1. 一般规范 3
2. HTML 规范 4
2.1. 文档类型 4
2.2. HTML 验证 5
2.3. 脚本加载 5
2.4. 语义化 6
2.5. 多媒体回溯 7
2.6. 关注点分离 7
2.7. HTML 内容至上 8
2.8. Type 属性 9
2.9. ID 和锚点 9
2.10. 格式化规则 9
2.11. HTML 引号 10
3. JavaScript 规范 10
3.1. 全局命名空间污染与 IIFE 10
3.2. IIFE（立即执行的函数表达式） 11
3.3. 变量声明 12
3.4. 理解 JavaScript 的定义域和定义域提升 12
3.5. 总是使用带类型判断的比较判断 13
3.6. 明智地使用真假判断 13
3.7. 变量赋值时的逻辑操作 14
3.8. 分号 14
3.9. 语句块内的函数声明 15
3.10. 异常 16
3.11. 使用闭包 16
3.12. 切勿在循环中创建函数 16
3.13. eval 函数（魔鬼） 18
3.14. this 关键字 18
3.15. 首选函数式风格 18
3.16. 数组和对象字面量 19
3.17. 修改内建对象的原型链 20
3.18. 字符串 20
3.19. 三元条件判断 20
4. CSS 和 Sass (SCSS) 规范 21
4.1. ID and class naming 21
4.2. 合理的避免使用 ID 21
4.3. CSS 选择器中避免标签名 22
4.4. 尽可能的精确 22
4.5. 缩写属性 23
4.6. 0 和 单位 23
4.7. 十六进制表示法 24
4.8. ID 和 Class（类） 名的分隔符 24
4.9. 声明顺序 24
4.10. 声明结束 25

4.11. 属性名结束 26
4.12. 规则分隔 26
4.13. CSS 引号 26
4.14. 选择器嵌套 (SCSS) 27
4.15. 嵌套中引入 空行 (SCSS) 28

前端编码规范

1.一般规范

- 文件/资源命名

不推荐
```
MyScript.js
myCamelCaseName.css
i_love_underscores.html
1001-scripts.js
my-file-min.css
```

推荐
```
my-script.js
my-camel-case-name.css
i-love-underscores.html
thousand-and-one-scripts.js
my-file.min.css
```

- 协议

不推荐
```
<script src="http://cdn.com/foundation.min.js"></script>
```
推荐
```
<script src="//cdn.com/foundation.min.js"></script>
```

不推荐
```
.example {
    background: url(http://static.example.com/images/bg.jpg);
}
```
推荐
```
.example {
    background: url(//static.example.com/images/bg.jpg);
}
```

- 文本缩进

一次缩进两个空格。
```
<ul>
  <li>Fantastic</li>
  <li>Great</li>
```

```
  <li><a href="#">Test</a></li>
</ul>

(function(){
  var x = 10;

  function y(a, b) {
    return {
      result: (a + b) * x
    }
  }

}());
```

- 注释

不推荐

```
var offset = 0;
if(includeLabels) {
    // Add offset of 20
    offset = 20;
}
```

推荐

```
var offset = 0;
if(includeLabels) {
    // If the labels are included we need to have a minimum
    // offset of 20 pixels
    // We need to set it explicitly because of the following
    // bug: http://somebrowservendor.com/issue-tracker/ISSUE-1
    offset = 20;
}
```

2. HTML 规范

2.1. 文档类型

推荐使用 HTML5 的文档类型申明： <!DOCTYPE html>
HTML 中最好不要将无内容元素[1] 的标签闭合，例如：使用
 而非

2.2. HTML 验证

不推荐

```
<title>Test</title>
<article>This is only a test.
```

推荐

```
<!DOCTYPE html>
<meta charset="utf-8">
<title>Test</title>
<article>This is only a test.</article>
```

这里是没有 html 标签，html5 规范此标签可以省略，但是开发时不建议省略，可读性降低，好处是页面小了，加载快。

2.3. 脚本加载

出于性能考虑，脚本异步加载很关键。一段脚本放置在 <head> 内，比如 <script src="main.js"></script>，其加载会一直阻塞 DOM 解析，直至它完全地加载和执行完毕。这会造成页面显示的延迟。特别是一些重量级的脚本，对用户体验来说那真是一个巨大的影响。

