目录

黄金法则

始终同意并遵循规范的每一条内容 -- 可以是这里列出的，也可以是你自己的。不对之处，请随时指出，继续了解或贡献内容，请 open an issue on GitHub。

> 不管有多少参与者，代码都应该像同一个人所写。

HTML

语法

- 使用两个空格的 soft tabs – 这是保证代码在各种环境下显示一致的唯一方式。
- 嵌套的节点应该缩进（两个空格）。
- 在属性上，使用双引号，不要使用单引号。
- 不好在自动闭合标签结尾处使用斜线 - HTML5 规范 指出他们是可选的。
- 不要忽略可选的关闭标签（例如， `</li>` 和 `</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

在每个 HTML 页面开头使用这个简单地 doctype 来启用标准模式，使其每个浏览器中尽可能

一致的展现。

```
<!DOCTYPE html>
<html>
  <head>
  </head>
</html>
```

Language 属性

根据 HTML5 规范：

> 鼓励网站作者在 *html* 元素上指定 *lang* 属性，来指出页面的语言。这样做有助于语言合成工具来确定发音方式，以及帮助翻译工具决定使用的规则，等等。

通过规范中的 `lang` 属性了解更多相关内容。

前往 Sitepoint 查看 language codes 列表。

```
<html lang="en-us">
  <!-- ... -->
</html>
```

```
<html lang="zh-cmn-Hans">
  <!-- ... -->
</html>
```

字符编码

通过声明一个明确的字符编码，让浏览器轻松、快速的确定适合网页内容的渲染方式。这样做之后，需要避免在 HTML 中出现字符实体，直接提供字符与文档一致的编码（通常是 UTF-8）。

```
<head>
  <meta charset="UTF-8">
```

```
</head>
```

IE 兼容模式

Internet Explorer 支持使用兼容性 `<meta>` 标签来指定使用什么版本的 IE 来渲染页面。如果不是特殊需要，通常通过 **edge mode** 来通知 IE 使用最新的兼容模式。

For more information, read this awesome Stack Overflow article.

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

引入 CSS 和 JavaScript

根据 HTML5 规范, 通常在引入 CSS 和 JavaScript 时不需要指明 `type`， 因为 `text/css` 和 `text/javascript` 分别是他们的默认值。

HTML5 规范链接

- 使用 link
- 使用 style
- 使用 script

```
<!-- External CSS -->
<link rel="stylesheet" href="code-guide.css">

<!-- In-document CSS -->
<style>
  /* ... */
</style>

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

实用高于完美

尽量遵循 HTML 标准和语义，但是不应该以浪费实用性作为代价。任何时候都要用尽量小的复杂度和尽量少的标签来解决问题。

属性顺序

HTML 属性应该按照特定的顺序出现以保证易读性。

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

Classes 是为高可复用组件设计的，所以他们处在第一位。Ids 更加具体而且应该尽量少使用（例如, 页内书签），所以他们处在第二位。

```
<a class="..." id="..." data-toggle="modal" href="#">
  Example link
</a>

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

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

Boolean 属性

Boolean 属性指不需要声明取值的属性。XHTML 需要每个属性声明取值，但是 HTML5 并不需要。

了解更多内容，参考 WhatWG section on boolean attributes:

> 一个元素中 *Boolean* 属性的存在表示取值 *true*，不存在则表示取值 *false*。

如果你必须为属性添加并不需要的取值，参照 WhatWG 的指引:

> 如果属性存在，他的取值必须是空字符串或者 *[...]* 属性的规范名称，不要在首尾包含空白字符。

简而言之，不要为 **Boolean** 属性添加取值。

```
<input type="text" disabled>

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

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

减少标签数量

在编写 HTML 代码时，需要尽量避免多余的父节点。很多时候，需要通过迭代和重构来使 HTML 变得更少。 参考下面的示例:

