在一些用户点击它的时候使用 outline 来表示高亮、可用，如果你要使用 outline，请确保不要因为它看起来像链接的高亮让用户迷惑。

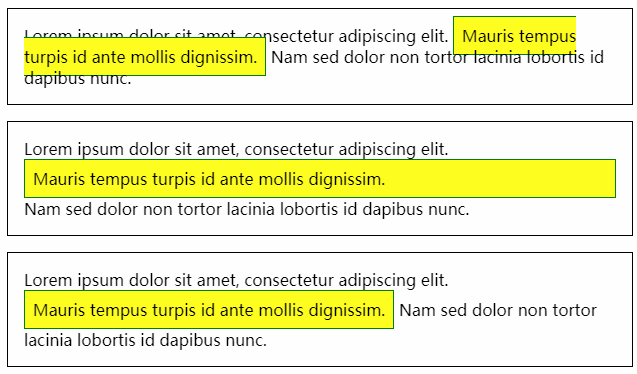
### CSS 框类型

之前我们说的所有对于框的应用都表示的是块级元素的，然而，CSS还有一些表现不同的其他框类型。我们可以通过display属性来设定元素的框类型。display属性有很多的属性值。在本篇文章，我们将关注三个最常见的类型：block, inline, and inline-block。

* 块框（ block box）是定义为堆放在其他框上的框（例如：其内容会独占一行），而且可以设置它的宽高，之前所有对于框模型的应用适用于块框 （ block box）
* 行内框（ inline box）与块框是相反的，它随着文档的文字流动（例如：它将会和周围的文字和其他行内元素出现在同一行，而且它的内容会像一段中的文字一样随着文字部分的流动而打乱），对行内盒设置宽高无效，设置padding, margin 和 border都会更新周围文字的位置，但是对于周围的的块框（ block box）不会有影响。
* 行内块状框（inline-block box） 像是上述两种的集合：它不会重新另起一行但会像行内框（ inline box）一样随着周围文字而流动，而且他能够设置宽高，并且像块框一样保持了其块特性的完整性，它不会在段落行中断开。（在下面的示例中，行内块状框会放在第二行文本上，因为第一行没有足够的空间，并且不会突破两行。然而，如果没有足够的空间，行内框会在多条线上断裂，而它会失去一个框的形状。）







### 外边距合并

块级元素的上外边距和下外边距有时会合并（或折叠）为一个外边距，其大小取其中的最大者，这种行为称为外边距折叠（margin collapsing），有时也翻译为外边距合并。注意浮动元素和绝对定位元素的外边距不会折叠。

* 相邻元素之间

毗邻的两个元素之间的外边距会折叠（除非后一个元素需要清除之前的浮动）。

* 父元素与其第一个或最后一个子元素之间

如果在父元素与其第一个子元素之间不存在边框、内边距、行内内容，也没有创建块格式化上下文、或者清除浮动将两者的 margin-top 分开；或者在父元素与其最后一个子元素之间不存在边框、内边距、行内内容、height、min-height、max-height将两者的 margin-bottom 分开，那么这两对外边距之间会产生折叠。此时子元素的外边距会“溢出”到父元素的外面。

* 空的块级元素

如果一个块级元素中不包含任何内容，并且在其 margin-top 与 margin-bottom 之间没有边框、内边距、行内内容、height、min-height 将两者分开，则该元素的上下外边距会折叠。

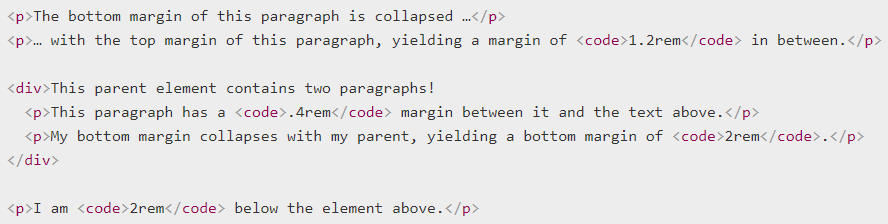
一些需要注意的地方：

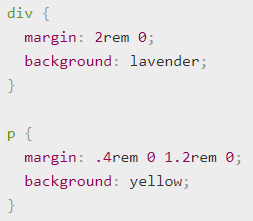
上述情况的组合会产生更复杂的外边距折叠。

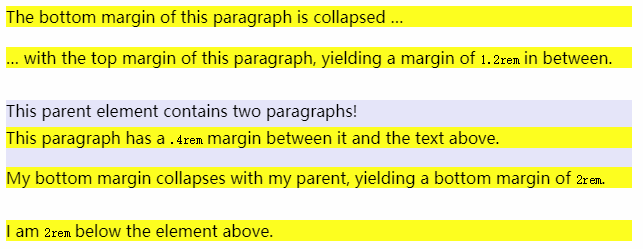
即使某一外边距为0，这些规则仍然适用。因此就算父元素的外边距是0，第一个或最后一个子元素的外边距仍然会“溢出”到父元素的外面。