异步加载脚本可缓解这种性能影响。如果只需兼容 IE10+，可将 HTML5 的 async 属性加至脚本中，它可防止阻塞 DOM 的解析，甚至你可以将脚本引用写在 <head> 里也没有影响。

如需兼容老旧的浏览器，实践表明可使用用来动态注入脚本的脚本加载器。你可以考虑 yepnope 或 labjs。注入脚本的一个问题是：一直要等到 CSS 对象文档已就绪，它们才开始加载（短暂地在 CSS 加载完毕之后），这就对需要及时触发的 JS 造成了一定的延迟，这多多少少也影响了用户体验吧。

终上所述，兼容老旧浏览器(IE9-)时，应该遵循以下最佳实践。

脚本引用写在 body 结束标签之前，并带上 async 属性。这虽然在老旧浏览器中不会异步加载脚本，但它只阻塞了 body 结束标签之前的 DOM 解析，这就大大降低了其阻塞影响。而在现代浏览器中，脚本将在 DOM 解析器发现 body 尾部的 script 标签才进行加载，此时加载属于异步加载，不会阻塞 CSSOM（但其执行仍发生在 CSSOM 之后）。

- 所有浏览器中，推荐

```
<html>
  <head>
    <link rel="stylesheet" href="main.css">
  </head>
  <body>
    <!-- body goes here -->
    <script src="main.js" async></script>
```

```
  </body>
</html>
```

注意代码中的 async，表示 js 文件是异步加载，异步加载的好处是不影响整体页面的加载，难点是多个相互依赖的 js 的加载问题，解决方案：可以把项目的 js 打包成一个文件,或者是使用 importScripts 在一个 js 中引入顺序引入其他 js，或者是使用

```
document.write(" <script language=javascript
src=' /js/import.js' ></script>" );
```

- 只在现代浏览器中，推荐

```
<html>
  <head>
    <link rel="stylesheet" href="main.css">
    <script src="main.js" async></script>
  </head>
  <body>
    <!-- body goes here -->
  </body>
</html>
```

2.4.语义化

- 不推荐

```
<b>My page title</b>
<div class="top-navigation">
  <div class="nav-item"><a href="#home">Home</a></div>
  <div class="nav-item"><a href="#news">News</a></div>
  <div class="nav-item"><a href="#about">About</a></div>
</div>
<div class="news-page">
  <div class="page-section news">
     <div class="title">All news articles</div>
     <div class="news-article">
       <h2>Bad article</h2>
       <div class="intro">Introduction sub-title</div>
       <div class="content">This is a very bademantics</div>
       <div class="article-side-notes">I thine main credits</div>
       <div class="article-foot-notes">
           This article wby David
       <div class="time">2014-01-01 00:00</div>
     </div>
  </div>
  <div class="section-footer">
```

```
    Related sections: Events, Public holidays
  </div>
</div>
<div class="page-footer"> Copyright 2014 </div>
```

- 推荐

```
<header>
    <h1>My page title</h1>
</header>
<nav class="top-navigation">
  <ul>
    <li class="nav-item"><a href="#home">Home</a></li>
    <li class="nav-item"><a href="#news">News</a></li>
    <li class="nav-item"><a href="#about">About</a></li>
  </ul>
</nav>
<footer class="page-footer">
  Copyright 2014
</footer>
```

2.5. 多媒体回溯

- 不推荐

```
<img src="luke-skywalker.jpg">
```

- 推荐

```
<img src="huge-spaceship-approaching-earth.jpg" alt="A huge spaceship
that is approaching the earth">
```

2.6. 关注点分离

- ✓ 不使用超过一到两张样式表（i.e. main.css, vendor.css）
- ✓ 不使用超过一到两个脚本（学会用合并脚本）
- ✓ 不使用行内样式（<style>.no-good {}</style>）
- ✓ 不在元素上使用 style 属性（<hr style="border-top: 5px solid black">）
- ✓ 不使用行内脚本（<script>alert('no good')</script>）
- ✓ 不使用表象元素（i.e. <b>, <u>, , <font>, <b>）
- ✓ 不使用表象 class 名（i.e. red, left, center）

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="base.css">
    <link rel="stylesheet" href="grid.css">
```

