级联层

本课旨在向您介绍级联层，这是一种更高级的功能，它建立在CSS 级联和CSS 特异性的基本概念之上。

如果您是 CSS 的新手，那么学习本课程可能看起来不太相关，并且比课程的其他部分更具学术性。但是，如果您在项目中遇到级联层，了解什么是级联层的基础知识是有帮助的。您使用 CSS 的次数越多，了解级联层并知道如何利用它们的力量将使您免于使用来自不同方、插件和开发团队的 CSS 管理代码库的许多痛苦。

当您使用来自多个来源的 CSS、存在冲突的 CSS 选择器和相互竞争的特性时，或者当您考虑使用 `!important` .

先决条件：	了解 CSS 的工作原理，包括级联和特异性（研究 CSS 第一步和级联、特异性和继承）。
客观的：	了解级联层的工作原理。

对于应用于元素的每个 CSS 属性，只能有一个值。您可以通过在浏览器的开发人员工具中检查元素来查看应用于元素的所有属性值。该工具的“样式”面板显示了应用于被检查元素的所有属性值，以及匹配的选择器和 CSS 源文件。具有优先权的来源选择器将其值应用于匹配元素。

除了应用的样式外，“样式”面板还显示与所选元素匹配但由于级联、特异性或源顺序而未应用的划线值。划掉的样式可能来自具有优先级但特异性较低的相同来源，或者具有匹配的来源和特异性，但在代码库中较早发现。对于任何应用的属性值，可能有从许多不同来源划掉的多个声明。如果你看到一个样式被划掉了，它有一个更具体的选择器，这意味着这个值缺乏来源或重要性。

通常，随着站点复杂性的增加，样式表的数量也会增加，这使得样式表的源代码顺序变得更加重要和复杂。级联层简化了跨此类代码库的样式表维护。级联层是明确的特异性容器，可以更简单、更好地控制最终应用的 CSS 声明，使 Web 开发人员能够确定 CSS 部分的优先级，而不必与特异性作斗争。

要了解级联层，您必须了解 CSS 级联。以下部分简要回顾了重要的级联概念。

级联概念回顾

CSS 中的 C 代表“级联”。这是样式级联在一起的方法。用户代理通过几个非常明确定义的步骤来确定分配给每个元素的每个属性的值。我们将在这里简要列出这些步骤，然后深入探讨第 4 步，即级联层，这就是您来到这里学习的内容：

1. **相关性：**查找所有声明块，其中每个元素的选择器都匹配。
2. **重要性：**根据规则是正常还是重要对规则进行排序。重要的样式是那些设置了 `!important` 标志的样式。
3. **来源：**在两个重要性桶中的每一个中，按作者、用户或用户代理来源对规则进行排序。
4. **层：**在六个起源重要性桶中的每一个中，按级联层排序。普通声明的层顺序是从创建的第一层到最后一层，然后是未分层的普通样式。重要样式的顺序相反，未分层的重要样式具有最低的优先级。
5. **Specificity：**对于原始层中具有优先级的竞争样式，按specificity对声明进行排序。
6. **出现顺序：**当原始层中具有优先级的两个选择器具有相同的特异性时，最后声明的具有最高特异性的选择器的属性值获胜。

对于每一步，只有“仍在运行中”的声明才能在下一步中“竞争”。如果只有一个声明在运行，则它“获胜”，并且后续步骤没有实际意义。

起源和级联

存在三种级联源类型：用户代理样式表、用户样式表和作者样式表。浏览器根据来源和重要性将每个声明分类到六个来源桶中。有八个优先级别：六个

原始桶、正在转换的属性和正在动画的属性。优先顺序从具有最低优先级的普通用户代理样式到当前应用的动画中的样式，再到重要的用户代理样式，然后是具有最高优先级的正在转换的样式：

1. 用户代理正常样式
2. 用户正常样式
3. 作者正常风格
4. 动画样式
5. 作者重要风格
6. 用户重要样式
7. 用户代理重要样式
8. 正在转换的样式

