级联、特异性和继承

本课的目的是加深您对 CSS 一些最基本概念的理解——级联、特异性和继承——它们控制着 CSS 如何应用于 HTML 以及如何解决样式声明之间的冲突。

虽然学习这节课可能看起来不太相关，并且比课程的其他部分更具学术性，但理解这些概念将使您以后免于很多痛苦！我们鼓励您仔细阅读本节，并在继续之前检查您是否理解这些概念。

先决条件：	基本的计算机知识、 安装的基本软件、使用文件的基本知识、HTML 基础知识（学习 HTML 简介）以及 CSS 的工作原理（学习 CSS 第一步）。
客观的：	了解级联和特异性，以及继承在 CSS 中的工作原理。

冲突规则

CSS 代表**Cascading Style Sheets**，理解第一个词*级联*非常重要——级联的行为方式是理解 CSS 的关键。

在某些时候，你将在一个项目上工作，你会发现你认为应该应用于元素的 CSS 不起作用。通常，问题是您创建了两个规则，将同一属性的不同值应用于同一元素。**级联和密切相关的特异性**概念是控制在存在此类冲突时应用哪个规则的机制。元素样式化规则可能不是您所期望的那样，因此您需要了解这些机制的工作原理。

这里同样重要的是继承的概念，这意味着一些 CSS 属性默认继承在当前元素的父元素上设置的值，而有些则没有。这也可能导致一些您可能意想不到的行为。

让我们首先快速浏览一下我们正在处理的关键事情，然后我们将依次查看每一个，看看它们如何与彼此以及您的 CSS 交互。这些看起来像是一组难以理解的概念。随着您对编写 CSS 的练习越来越多，它的工作方式对您来说将变得更加明显。

级联

样式**表级联**——在一个非常简单的层面上，这意味着 CSS 规则的起源、级联层和顺序很重要。当来自同一个级联层的两个规则应用并且都具有相同的特异性时，样式表中最后定义的那个将被使用。

在下面的示例中，我们有两个可应用于元素的规则 `<h1>` 。内容 `<h1>` 最终变成蓝色。这是因为这两个规则来自相同的源，具有相同的元素选择器，因此具有相同的特异性，但源顺序中的最后一个规则获胜。

Interactive editor

```
h1 {
    color: red;
}
h1 {
    color: blue;
}
```

```
<h1>This is my heading.</h1>
```

Reset

特异性

特异性是浏览器用来决定将哪个属性值应用于元素的算法。如果多个样式块具有不同的选择器，这些选择器将相同的属性配置为不同的值并以相同的元素为目标，则特异性决定了应用于该元素的属性值。特异性基本上是衡量选择器选择的具体程度的指标：

- 元素选择器不太具体；它将选择出现在页面上的所有该类型的元素，因此它的权重较小。伪元素选择器与常规元素选择器具有相同的特异性。
- 类选择器更具体；它只会选择页面上具有特定属性值的元素 `class`，因此它具有更大的权重。属性选择器和伪类与类具有相同的权重。

下面，我们再次有两个可应用于该 `<h1>` 元素的规则。下面的内容 `<h1>` 最终被涂成红色，因为类选择器 `main-heading` 为其规则提供了更高的特异性。因此，即使带有 `<h1>` 元素选择器的规则在源代码顺序中出现得更靠后，使用类选择器定义的具有更高特异性的规则也会被应用。

Interactive editor

```
.main-heading {
    color: red;
}

h1 {
    color: blue;
}
```

```
<h1 class="main-heading">This is my
heading.</h1>
```

Reset

稍后我们将解释特异性算法。

遗产

在此上下文中也需要理解继承——在父元素上设置的某些 CSS 属性值会被其子元素继承，而有些则不会。

例如，如果你在一个元素上设置了一个 `color` and ，它里面的每个元素也将使用该颜色和字体设置样式，除非你已经直接对它们应用了不同的颜色和字体值。 `font-family`

As the body has been set to have a color of blue this is inherited through the descendants.