```
    <link rel="stylesheet" href="type.css">
    <link rel="stylesheet" href="modules/teaser.css">
  </head>
  <body>
    <h1 style="font-size: 3rem"></h1>
    <b>I'm a subtitle and I'm bold!</b>
    <center>Dare you center me!</center>
    <script> alert('Just dont...'); </script>
    <div class="red">I'm important!</div>
  </body>
</html>
```

仔细看看上面的问题

2.7. HTML 内容至上

✓ 不要引入一些特定的 HTML 结构来解决一些视觉设计问题
✓ 不要将 img 元素当做专门用来做视觉设计的元素

不推荐

```
<span class="text-box">
  <span class="square"></span>
  See the square next to me?
</span>
.text-box > .square {
  display: inline-block;
  width: 1rem;
  height: 1rem;
  background-color: red;
}
```

推荐

```
<span class="text-box">
  See the square next to me?
</span>
.text-box:before {
  content: "";
  display: inline-block;
  width: 1rem;
  height: 1rem;
  background-color: red;
}
```

2.8. Type 属性

HTML5 中以上两者默认的 type 值就是 text/css 和 text/javascript，所以 type 属性一般是可以忽略掉的

不推荐

```
<link rel="stylesheet" href="main.css" type="text/css">
<script src="main.js" type="text/javascript"></script>
```

推荐

```
<link rel="stylesheet" href="main.css">
<script src="main.js"></script>
```

2.9. ID 和锚点

在浏览器中输入 URL http://your-site.com/about#best-practices，浏览器将定位至以下 H3 上

```
<h3 id="best-practices">Best practices</h3>
```

2.10. 格式化规则

在每一个块状元素，列表元素和表格元素后，加上一新空白行，并对其子孙元素进行缩进。内联元素写在一行内，块状元素还有列表和表格要另起一行。

（如果由于换行的空格引发了不可预计的问题，那将所有元素并入一行也是可以接受的，格式警告总好过错误警告）。

```
<blockquote>
  <p><em>Space</em>, the final frontier.</p>
</blockquote>

<ul>
    <li>Moe</li>
    <li>Larry</li>
    <li>Curly</li>
</ul>

<table>
  <thead>
    <tr>
      <th scope="col">Income</th>
      <th scope="col">Taxes</th>
    </tr>
  </thead>
```

```
  <tbody>
    <tr>
      <td>$ 5.00</td>
      <td>$ 4.50</td>
    </tr>
   </tbody>
</table>
```

2.11. HTML 引号

不推荐

```
<div class='news-article'></div>
```

推荐

```
<div class="news-article"></div>
```

使用双引号(“”) 而不是单引号(”)

3. JavaScript 规范

3.1. 全局命名空间污染与 IIFE

不推荐

```
var x = 10, y = 100;
// Declaring variables in the global scope is resulting in global scope pollution.
All variables declared like this
// will be stored in the window object. This is very unclean and needs to be avoided.
console.log(window.x + ' ' + window.y);
```

推荐

```
// We declare a IIFE and pass parameters into the function that we will use from the global space
(function(log, w, undefined){
   'use strict';

   var x = 10, y = 100;
   // Will output 'true true'
   log((w.x === undefined) + ' ' + (w.y === undefined));

 }(window.console.log, window));
```

适当使用闭包隔离代码

3.2. IIFE（立即执行的函数表达式）

无论何时，想要创建一个新的封闭的定义域，那就用 IIFE。它不仅避免了干扰，也使得内存在执行完后立即释放。

所有脚本文件建议都从 IIFE 开始。

立即执行的函数表达式的执行括号应该写在外包括号内。虽然写在内还是写在外都是有效的，但写在内使得整个表达式看起来更像一个整体，因此推荐这么做。

不推荐

```
(function(){})();
```

推荐

```
(function(){}());
```

用下列写法来格式化你的 IIFE 代码：

```
(function(){
    //严格模式标志
    'use strict';

    // Code goes here

}());
```

如果你想引用全局变量或者是外层 IIFE 的变量，可以通过下列方式传参：

```
(function($, w, d){
    'use strict'; //严格模式标志

    $(function(){
        w.alert(d.querySelectorAll('div').length);
    });

}(jQuery, window, document));
```

不推荐

```
// Script starts here
'use strict';