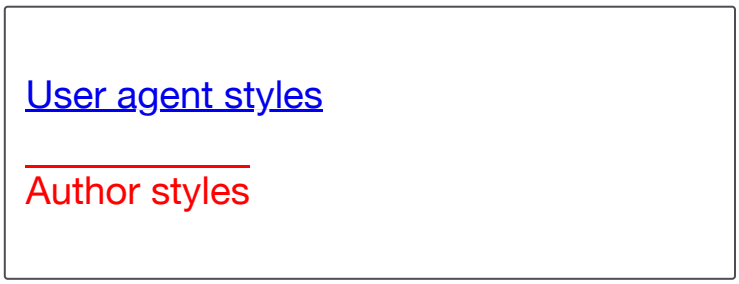
“用户代理”是浏览器。“用户”是站点访问者。“作者”是你，开发者。直接在带有元素的元素上声明的样式 `<style>` 是作者样式。不包括动画和过渡样式，用户代理正常样式的优先级最低；user-agent 重要样式最高。

起源和特异性

对于每个属性，“获胜”的声明是来自起源的声明，其优先级基于权重（正常或重要）。暂时忽略图层，应用来自具有最高优先级的原点的值。如果获胜的来源对一个元素有多个属性声明，则比较这些竞争属性值的选择器的特异性。从不比较来自不同来源的选择器之间的特异性。

在下面的示例中，有两个链接。第一个没有应用作者样式，因此仅应用用户代理样式（以及您的个人用户样式，如果有的话）。尽管作者样式表中的选择器具有 . 的特殊性，但第二个具有 `text-decoration` 和由作者样式设置。作者样式“获胜”的原因是，当存在来自不同来源的冲突样式时，将应用来自具有优先权的来源的规则，而不考虑没有优先权的来源的特殊性。

`color` `0-0-0`



Interactive editor

```
:where(a.author) {
  text-decoration: overline;
  color: red;
}
```

```
<p><a
href="https://example.org">User
```

Reset

在撰写本文时，用户代理样式表中的“竞争”选择器是 `a:any-link`，其权重为 `0-1-1`。虽然这大于 `0-0-0` 作者样式表中的选择器，但即使当前用户代理中的选择器不同，也没关系：作者和用户代理来源的特异性权重永远不会进行比较。详细了解如何计算特异性权重。

来源优先级总是胜过选择器特异性。如果元素属性在多个来源中使用普通样式声明进行样式化，则作者样式表将始终覆盖在用户或用户代理样式表中声明的冗余普通属性。如果样式很重要，用户代理样式表将始终胜过作者和用户样式。级联起源优先级确保起源之间的特异性冲突永远不会发生。

在继续之前要注意的最后一件事：仅当优先来源中的竞争声明具有相同的特异性时，出现顺序才有意义。

级联层概述

我们现在明白了“cascade origin precedence”，但是什么是“cascade layer precedence”呢？我们将通过解决级联层是什么、它们如何排序以及

样式如何分配给级联层来回答这个问题。我们将介绍常规层、嵌套层和匿名层。我们先讨论什么是级联层，它们解决什么问题。

级联层优先顺序

类似于我们基于来源和重要性有六个优先级的方式，级联层使我们能够在任何这些来源中创建子来源优先级。

在六个原始桶中的每一个中，都可以有多个级联层。图层创建的顺序很重要。创建的顺序设置了原点内各层之间的优先顺序。

在正常的原始桶中，层按照每个层的创建顺序排序。优先顺序是从创建的第一层到最后一层，然后是未分层的普通样式。

对于重要的样式，此顺序是相反的。所有未分层的重要样式级联在一起成为一个隐式层，该层优先于所有非过渡正常样式。未分层的重要样式的优先级低于任何重要的分层样式。较早声明的图层中的重要样式优先于相同来源的后续声明图层中的重要样式。