We can change the color by targeting the element with a selector, such as this span.

Interactive editor

```
body {
    color: blue;
}

span {
    color: black;
}
```

```
<p>As the body has been set to have
a color of blue this is inherited
through the descendants.</p>
<p>We can change the color by
targeting the element with a
selector, such as this
```

Reset

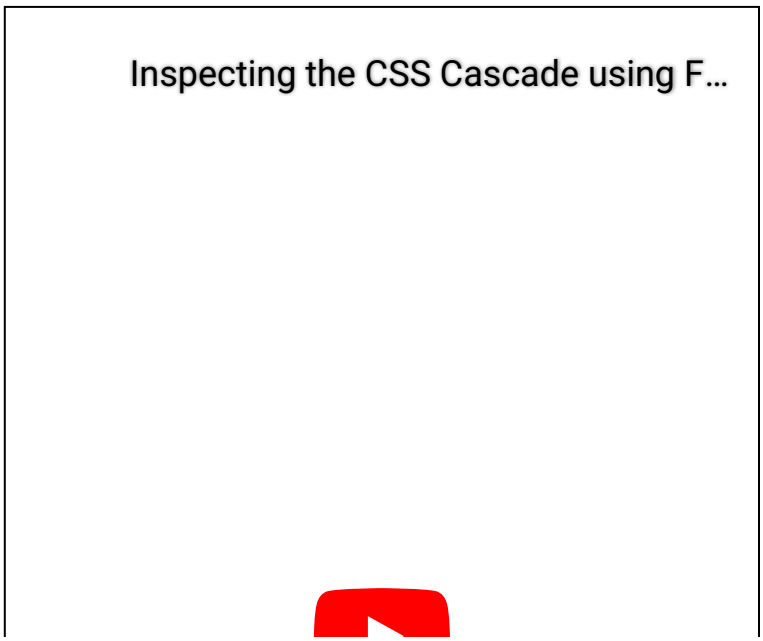
某些属性不会继承——例如，如果您将 `width` 一个元素的 a 设置为 50%，则其所有后代的宽度都不会达到其父元素宽度的 50%。如果是这样的话，CSS 使用起来会非常令人沮丧！

> **注意：**在 MDN CSS 属性参考页面上，您可以找到一个名为“正式定义”的技术信息框，其中列出了有关该属性的许多数据点，包括它是否被继承。以颜色属性形式定义部分为例。

了解这些概念如何协同工作

这三个概念（级联、特异性和继承）共同控制了哪些 CSS 应用于哪些元素。在下面的部分中，我们将看到它们如何协同工作。它有时看起来有点复杂，但随着你对 CSS 的经验越来越丰富，你会开始记住它们，如果你忘记了，你可以随时查找细节！即使是经验丰富的开发人员也不记得所有的细节。

以下视频展示了如何使用 Firefox DevTools 检查页面的级联、特异性等：



理解继承

我们将从继承开始。在下面的示例中，我们有一个 `<ul>` 元素，其中嵌套了两层无序列表。我们已经为外部提供了 `<ul>` 边框、填充和字体颜色。

该 `color` 财产是继承财产。因此， `color` 属性值将应用于直接子级和间接子级——直接子级 `<li>` 和第一个嵌套列表中的子级。然后我们将该类添加 `special` 到第二个嵌套列表并为其应用不同的颜色。然后通过它的孩子继承下来。

- Item One
- Item Two
  - 2.1
  - 2.2
- Item Three
  - **3.1**
    - **3.1.1**
    - **3.1.2**
  - **3.2**

Interactive editor

```
.main {
    color: rebeccapurple;
    border: 2px solid #ccc;
    padding: 1em;
}

.special {
    color: black;
    font-weight: bold;
}
```

```
<ul class="main">
    <li>Item One</li>
    <li>Item Two
        <ul>
            <li>2.1</li>
            <li>2.2</li>
        </ul>
    </li>
    <li>Item Three
        <ul class="special">
            <li>3.1
                <ul>
                    <li>3.1.1</li>
                    <li>3.1.2</li>
                </ul>
            </li>
            <li>3.2</li>
        </ul>
    </li>
</ul>
```

