CSS 初级入门

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。

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。这些影响主要体现在两点：层叠性以及继承性。

开胃菜

CSS 层叠性

CSS 的层叠性取决于三个因素（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）：

1. 重要性（Importance）
2. 专用性（Specificity）
3. 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`。

重载 `!important` 声明的唯一方法是在后面的源码或者是一个拥有更高专用性的源码中包含相同的 `!important` 特性的声明。所以建议不要使用该特性，否则代码会特别难维护。

我很重要

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：

1. 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)。
2. 用户样式表中的普通声明（由用户设置的自定义样式）。
3. 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）。
4. 作者样式表中的重要声明
5. 用户样式表中的重要声明

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。通俗来说，就是我们通过计算各个选择器的权重值，然后互相比较得到各自的生效范围。这部分是比较常用的，我们日常写代码处理的最多的就是各种选择器的权重比较。

一般来说，我们计算权重有下面几个规则：

1. `style` **属性** 的权值为 1000，**ID 选择器**的权值为 100，**类选择器、属性选择器、或者伪类** 的权值为 10，**元素选择器或伪元素** 的权值为 1 ， **通用选择器 (`*`), 复合选择器 (`+`, `>`, `~`, ' ') 和否定伪类 (`:not`)** 权值 为 0，也就是没有影响。 一道计算题
2. 如果计算出的权重相同，则取决于源代码次序。
3. 权值的计算不会发生进位情况，举例来说就是：100 个 **类选择器** 也打不过一个 **ID 选择器** 。

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。

选择器	千位	百位	十位	个位	合计值
`h1`	0	0	0	1	0001
`#indentifier`	0	1	0	0	0100
`h1 + p::first-letter`	0	0	0	3	0003
`li > a[href*="zh-CN"] > .inline-warning`	0	0	2	2	0022
没有选择器, 规则在一个元素的 `` 属性里	1	0	0	0	1000

小孩子才比来比去

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。 近水楼台先得月

继承

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。具体的那些默认被继承哪些不被继承大部分符合常识，不太确定的可以参考CSS参考资料—— 每个单独的属性页都会从一个汇总表开始，其中包含有关该元素的各种详细信息，包括是否被继承，或者直接写一下然后在浏览器中看一下效果。

- 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 `<htm>` 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！

哪些属性默认被继承哪些不被继承大部分符合常识。如果你想确定，你可以参考CSS参考资料—— 每个单独的属性页都会从一个汇总表开始，其中包含有关该元素的各种详细信息，包括是否被继承。

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` ： 将该属性恢复到它没有设置任何样式时的值。

> **注意**: `initial` 和 `unset` 不被IE支持。 `revert`仅被 Firefox 和 Safari 支持。

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。

Vue 中的 `scoped`

Vue 中的 CSS 并没有其他特殊规则，属性是否生效也是使用上面的那些规则。

这时就有人要提出反对意见了，不对啊，Vue 里面 `<style>` 标签有一个 `scoped` 属性老神奇了，加上之后攻击力 +max ，就算是 ID 选择器都打不过子组件里面的标签选择器。

下面我们就说说 `scoped`。

当 `<style>` 标签有 `scoped` 属性时，它的 CSS 只作用于当前组件中的元素。它通过使用 PostCSS 来实现以下转换：

```
<style scoped>
.example {
  color: red;
}
</style>

<template>
  <div class="example">hi</div>
</template>
```

转换结果：

```
<style>
.example[data-v-f3f3eg9] {
  color: red;
}
</style>

<template>
  <div class="example" data-v-f3f3eg9>hi</div>
</template>
```

也就是说，`scoped` 属性的工作原理是：通过给每个组件的 HTML 节点加上不同的特定属性值，同时将有 `scoped` 属性的 `<style>` 标签内部的所有选择器加上对应的属性选择器，这样就实现了该样式只会对该组件生效，毕竟选中了才能进行权重的比较。万变不离其宗

那么我们既要在父组件中使用 `scoped` 属性，又要重置子组件的样式要怎么做呢？

如果你希望 `scoped` 样式中的一个选择器能够作用得“更深”，例如影响子组件，你可以使用 `>>>` 操作符：

```
<style scoped>
.a >>> .b { /* ... */ }
</style>
```

上述代码将会编译成：

```
.a[data-v-f3f3eg9] .b { /* ... */ }
```

有些像 Sass 之类的预处理器无法正确解析 `>>>`。这种情况下你可以使用 `/deep/` 或 `::v-deep` 操作符取而代之——两者都是 `>>>` 的别名，同样可以正常工作。

CSS 预处理器

预处理器有：Sass（SCSS）、Less、Stylus，这三者功能基本相同，写法略有差异，在这里不对这些预处理器做评论，讲到这些预处理器也是想讲一讲其中的嵌套 CSS 规则部分。下面以 Sass 为例。

`css`中重复写选择器是非常恼人的。如果要写一大串指向页面中同一块的样式时，往往需要 一遍又一遍地写同一个`ID`：

```
#content article h1 { color: #333 }
#content article p { margin-bottom: 1.4em }
#content aside { background-color: #EEE }
```

像这种情况，`sass`可以让你只写一遍，且使样式可读性更高。在Sass中，你可以像俄罗斯套娃那样在规则块中嵌套规则块。`sass`在输出`css`时会帮你把这些嵌套规则处理好，避免你的重复书写。

```
#content {
  article {
    h1 { color: #333 }
    p { margin-bottom: 1.4em }
  }
  aside { background-color: #EEE }
}
 /* 编译后 */
#content article h1 { color: #333 }
#content article p { margin-bottom: 1.4em }
#content aside { background-color: #EEE }
```

诶，すごい（厉害）~

但是，好多同学就完全搞错了意思，不是说避免重复书写嘛，管他三七二十一，只要有自然嵌套我就用嵌套写法。咋一看好像没什么问题，但是，凡事都怕但是，你真的需要重复写选择器嘛？你本来就只用写一层的好吧，没有任何理由增加权重，因为基本没有其他选择器来和你的样式进行竞争。

恐怖的嵌套

下面列一下使用嵌套的坏处：

1. 增加重置样式的成本，需要很高的权重才能重置，而这本来是没有必要的。
2. 降低性能。没错，嵌套写法性能更低，因为浏览器解析子代选择器是从右到左解析的。例如：.main .item p 的解析步骤如下：
    1. 查找到页面上的所有 p 标签元素！！！
    2. 依次判断该元素的祖先元素上有没有 item 类名。
    3. 再次判断再上层祖先元素有没有 main 类名（浏览器要累死了好吧）。
    4. 如果还有嵌套就再网上找，也就是多一层就多一次判断，而不是本来以为的缩小范围。。。

完结撒花

不鼓个掌嘛、、、