开发规范

HTML

HTML 风格指南~

文章翻译自 w3cschool。

始终声明文档类型

始终将文档类型声明为文档的第一行。

HTML 的正确文档类型是：

```
<!DOCTYPE html>
```

使用小写元素名称

HTML 允许在元素名称中混合大写和小写字母。

但是，我们建议使用**小写**的元素名称，因为：

- 混合大写和小写名称看起来很糟糕
- 开发人员通常使用小写名称
- 小写看起来更干净
- 小写更容易写

Good:

```
<body>
  <p>This is a paragraph.</p>
</body>
```

Bad:

```
<BODY>
<P>This is a paragraph.</P>
</BODY>
```

闭合元素标签

在 HTML 中，不必关闭所有元素（例如 `<p>` 元素）。

但是，我们**强烈建议关闭**所有 HTML 元素，像这样：

Good:

```
<section>
  <p>This is a paragraph.</p>
  <p>This is a paragraph.</p>
</section>
```

Bad:

```
<section>
  <p>This is a paragraph.
  <p>This is a paragraph.
</section>
```

使用小写的属性名

HTML 允许在属性名称中混合大写和小写字母。

但是，我们**建议使用小写**的属性名称，因为：

- 混合大写和小写名称看起来很糟糕
- 开发人员通常使用小写名称
- 小写看起来更干净
- 小写更容易写

Good:

```
<a href="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

Bad:

```
<a HREF="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

属性值始终添加双引号

HTML 允许不带引号的属性值。

但是，我们建议引用属性值，因为：

- 开发人员通常会引用属性值
- 引用的值更容易阅读
- 如果值包含空格，则必须使用引号

Good:

```
<table class="striped">
```

Bad:

```
<table class=striped>
```

Very Bad:

这不起作用，因为该值包含空格：

```
<table class=table striped>
```

始终为 img 指定 `alt`、`width` 和 `height`

始终为图像指定 `alt` 属性。如果由于某种原因无法显示图像，则此属性很重要。

此外，始终定义图像的 `width` 和 `height`。**这减少了闪烁，因为浏览器可以在加载之前为图像保留空间。**

Good:

```
<img src="html5.gif" alt="HTML5" style="width:128px;height:128px">
```

Bad:

```
<img src="html5.gif">
```

空格和等号

HTML 允许等号周围有空格。但是**无空格更容易阅读**，并且可以更好地将实体组合在一起。

Good:

```
<link rel="stylesheet" href="styles.css">
```

Bad:

```
<link rel = "stylesheet" href = "styles.css">
```

避免长代码行

使用 HTML 编辑器时，左右滚动来阅读 HTML 代码并不方便。

尽量避免太长的代码行。

空行和缩进

不要无故添加空行、空格或缩进。

为了便于阅读，请添加空行来分隔大型或逻辑代码块。

为了可读性，添加两个缩进空格。不要使用制表键。

Good:

```
<body>

<h1>Famous Cities</h1>

<h2>Tokyo</h2>
<p>Tokyo is the capital of Japan, the center of the Greater Tokyo Area, and the most populous metropolitan area in the world.</p>

<h2>London</h2>
<p>London is the capital city of England. It is the most populous city in the United Kingdom.</p>

<h2>Paris</h2>
```

```
<p>Paris is the capital of France. The Paris area is one of the largest population centers in Europe.</p>

</body>
```

Bad:

```
<body>
<h1>Famous Cities</h1>
<h2>Tokyo</h2><p>Tokyo is the capital of Japan, the center of the Greater Tokyo Area, and the most populous metropolitan area in the world.</p>
<h2>London</h2><p>London is the capital city of England. It is the most populous city in the United Kingdom.</p>
<h2>Paris</h2><p>Paris is the capital of France. The Paris area is one of the largest population centers in Europe.</p>
</body>
```

Good `Table` Example:

```
<table>
  <tr>
    <th>Name</th>
    <th>Description</th>
  </tr>
  <tr>
    <td>A</td>
    <td>Description of A</td>
  </tr>
  <tr>
    <td>B</td>
    <td>Description of B</td>
  </tr>
</table>
```

Good List Example:

```
<ul>
  <li>London</li>
  <li>Paris</li>
  <li>Tokyo</li>
</ul>
```

