✍

编码规范 by @mdo

编写灵活、稳定、高质量的 HTML 和 CSS 代码的规范。(2016.08.09)

目录

[HTML](http://codeguide.bootcss.com/#html)

* [语法](http://codeguide.bootcss.com/#html-syntax)
* [HTML5 doctype](http://codeguide.bootcss.com/#html-doctype)
* [语言属性（Language attribute）](http://codeguide.bootcss.com/#html-lang)
* [字符编码](http://codeguide.bootcss.com/#html-encoding)
* [IE 兼容模式](http://codeguide.bootcss.com/#html-ie-compatibility-mode)
* [引入 CSS 和 JavaScript 文件](http://codeguide.bootcss.com/#html-style-script)
* [实用为王](http://codeguide.bootcss.com/#html-practicality)
* [属性顺序](http://codeguide.bootcss.com/#html-attribute-order)
* [布尔（boolean）型属性](http://codeguide.bootcss.com/#html-boolean-attributes)
* [减少标签的数量](http://codeguide.bootcss.com/#html-reducing-markup)
* [JavaScript 生成的标签](http://codeguide.bootcss.com/#html-javascript)

[CSS](http://codeguide.bootcss.com/#css)

* [语法](http://codeguide.bootcss.com/#css-syntax)
* [声明顺序](http://codeguide.bootcss.com/#css-declaration-order)
* [媒体查询（Media query）的位置](http://codeguide.bootcss.com/#css-media-queries)
* [带前缀的属性](http://codeguide.bootcss.com/#css-prefixed-properties)
* [单行规则声明](http://codeguide.bootcss.com/#css-single-declarations)
* [简写形式的属性声明](http://codeguide.bootcss.com/#css-shorthand)
* [Less 和 Sass 中的嵌套](http://codeguide.bootcss.com/#css-nesting)
* [注释](http://codeguide.bootcss.com/#css-comments)
* [class 命名](http://codeguide.bootcss.com/#css-classes)
* [选择器](http://codeguide.bootcss.com/#css-selectors)
* [代码组织](http://codeguide.bootcss.com/#css-organization)

黄金定律

永远遵循同一套编码规范 -- 可以是这里列出的，也可以是你自己总结的。如果你发现本规范中有任何错误，敬请指正。通过 [open an issue on GitHub](https://github.com/mdo/code-guide/issues/new)为本规范添加或贡献内容。

*不管有多少人共同参与同一项目，一定要确保每一行代码都像是同一个人编写的。*

HTML

语法

* 用两个空格来代替制表符（tab） -- 这是唯一能保证在所有环境下获得一致展现的方法。
* 嵌套元素应当缩进一次（即两个空格）。
* 对于属性的定义，确保全部使用双引号，绝不要使用单引号。
* 不要在自闭合（self-closing）元素的尾部添加斜线 -- [HTML5 规范](http://dev.w3.org/html5/spec-author-view/syntax.html#syntax-start-tag)中明确说明这是可选的。
* 不要省略可选的结束标签（closing tag）（例如，</li> 或</body>）。

<!DOCTYPE html>

<html>

<head>

<title>Page title</title>

</head>

<body>

<img src="images/company-logo.png" alt="Company">

<h1 class="hello-world">Hello, world!</h1>

</body>

</html>

HTML5 doctype

为每个 HTML 页面的第一行添加标准模式（standard mode）的声明，这样能够确保在每个浏览器中拥有一致的展现。

<!DOCTYPE html>

<html>

<head>

</head>

</html>

语言属性

根据 HTML5 规范：

*强烈建议为 html 根元素指定 lang 属性，从而为文档设置正确的语言。这将有助于语音合成工具确定其所应该采用的发音，有助于翻译工具确定其翻译时所应遵守的规则等等。*

更多关于 lang 属性的知识可以从 [此规范](http://www.w3.org/html/wg/drafts/html/master/semantics.html#the-html-element) 中了解。

这里列出了[语言代码表](http://reference.sitepoint.com/html/lang-codes)。

<html lang="zh-CN">

<!-- ... -->

</html>

IE 兼容模式

IE 支持通过特定的 <meta> 标签来确定绘制当前页面所应该采用的 IE 版本。除非有强烈的特殊需求，否则最好是设置为 **edge mode**，从而通知 IE 采用其所支持的最新的模式。

[阅读这篇 stack overflow 上的文章](http://stackoverflow.com/questions/6771258/whats-the-difference-if-meta-http-equiv-x-ua-compatible-content-ie-edge-e)可以获得更多有用的信息。

<meta http-equiv="X-UA-Compatible" content="IE=Edge">

字符编码

通过明确声明字符编码，能够确保浏览器快速并容易的判断页面内容的渲染方式。这样做的好处是，可以避免在 HTML 中使用字符实体标记（character entity），从而全部与文档编码一致（一般采用 UTF-8 编码）。

