CSS 样式书写规范

2017-07-19 伯乐在线 前端大全

（点击上方公众号，可快速关注）

> 作者：伯乐在线/chokcoco
> http://web.jobbole.com/91792/
> 如有好文章投稿，请点击 → 这里了解详情

可能不同团队都有各自的规范，又或者很多人在写 CSS 的时候还是想到什么就写什么，不存在太多的约束。

我觉得 CSS 代码规范还是有存在的必要的，尤其是在团队配合，多人协作下，规范就显得尤为重要。

本文的所列是实践当中得出的一套比较不错的 CSS 书写规范，并不希望大家完全采用，而是希望可以结合自己的团队需要，发展出一套适合自己的 CSS 代码规范。

也希望可以有更多的建议，共同的完善。本规范也可以在我的 Github 上看到，欢迎留言或者提 PR。

> 我觉得不同的规范都有各自的长处与缺陷，对待所谓的规范最好的方式不是人云亦云，拿来就用，而是应该结合实际情况及需求，取长补短，取其精华去其糟粕。

编码设置

采用 UTF-8 编码，在 CSS 代码头部使用：

```
@charset "utf-8";
```

注意，必须要定义在 CSS 文件所有字符的前面（包括编码注释），@charset 才会生效。

例如，下面的例子都会使得 @charset 失效：

```
/* 字符编码 */
@charset "utf-8";
html,
body {
  height: 100%;
}
```

```
@charset "utf-8";
```

命名空间规范

- 布局：以 g 为命名空间，例如：.g-wrap 、.g-header、.g-content。
- 状态：以 s 为命名空间，表示动态的、具有交互性质的状态，例如：.s-current、s-selected。
- 工具：以 u 为命名空间，表示不耦合业务逻辑的、可复用的的工具，例如：u-clearfix、u-ellipsis。
- 组件：以 m 为命名空间，表示可复用、移植的组件模块，例如：m-slider、m-dropMenu。
- 钩子：以 j 为命名空间，表示特定给 JavaScript 调用的类名，例如：j-request、j-open。

命名空间思想

没有选择 BEM 这种命名过于严苛及样式名过长过丑的规则，采取了一种比较折中的方案。

不建议使用下划线 _ 进行连接

- 节省操作，输入的时候少按一个 shift 键
- 能良好区分 JavaScript 变量命名

字符小写

定义的选择器名，属性及属性值的书写皆为小写。

选择器

当一个规则包含多个选择器时，每个选择器独占一行。

> 、+、~、> 选择器的两边各保留一个空格。

```
.g-header > .g-header-des,
.g-content ~ .g-footer {

}
```

命名短且语义化良好

对于选择器的命名，尽量简洁且具有语义化，不应该出现 g-abc 这种语义模糊的命名。

规则声明块

- 当规则声明块中有多个样式声明时，每条样式独占一行。
- 在规则声明块的左大括号 { 前加一个空格。
- 在样式属性的冒号 : 后面加上一个空格，前面不加空格。
- 在每条样式后面都以分号 ; 结尾。
- 规则声明块的右大括号 } 独占一行。
- 每个规则声明间用空行分隔。
- 所有最外层引号使用单引号 ‘ 。
- 当一个属性有多个属性值时，以逗号 , 分隔属性值，每个逗号后添加一个空格，当单个属性值过长时，每个属性值独占一行。

完整示例如下：

```
.g-footer,
.g-header {
  position: relative;
}

.g-content {
  background:
    linear-gradient(135deg, deeppink 25%, transparent 25%) -50px 0,
    linear-gradient(225deg, deeppink 25%, transparent 25%) -50px 0,
    linear-gradient(315deg, deeppink 25%, transparent 25%),
    linear-gradient(45deg, deeppink 25%, transparent 25%);
  }

.g-content::before {
  content: '';
}
```

数值与单位

- 当属性值或颜色参数为 0 – 1 之间的数时，省略小数点前的 0 。color: rgba(255, 255, 255, 0.5)color: rgba(255, 255, 255, .5);
- 当长度值为 0 时省略单位。margin: 0px automargin: 0 auto
- 十六进制的颜色属性值使用小写和尽量简写。color: #ffcc00color: #fc0

样式属性顺序

单个样式规则下的属性在书写时，应按功能进行分组，并以 Positioning Model > Box Model > Typographic > Visual 的顺序书写，提高代码的可读性。

- 如果包含 content 属性，应放在最前面；
- Positioning Model 布局方式、位置，相关属性包括：position / top / right / bottom / left / z-index / display / float / ...
- Box Model 盒模型，相关属性包括：width / height / padding / margin / border / overflow / ...
- Typographic 文本排版，相关属性包括：font / line-height / text-align / word-wrap / ...
- Visual 视觉外观，相关属性包括：color / background / list-style / transform / animation / transition / ...