对于本教程的其余部分，我们将把讨论限制在作者样式上，但请记住，层也可以存在于用户和用户代理样式表中。

级联层可以解决的问题

大型代码库可能具有来自多个团队、组件库、框架和第三方的样式。无论包含多少样式表，所有这些样式都在一个来源中级联在一起：*作者*样式表。

将来自多个来源的样式级联在一起，尤其是来自不同团队的样式，可能会产生问题。不同的团队可能有不同的方法；一个人可能有降低特异性的最佳实践，而另一个人可能有一个在每个选择器中包含一个的标准 `id` 。

特异性冲突会迅速升级。Web 开发人员可以通过添加标志来创建“快速修复” `!important` 。虽然这可能感觉是一个简单的解决方案，但它通常只是将特异性战争从正常声明转移到重要声明。

与级联起源在用户、用户代理和作者风格之间提供权力平衡的方式相同，级联层提供了一种结构化的方式来组织和平衡单个起源内的关注点，就好像起

源中的每一层都是一个子层一样。起源。可以为每个团队、组件和第三方创建一个图层，其样式优先级基于图层顺序。

层内的规则级联在一起，而不与层外的样式规则竞争。级联层使整个样式表优先于其他样式表，而不必担心这些子源之间的特异性。

层优先级总是胜过选择器特异性。具有优先级的图层中的样式“胜过”具有较低优先级的图层。丢失层中选择器的特异性无关紧要。特异性对于层内的竞争属性值仍然很重要，但层与层之间没有特异性问题，因为每个属性只考虑最高优先级的层。

嵌套级联层可以解决的问题

级联层允许创建嵌套层。每个级联层都可以包含嵌套层。

例如，可以将组件库导入到 `components` 图层中。常规级联层会将组件库添加到作者原点，从而消除与其他作者样式的任何特异性冲突。在层内 `components`，开发人员可以选择定义各种主题，每个主题都作为单独的嵌套层。这些嵌套主题图层的顺序可以根据媒体查询（请参阅下面的图层创建和媒体查询部分）定义，例如视口大小或方向。这些嵌套层提供了一种创建基于特定性不冲突的主题的方法。

嵌套层的能力对于任何致力于开发组件库、框架、第三方小部件和主题的人来说都非常有用。

创建嵌套图层的能力也消除了图层名称冲突的担忧。我们将在嵌套层部分对此进行介绍。

> “作者可以创建层来表示元素默认值、第三方库、主题、组件、覆
>
> mdn web docs
>
> 跨层冲突。
>
> —级联和继承规范 。

创建级联层

可以使用以下任何一种方法创建图层：

- 语句 `@layer` at-rule，声明层 using `@layer` 后跟一个或多个层的名称。这将创建命名图层而不为其分配任何样式。
- 块 `@layer` 规则，其中块内的所有样式都添加到名称或未命名层。
- `@import` 带有 `layer` 关键字或函数的规则 `layer()`，它将导入文件的内容分配到该层。

如果尚未初始化具有该名称的层，则所有三种方法都会创建一个层。`@layer` 如果在规则或 `@import` with中未提供图层名称 `layer()`，则会创建一个新的匿名（未命名）图层。

> **注意：**图层的优先顺序是它们的创建顺序。不在层中的样式或“未分层样式”层叠在一起成为最终的隐式标签。

在讨论嵌套层之前，让我们更详细地介绍创建层的三种方法。

命名层的 @layer 语句规则

层的顺序由层在 CSS 中出现的顺序设置。使用 `@layer` 后跟一个或多个层的名称而不指定任何样式来声明层是定义层顺序的一种方法。

CSS `@layer` at-rule 用于声明级联层，并在有多个级联层时定义优先顺序。以下 at 规则按列出的顺序声明了三个层：

```
@layer theme, layout, utilities;
```

很多时候，您会希望 CSS 的第一行是此 `@layer` 声明（当然，使用对您的站点有意义的图层名称）以完全控制图层顺序。

如果上述语句是站点 CSS 的第一行，则层顺序将为 `theme` ， `layout` , 和 `utilities` 。如果在上述语句之前创建了一些层，只要这些名称的层不存在，这三个层将被创建并添加到现有层列表的末尾。但是，如果同名层已经存在，那么上面的语句只会创建两个新层。例如，如果 `layout` 已经存在，则只创建 `theme` 和 `utilities` ， 但在这种情况下层的顺序将是 `layout` , `theme` , 和 `utilities` 。