<head>

<meta charset="UTF-8">

</head>

引入 CSS 和 JavaScript 文件

根据 HTML5 规范，在引入 CSS 和 JavaScript 文件时一般不需要指定type 属性，因为 text/css 和 text/javascript 分别是它们的默认值。

HTML5 spec links

* [Using link](http://www.w3.org/TR/2011/WD-html5-20110525/semantics.html#the-link-element)
* [Using style](http://www.w3.org/TR/2011/WD-html5-20110525/semantics.html#the-style-element)
* [Using script](http://www.w3.org/TR/2011/WD-html5-20110525/scripting-1.html#the-script-element)

<!-- External CSS -->

<link rel="stylesheet" href="code-guide.css">

<!-- In-document CSS -->

<style>

/\* ... \*/

</style>

<!-- JavaScript -->

<script src="code-guide.js"></script>

实用为王

尽量遵循 HTML 标准和语义，但是不要以牺牲实用性为代价。任何时候都要尽量使用最少的标签并保持最小的复杂度。

属性顺序

HTML 属性应当按照以下给出的顺序依次排列，确保代码的易读性。

* class
* id, name
* data-\*
* src, for, type, href
* title, alt
* aria-\*, role

class 用于标识高度可复用组件，因此应该排在首位。id 用于标识具体组件，应当谨慎使用（例如，页面内的书签），因此排在第二位。

<a class="..." id="..." data-modal="toggle" href="#">

Example link

</a>

<input class="form-control" type="text">

<img src="..." alt="...">

布尔（boolean）型属性

布尔型属性可以在声明时不赋值。XHTML 规范要求为其赋值，但是 HTML5 规范不需要。

更多信息请参考 [WhatWG section on boolean attributes](http://www.whatwg.org/specs/web-apps/current-work/multipage/common-microsyntaxes.html#boolean-attributes)：

*元素的布尔型属性如果有值，就是 true，如果没有值，就是 false。*

如果*一定*要为其赋值的话，请参考 WhatWG 规范：

*如果属性存在，其值必须是空字符串或 [...] 属性的规范名称，并且不要再收尾添加空白符。*

**简单来说，就是不用赋值。**

<input type="text" disabled>

<input type="checkbox" value="1" checked>

<select>

<option value="1" selected>1</option>

</select>

减少标签的数量

编写 HTML 代码时，尽量避免多余的父元素。很多时候，这需要迭代和重构来实现。请看下面的案例：

<!-- Not so great -->

<span class="avatar">

<img src="...">

</span>

<!-- Better -->

<img class="avatar" src="...">

JavaScript 生成的标签

通过 JavaScript 生成的标签让内容变得不易查找、编辑，并且降低性能。能避免时尽量避免。

CSS

语法

* 用两个空格来代替制表符（tab） -- 这是唯一能保证在所有环境下获得一致展现的方法。
* 为选择器分组时，将单独的选择器单独放在一行。
* 为了代码的易读性，在每个声明块的左花括号前添加一个空格。
* 声明块的右花括号应当单独成行。
* 每条声明语句的 : 后应该插入一个空格。
* 为了获得更准确的错误报告，每条声明都应该独占一行。
* 所有声明语句都应当以分号结尾。最后一条声明语句后面的分号是可选的，但是，如果省略这个分号，你的代码可能更易出错。
* 对于以逗号分隔的属性值，每个逗号后面都应该插入一个空格（例如，box-shadow）。
* 不要在 rgb()、rgba()、hsl()、hsla() 或 rect() 值的*内部*的逗号后面插入空格。这样利于从多个属性值（既加逗号也加空格）中区分多个颜色值（只加逗号，不加空格）。
* 对于属性值或颜色参数，省略小于 1 的小数前面的 0 （例如，.5代替 0.5；-.5px 代替 -0.5px）。
* 十六进制值应该全部小写，例如，#fff。在扫描文档时，小写字符易于分辨，因为他们的形式更易于区分。
* 尽量使用简写形式的十六进制值，例如，用 #fff 代替 #ffffff。
* 为选择器中的属性添加双引号，例如，input[type="text"]。[只有在某些情况下是可选的](http://mathiasbynens.be/notes/unquoted-attribute-values#css)，但是，为了代码的一致性，建议都加上双引号。
* 避免为 0 值指定单位，例如，用 margin: 0; 代替 margin: 0px;。

对于这里用到的术语有疑问吗？请参考 Wikipedia 上的 [syntax section of the Cascading Style Sheets article](http://en.wikipedia.org/wiki/Cascading_Style_Sheets#Syntax)。

