前言

有试过样式覆盖导致排版出问题或布局时不知使用什么长度单位声明属性吗？这些问题应该都是比较常见的问题，笔者有位同事遇到样式覆盖的问题时一言不合使用 `!important` 暴力解决，这样真的好吗？

优先级别

样式覆盖的根本原因是未处理好规则间的优先级别，虽然使用 `!important` 能解决问题，但是不能什么情况都由 `!important` 暴力解决。

为何样式需优先级别呢？当创建的样式越来越复杂时，一个节点的样式将会受到越来越多的影响，这种影响可能来自周围节点也可能来自自身。通过相关规范分配给规则一个权重，那么样式可按照权重计算，呈现页面最终的效果。

从以下几方面了解优先级别，相信能更好把握优先级别的使用场景。

特点

- 使用 就近原则
- 继承样式 的优先级别最低
- `!important样式` 的优先级别最高，若冲突则重新计算

权重

> 直观权重

- **10000**： `!important`
- **1000**： 内联样式 、 外联样式
- **100**： ID选择器
- **10**： 类选择器 、 伪类选择器 、 属性选择器
- **1**： 元素选择器 、 伪元素选择器
- **0**： 通配选择器 、 后代选择器 、 兄弟选择器

> 微观权重

- **1,0,0,0,0**： `!important`
- **0,1,0,0,0**： 内联样式 、 外联样式
- **0,0,1,0,0**： ID选择器
- **0,0,0,1,0**： 类选择器 、 伪类选择器 、 属性选择器
- **0,0,0,0,1**： 元素选择器 、 伪元素选择器
- **0,0,0,0,0**： 通配选择器 、 后代选择器 、 兄弟选择器

总体来说，直观权重和微观权重只是表达方式不同，实际意义还是一致的，使用公式可表达成这样。

`!important > 内联样式 = 外联样式 > ID选择器 > 类选择器 = 伪类选择器 = 属性选择器 > 元素选择器 = 伪元素选择器 > 通配选择器 = 后代选择器 = 兄弟选择器`

计算

> 优先级别相同的规则使用最后出现的规则

```
.input-box {
    color: #f66;
}
:focus {
    color: #66f;
}
[type=text] {
    color: #f90;
}
```

虽然 `类选择器` 、 `伪类选择器` 和 `属性选择器` 三者的优先级别相同，但是最后出现的规则其优先级别最高，所以 `<input>` 最终会显示 `#f90` 。

> 优先级别无视节点在DOM树中的距离

```
html h1 {
    color: #f66;
}
body h1 {
    color: #66f;
}
```

虽然 `<html>` 包含着 `<body>` ，但是依据就近原则，所以 `<h1>` 最终会显示 `#66f` 。

> 不同规则作用于相同节点使用优先级别最高的规则

```
#bruce {
    color: #f66;
}
[id=bruce] {
    color: #66f;
}
```

虽然两者规则都作用于ID为bruce的 `<div>` ，但是 `ID选择器` 的优先级比 `属性选择器` 高，所以 `<div>` 最终会显示 `#f66` 。

> :not()不参与优先级别的计算

`:not()` 在优先级别计算中不会被看成 `伪类` ，但是会把 `:not()` 里的选择器当作普通选择器计数。简单来说就是忽略 `:not()` ，其他伪类照常参与优先级别计算。

规则

- 规则的权值不同时，权值高的规则优先
- 规则的权值相同时，后定义的规则优先
- 属性后面追加 `!important` 时，规则无条件绝对优先

长度单位

粗糙的干活可能只需 `px` 和 `%` 两个长度单位即可，随着终端设备分辨率的多样性，CSS衍生出越来越多的长度单位，灵活结合这些长度单位能为页面的布局方案提供更多可能性。

单位	定义	类型	描述
px	像素	绝对单位	-
pt	点	绝对单位	1pt = 1/72in
pc	派	绝对单位	1pc = 12pt
mm	毫米	绝对单位	-
单位	定义	类型	描述
cm	厘米	绝对单位	-

in	英寸	绝对单位	`1in = 96px = 2.54cm`
%	百分比	相对单位	相对父节点，宽度对应，高度不一定对应
em	M的宽度	相对单位	相对当前节点字体
rem	M的宽度	相对单位	相对根结点字体
ch	0的宽度	相对单位	相对当前节点字体
ex	x的宽度	相对单位	相对当前节点字体
vw	1%视窗宽度	相对单位	相对视窗
vh	1%视窗高度	相对单位	相对视窗
vmin	vw/vh最小者	相对单位	相对视窗
vmax	vw/vh最大者	相对单位	相对视窗