如果参与折叠的外边距中包含负值，折叠后的外边距的值为最大的正边距与最小的负边距（即绝对值最大的负边距）的和。

如果所有参与折叠的外边距都为负，折叠后的外边距的值为最小的负边距的值。这一规则适用于相邻元素和嵌套元素。







## 层叠上下文

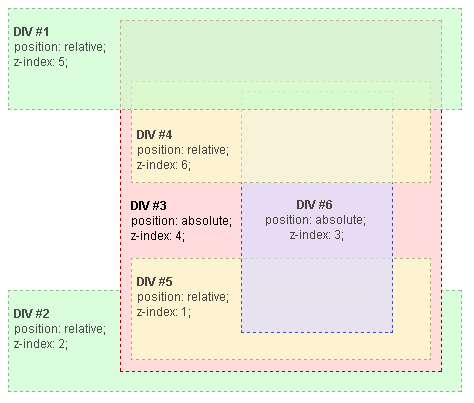
层叠上下文是HTML元素的三维概念，这些HTML元素在一条假想的相对于面向（电脑屏幕的）视窗或者网页的用户的z轴上延伸，HTML元素依据其自身属性按照优先级顺序占用层叠上下文的空间。

在之前的增加 z-index 的例子中, 某些 DIV 的渲染顺序是由 z-index 的值影响的。这是因为这些 DIV 具有 使他们形成一个层叠上下文 的特殊属性。

文档中的层叠上下文由满足以下任意一个条件的元素形成：

* 根元素 (HTML),
* z-index 值不为 "auto"的 绝对/相对定位，
* 一个 z-index 值不为 "auto"的 flex 项目 (flex item)，即：父元素 display: flex|inline-flex，
* opacity 属性值小于 1 的元素，
* transform 属性值不为 "none"的元素，
* mix-blend-mode 属性值不为 "normal"的元素，
* filter值不为“none”的元素，
* perspective值不为“none”的元素，
* isolation 属性被设置为 "isolate"的元素，
* position: fixed
* 在 will-change 中指定了任意 CSS 属性，即便你没有直接指定这些属性的值
* -webkit-overflow-scrolling 属性被设置 "touch"的元素

在层叠上下文中，其子元素同样也按照上面解释的规则进行层叠。 特别值得一提的是，其子元素的 z-index 值只在父级层叠上下文中有意义。子级层叠上下文被自动视为父级层叠上下文的一个独立单元。



## 块格式化上下文

块格式化上下文（Block Formatting Context，BFC） 是Web页面的可视化CSS渲染的一部分，是块盒子的布局过程发生的区域，也是浮动元素与其他元素交互的区域。

下列方式会创建块格式化上下文：

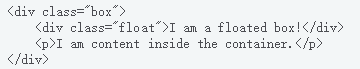
* 根元素或包含根元素的元素
* 浮动元素（元素的 float 不是 none）
* 绝对定位元素（元素的 position 为 absolute 或 fixed）
* 行内块元素（元素的 display 为 inline-block）
* 表格单元格（元素的 display为 table-cell，HTML表格单元格默认为该值）
* 表格标题（元素的 display 为 table-caption，HTML表格标题默认为该值）
* 匿名表格单元格元素（元素的 display为 table、table-row、 table-row-group、table-header-group、table-footer-group（分别是HTML table、row、tbody、thead、tfoot的默认属性）或 inline-table）
* overflow 值不为 visible 的块元素
* display 值为 flow-root 的元素
* contain 值为 layout、content或 strict 的元素
* 弹性元素（display为 flex 或 inline-flex元素的直接子元素）
* 网格元素（display为 grid 或 inline-grid 元素的直接子元素）
* 多列容器（元素的 column-count 或 column-width 不为 auto，包括 column-count 为 1）
* column-span 为 all 的元素始终会创建一个新的BFC，即使该元素没有包裹在一个多列容器中（标准变更，Chrome bug）。

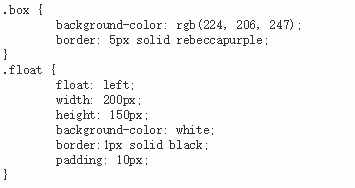
块格式化上下文包含创建它的元素内部的所有内容.