/\* Bad CSS \*/

.selector, .selector-secondary, .selector[type=text] {

padding:15px;

margin:0px 0px 15px;

background-color:rgba(0, 0, 0, 0.5);

box-shadow:0px 1px 2px #CCC,inset 0 1px 0 #FFFFFF

}

/\* Good CSS \*/

.selector,

.selector-secondary,

.selector[type="text"] {

padding: 15px;

margin-bottom: 15px;

background-color: rgba(0,0,0,.5);

box-shadow: 0 1px 2px #ccc, inset 0 1px 0 #fff;

}

声明顺序

相关的属性声明应当归为一组，并按照下面的顺序排列：

1. Positioning
2. Box model
3. Typographic
4. Visual

由于定位（positioning）可以从正常的文档流中移除元素，并且还能覆盖盒模型（box model）相关的样式，因此排在首位。盒模型排在第二位，因为它决定了组件的尺寸和位置。

其他属性只是影响组件的*内部（inside）*或者是不影响前两组属性，因此排在后面。

完整的属性列表及其排列顺序请参考 [Recess](http://twitter.github.com/recess)。

.declaration-order {

/\* Positioning \*/

position: absolute;

top: 0;

right: 0;

bottom: 0;

left: 0;

z-index: 100;

/\* Box-model \*/

display: block;

float: right;

width: 100px;

height: 100px;

/\* Typography \*/

font: normal 13px "Helvetica Neue", sans-serif;

line-height: 1.5;

color: #333;

text-align: center;

/\* Visual \*/

background-color: #f5f5f5;

border: 1px solid #e5e5e5;

border-radius: 3px;

/\* Misc \*/

opacity: 1;

}

不要使用 @import

与 <link> 标签相比，@import 指令要慢很多，不光增加了额外的请求次数，还会导致不可预料的问题。替代办法有以下几种：

* 使用多个 <link> 元素
* 通过 Sass 或 Less 类似的 CSS 预处理器将多个 CSS 文件编译为一个文件
* 通过 Rails、Jekyll 或其他系统中提供过 CSS 文件合并功能

请参考 [Steve Souders 的文章](http://www.stevesouders.com/blog/2009/04/09/dont-use-import/)了解更多知识。

<!-- Use link elements -->

<link rel="stylesheet" href="core.css">

<!-- Avoid @imports -->

<style>

@import url("more.css");

</style>

媒体查询（Media query）的位置

将媒体查询放在尽可能相关规则的附近。不要将他们打包放在一个单一样式文件中或者放在文档底部。如果你把他们分开了，将来只会被大家遗忘。下面给出一个典型的实例。

.element { ... }

.element-avatar { ... }

.element-selected { ... }

@media (min-width: 480px) {

.element { ...}

.element-avatar { ... }

.element-selected { ... }

}

带前缀的属性

当使用特定厂商的带有前缀的属性时，通过缩进的方式，让每个属性的值在垂直方向对齐，这样便于多行编辑。

在 Textmate 中，使用 **Text → Edit Each Line in Selection** (⌃⌘A)。在 Sublime Text 2 中，使用 **Selection → Add Previous Line** (⌃⇧↑) 和 **Selection → Add Next Line** (⌃⇧↓)。

/\* Prefixed properties \*/

.selector {

-webkit-box-shadow: 0 1px 2px rgba(0,0,0,.15);

box-shadow: 0 1px 2px rgba(0,0,0,.15);

}

单行规则声明

对于**只包含一条声明**的样式，为了易读性和便于快速编辑，建议将语句放在同一行。对于带有多条声明的样式，还是应当将声明分为多行。

这样做的关键因素是为了错误检测 -- 例如，CSS 校验器指出在 183 行有语法错误。如果是单行单条声明，你就不会忽略这个错误；如果是单行多条声明的话，你就要仔细分析避免漏掉错误了。

/\* Single declarations on one line \*/

.span1 { width: 60px; }

.span2 { width: 140px; }

.span3 { width: 220px; }

/\* Multiple declarations, one per line \*/

.sprite {

display: inline-block;

width: 16px;

height: 15px;

background-image: url(../img/sprite.png);

}

.icon { background-position: 0 0; }

.icon-home { background-position: 0 -20px; }

.icon-account { background-position: 0 -40px; }

简写形式的属性声明

在需要显示地设置所有值的情况下，应当尽量限制使用简写形式的属性声明。常见的滥用简写属性声明的情况如下：

* padding
* margin
* font
* background
* border
* border-radius

大部分情况下，我们不需要为简写形式的属性声明指定所有值。例如，HTML 的 heading 元素只需要设置上、下边距（margin）的值，因此，在必要的时候，只需覆盖这两个值就可以。过度使用简写形式的属性声明会导致代码混乱，并且会对属性值带来不必要的覆盖从而引起意外的副作用。