从不省略 `<title>` 元素

`<title>` 元素在 HTML 中是必需的。

页面标题的内容对于搜索引擎优化 (SEO) 非常重要！搜索引擎算法使用页面标题来决定在搜索结果中列出页面时的顺序。

`<title>` 元素：

- 在浏览器工具栏中定义标题
- 为页面添加到收藏夹时提供标题
- 在搜索引擎结果中显示页面的标题

所以，使标题尽可能准确和有意义：

```
<title>HTML Style Guide and Coding Conventions</title>
```

省略 `<html>` 和 `<body>`？

HTML 页面将在没有 `<html>` 和 `<body>` 标签的情况下进行验证：

案例：

```
<!DOCTYPE html>
<head>
  <title>Page Title</title>
</head>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>
```

Try it Yourself

但是，我们强烈建议始终添加 `<html>` 和 `<body>` 标签！

省略 `<body>` 会在旧浏览器中产生错误。

省略 `<html>` 和 `<body>` 也会导致 DOM 和 XML 软件崩溃。

省略 `<head>`？

HTML `<head>` 标签也可以省略。

浏览器会将 `<body>` 之前的所有元素添加到默认的 `<head>` 元素中。

案例：

```
<!DOCTYPE html>
<html>
<title>Page Title</title>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Try it Yourself

但是，我们建议使用 `<head>` 标签。

关闭空 HTML 元素?

在 HTML 中，关闭空元素是可选的。

Allowed:

```
<meta charset="utf-8">
```

Also Allowed:

```
<meta charset="utf-8" />
```

如果您希望 XML/XHTML 软件访问您的页面，请保留右斜杠 (/)，因为它在 XML 和 XHTML 中是必需的。

添加 lang 属性

您应该始终在 `<html>` 标记内包含 `lang` 属性，以声明网页的语言。这是为了帮助搜索引擎和浏览器。

案例：

```
<!DOCTYPE html>
<html lang="en-us">
```

```
<head>
  <title>Page Title</title>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Try it Yourself

元数据

为了确保正确的解释和正确的搜索引擎索引，语言和字符编码 `<meta charset="charset">` 应该尽早在 HTML 文档中定义：

```
<!DOCTYPE html>
<html lang="en-us">
<head>
  <meta charset="UTF-8">
  <title>Page Title</title>
</head>
```

设置视口

视口是网页的用户可见区域。它因设备而异 - 在手机上会比在电脑屏幕上小。 您应该在所有网页中包含以下 `<meta>` 元素：

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

这为浏览器提供了有关如何控制页面尺寸和缩放的说明。

`width=device-width` 部分设置页面的宽度以跟随设备的屏幕宽度（这将因设备而异）。

`initial-scale=1.0` 部分设置浏览器首次加载页面时的初始缩放级别。

这是一个没有视口元标记的网页示例，以及具有视口元标记的同一网页：

信息

如果您使用手机或平板电脑浏览此页面，您可以点击下面的两个链接查看不同之处。

HTML 注释

简短的注释应该写在一行上，如下所示：

```
<!-- This is a comment -->
```

超过一行的注释应该这样写：

```
<!--
  This is a long comment example. This is a long comment example.
  This is a long comment example. This is a long comment example.
-->
```

长注释如果缩进两个空格更容易观察。

使用样式表

使用简单的语法链接到样式表（不需要 `type` 属性）：

```
<link rel="stylesheet" href="styles.css">
```

短的 CSS 规则可以写成压缩的，像这样：

```
p.intro {font-family:Verdana;font-size:16em;}
```

长的 CSS 规则应该写成多行：

```
body {
  background-color: lightgrey;
  font-family: "Arial Black", Helvetica, sans-serif;
  font-size: 16em;
  color: black;
}
```

- 将左括号与选择器放在同一行
- 在左括号前使用一个空格
- 使用两个缩进空格
- 在每个属性值对之后使用分号，包括最后一个
- 如果值包含空格，则仅在值周围使用引号
- 将右括号放在新行上，不带前导空格

在 HTML 中加载 JavaScript

使用简单的语法加载外部脚本（不需要 `type` 属性）：

```
<script src="myscript.js">
```

使用 JavaScript 访问 HTML 元素

使用“不规范”的 HTML 代码可能会导致 JavaScript 错误。

这两个 JavaScript 语句会产生不同的结果：

例子

```
getElementById("Demo").innerHTML = "Hello";
getElementById("demo").innerHTML = "Hello";
```