Reset

像 `width` （如前所述）、 `margin` 、 `padding` 和 之类的属性 `border` 不是继承属性。如果在此列表示例中子项继承了边框，则每个列表和列表项都会获得一个边框——这可能不是我们想要的效果！

尽管每个 CSS 属性页都列出了该属性是否被继承，但如果您知道属性值将在哪个方面设置样式，您通常可以凭直觉做出相同的猜测。

控制继承

CSS 提供了五个特殊的通用属性值来控制继承。每个 CSS 属性都接受这些值。

`inherit`

将应用于选定元素的属性值设置为其父元素的属性值。实际上，这“开启了继承”。

`initial`

将应用于选定元素的属性值设置为该属性的初始值。

`revert`

将应用于选定元素的属性值重置为浏览器的默认样式，而不是应用于该属性的默认值。 `unset` 在许多情况下，此值的作用是一样的。

`revert-layer`

将应用于选定元素的属性值重置为在前一个级联层中建立的值。

`unset`

将属性重置为其自然值，这意味着如果属性是自然继承的，则它的行为类似于 `inherit` ，否则它的行为类似于 `initial` 。

> **注意：** 有关每种类型及其工作方式的更多信息，请参阅来源类型。

我们可以查看链接列表并探索普世价值是如何运作的。下面的示例允许您使用 CSS 并查看进行更改时会发生什么。玩代码确实是更好地理解 HTML 和 CSS 的最佳方式。

例如：

1. 第二个列表项应用了类 `my-class-1` 。 `<a>` 这会将嵌套在其中的元素的颜色设置为 `inherit` 。如果删除规则，它如何改变链接的颜色?
2. 你明白为什么第三个和第四个链接是它们现在的颜色吗？第三个链接设置为 `initial` ，这意味着它使用属性的初始值（在本例中为黑色），而不是浏览器默认的链接，即蓝色。第四个设置为 `unset` 意味着链接文本使用父元素的颜色，绿色。
3. 如果您为元素定义新颜色，哪个链接会改变颜色 `<a>` ——例如 `a { color: red; }` ?
4. 阅读下一节有关重置所有属性的内容后，返回并将 `color` 属性更改为 `all` . 请注意第二个链接是如何在新行上并带有项目符号的。你认为哪些属性是遗传的?

- Default link color
- Inherit the link color
- Reset the link color
- Unset the link color

Interactive editor

```
body {
    color: green;
}

.my-class-1 a {
    color: inherit;
}

.my-class-2 a {
    color: initial;
}

.my-class-3 a {
    color: unset;
}
```

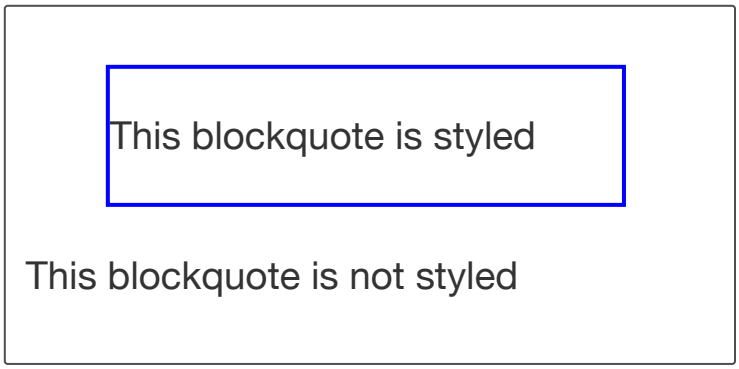
```
<ul>
    <li>Default <a href="#">link</a>
color</li>
    <li class="my-class-1">Inherit
the <a href="#">link</a> color</li>
    <li class="my-class-2">Reset the
<a href="#">link</a> color</li>
    <li class="my-class-3">Unset the
<a href="#">link</a> color</li>
```