命名层和匿名层的 @layer 块规则

`@layer` 可以使用块规则创建层。如果 `@layer` at 规则后跟一个标识符和一个样式块，则该标识符用于命名图层，并将该 at 规则中的样式添加到图层的样式中。如果具有指定名称的图层不存在，将创建一个新图层。如果具有指定名称的图层已经存在，则样式将添加到先前存在的图层中。如果在使用创建样式块时未指定名称 `@layer` ， 则 at 规则中的样式将添加到新的匿名层。

在下面的示例中，我们使用了四个块和一个内联 `@layer` 规则。此 CSS 按列出的顺序执行以下操作：

1. 创建一个命名 `layout` 层
2. 创建一个未命名的匿名层
3. 声明一个包含三层的列表，并且只创建两个新层， `theme` 并且 `utilities` ， 因为 `layout` 已经存在
4. 向现有 `layout` 图层添加其他样式
5. 创建第二个未命名的匿名层

```
/* file: layers1.css */

/* unlayered styles */
body {
  color: #333;
}

/* creates the first layer: `layout` */
@layer layout {
  main {
    display: grid;
  }
```

```
}

/* creates the second layer: an unnamed, anonymous layer */
@layer {
  body {
    margin: 0;
  }
}

/* creates the third and fourth layers: `theme` and `utilities`
*/
@layer theme, layout, utilities;

/* adds styles to the already existing `layout` layer */
@layer layout {
  main {
    color: #000;
  }
}

/* creates the fifth layer: an unnamed, anonymous layer */
@layer {
  body {
    margin: 1vw;
  }
}
```

在上面的 CSS 中，我们创建了五个层： `layout` , `<anonymous(01)>` , `theme` , `utilities` , 和 `<anonymous(02)>` - 按此顺序 - 样式块中包含第六层隐式未分层样式 `body` 。图层顺序是创建图层的顺序，未分层样式的隐式图层始终位于最后。一旦创建，就无法更改图层顺序。

We assigned some styles to the layer named `layout` . If a named layer doesn't already exist, then specifying the name in an `@layer` at-rule, with or without assigning styles to the layer, creates the layer; this adds the layer to the end of the series of existing layer names. If the named layer already exists, all styles within the named block get appended to styles in the previously existing layer – specifying styles in a block by reusing an existing layer name does not create a new layer.

Anonymous layers are created by assigning styles to a layer without naming the layer. Styles can be added to an unnamed layer only at the time of its creation.

> **Note:** Subsequent use of `@layer` with no layer name creates additional unnamed layers; it does not append styles to a previously existing unnamed layer.

The `@layer` at-rule creates a layer, named or not, or appends styles to a layer if the named layer already exists. We called the first anonymous layer `<anonymous(01)>` and the second `<anonymous(02)>`, this is just so we can explain them. These are actually unnamed layers. There is no way to reference them or add additional styles to them.

All styles declared outside of a layer are joined together in an implicit layer. In the example code above, the first declaration set the `color: #333` property on `body`. This was declared outside of any layer. Normal unlayered declarations take precedence over the normal layered declarations even if the unlayered styles have a lower specificity and come first in the order of appearance. This explains why even though the unlayered CSS was declared first in the code block, the implicit layer containing these unlayered styles takes precedence as if it was the last declared layer.

In the line `@layer theme, layout, utilities;`, in which a series of layers were declared, only the `theme` and `utilities` layers were created; `layout` was already created in the first line. Note that this declaration does not change the order of already created layers. There is currently no way to re-order layers once declared.

In the following interactive example, we assign styles to two layers, creating them and naming them in the process. Because they already exist, being created when first used, declaring them on the last line does nothing.

Is this heading underlined?