使用小写文件名

一些 Web 服务器（Apache、Unix）对文件名区分大小写：“london.jpg” 不能作为“London.jpg” 访问。

其他 Web 服务器（Microsoft、IIS）不区分大小写：“london.jpg” 可以作为 “London.jpg” 访问。

如果混合使用大写和小写，则必须注意这一点。

如果您从不区分大小写的服务器转移到区分大小写的服务器，即使是小错误也会破坏您的网络！

为避免这些问题，请始终使用小写文件名！

文件扩展名

HTML 文件应具有 `.html` 扩展名（允许使用 `.htm`）。

CSS 文件应具有 `.css` 扩展名。

JavaScript 文件应具有 `.js` 扩展名。

`.htm` 和 `.html` 之间的区别？

`.htm` 和 `.html` 文件扩展名之间**没有区别**！

两者都将被任何 Web 浏览器和 Web 服务器视为 HTML。

默认文件名

当 URL 没有在末尾指定文件名时（如 https://www.w3schools.com/), 服务器只添加一个默认文件名，如 `index.html`、`index.htm`、`default.html` 或 `default.htm`。

如果您的服务器仅配置了 `index.html` 作为默认文件名，则您的文件必须命名为 `index.html`，而不是 `default.html`。

但是，服务器可以配置多个默认文件名；通常您可以根据需要设置任意数量的默认文件名。

CSS

CSS 风格指南~

工具保障

使用 `Stylelint` 来强制约束。

```
1 pnpm add stylelint stylelint-config-prettier stylelint-config-standard -D
```

package.json

```
"stylelint": {
  "extends": [
    "stylelint-config-standard",
    "stylelint-config-prettier"
  ],
  // 自定义规则，不应该超过 5 条
  // 比如我们为了方便使用 Module CSS，约束只允许存在下划线 _
  "rules": {
    "selector-class-pattern": [
      "^([a-z][a-z0-9]*)(_[a-z0-9]+)*$",
      {
        "message": "Expected class selector to be snake_case: %s"
      }
    ]
  }
},
...
```

然后再配置 Git Hooks：

package.json

```
"lint-staged": {
  "*.css": "stylelint --fix",
  ...
}
```

选择器约束

大小写

一般场景都是用**小写**

标签、属性选择器，统一使用小写

我们**不建议**此类选择器写法。

信息

浏览器**不区分**大小写。

```
/* bad */
```

```
P {
  padding: 0 0.3rem;
}

/* good */
p {
  font-size: 0.28rem;
}
```

属性值，统一使用小写

警告

浏览器**区分**大小写

```
/* bad */
.SomeContent {
  color: #DDD;
}

/* good */
.some-content {
  color: #ddd;
}
```

不使用 ID 选择器

尽量**不使用** id 选择器，理由是：

- 优先级高，无法用 class 选择器覆盖
- 样式不能复用
- 常用于 JS 获取 DOM，同时和样式关联会使代码可维护性降低

少用 * 和标签选择器

提示

CSS 的渲染方式是“从右往左”渲染的，大量的使用 * 通配符选择器会降低页面执行效率

```
/* bad */
/* 会去遍历所有标签，影响性能 */
.some-content * {
    padding:0 .3rem;
}
```

```
6
7  /* 同样会去遍历所有<a>标签，影响性能 */
8  .some-content a {
9      font-size:0.28rem;
10 }
```

checked disabled 等状态属性

`checked`、`disabled` 等状态不使用属性选择器，**优先使用伪类**，理由是：

- 伪类只能匹配表单元素，更精确，`checked 属性` 可以加在任意属性上
- 用 JS 修改属性时，`checked 属性` 会出现匹配不准的情况
- 伪类可以继承祖先元素的状态

```
1  /* bad */
2  [checked] {
3    border: 1px solid #ff0;
4  }
5
6  /* good */
7  :checked {
8    border: 1px solid #ff0;
9  }
```

属性格式规范

属性书写顺序

1. 位置属性（position, top, right, z-index, display, float 等）
2. 盒子属性 (width, height, padding, margin 等)
3. 文字相关 (font, lineheight, color, text-alight 等)
4. 背景相关 (shadow, border, backgoround 等)
5. 其他 CSS3 属性 (transform、transition 等)

```
1  /* good */
2  .box {
3    display: block;
4    position: absolute;
5    left: 30%;
6    right: 30%;
```

```
  overflow: hidden;
  margin: 1em;
  padding: 1em;
  background-color: #eee;
  border: 3px solid #ddd;
  font-family: 'Arial', sans-serif;
  font-size: 1.5rem;
  text-transform: uppercase;
}
```

属性缩写

对于可以有缩写的 CSS 属性 例如。`border`、`margin`、`padding`、`background` 等。尽量采用缩写形式。可读性强又可以减小代码体积。

```
/* bad */
margin-top: 10px;
margin-bottom: 0;
margin-left: 5px;
margin-right: 5px;

/* good */
margin: 10px 5px 0 5px;
```