```
<!-- Not so great -->
<span class="avatar">
  <img src="...">
</span>

<!-- Better -->
<img class="avatar" src="...">
```

JavaScript 生成标签

在 JavaScript 文件中生成标签让内容变得更难查找，更难编辑，性能更差。应该尽量避免这种情况的出现。

CSS

语法

- 使用两个空格的 soft tabs – 这是保证代码在各种环境下显示一致的唯一方式。
- 使用组合选择器时，保持每个独立的选择器占用一行。
- 为了代码的易读性，在每个声明的左括号前增加一个空格。
- 声明块的右括号应该另起一行。
- 每条声明 `:` 后应该插入一个空格。
- 每条声明应该只占用一行来保证错误报告更加准确。
- 所有声明应该以分号结尾。虽然最后一条声明后的分号是可选的，但是如果没有他，你的代码会更容易出错。
- 逗号分隔的取值，都应该在逗号之后增加一个空格。
- 不要在颜色值 `rgb()`，`rgba()`，`hsl()`，`hsla()`，和 `rect()` 中增加空格
- 不要在属性取值或者颜色参数前面添加不必要的 0 (比如，使用 `.5` 替代 `0.5` 和 `-.5px` 替代 `0.5px`)。
- 所有的十六进制值都应该使用小写字母，例如 `#fff`。因为小写字母有更多样的外形，在浏览文档时，他们能够更轻松的被区分开来。
- 尽可能使用短的十六进制数值，例如使用 `#fff` 替代 `#ffffff`。
- 为选择器中得属性取值添加引号，例如 `input[type="text"]`。他们只在某些情况下可有可无，所以都使用引号可以增加一致性。
- 不要为 0 指明单位，比如使用 `margin: 0;` 而不是 `margin: 0px;`。

对这里提到的规则有问题吗？参考 Wikipedia 中的 CSS 语法部分。

```
/* Bad CSS */
.selector, .selector-secondary, .selector[type=text] {
  padding:15px;
  margin:0px 0px 15px;
  background-color:rgba(0, 0, 0, 0.5);
  box-shadow:0 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: 0px 1px 2px #ccc, inset 0 1px 0 #fff;
}
```

声明顺序

相关的属性声明应该以下面的顺序分组处理：

1. Positioning
2. Box model 盒模型
3. Typographic 排版
4. Visual 外观

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。

关于完整的属性以及他们的顺序，请参考 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;
}
```

媒体查询位置

尽量将媒体查询的位置靠近他们相关的规则。不要将他们一起放到一个独立的样式文件中，或者丢在文档的最底部。这样做只会让大家以后更容易忘记他们。这里是一个典型的案例。

```
.element { ... }
.element-avatar { ... }
.element-selected { ... }

@media (min-width: 480px) {
  .element { ...}
  .element-avatar { ... }
  .element-selected { ... }
}
```

不要使用 `@import`

与 `<link>` 相比，`@import` 更慢，需要额外的页面请求，并且可能引发其他的意想不到的问题。应该避免使用他们，而选择其他的方案：

- 使用多个 `<link>` 元素
- 使用 CSS 预处理器例如 Sass 或 Less 将样式编译到一个文件中
- 使用 Rails, Jekyll 或其他环境提供的功能，来合并 CSS 文件。

了解更多信息, 参照 Steve Souders 的这篇文章。

```
<!-- Use link elements -->
<link rel="stylesheet" href="core.css">

<!-- Avoid @imports -->
<style>
  @import url("more.css");
```

```
</style>
```

前缀属性

当使用厂商前缀属性时，通过缩进使取值垂直对齐以便多行编辑。

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);
}
```

单条声明的声明块

在一个声明块中只包含一条声明的情况下，为了易读性和快速编辑可以考虑移除其中的换行。所有包含多条声明的声明块应该分为多行。

这样做的关键因素是错误检测 - 例如，一个 CSS 验证程序显示你在 183 行有一个语法错误，如果是一个单条声明的行，那就是他了。在多个声明的情况下，你必须为哪里出错了费下脑子。