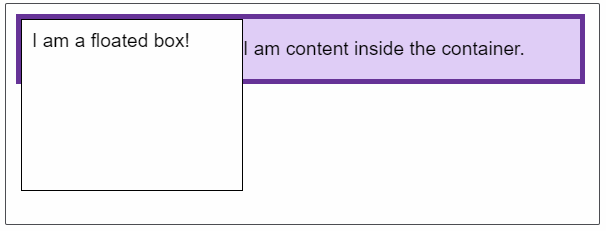
块格式化上下文对浮动定位（参见 float）与清除浮动（参见 clear）都很重要。浮动定位和清除浮动时只会应用于同一个BFC内的元素。浮动不会影响其它BFC中元素的布局，而清除浮动只能清除同一BFC中在它前面的元素的浮动。外边距折叠（Margin collapsing）也只会发生在属于同一BFC的块级元素之间。

**让浮动内容和周围的内容等高**

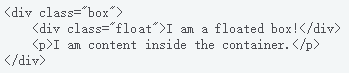
在下面的例子中，我们让 <div> 元素浮动，并给它一个边框效果。<div>里的内容现在已经在浮动元素周围浮动起来了。由于浮动的元素比它旁边的元素高，所以<div>的边框穿出了浮动。正如我们在这篇文章中 guide to in-flow and out of flow elements 解释的，浮动脱离了文档流，所以 <div> 的 background 和 border 仅仅包含了内容，不包含整个浮动。

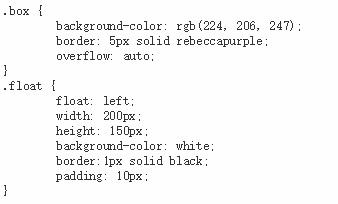


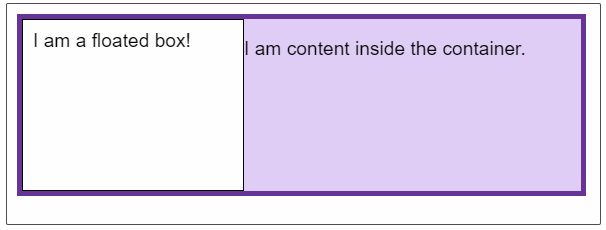




创建一个会包含这个浮动的BFC，通常的做法是设置父元素 overflow: auto 或者设置其他的非默认的 overflow: visible 的值。







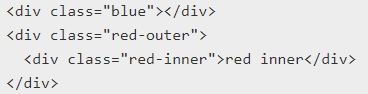
设置 overflow: auto 创建一个新的BFC来包含这个浮动。我们的 <div> 元素现在变成布局中的迷你布局。任何子元素都会被包含进去。

使用 overflow 来创建一个新的BFC，是因为 overflow 属性告诉浏览器你想要怎样处理溢出的内容。当你使用这个属性只是为了创建BFC的时候，你可能会发现一些不想要的问题，比如滚动条或者一些剪切的阴影，需要注意。另外，对于后续的开发，可能不是很清楚当时为什么使用overflow。所以你最好添加一些注释来解释为什么这样做。