书写规范

良好的 CSS 书写规范可以让代码具有可读性可维护性。让后续的开发者可以快速的投入到项目中。

现在一般交由 `stylelint` 和 `prettier` 工具保障。

- 使用软制表符（2 个空格）进行缩进。
- 在类名中，最好使用破折号而不是驼峰式铸造。

```
/* bad */
.pageContainer {
}

/* good */
.page-container {
}
```

- 如果使用的是 BEM，则可以使用下划线和 PascalCasing（请参见下面的 OOCSS 和 BEM）。

- 不要使用 ID 选择器。

```
/* bad */
#card-list {
}

/* good */
.card-list {
}
```

- 在规则声明中使用多个选择器时，请为每个选择器分配单独行。

```
/* bad */
.a-container, .b-container, .c-container { }

/* good */
.a-container,
.b-container,
.c-container {
}
```

- 在规则声明的左大括号 `{` 之前放置一个空格。

```
/* bad */
.avatar{
  border-radius: 50%;
  border: 2px solid white;
}

/* good */
.avatar {
  border-radius: 50%;
  border: 2px solid white;
}
```

- 在属性中，在 `:` 字符之后（**但不能在字符之前**）放置一个空格。 将 `}` 规则声明的右花括号放在新行上。

```
/* bad */
.avatar {
```

```
  border-radius: 50%;
  border: 2px solid white; }

/* good */
.avatar {
  border-radius: 50%;
  border: 2px solid white;
}
```

- 在规则声明之间放置空白行

```
/* bad */
.avatar {
  border-radius: 50%;
  border: 2px solid white;
}
.avatar-info {
  color: #ccc;
}

/* good */
.avatar {
  border-radius: 50%;
  border: 2px solid white;
}

.avatar-info {
  color: #ccc;
}
```

- 建议使用行注释 (在 SASS 中是 `//`) 代替块注释
- 建议注释独占一行, 避免行末注释

```
/* bad */
.avatar {
  border-radius: 50%;
  border: 2px solid white;
}
/* 这是一个行末结尾 */

/* good */
/* 这是一个头像 */
.avatar {
```

```
  border-radius: 50%;
  border: 2px solid white;
}
```

命名规范

CSS 选择器的命名问题最困扰开发者的事情之一, 究竟是面向 CSS 属性命令好，还是面向 HTML 语意命名？ 是使用长命名还是短命名？

命名字符

CSS 命名只包含 [a-z0-9_-]，不建议一些花里胡哨的字符

```
<!-- bad -->
<span class="1😊。哈哈">哈哈</span>
<style>
.\31😊。哈哈 {
  color: red;
}
</style>
```

使用短命名

在不影响阅读的前提下，尽量使用短命名 (书写效率和文件大小，类名无法压缩)；例如

- `information` 写成 `info`;
- `description` 写成 `desc`;
- `introduction` 写成 `intro`;

```
/* bad */
.some-information {}

/* good */
.some-info {}
```

```
/* bad */
.article-description {}

/* good */
.article-desc {}
```

状态类不加前缀

`active`、`selected` 等状态类名不加前缀，且不单独写样式。好处是：

- 命名不会太长
- 样式不会冲突
- JS 操作时更方便、统一

类似的类名还有 `disabled`、`checked`、`invalid` 等

```
/* bad */
.some-content-active,
.active {
  color: blue;
}

/* good */
.some-content.active,
.some-content > .active {
  color: blue;
}
```

BEM & OOCSS

在这里鼓励使用 `OOCSS` 和 `BEM` 的某种组合命名规范：

- 可以帮助我们理清 CSS 和 HTML 之间清晰且严谨的关系
- 可以帮助我们创建出可重用、易装配的组件
- 可以减少嵌套，降低特定性
- 可以帮助我们创建出可扩展的样式表
-

OOCSS

Object Oriented CSS（面向对象的 CSS），是一种写 CSS 的方法，其思想就是鼓励你把样式表看作“对象” 的集合：创建可重用性、可重复性的代码段让你可以在整个网站中多次使用。

OOCSS (Object-Oriented CSS 翻译为 面向对象 CSS) 是组织 CSS 的领先的模块化或基于组件的系统。它是 Nicole Sullivan 在 2008 年在 Web Directions North 大会上首次提出的。