这么多单位，到底如何区别呢？首先要明确一点，那就是**屏幕分辨率**。

屏幕分辨率指横纵向上的像素点数，单位是 `px` 。 屏幕分辨率 确定计算机屏幕上能显示多少信息的，以水平和垂直像素衡量。屏幕尺寸一致的情况下， 屏幕分辨率 越低在屏幕上显示的像素就越少，单个像素尺寸也比较大， 屏幕分辨率 越高在屏幕上显示的像素越多，单个像素尺寸也比较小。

屏幕分辨率 就是屏幕上显示的像素个数，分辨率 `1920×1080` 意味着水平方向含有 `1920` 个像素数，垂直方向含有 `1080` 个像素数。屏幕尺寸一致的情况下， 屏幕分辨率 越高，显示效果就越细腻。这也是为何 `iPhone` 经常亮瞎眼睛的原因。

所以在同一个网页里，以 `px` 作为长度单位时，在不同 屏幕分辨率 下显示的大小是不同的。在低 屏幕分辨率 下像素比较大，显示的页面元素也偏大偏模糊。实际上，所有单位无论是 绝对单位 还是 相对单位 ，最终

都是转化为 px 在屏幕上显示。因此在设计和开发过程中都以 px 为准。

> em/rem区别

em 和 rem 是移动端布局上常用的长度单位，两者的后缀都一致。 rem 全称是 root em ，意思是相对根节点作为参考的长度单位。

- **em**：当前节点字体宽度，准确来说是一个M的宽度
- **rem**：默认字体宽度，准确来说是一个M的宽度

两者区别在于：**em相对父节点，rem相对根节点**。 em 以当前节点字体宽度作为参考， rem 以根节点 <html> 字体宽度作为参考，默认是 16px 。很多同学错误地以为 em 是根据父节点作为参考的，实际上是当前节点继承了父节点的属性后产生的错觉。

em 和 rem 都是很灵活且可扩展的长度单位，由浏览器转换为 px ，具体取决于设计图中的字体大小。

针对移动端，笔者通常会结合JS依据屏幕宽度与设计图宽度的比例动态声明 <html> 的 font-size ，以 rem 为长度单位声明所有节点的几何属性，这样就能做到大部分移动设备的页面兼容，兼容出入比较大的地方再通过 媒体查询 做特别处理。

```
function AutoResponse(width = 750) {
    const target = document.documentElement;
    if (target.clientWidth >= 600) {
        target.style.fontSize = "80px";
    } else {
        target.style.fontSize = target.clientWidth / width * 100 + "px";
    }
}

AutoResponse();
```

前提还需在 <html> 中声明以下代码，阻止用户缩放屏幕。

```
<meta name="viewport" content="width=device-width, user-scalable=no, in
```

> 视窗比例单位

在CSS3中增加了与 viewport 相关的四个长度单位，随着时间推移，目前大部分浏览器对这四个长度单位都有比较好的兼容，这也是未来最建议在伸缩方案中使用的长度单位。

- 1vw 表示 1% 视窗宽度
- 1vh 表示 1% 视窗高度
- 1vmin 表示 1% 视窗宽度和 1% 视窗高度中最小者
- 1vmax 表示 1% 视窗宽度和 1% 视窗高度中最大者

视窗宽高在JS中分别对应 window.innerWdith 和 window.innerHeight 。若不考虑低版本浏览器的兼容，完全可用一行CSS代码秒杀所有移动端的伸缩方案。

```
/* 基于UI width=750px DPR=2的页面 */
html {
    font-size: calc(100vw / 7.5);
}
```

这是 calc() 的一个神操作，在第7章**函数计算**会详细讲解 calc() 怎么玩。这行CSS代码就留个给位同学思考，为何这样处理能，细心的同学可能发现这段代码可代替上述那段JS代码。