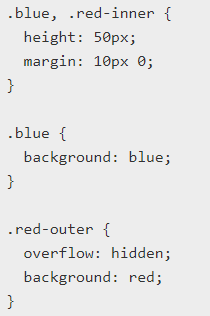
**外边距塌陷**

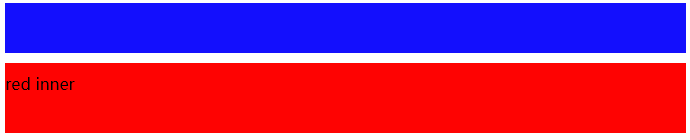
创建新的BFC避免两个相邻 <div> 之间的 外边距合并 问题

**HTML**



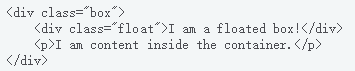
**CSS**

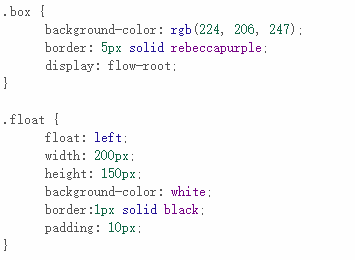


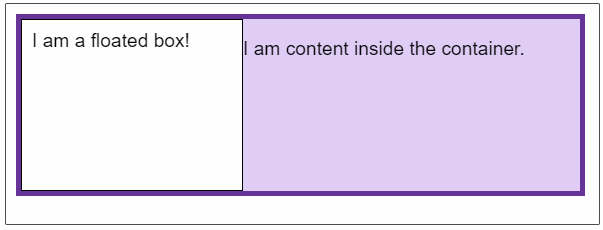


**使用display: flow-root**

一个新的 display 属性的值，它可以创建无副作用的BFC。在父级块中使用 display: flow-root 可以创建新的BFC。







给 <div> display: flow-root 属性后，<div> 中的所有内容都会参与BFC，浮动的内容不会从底部溢出。

flow-root的值名是有意义的，当您理解您正在创建一个类似于根元素(浏览器中的<html>元素)的东西时，它是如何为其中的流布局创建一个新的上下文的。

## 调试 CSS

### CSS 和 调试

就像HTML一样，CSS是宽容的（在继续阅读前阅读宽容代码。）在CSS的情况下，如果一个声明是无效的（包含语法错误，或者浏览器不支持该特性），浏览器完全忽略了它，然后转向它找到的下一个。

如果选择器是无效的，那么它就不会选择任何东西，而整个规则也不会再做任何事情，浏览器只会继续执行下一个规则。

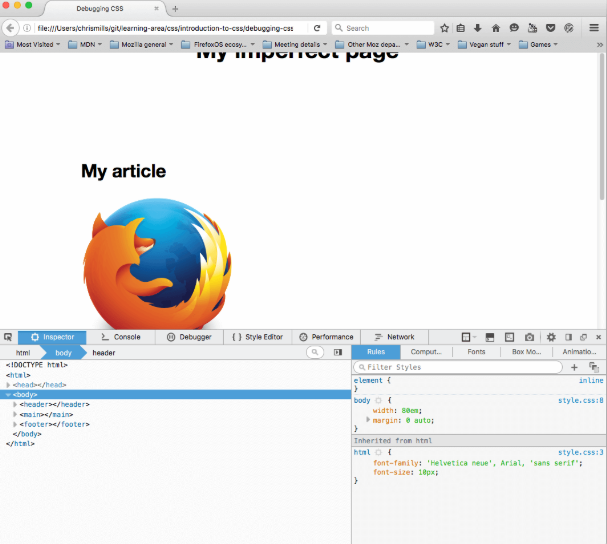
这在很多情况下都是很好的，在大多数情况下，你的内容会显示给你的用户，即使它不是很正确。但是，当您试图调试这个问题时，这并不是很有帮助，您甚至没有得到任何错误消息来帮助您找到它。当你的用户无法看到内容的时候，这就更令人头疼了——也许关键的风格没有被应用，导致布局出现了严重的错误？

幸运的是，有一些工具可以帮助您。让我们看看这些。

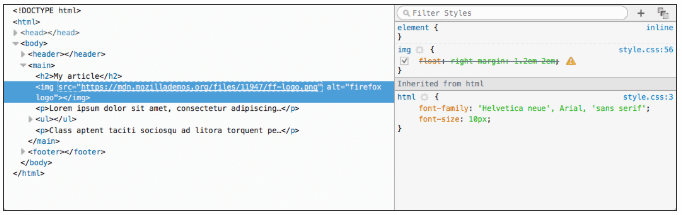
#### 检查 DOM 和 CSS

现在，所有的web浏览器都提供了帮助您检查和理解web页面的开发工具。在他们提供的各种工具中，有两种可以在所有浏览器中使用:DOM检查器和CSS编辑器，它们都可以在Firefox的page inspector tool中找到。我们已经在调试HTML中查看了DOM检查器，以及如何使用它来检查HTML。在这里，我们将看到这个和CSS编辑器，以及如何将它们一起使用来调试CSS问题。

在火狐中你可以使用 Cmd/Ctrl + I 打开页面检查器（或者通过在菜单中选择Tools > Web Developer > Inspector），这将会给你一组工具在浏览器底部的窗口中打开，就像这样：



如果单击左边的DOM检查器中的一个元素，右边的CSS编辑器将更新显示应用到该元素的所有CSS规则。这是非常有用的，特别是当任何无效的属性出现在它们的一行和旁边的一个警告符号的时候。这将会很方便！