```
/* 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; }
```

属性简写

坚持限制属性取值简写的使用，属性简写需要你必须显式设置所有取值。常见的属性简写滥用包括:

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

大多数情况下，我们并不需要设置属性简写中包含的所有值。例如，HTML 头部只设置上下的 margin，所以如果需要，只设置这两个值。过度使用属性简写往往会导致更混乱的代码，其中包含不必要的重写和意想不到的副作用。

Mozilla Developer Network 有一篇对不熟悉属性简写及其行为的人来说很棒的关于 shorthand properties 的文章。

```
/* 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

避免不必要的嵌套。可以进行嵌套，不意味着你应该这样做。只有在需要给父元素增加样式并且同时存在多个子元素时才需要考虑嵌套。

```
// Without nesting
.table > thead > tr > th { … }
.table > thead > tr > td { … }

// With nesting
.table > thead > tr {
  > th { … }
  > td { … }
}
```

LESS 和 SASS 中的运算符

为了提高代码可读性，在数学运算外增加括号，并且在取值，变量和运算符之间增加空格。

```
// Bad example
.element {
  margin: 10px 0 @variable*2 10px;
}

// Good example
.element {
  margin: 10px 0 (@variable * 2) 10px;
}
```

代码注释

代码是由人来编写和维护的。保证你的代码是描述性的，包含好的注释，并且容易被他人理解。好的代码注释传达上下文和目标。不要简单地重申组件或者 class 名称。

Be sure to write in complete sentences or larger comments and succinct phrases for general notes.

```
/* Bad example */
/* Modal header */
.modal-header {
  ...
}

/* Good example */
/* Wrapping element for .modal-title and .modal-close */
.modal-header {
  ...
}
```

Class 命名

- 保持 Class 命名为全小写，可以使用短划线（不要使用下划线和 camelCase 命名）。短划线应该作为相关类的自然间断。(例如， `.btn` 和 `.btn-danger`)。
- 避免过度使用简写。 `.btn` 可以很好地描述 *button*，但是 `.s` 不能代表任何元素。
- Class 的命名应该尽量短，也要尽量明确。
- 使用有意义的名称；使用结构化或者作用目标相关，而不是抽象的名称。
- 命名时使用最近的父节点或者父 class 作为前缀。
- 使用 `.js-*` classes 来表示行为(相对于样式)，但是不要在 CSS 中定义这些 classes。

这些规则在创建 Sass 和 Less 变量名时同样有用。

```
/* Bad example */
.t { ... }
.red { ... }
.header { ... }

/* Good example */
.tweet { ... }
.important { ... }
.tweet-header { ... }
```

选择器

- 使用 classes 而不是通用元素标签来优化渲染性能。
- 避免在经常出现的组件中使用一些属性选择器 (例如， `[class^="..."]`)。浏览器性能会受到这些情况的影响。
- 减少选择器的长度，每个组合选择器选择器的条目应该尽量控制在 3 个以内。
- 只在必要的情况下使用后代选择器 (例如，没有使用带前缀 classes 的情况).

扩展阅读:

- Scope CSS classes with prefixes
- 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。
- 制定一个一致的注释层级结构。
- 使用一致的空白来分割代码块，这样做在查看大的文档时更有优势。
- 当使用多个 CSS 文件时，通过组件而不是页面来区分他们。页面会被重新排列组合，而组件是可以移动的。

```
/*
 * 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

根据以下的设置来配置你的编辑器，来避免常见的代码不一致和丑陋的 diffs。

- 使用两个空格的 soft-tabs。
- 在保存时删除尾部的空白字符。
- 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，将这些设置应用到项目的 `.editorconfig` 文件。 例如，Bootstrap 中的 `.editorconfig` 文件。 通过 关于 EditorConfig 了解更多内容。

<3

深受 Idiomatic CSS 和 GitHub Styleguide 启发。 Made with all the love in the world by @mdo.

由 ZoomZhao 翻译为中文，中文翻译问题请参见 Github 仓库。

Follow @mdo Tweet