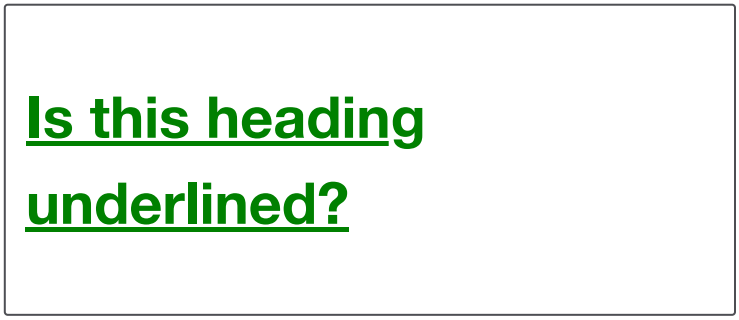
Interactive editor

```
@layer page {
  h1 {
    text-decoration: overline;
    color: red;
  }
}

@layer site {
  h1 {
    text-decoration: underline;
    color: green;
  }
}

/* this does nothing */
@layer site, page;
```

Try moving the last line, `@layer site, page;` , to make it the first line. What happens?

Layer creation and media queries

If you define a layer using media or feature queries, and the media is not a match or the feature is not supported, the layer is not created. The example below shows how changing the size of your device or browser may change the layer order. In this example, we create the `site` layer only in wider browsers. We then assign styles to the `page` and `site` layers, in that order.



Interactive editor

```
@media (min-width: 50em) {
  @layer site;
}

@layer page {
  h1 {
    text-decoration: overline;
    color: red;
```

```
  <h1>Is this heading
underlined?</h1>
```

Reset

In wide screens, the `site` layer is declared in the first line, meaning `site` has less precedence than `page`. Otherwise, `site` has precedence over `page` because it is declared later on narrow screens. If that doesn't work, try changing the `50em` in the media-query to `10em` or `100em`.

Importing style sheets into named and anonymous layers with @import

The `@import` rule allows users to import style rules from other style sheets either directly into a CSS file or into a `<style>` element.

When importing stylesheets, the `@import` statement must be defined before any CSS styles within the stylesheet or `<style>` block. The `@import` statement must come first, before any styles, but can be preceded by an `@layer` at-rule that creates one or more layers

without assigning any styles to the layers. (`@import` can also be preceded by an `@charset` rule.)

You can import a stylesheet into a named layer, a nested named layer, or an anonymous layer. The following layer imports the style sheets into a `components` layer, a nested `dialog` layer within the `components` layer, and an un-named layer, respectively:

```
@import url("components-lib.css") layer(components);
@import url("dialog.css") layer(components.dialog);
@import url("marketing.css") layer();
```

You can import more than one CSS file into a single layer. The following declaration imports two separate files into a single `social` layer:

```
@import url(comments.css) layer(social);
@import url(sm-icons.css) layer(social);
```

You can import styles and create layers based on specific conditions using media queries and feature queries. The following imports a style sheet into an `international` layer only if the browser supports `display: ruby` , and the file being imported is dependent on the width of the screen.

```
@import url("ruby-narrow.css") layer(international)
supports(display: ruby) and
  (width < 32rem);
@import url("ruby-wide.css") layer(international)
supports(display: ruby) and
  (width >= 32rem);
```

> **Note:** There is no equivalent of the `<link>` method of linking stylesheets. Use `@import` to import a stylesheet into a layer when you can't use `@layer` within the stylesheet.

Overview of nested cascade layers

Nested layers are layers within a named or an anonymous layer. Each cascade layer, even an anonymous one, can contain nested layers. Layers imported into another layer become nested layers within that layer.

Advantages of nesting layers

The ability to nest layers enables teams to create cascade layers without worrying about whether other teams will import them into a layer. Similarly, nesting enables you to import third party style sheets into a layer without worrying if that style sheet itself has layers. Because layers can be nested, you don't have to worry about having conflicting layer names between external and internal style sheets.