她同时提到，在构建 OOCSS 时，抽象是首先要考虑的，但还有两个基本原则要遵循：

1. **分离结构（structure）和皮肤（skin）。** 您应该在基础对象中保留结构和位置，并在扩展类中保留视觉特征（如 `background` 或 `border`）。这样您就不必覆盖视觉属性。
2. **分离容器（container）和内容（content）。** 永远不要在 CSS 中模仿 HTML 的结构。换句话说，**不要在样式表中引用标签或 ID**。相反，尝试创建和应用描述相关标签使用的类。并将嵌套类保持在最低限度。

分离结构和皮肤

皮肤是我们可见的视觉属性，例如：

- Colors 颜色
- Fonts 字体
- Shadows 阴影
- Gradients 渐变
- BackgroundColos 背景

结构当然就是我们不可见的视觉属性，例如：

- Height 高度
- Width 宽度
- Position 位置
- Margin
- Padding
- Overflow

这个好例子就是我们上面举的这个例子：

```
.box-border {
  border: 1px solid #ccc;
  border-radius: 10px;
}

.box-1 {
  width: 200px;
  height: 200px;
}

.box-2 {
  width: 120px;
  height: 120px;
}
```

在 HTML 结构：

```
<div class="box-border box-1">Learn OOP</div>
<div class="box-border box-2">Learn CSS</div>
```

分离容器和内容

我们对着下面这个例子讲解：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <style>
      div {
        font-size: 20px;
      }
      div h2 {
        font-size: 20px;
      }
    </style>
  </head>
  <body>
    <div>
      <h2></h2>
      <p></p>
    </div>
  </body>
</html>
```

上面这个例子，`h2` 被锁定在 `div` 这个容器里面了，如果一不小心改变了 HTML 的结构就会导致我们写的 CSS 无效，非常的不便于维护，而且作用于 h2 标签上的样式还无法复用，真是让人头疼。

根据容器和内容分离的原则，我们应该让容器和内容有各自的样式，同时避免使用标签选择器，改写得到如下代码。

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <style>
      .menu {
        width: 200px;
        height: 200px;
      }
```

```
      .menu-title {
        font-size: 20px;
      }
    </style>
  </head>
  <body>
    <div class="menu">
      <h2 class="menu-title"></h2>
      <p></p>
    </div>
  </body>
</html>
```

BEM

Block-Element-Modifier，是一种用于 HTML 和 CSS 类名的命名约定。BEM 最初是由 Yandex 提出的，要知道他们拥有巨大的代码库和可伸缩性，BEM 就是为此而生的，并且可以作为一套遵循 OOCSS 的参考指导规范。

所谓 BEM 是块（block）、元素（element）、修饰符（modifier）的简写

Block

一个本身就是有意义的独立实体。尽管一个 Block 可以和其他 Block 嵌套和共同发挥作用，但语义上所有 Blocks 是平等的，并没有等级或者优先级之分。没有表现为 DOM 的整体实体（例如控制器或模型）也可以是块。

块名称可以包含拉丁字母，数字和短划线。

任何有 class name 的 DOM 节点都可以成为 Block。在一页中，没有对其他 Blocks、Elements 的依赖。例如：header, container, menu, checkbox, input

```
.articel-list {
}
```

Element

Block 的一部分，没有独立的意义，而且在语义上依附于 Block 存在。

例如: menu item, list item, checkbox caption, header title

元素名称可以包含拉丁字母，数字，短划线和下划线。**主要是以两个下划线接在 Block 之后**，来表示从属关系，在一个 Block 下的任何 DOM 节点都可以是 Element。没有对同一页上其他 blocks/elements 的依赖。

```

```

```
.articel-list__item {
}
```

Modifier

修饰符，一个 block 或者 element 上的 flag，体现了外观或者行为的变化。修饰符名称可能包含拉丁字母，数字，短划线和下划线。**以两个短划线接在 element/block 的类名之后**，诸如 `.block--mod`，`.block__elem--mod`， `.block--color-black`， `block-- color-red`。在复杂修饰符中的空格用短划线来代替。

例如: disabled, highlighted, checked, fixed, size big, bg yellow

修饰符是您添加到块/元素 DOM 节点的额外类名。仅将修饰符类添加到它们修改的块/元素，并保留原始类名。是添加不是替换。

```
.articel-list__item--highlighted {
}
```

下面是 HTML 结构：

```
<ul class="articel-list">
  <li class="articel-list_item">This is an element inside a block.</li>
  <li class="articel-list_item articel-list__item--highlighted">
    This is an element inside a block, with a modifier.
  </li>
</ul>
```