Reset

重置所有属性值

CSS 速记属性 `all` 可用于将这些继承值之一立即应用于（几乎）所有属性。它的值可以是任何一个继承值（ `inherit` 、 `initial` 、 `revert` 、 `revert-layer` 或 `unset` ）。这是撤消对样式所做更改的便捷方式，以便您可以在开始新更改之前返回到已知起点。

在下面的示例中，我们有两个块引用。第一个将样式应用于 blockquote 元素本身。第二个有一个应用于 blockquote 的类，它将 的值设置 `all` 为 `unset` 。



Interactive editor

```
blockquote {
    background-color: orange;
    border: 2px solid blue;
}

.fix-this {
    all: unset;
}
```

```
        <blockquote>
            <p>This blockquote is
styled</p>
        </blockquote>

        <blockquote class="fix-
this">
            <p>This blockquote is
not styled</p>
```

Reset

尝试将 的值设置 `all` 为其他一些可用值并观察有何不同。

了解级联

我们现在明白了继承是为什么嵌套在 HTML 结构深处的段落与应用于正文的 CSS 颜色相同的原因。从介绍性课程中，我们了解了如何更改应用于文档中任何位置的某些内容的 CSS——无论是通过将 CSS 分配给元素还是通过创建类。我们现在将了解当多个样式块将相同属性但具有不同值应用于同一元素时，级联如何定义应用哪些 CSS 规则。

需要考虑三个因素，此处按重要性递增的顺序列出。后来的推翻了前面的：

1. **货源顺序**
2. **特异性**
3. **重要性**

我们将研究这些以了解浏览器如何准确地确定应该应用什么 CSS。

货源顺序

我们已经看到源代码顺序对级联有何影响。如果你有多个规则，所有规则都具有完全相同的权重，那么在 CSS 中排在最后的规则将获胜。您可以将其视为：靠近元素本身的规则会覆盖较早的规则，直到最后一个规则获胜并开始设置元素的样式。

源顺序仅在规则的特异性权重相同时才重要，所以让我们看一下特异性：

特异性

您经常会遇到这样一种情况，您知道规则出现在样式表的后面，但是应用了一个更早的、冲突的规则。发生这种情况是因为较早的规则具有**更高的特异性**——它更具体，因此被浏览器选择为应该为元素设置样式的规则。

正如我们在本课前面看到的，类选择器比元素选择器具有更大的权重，因此类样式块中定义的属性将覆盖元素样式块中定义的属性。

这里需要注意的是，虽然我们正在考虑选择器和应用于它们选择的文本或组件的规则，但并不是整个规则被覆盖，只是在多个地方声明的属性。

此行为有助于避免 CSS 中的重复。一种常见的做法是为基本元素定义通用样式，然后为那些不同的元素创建类。例如，在下面的样式表中，我们为 2 级标题定义了通用样式，然后创建了一些仅更改部分属性和值的类。最初定义的值应用于所有标题，然后将更具体的值应用于带有类别的标题。

Heading with no class

Heading with class of small

Heading with class of bright

Interactive editor

```
h2 {
    font-size: 2em;
    color: #000;
    font-family: Georgia, 'Times New
Roman', Times, serif;
}

.small {
    font-size: 1em;
}

.bright {
    color: rebeccapurple;
}
```

```
<h2>Heading with no class</h2>
<h2 class="small">Heading with class of
small</h2>
<h2 class="bright">Heading with class of
bright</h2>
```

Reset

现在让我们看看浏览器如何计算特异性。我们已经知道元素选择器的特异性很低，可以被类覆盖。本质上，不同类型的选择器会获得一个以点为单位的值，将这些值相加即可得出该特定选择器的权重，然后可以根据其他潜在匹配项对其进行评估。