Positioning 处在第一位，因为他可以使一个元素脱离正常文本流，并且覆盖盒模型相关的样式。盒模型紧跟其后，因为他决定了一个组件的大小和位置。其他属性只在组件内部起作用或者不会对前面两种情况的结果产生影响，所以他们排在后面。

合理使用使用引号

在某些样式中，会出现一些含有空格的关键字或者中文关键字。

font-family 内使用引号

当字体名字中间有空格，中文名字体及 Unicode 字符编码表示的中文字体，为了保证兼容性，都建议在字体两端添加单引号或者双引号：

```
body {
  font-family: 'Microsoft YaHei', '黑体-简', '\5b8b\4f53';
}
```

background-image 的 url 内使用引号

如果路径里面有空格，旧版 IE 是无法识别的，会导致路径失效，建议不管是否存在空格，都添加上单引号或者双引号：

```
div {
  background-image: url('...');
}
```

避免使用 !important

除去某些极特殊的情况，尽量不要不要使用 !important。

!important 的存在会给后期维护以及多人协作带来噩梦般的影响。

当存在样式覆盖层叠时，如果你发现新定义的一个样式无法覆盖一个旧的样式，只有加上 !important 才能生效时，是因为你新定义的选择器的优先级不够旧样式选择器的优先级高。所以，合理的书写新样式选择器，是完全可以规避一些看似需要使用 !important 的情况的。

代码注释

单行注释

星号与内容之间必须保留一个空格。

```
/* 表格隔行变色 */
```

多行注释

星号要一列对齐，星号与内容之间必须保留一个空格。

```
/**
* Sometimes you need to include optional context for the entire component. Do that up here if it's important enough.
*/
```

规则声明块内注释

使用 // 注释，// 后面加上一个空格，注释独立一行。

```
.g-footer {
    border: 0;
    // ....
}
```

文件注释

文件顶部必须包含文件注释，用 @name 标识文件说明。星号要一列对齐，星号与内容之间必须保留一个空格，标识符冒号与内容之间必须保留一个空格。

```
/**
* @name: 文件名或模块名
* @description: 文件或模块描述
```

```
* @author: author-name(mail-name@domain.com)
*          author-name2(mail-name2@domain.com)
* @update: 2015-04-29 00:02
*/
```

- @description为文件或模块描述。
- @update为可选项，建议每次改动都更新一下。

当该业务项目主要由固定的一个或多个人负责时，需要添加@author标识，一方面是尊重劳动成果，另一方面方便在需要时快速定位责任人。

SASS 使用建议

嵌套层级规定

使用 SASS 、 LESS 等预处理器时，建议嵌套层级不超过 3 层。

组件/公用类的使用方法

组件/公用类使用 %placeholders 定义，使用 @extend 引用。如：

```
%clearfix {

  overflow: auto;
  zoom: 1;
}

.g-header {
  @extend %clearfix;
}
```

组件类的思考

使用 SASS ，经常会预先定义好一些常用公用组件类，譬如清除浮动，水平垂直居中，文字 ellipsis。又或者多个元素具有同样的样式，我们希望能够少写这部分代码，公共部分抽离出来只写一次，达到复用。

但是复用的方式在 SASS 中有多种，那么是使用单独使用一个类定义，给需要的标签添加，还是使用 @include 或者 @extend在定义的类中引入一个 @mixin，或者一个 @function 呢?

基于让 CSS 更简洁以及代码的复用考虑，采用上面的使用 %placeholders 定义，使用 @extend 引用的方案。

- %placeholders，只是一个占位符，只要不通过 @extend 调用，编译后不会产生任何代码量
- 使用 @extend 引用，则是因为每次调用相同的 %placeholders 时，编译出来相同的 CSS 样式会进行合并（反之，如果使用 @include 调用定义好的 @mixin，编译出来相同的 CSS 样式不会进行合并）
- 这里的组件类特指那些不会动态改变的 CSS 样式，注意与那些可以通过传参生成不同数值样式的 @mixin 方法进行区分

尽量避免使用标签名

使用 SASS ，或者说在 CSS 里也有这种困惑。

假设我们有如下 html 结构：

```
<span>
 <div class="g-content">
  <ul class="g-content-list"><li class="item"/>
     <li class="item"/>
     <li class="item"/>
     <li class="item"/>
  </ul></div>
</span>
```

在给最里层的标签命名书写样式的时候，我们有两种选择：

```
.g-content {
 .g-content-list {
  li {
   ...
  }
 }
}
```

或者是

```
.g-content {
 .g-content-list {
  .item {
```

```
    ...
  }
 }
}
```

也就是，编译之后生成了下面这两个，到底使用哪个好呢？

- .g-content .g-content-list li { }
- .g-content .g-content-list .item { }

基于 CSS 选择器的解析规则（从右向左），建议使用上述第二种 .g-content .g-content-list .item { }，避免使用通用标签名作为选择器的一环可以提高 CSS 匹配性能。

> 浏览器的排版引擎解析 CSS 是基于从右向左（right-to-left）的规则，这么做是为了使样式规则能够更快地与渲染树上的节点匹配。

本规范也可以在我的 Github （https://github.com/chokcoco/CSSWritingRules）上看到，欢迎留言或者提 PR。

到此本文结束，如果还有什么疑问或者建议，可以多多交流，原创文章，文笔有限，才疏学浅，文中若有不正之处，万望告知。