注意：每个属性旁边还有一个复选框，可以选择checked/unchecked来启用/禁用您的CSS属性。这对于找出可能导致错误的原因也非常有用。

# CSS属性集合

此部分将目前CSS中定义的CSS属性按照功能划分进行归类，以便于使用时能快速查找定位。

## 页面布局

|  |  |
| --- | --- |
| **属性名** | **功能简述** |
| 弹性盒子 | |
| [align-content](#alignContent) | align-content 属性在弹性容器内的各项没有占用交叉轴上所有可用的空间时对齐容器内的各项（垂直）。 |
| [align-items](#alignItems) | align-items 属性定义flex子项在flex容器的当前行的侧轴（纵轴）方向上的对齐方式。 |
| [align-self](#alignSelf) | align-self 属性定义flex子项单独在侧轴（纵轴）方向上的对齐方式。 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## 动画属性

|  |  |
| --- | --- |
| **属性名** | **功能简述** |
| [animation](#animation) | 这是一个简写属性，定义动画相关属性值。 |
| [animation-name](#animationName) | animation-name 属性为 @keyframes 动画指定名称。 |
| [animation-duration](#animationDuration) | animation-duration属性定义动画完成一个周期需要多少秒或毫秒。 |
| [animation-timing-function](#animationTimingFunction) | animation-timing-function指定动画将如何完成一个周期。速度曲线定义动画从一套 CSS 样式变为另一套所用的时间。速度曲线用于使变化更为平滑。 |
| [animation-delay](#animationDelay) | animation-delay 属性定义动画什么时候开始。 |
| [animation-iteration-count](#animationIterationCoun) | animation-iteration-count属性定义动画应该播放多少次。 |
| [animation-direction](#animationDirection) | animation-direction 属性定义是否循环交替反向播放动画。 |
| [animation-fill-mode](#animationFillMode) | animation-fill-mode 属性规定当动画不播放时（当动画完成时，或当动画有一个延迟未开始播放时），要应用到元素的样式。  默认情况下，CSS 动画在第一个关键帧播放完之前不会影响元素，在最后一个关键帧完成后停止影响元素。animation-fill-mode 属性可重写该行为。 |
| [animation-play-state](#animationPlayState) | animation--play-state属性指定动画是否正在运行或已暂停。 |

## 其他属性

|  |  |
| --- | --- |
| **属性名** | **功能简述** |
| [all](#all) | all 属性用于重置所有属性，除了 unicode-bidi 和 direction。 |
| [appearance](#appearance) | appearance 属性允许您使元素看上去像标准的用户界面元素。 |
|  |  |
|  |  |

* align-content

align-content 属性在弹性容器内的各项没有占用交叉轴上所有可用的空间时对齐容器内的各项（垂直）。

提示：使用 justify-content 属性对齐主轴上的各项（水平）。

注意：容器内必须有多行的项目，该属性才能渲染出效果。

下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-align-content。

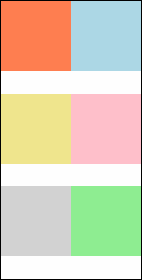


属性值：

1. stretch

默认值。元素被拉伸以适应容器。

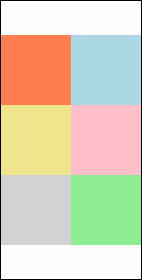
各行将会伸展以占用剩余的空间。如果剩余的空间是负数，该值等效于'flex-start'。在其它情况下，剩余空间被所有行平分，以扩大它们的侧轴尺寸。



1. center

元素位于容器的中心。

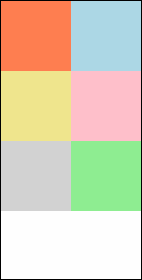
各行向弹性盒容器的中间位置堆叠。各行两两紧靠住同时在弹性盒容器中居中对齐，保持弹性盒容器的侧轴起始内容边界和第一行之间的距离与该容器的侧轴结束内容边界与第最后一行之间的距离相等。（如果剩下的空间是负数，则各行会向两个方向溢出的相等距离。）



1. flex-start

元素位于容器的开头。

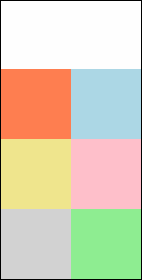
各行向弹性盒容器的起始位置堆叠。弹性盒容器中第一行的侧轴起始边界紧靠住该弹性盒容器的侧轴起始边界，之后的每一行都紧靠住前面一行。