(function(){

// Your code starts here

}());
```

推荐

```
(function(){
    'use strict';
    // Your code starts here

}());
```

3.3. 变量声明

不推荐

```
x = 10;
y = 100;
```

推荐

```
var x = 10, y = 100;
```

注意：两个不同点，变量不用 var 声明，变量将被隐式地声明为全局变量，到时候变量的值就难以控制。

3.4. 理解 JavaScript 的定义域和定义域提升

啥子叫定义域提升，JavaScript 只有 function 级的定义域，在此内定义的变量，即使是某语句和循环体中定义的变量都砸 function 的定义域中，在此域中都能访问到，这就是被提升了。在执行时，变量和方法会自动提升到执行之前。

```
(function(log){
  'use strict';
  var a = 10;
  for(var i = 0; i < a; i++) {
    var b = i * i;
    log(b);
  }

  if(a === 10) {
    var f = function() {
      log(a);
    };

    f();
  }

  function x() {
    log('Mr. X!');
```

```
  }

  x();

}(window.console.log));
```

3.5. 总是使用带类型判断的比较判断

注意条件判断“==”和“===”的区别：
“==”判断是否相等，带有强制类型转换，所带来的强制类型转换使得判断结果跟踪变得复杂。
“===”不带有转换，可以比较类型是否相同
例如：

```
(function(log){
  'use strict';

  log('0' == 0);// true
  log('' == false); // true
  log('1' == true); // true
  log(null == undefined); // true

  var x = {
    valueOf: function() {
      return 'X';
    }
  };

  log(x == 'X');

}(window.console.log));
```

3.6. 明智地使用真假判断

当我们在一个 if 条件语句中使用变量或表达式时，会做真假判断。if(a == true) 是不同于 if(a) 的。后者的判断比较特殊，我们称其为真假判断。这种判断会通过特殊的操作将其转换为 true 或 false，下列表达式统统返回 false：false, 0, undefined, null, NaN, ''（空字符串）.

这种真假判断在我们只求结果而不关心过程的情况下，非常的有帮助。

```
(function(log){
    'use strict';
    function logTruthyFalsy(expr) {
        if(expr) {
            log('truthy');
```

```
        } else {
            log('falsy');
        }
    }

    logTruthyFalsy(true); // truthy
    logTruthyFalsy(1); // truthy
    logTruthyFalsy({}); // truthy
    logTruthyFalsy([]); // truthy
    logTruthyFalsy('0'); // truthy
    logTruthyFalsy(false); // falsy
    logTruthyFalsy(0); // falsy
    logTruthyFalsy(undefined); // falsy
    logTruthyFalsy(null); // falsy
    logTruthyFalsy(NaN); // falsy
    logTruthyFalsy(''); // falsy

}(window.console.log));
```

3.7. 变量赋值时的逻辑操作

逻辑操作符 || 和 && 也可被用来返回布尔值。如果操作对象为非布尔对象，那每个表达式将会被自左向右地做真假判断。基于此操作，最终总有一个表达式被返回回来。这在变量赋值时，是可以用来简化你的代码的。

不推荐

```
if(!x) {
    if(!y) {
        x = 1;
    } else {
        x = y;
    }
}
```

推荐

```
x = x || y || 1;
```

3.8. 分号

总是使用分号，因为隐式的代码嵌套会引发难以察觉的问题。当然我们更要从根本上来杜绝这些问题。以下几个示例展示了缺少分号的危害：

```
// 1.
```

```
MyClass.prototype.myMethod = function() {
    return 42;
}
// No semicolon here.
(function() {
// Some initialization code wrapped in a function to create a scope for
locals.

 })();

var x = { 'i': 1, 'j': 2 }
// No semicolon here.
// 2. Trying to do one thing on Internet Explorer and another on Firefox.
// I know you'd never write code like this, but throw me a bone.
[ffVersion, ieVersion][isIE]();

var THINGS_TO_EAT = [apples, oysters, sprayOnCheese]