Creating nested cascade layers

Nested layers can be created using the same methods as described for regular layers. For example, they can be created using `@layer` at-rule followed by the names of one or more layers, using a dot notation. Multiple dots and layer names signify multiple nesting.

If you nest a block `@layer` at-rule inside another block `@layer` at-rule, with or without a name, the nested block becomes a nested layer. Similarly, when a style sheet is imported with an `@import` declaration containing the `layer` keyword or `layer()` function, the styles get assigned to that named or anonymous layer. If the `@import` statement contains layers, those layers become nested layers within that anonymous or named layer.

Let's look at the following example:

```
@import url("components-lib.css") layer(components);
@import url("narrowtheme.css") layer(components.narrow);
```

In the first line, we import `components-lib.css` into the `components` layer. If that file contains any layers, named or not, those layers become nested layers within the `components` layer.

The second line imports `narrowtheme.css` into the `narrow` layer, which is a sub-layer of `components`. The nested `components.narrow` gets created as the last layer within the `components` layer, unless `components-lib.css` already contains a `narrow` layer, in which case, the contents of `narrowtheme.css` would be appended to the `components.narrow` nested layer. Additional nested named layers can be added to the `components` layer using the pattern `components.<layerName>`. As mentioned before, unnamed layers can be created but they cannot be accessed subsequently.

Let's look at another example, where we import `layers1.css` into a named layer using the following statement:

```
@import url(layers1.css) layer(example);
```

This will create a single layer named `example` containing some declarations and five nested layers - `example.layout`, `example.<anonymous(01)>`, `example.theme`, `example.utilities`, and `example.<anonymous(02)>`.

To add styles to a named nested layer, use the dot notation:

```
@layer example.layout {
  main {
    width: 50vw;
  }
}
```

Determining the precedence based on the order of layers

The order of layers determines their order of precedence. Therefore, the order of layers is very important. In the same way as the cascade

sorts by origin and importance, the cascade sorts each CSS declaration by origin layer and importance.

Precedence order of regular cascade layers

```
@import url(A.css) layer(firstLayer);
@import url(B.css) layer(secondLayer);
@import url(C.css);
```

The above code creates two named layers and one unnamed layer. Let us assume that the three files (`A.css` , `B.css` , and `C.css`) do not contain any additional layers within them. The following list shows where styles declared inside and outside of these files will be sorted from least (1) precedence to highest (10).

1. firstLayer normal styles (`A.css`)
2. secondLayer normal styles (`B.css`)
3. unlayered normal styles (`C.css`)
4. inline normal styles
5. animating styles
6. unlayered important styles (`C.css`)
7. secondLayer important styles (`B.css`)
8. firstLayer important styles (`A.css`)
9. inline important styles
10. transitioning styles

Normal styles declared inside layers receive the lowest priority and are sorted by the order in which the layers were created. Normal styles in the first created layer have the lowest precedence, and normal styles in the layer created last have the highest precedence among the layers. In other words, normal styles declared within `firstLayer` will be overridden by any subsequent stylings on the list if any conflicts exist.

Next up are any styles declared outside of layers. The styles in `C.css` were not imported into a layer and will override any conflicting styles from `firstLayer` and `secondLayer`. Normal styles not declared in a layer always have higher precedence than normal-importance layered styles.

Inline styles are declared using the `style` attribute. Normal styles declared in this way will take precedence over normal styles found in the unlayered and layered style sheets (`firstLayer — A.css` , `secondLayer — B.css` , and `C.css`).

Animating styles have higher precedence than all normal styles, including inline normal styles.

Important styles, that is, property values that include the `!important` flag, take precedence over any styles previously mentioned in our list. They are sorted in reverse order of normal styles. Any important styles declared outside of a layer have less precedence than those declared within a layer. Important styles found within layers are also sorted in order of layer creation. For important styles, the last created layer has the lowest precedence, and the first created layer has the highest precedence among declared layers.

Inline important styles again have higher precedence than important styles declared elsewhere.