1. flex-end

元素位于容器的结尾。

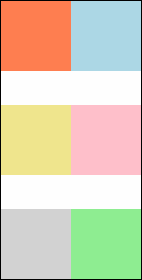
各行向弹性盒容器的结束位置堆叠。弹性盒容器中最后一行的侧轴起结束界紧靠住该弹性盒容器的侧轴结束边界，之后的每一行都紧靠住前面一行。



1. space-between

元素位于各行之间留有空白的容器内。

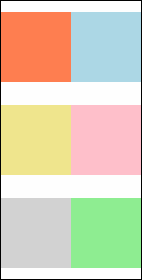
各行在弹性盒容器中平均分布。如果剩余的空间是负数或弹性盒容器中只有一行，该值等效于'flex-start'。在其它情况下，第一行的侧轴起始边界紧靠住弹性盒容器的侧轴起始内容边界，最后一行的侧轴结束边界紧靠住弹性盒容器的侧轴结束内容边界，剩余的行则按一定方式在弹性盒窗口中排列，以保持两两之间的空间相等。



1. space-around

元素位于各行之前、之间、之后都留有空白的容器内。

各行在弹性盒容器中平均分布，两端保留子元素与子元素之间间距大小的一半。如果剩余的空间是负数或弹性盒容器中只有一行，该值等效于'center'。在其它情况下，各行会按一定方式在弹性盒容器中排列，以保持两两之间的空间相等，同时第一行前面及最后一行后面的空间是其他空间的一半。



1. initial

设置该属性为它的默认值。即stretch值。

* **align-items**

align-items 属性定义flex子项在flex容器的当前行的侧轴（纵轴）方向上的对齐方式。

提示：使用每个弹性对象元素的 align-self 属性可重写 align-items 属性。

下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-align-items。



属性值：

1. stretch

默认值。元素被拉伸以适应容器。

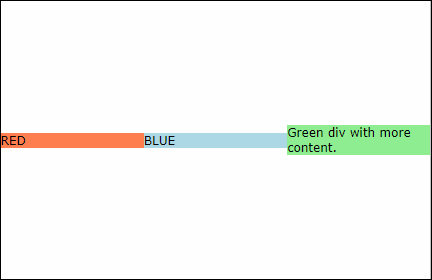
如果指定侧轴大小的属性值为'auto'，则其值会使项目的边距盒的尺寸尽可能接近所在行的尺寸，但同时会遵照'min/max-width/height'属性的限制。



1. center

元素位于容器的中心。

弹性盒子元素在该行的侧轴（纵轴）上居中放置。（如果该行的尺寸小于弹性盒子元素的尺寸，则会向两个方向溢出相同的长度）。



1. flex-start

元素位于容器的开头。

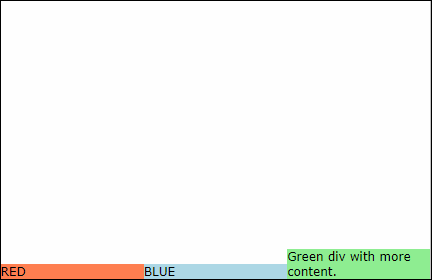
弹性盒子元素的侧轴（纵轴）起始位置的边界紧靠住该行的侧轴起始边界。



1. flex-end

元素位于容器的结尾。

弹性盒子元素的侧轴（纵轴）起始位置的边界紧靠住该行的侧轴结束边界。



1. baseline

元素位于容器的基线上。

如弹性盒子元素的行内轴与侧轴为同一条，则该值与'flex-start'等效。其它情况下，该值将参与基线对齐。



1. initial

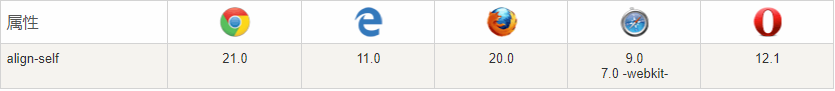
设置该属性为它的默认值。

* **align-self**

align-self 属性定义flex子项单独在侧轴（纵轴）方向上的对齐方式。

注意：align-self 属性可重写灵活容器的 align-items 属性。

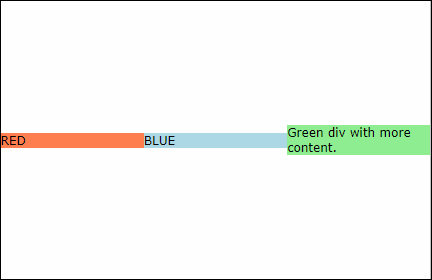
下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-align-self