选择器具有的特异性的数量是使用三个不同的值（或组件）来衡量的，可以将其视为百位、十位和个位的 ID、CLASS 和 ELEMENT 列：

- **Identifiers**：为包含在整体选择器中的每个 ID 选择器在此列中打一分。
- **类**：在该列中为包含在整个选择器中的每个类选择器、属性选择器或伪类打一分。
- **元素**：在此列中为每个元素选择器或包含在整体选择器中的伪元素打一分。

> **注意：** 通用选择器（`*`）、组合器（`+`、`>`、`~`、`' '`）和特异性调整选择器（`:where()`）及其参数对特异性没有影响。

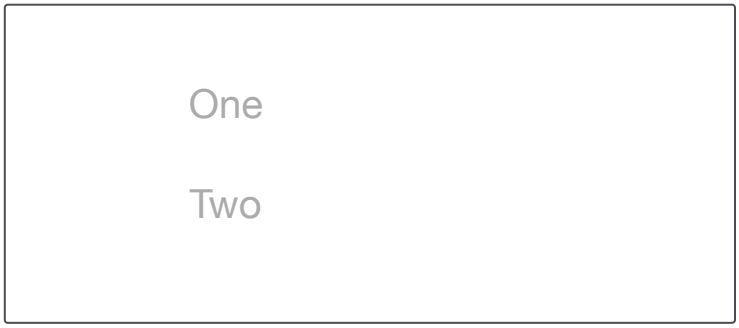
否定（`:not()`）、关系选择器（`:has()`）和 matches-any（`:is()`）伪类本身对特异性没有影响，但它们的参数有影响。每个对特异性算法贡献的特异性是选择器在具有最大权重的参数中的特异性。

下表显示了一些独立的示例，可以让您了解情况。尝试通过这些，并确保您理解为什么它们具有我们赋予它们的特殊性。我们还没有详细介绍选择器，但您可以在 MDN选择器参考中找到每个选择器的详细信息。

选择器	身份标识	班级	元素	总特异性
`h1`	0	0	1个	0-0-1
`h1 + p::first-letter`	0	0	3个	0-0-3
`li > a[href*="en-US"] >`	0	2个	2个	0-2-2

选择器	身份标识	班级	元素	总特异性
`.inline-warning`				
`#identifier`	1个	0	0	1-0-0
`button:not(#mainBtn, .cta)`	1个	0	1个	1-0-1

在我们继续之前，让我们看一个实际的例子。



Interactive editor

```
/* 1. specificity: 1-0-1 */
#outer a {
    background-color: red;
}

/* 2. specificity: 2-0-1 */
#outer #inner a {
    background-color: blue;
}
```



```
/* 4. specificity: 1-1-3 */
```

```
<div id="outer" class="container">
    <div id="inner"
class="container">
        <ul>
            <li class="nav"><a
href="#">One</a></li>
            <li class="nav"><a
href="#">Two</a></li>
        </ul>
```

Reset

那么这是怎么回事？首先，我们只对本示例的前七个规则感兴趣，您会注意到，我们在每个规则之前的注释中包含了它们的特异性值。

- 前两个选择器竞争链接背景颜色的样式。第二个获胜并使背景颜色为蓝色，因为它在链中有一个额外的 ID 选择器：它的特异性是 2-0-1 与 1-0-1。

- 选择器 3 和 4 正在竞争链接文本颜色的样式。第二个获胜并使文本变白，因为虽然它少了一个元素选择器，但缺少的选择器被换成了类选择器，它比无限元素选择器具有更大的权重。获胜的特殊性是 1-1-3 对 1-0-4。
- 选择器 5-7 在悬停时竞争链接边框的样式。选择器 6 显然输给了选择器 5，其特异性为 0-2-3 vs. 0-2-4；它在链中少了一个元素选择器。然而，选择器 7 击败了选择器 5 和 6，因为它在链中具有与选择器 5 相同数量的子选择器，但是一个元素已被换出为类选择器。所以获胜的特殊性是 0-3-3 对 0-2-3 和 0-2-4。