Transitioning styles have the highest precedence. When a normal property value is being transitioned, it takes precedence over all other property value declarations, even inline important styles; but only while transitioning.

In this example, there are two inline layers `A` and `B` without styles, a block of unlayered styles, and two blocks of styles in named layers `A` and `B`.

The inline styles added on the `h1` element using the `style` attribute, set a normal `color` and an important `background-color`. Normal inline styles override all layered and unlayered normal styles. Important inline styles override all layered and unlayered normal and important author styles. There is no way for author styles to override important inline styles.

The normal `text-decoration` and important `box-shadow` are not part of the `style` inline styles and can therefore be overridden. For normal non-inline styles, unlayered styles have precedence. For important styles, layer order matters too. While normal unlayered styles override all normal styles set in a layer, with important styles, the precedence

order is reversed; unlayered important styles have lower precedence than layered styles.

The two styles declared only within layers are `font-style`, with normal importance, and `font-weight` with an `!important` flag. For normal styles, the `B` layer, declared last, overrides styles in the earlier declared layer `A`. For normal styles, later layers have precedence over earlier layers. The order of precedence is reversed for important styles. For the important `font-weight` declarations, layer `A`, being declared first, has precedence over the last declared layer `B`.

You can reverse the layer order by changing the first line from `@layer A, B;` to `@layer B, A;`. Try that. Which styles get changed by this, and which stay the same? Why?

The order of layers is set by the order in which the layers appear in your CSS. In our first line, we declared layers without assigning any styles using `@layer` followed by the names of our layers, ending with a semi-colon. Had we omitted this line, the results would have been the same. Why? We assigned styles rules in named @layer blocks in the order A then B. The two layers were created in that first line. Had they not been, these rule blocks would have created them, in that order.

We included that first line for two reasons: first, so you could easily edit the line and switch the order, and second, because often times you'll find declaring the order layer up front to be the best practice for your layer order management.

To summarize:

- The order of precedence of layers is the order in which the layers are created.
- Once created, there is no way to change the layer order.

- Layer precedence for normal styles is the order in which the layers are created.
- Unlayered normal styles have precedence over normal layered styles.
- Layer precedence for important styles is reversed, with earlier created layers having precedence.
- All layered important styles have precedence over unlayered important (and normal) styles.
- Normal inline styles take precedence over all normal styles, layered or not.
- Important inline styles take precedence over all other styles, with the exception of styles being transitioned.
- There is no way for author styles to override important inline styles (other than transitioning them, which is temporary).

Precedence order of nested cascade layers

The cascade precedence order for nested layers is similar to that of regular layers, but contained within the layer. The precedence order is based on the order of nested layer creation. Non-nested styles in a layer have precedence over nested normal styles, with the precedence order reversed for important styles. Specificity weight between nested layers does not matter, though it does matter for conflicting styles within a nested layer.

The following creates and adds styles to the `components` layer and `components.narrow` nested layer and creates and appends styles to a new `components.wide` layer:

```
@import url("components-lib.css") layer(components);
@import url("narrowtheme.css") layer(components.narrow);

@layer components {
  :root {
    --theme: red;
    font-family: serif !important;
  }
```

```
}
@layer components.narrow {
  :root {
    --theme: blue;
    font-family: sans-serif !important;
  }
}
@layer components.wide {
  :root {
    --theme: purple;
    font-family: cursive !important;
  }
}
```

Because unlayered normal styles have precedence over layered normal styles, and within a layer, non-nested styles have precedence over normal nested styles, `red` wins over the other `theme` colors.

With important styles, layered styles take precedence over unlayered styles, with important styles in earlier declared layers having precedence over later declared layers. In this example, the order of nested layer creation is `components.narrow`, then `components.wide`, so important styles in `components.narrow` have precedence over important styles in `components.wide`, meaning `sans-serif` wins.

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, Task 2.

Summary

If you understood most of this article, then well done — you're now familiar with the fundamental mechanics of CSS cascade layers. Next up, we'll look at the box model in detail.

This page was last modified on Apr 12, 2023 by MDN contributors.