属性值：

1. auto

默认值。元素继承了它的父容器的 align-items 属性。如果没有父容器则为 "stretch"。



1. stretch

元素被拉伸以适应容器。

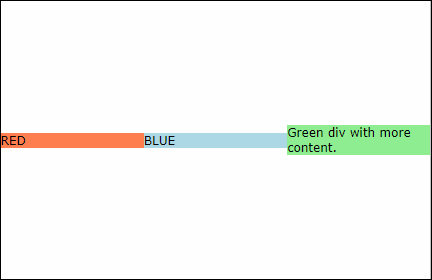
如果指定侧轴大小的属性值为'auto'，则其值会使项目的边距盒的尺寸尽可能接近所在行的尺寸，但同时会遵照'min/max-width/height'属性的限制。



1. center

元素位于容器的中心。

弹性盒子元素在该行的侧轴（纵轴）上居中放置。（如果该行的尺寸小于弹性盒子元素的尺寸，则会向两个方向溢出相同的长度）。



1. flex-start

元素位于容器的开头。

弹性盒子元素的侧轴（纵轴）起始位置的边界紧靠住该行的侧轴起始边界。



1. flex-end

元素位于容器的结尾。

弹性盒子元素的侧轴（纵轴）起始位置的边界紧靠住该行的侧轴结束边界。



1. baseline

元素位于容器的基线上。

如弹性盒子元素的行内轴与侧轴为同一条，则该值与'flex-start'等效。其它情况下，该值将参与基线对齐。



1. initial

设置该属性为它的默认值。

* **all**

all 属性用于重置所有属性，除了 unicode-bidi 和 direction。



属性值：

1. initial

修改所有元素属性或父元素的值为其初始化值

1. inherit

修改所有元素属性或父元素的值为其父元素的值

1. unset

修改所有元素属性或父元素的值为其父元素的值(如果有继承)或其初始值

* **animation**

animation属性是定义动画相关的简写属性。

下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-animation



属性值：

该属性为简写属性，属性值为对应完整属性名的值，该简写属性的属性值顺序如下。

animation: name duration timing-function delay iteration-count direction fill-mode play-state;

name：指定要绑定到选择器的关键帧的名称（必须）。

duration：动画指定需要多少秒或毫秒完成（必须）。

timing-function：设置动画将如何完成一个周期（可省略）。

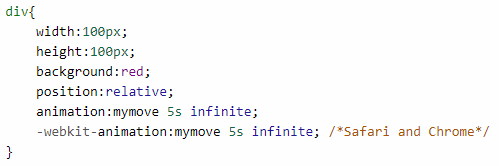
delay：设置动画在启动前的延迟间隔。（可省略）。

iteration-count：定义动画的播放次数。（可省略）。

direction：指定是否应该轮流反向播放动画。（可省略）。

fill-mode：规定当动画不播放时（当动画完成时，或当动画有一个延迟未开始播放时），要应用到元素的样式。（可省略）。

play-state：指定动画是否正在运行或已暂停。（可省略）。



* **animation-name**

animation-name 属性为 @keyframes 动画指定名称。

下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-animation-name



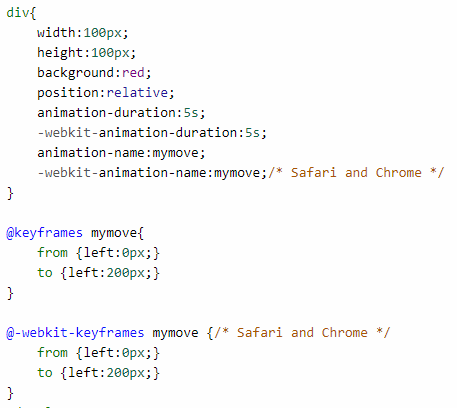
属性值：

1. keyframename

指定要绑定到选择器的关键帧的名称

1. none

指定有没有动画（可用于覆盖从级联的动画）



* **animation-duration**

animation-duration属性定义动画完成一个周期需要多少秒或毫秒。

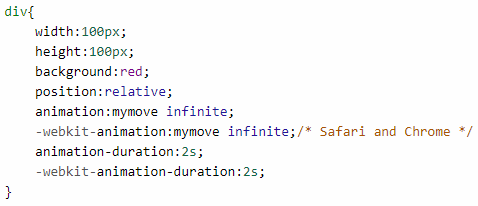
下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-animation-duration