MDN（Mozilla Developer Network）上一片非常好的关于[shorthand properties](https://developer.mozilla.org/en-US/docs/Web/CSS/Shorthand_properties) 的文章，对于不太熟悉简写属性声明及其行为的用户很有用。

/\* Bad example \*/

.element {

margin: 0 0 10px;

background: red;

background: url("image.jpg");

border-radius: 3px 3px 0 0;

}

/\* Good example \*/

.element {

margin-bottom: 10px;

background-color: red;

background-image: url("image.jpg");

border-top-left-radius: 3px;

border-top-right-radius: 3px;

}

Less 和 Sass 中的嵌套

避免非必要的嵌套。这是因为虽然你可以使用嵌套，但是并不意味着应该使用嵌套。只有在必须将样式限制在父元素内（也就是后代选择器），并且存在多个需要嵌套的元素时才使用嵌套。

// Without nesting

.table > thead > tr > th { … }

.table > thead > tr > td { … }

// With nesting

.table > thead > tr {

> th { … }

> td { … }

}

注释

代码是由人编写并维护的。请确保你的代码能够自描述、注释良好并且易于他人理解。好的代码注释能够传达上下文关系和代码目的。不要简单地重申组件或 class 名称。

对于较长的注释，务必书写完整的句子；对于一般性注解，可以书写简洁的短语。

/\* Bad example \*/

/\* Modal header \*/

.modal-header {

...

}

/\* Good example \*/

/\* Wrapping element for .modal-title and .modal-close \*/

.modal-header {

...

}

class 命名

* class 名称中只能出现小写字符和破折号（dashe）（不是下划线，也不是驼峰命名法）。破折号应当用于相关 class 的命名（类似于命名空间）（例如，.btn 和 .btn-danger）。
* 避免过度任意的简写。.btn 代表 *button*，但是 .s 不能表达任何意思。
* class 名称应当尽可能短，并且意义明确。
* 使用有意义的名称。使用有组织的或目的明确的名称，不要使用表现形式（presentational）的名称。
* 基于最近的父 class 或基本（base） class 作为新 class 的前缀。
* 使用 .js-\* class 来标识行为（与样式相对），并且不要将这些 class 包含到 CSS 文件中。

在为 Sass 和 Less 变量命名是也可以参考上面列出的各项规范。

/\* Bad example \*/

.t { ... }

.red { ... }

.header { ... }

/\* Good example \*/

.tweet { ... }

.important { ... }

.tweet-header { ... }

选择器

* 对于通用元素使用 class ，这样利于渲染性能的优化。
* 对于经常出现的组件，避免使用属性选择器（例如，[class^="..."]）。浏览器的性能会受到这些因素的影响。
* 选择器要尽可能短，并且尽量限制组成选择器的元素个数，建议不要超过 3 。
* **只有**在必要的时候才将 class 限制在最近的父元素内（也就是后代选择器）（例如，不使用带前缀的 class 时 -- 前缀类似于命名空间）。

扩展阅读：

* [Scope CSS classes with prefixes](http://markdotto.com/2012/02/16/scope-css-classes-with-prefixes/)
* [Stop the cascade](http://markdotto.com/2012/03/02/stop-the-cascade/)

/\* Bad example \*/

span { ... }

.page-container #stream .stream-item .tweet .tweet-header .username { ... }

.avatar { ... }

/\* Good example \*/

.avatar { ... }

.tweet-header .username { ... }

.tweet .avatar { ... }

代码组织

* 以组件为单位组织代码段。
* 制定一致的注释规范。
* 使用一致的空白符将代码分隔成块，这样利于扫描较大的文档。
* 如果使用了多个 CSS 文件，将其按照组件而非页面的形式分拆，因为页面会被重组，而组件只会被移动。

/\*

\* Component section heading

\*/

.element { ... }

/\*

\* Component section heading

\*

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

\*/

.element { ... }

/\* Contextual sub-component or modifer \*/

.element-heading { ... }

编辑器配置

将你的编辑器按照下面的配置进行设置，以避免常见的代码不一致和差异：

* 用两个空格代替制表符（soft-tab 即用空格代表 tab 符）。
* 保存文件时，删除尾部的空白符。
* 设置文件编码为 UTF-8。
* 在文件结尾添加一个空白行。

参照文档并将这些配置信息添加到项目的 .editorconfig 文件中。例如：[Bootstrap 中的 .editorconfig 实例](https://github.com/twbs/bootstrap/blob/master/.editorconfig)。更多信息请参考 [about EditorConfig](http://editorconfig.org/)。