> **注意：** 每种选择器类型都有自己的特定级别，不能被具有较低特定级别的选择器覆盖。例如，*一百万个* **类**选择器组合起来将无法覆盖*一个* **id**选择器的特异性。
>
> 评估特异性的最佳方法是对特异性水平进行单独评分，从最高开始，并在必要时向最低移动。只有当特异性列中的选择器分数之间存在联系时，您才需要评估下一列；否则，您可以忽略较低特异性的选择器，因为它们永远无法覆盖较高特异性的选择器。

行内样式

内联样式，即 `style` 属性内的样式声明，无论其特殊性如何，都优先于所有普通样式。此类声明没有选择器，但它们的特异性可以解释为 1-0-0-0；无论选择器中有多少个 ID，它总是比任何其他特异性权重都大。

！重要的

有一个特殊的 CSS 片段可以用来否决上述所有计算，甚至是内联样式 - 标志 `!important` 。但是，您在使用它时应该非常小心。此标志用于使单个属性和值对成为最具体的规则，从而覆盖级联的正常规则，包括正常的内联样式。

> **注意：** 了解该标志的存在很有用 `!important` ，这样当您在其他人的代码中遇到它时，您就知道它是什么。**但是，我们强烈建议您除非万不得已，否则永远不要使用它。**该 `!important` 标志改变

了级联正常工作的方式，因此它可以使调试 CSS 问题变得非常困难，尤其是在大型样式表中。

看看这个例子，我们有两个段落，其中一个有一个 ID。

This is a paragraph.

One selector to rule them all!

Interactive editor

```
#winning {
    background-color: red;
    border: 1px solid black;
}

.better {
    background-color: gray;
    border: none !important;
}

p {
    background-color: blue;
    color: white;
    padding: 5px;
}
```

```
<p class="better">This is a paragraph.
</p>
<p class="better" id="winning">One
selector to rule them all!</p>
```

Reset

让我们来看看发生了什么——尝试删除一些属性，看看如果你觉得难以理解会发生什么：

1. 您会看到已应用第三条规则 `color` 和值，但尚未应用。为什么？实际上，这三者都应该适用，因为源代码顺序中后面的规则通常会覆盖前面的规则。 `padding` `background-color`
2. 但是，它上面的规则获胜，因为类选择器比元素选择器具有更高的特异性。
3. 两个元素都有一个 `class` of `better` ，但第二个元素也有一个 `id` of `winning` 。由于 ID 比类具有*更高的*特异性（页面上每个元素只能有一个具有唯一 ID 的元素，但许多元素具有相同的类——ID 选择器的目标非常具体），红色背景颜色和*1px*黑色边框应该应用于第二个元素，第一个元素获得灰色背景颜色，并且没有边框，如类所指定的那样。
4. 第二个元素*确实*获得了红色背景颜色，但没有边框。为什么？因为 `!important` 第二条规则中的标志。 `!important` 在后面添加标志 `border: none` 意味着此声明将胜过 `border` 先前规则中的值，即使 ID 选择器具有更高的特异性。

> **注意：** 覆盖重要声明的唯一方法是在源代码顺序的后面包含另一个*具有相同特异性*的重要声明，或者具有更高特异性的声明，或者在前面的级联层中包含一个重要声明（我们没有涵盖级联层）。

您可能必须使用该 `!important` 标志的一种情况是，当您在无法编辑核心 CSS 模块的 CMS 上工作时，并且您确实想要覆盖内联样式或无法覆盖的重要声明任何其他方式。但实际上，如果可以避免，请不要使用它。