属性值：

1. time

指定动画播放完成花费的时间。默认值为 0，意味着没有动画效果。值为以秒（s）为单位的数值。



* **animation-timing-function**

animation-timing-function指定动画将如何完成一个周期。速度曲线定义动画从一套 CSS 样式变为另一套所用的时间。速度曲线用于使变化更为平滑。

下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-animation-timing-function



属性值：

animation-timing-function使用的数学函数，称为三次贝塞尔曲线，速度曲线。使用此函数，您可以使用您自己的值，或使用预先定义的值之一：

1. linear

动画从头到尾的速度是相同的。

1. ease

默认。动画以低速开始，然后加快，在结束前变慢。

1. ease-in

动画以低速开始。

1. ease-out

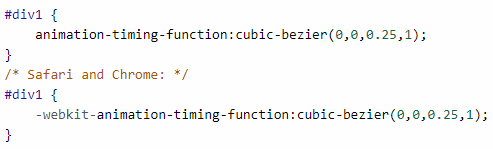
动画以低速结束。

1. ease-in-out

动画以低速开始和结束。

1. cubic-bezier(n,n,n,n)

在 cubic-bezier 函数中自己的值。可能的值是从 0 到 1 的数值。



* **animation-delay**

animation-delay 属性定义动画什么时候开始。animation-delay 值单位可以是秒（s）或毫秒（ms）。

提示: 允许负值，-2s 使动画马上开始，但跳过 2 秒进入动画。

下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-animation-delay



属性值：

1. time

可选。定义动画开始前等待的时间，以秒或毫秒计。默认值为0

* **animation-iteration-count**

animation-iteration-count属性定义动画应该播放多少次。

下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-animation-iteration-count



属性值：

1. n

一个数字，定义应该播放多少次动画

1. infinite

指定动画应该播放无限次（永远）

* **animation-direction**

animation-direction 属性定义是否循环交替反向播放动画。

注意：如果动画被设置为只播放一次，该属性将不起作用。

下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-animation-direction



属性值：

1. normal

默认值。动画按正常播放。

1. reverse

动画反向播放。

1. alternate

动画在奇数次（1、3、5...）正向播放，在偶数次（2、4、6...）反向播放。

1. alternate-reverse

动画在奇数次（1、3、5...）反向播放，在偶数次（2、4、6...）正向播放。

1. initial

设置该属性为它的默认值。

* **animation-fill-mode**

animation-fill-mode 属性规定当动画不播放时（当动画完成时，或当动画有一个延迟未开始播放时），要应用到元素的样式。

默认情况下，CSS 动画在第一个关键帧播放完之前不会影响元素，在最后一个关键帧完成后停止影响元素。animation-fill-mode 属性可重写该行为。

下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-animation-fill-mode



属性值：

1. none

默认值。动画在动画执行之前和之后不会应用任何样式到目标元素。

1. forwards

在动画结束后（由 animation-iteration-count 决定），动画将应用该属性值。

1. backwards

动画将应用在 animation-delay 定义期间启动动画的第一次迭代的关键帧中定义的属性值。这些都是 from 关键帧中的值（当 animation-direction 为 "normal" 或 "alternate" 时）或 to 关键帧中的值（当 animation-direction 为 "reverse" 或 "alternate-reverse" 时）。

1. both

动画遵循 forwards 和 backwards 的规则。也就是说，动画会在两个方向上扩展动画属性。

1. initial

设置该属性为它的默认值。

* **animation-play-state**

animation--play-state属性指定动画是否正在运行或已暂停。

注意：在JavaScript中使用此属性在一个周期中暂停动画。

下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-animation--play-state



属性值：

1. paused

指定暂停动画

1. running

指定正在运行的动画

* **appearance**

appearance 属性允许您使元素看上去像标准的用户界面元素。

下表中说明了到现目前为止该属性在各大浏览器的支持情况，-webkit-, -ms- 或 -moz- 表明该属性要想在对应浏览器上实现效果需要在该属性名称前添加对应的前缀。例如：-webkit-appearance



属性值：

1. normal

正常呈现元素

1. icon

作为一个小图片的呈现元素

1. window

作为一个视口呈现元素

1. button

作为一个按钮，呈现元素

1. menu

作为一个用户选项设定呈现元素选择

1. field

作为一个输入字段呈现元素