CSS定位的效果

最后，重要的是要注意 CSS 声明的优先级取决于它在哪个样式表和级联层中指定。

用户可以设置自定义样式表来覆盖开发人员的样式。例如，视障用户可能希望将他们访问的所有网页的字体大小设置为正常大小的两倍，以便于阅读。

也可以在级联层中声明开发人员样式：您可以使非分层样式覆盖在层中声明的样式，或者您可以使在后面的层中声明的样式覆盖较早声明的层中的样式。例如，作为开发人员，您可能无法编辑第三方样式表，但您可以将外部样式表导入级联层，这样您的所有样式都可以轻松覆盖导入的样式，而不必担心第三方选择器的特殊性。

覆盖声明的顺序

冲突的声明将按以下顺序应用，后面的将覆盖前面的：

1. 用户代理样式表中的声明（例如，浏览器的默认样式，在未设置其他样式时使用）。
2. 用户样式表中的正常声明（用户设置的自定义样式）。
3. 作者样式表中的正常声明（这些是我们 Web 开发人员设置的样式）。
4. 作者样式表中的重要声明。
5. 用户样式表中的重要声明。
6. 用户代理样式表中的重要声明。

> **注意：** 对于标记有 的样式，优先顺序是相反的 `!important` 。Web 开发人员的样式表覆盖用户样式表是有意义的，因此设计可以保持预期；然而，有时用户有充分的理由覆盖 Web 开发人员样式，如上所述，这可以通过 `!important` 在他们的规则中使用来实现。

级联层的顺序

尽管级联层是一个高级主题并且您可能不会立即使用此功能，但了解层级联的方式很重要。

当您在级联层中声明 CSS 时，优先顺序由声明层的顺序决定。在任何层之外声明的 CSS 样式按照这些样式声明的顺序组合在一起，形成一个未命名的层，就好像它是最后声明的层一样。对于相互竞争的普通（不重要）样式，后面的层优先于前面定义的层。然而，对于标记为 的样式 `!important` ，顺序是相反的，较早层中的重要样式优先于后续层中或任何层外声明的重要样式。内联样式优先于所有作者样式，无论图层如何。

当不同层中有多个样式块为单个元素的属性提供竞争值时，声明层的顺序决定了优先级。层之间的特异性并不重要，但单层内的特异性仍然重要。

A paragraph with a border and background

Interactive editor

```
@layer firstLayer, secondLayer;

p { /* 0-0-1 */
  background-color: red;
  color: grey !important;
  border: 5px inset purple;
}
p#addSpecificity { /* 1-0-1 */
  border-style: solid !important;
}

@layer firstLayer {
  #addSpecificity { /* 1-0-0 */
    background-color: blue;
    color: white !important;
    border-width: 5px;
    border-style: dashed !important;
  }
```

```
<p id="addSpecificity">
  A paragraph with a border and background
</p>
```

Reset

让我们从上面的例子中讨论一些事情来理解发生了什么。已经声明了两层，`firstLayer` 并且 `secondLayer` , 按此顺序。即使 in 的特异性 `secondLayer` 最高，也不会使用该声明中的任何属性。为什么？因为非分层普通样式优先

于分层普通样式，无论特殊性如何，重要的分层样式优先于后面层中声明的重要样式，同样，无论特殊性如何。

If you change the first line of CSS in this example to read `@layer secondLayer, firstLayer;`, you will change the layer declaration order, and all the important styles from `firstLayer` will be changed to their respective values in `secondLayer`.

Test your skills!

You've reached the end of this article, but can you remember the most important information? You can find some further tests to verify that you've retained this information before you move on — see Test your skills: The Cascade.

Summary

If you understood most of this article, then well done — you've started getting familiar with the fundamental mechanics of CSS. Next up, we'll take a deeper look at Cascade Layers.

If you didn't fully understand the cascade, specificity, and inheritance, then don't worry! This is definitely the most complicated thing we've covered so far in the course and is something that even professional web developers sometimes find tricky. We'd advise that you return to this article a few times as you continue through the course, and keep thinking about it.

Refer back here if you start to come across strange issues with styles not applying as expected. It could be a specificity issue.

This page was last modified on 2023 年 4 月 14 日由MDN 贡献者提供。