// No semicolon here.
// 3. conditional execution a la bash

-1 == resultOfOperation() || die();
```

以上代码发生了什么：

- ✧ JavaScript 错误 —— 首先返回 42 的那个 function 被第二个 function 当中参数传入调用，接着数字 42 也被“调用”而导致出错。
- ✧ 八成你会得到 ‘no such property in undefined’ 的错误提示，因为在真实环境中的调用是这个样子：x[ffVersion, ieVersion][isIE]().
- ✧ die 总是被调用。因为数组减 1 的结果是 NaN，它不等于任何东西（无论 resultOfOperation 是否返回 NaN）。所以最终的结果是 die() 执行完所获得值将赋给 THINGS_TO_EAT.

JavaScript 中语句要以分号结束，否则它将会继续执行下去，不管换不换行。

3.9. 语句块内的函数声明

切勿在语句块内声明函数，在 ECMAScript 5 的严格模式下，这是不合法的。
不推荐

```
if (x) {
    function foo() {}
}
```

推荐

```
if (x) {
    var foo = function() {};
}
```

3.10. 异常

基本上你无法避免出现异常，特别是在做大型开发时（使用应用开发框架等等）。

在没有自定义异常的情况下，从有返回值的函数中返回错误信息一定非常的棘手，更别提多不优雅了。不好的解决方案包括了传第一个引用类型来接纳错误信息，或总是返回一个对象列表，其中包含着可能的错误对象。以上方式基本上是比较简陋的异常处理方式。适时可做自定义异常处理。

在复杂的环境中，你可以考虑抛出对象而不仅仅是字符串（默认的抛出值）。

```
if(name === undefined) {
    throw {
        name: 'System Error',
        message: 'A name should always be specified!'
    }
}
```

3.11. 使用闭包

怎么使用闭包，前面已经说过，javascript 里面只有 function 定义域，故使用闭包有且只有一种方式，使用函数来定义闭包。

3.12. 切勿在循环中创建函数

```
(function(log, w){
    'use strict';
    // numbers and i is defined in the current function closure
    var numbers = [1, 2, 3], i;

    for(i = 0; i < numbers.length; i++) {
        w.setTimeout(function() {
    // At the moment when this gets executed the i variable, coming from
the outer function scope
```

```
    // is set to 3 and the current program is alerting the message 3 times
// 'Index 3 with number undefined
    // If you understand closures in javascript you know how to deal with
those cases
    // It's best to just avoid functions / new closures in loops as this
prevents those issues
            w.alert('Index ' + i + ' with number ' + numbers[i]);
        }, 0);
    }

}(window.console.log, window));
```

以上代码的问题，需要结合上面讲到的定义域提升来考虑。
不完全推荐

```
(function(log, w){
    'use strict';
    // numbers and i is defined in the current function closure
    var numbers = [1, 2, 3], i;
    // Create a function outside of the loop that will accept arguments
to create a
    // function closure scope. This function will return a function that
executes in this
    // closure parent scope.

    function alertIndexWithNumber(index, number) {
        return function() {
            w.alert('Index ' + index + ' with number ' + number);
        };
    }
    // First parameter is a function call that returns a function.
    // ---
    // This solves our problem and we don't create a function inside our
loop

    for(i = 0; i < numbers.length; i++){
        w.setTimeout(alertIndexWithNumber(i, numbers[i]),0);
    }
}(window.console.log, window));
```

推荐

```
(function(log, w){
    'use strict';
    // numbers and i is defined in the current function closure
```

```
    var numbers = [1, 2, 3], i;

    numbers.forEach(function(number, index) {
        w.setTimeout(function() {
            w.alert('Index ' + index + ' with number ' + number);
        }, 0);
    });

}(window.console.log, window));
```

3.13. eval 函数（魔鬼）

eval() 不但混淆语境还很危险，总会有比这更好、更清晰、更安全的另一种方案来写你的代码。

3.14. this 关键字

只在对象构造器、方法和在设定的闭包中使用 this 关键字。this 的语义在此有些误导。它时而指向全局对象（大多数时），时而指向调用者的定义域（在 eval 中），时而指向 DOM 树中的某一节点（当用事件处理绑定到 HTML 属性上时），时而指向一个新创建的对象（在构造器中），还时而指向其它的一些对象（如果函数被 call() 和 apply() 执行和调用时）。

正因为它是如此容易地被搞错，请限制它的使用场景：

- 在构造函数中
- 在对象的方法中（包括由此创建出的闭包内）

3.15. 首选函数式风格

函数式编程让你可以简化代码并缩减维护成本，因为它容易复用，又适当地解耦和更少的依赖。

例外：往往在重代码性能轻代码维护的情况之下，要选择最优性能的解决方案而非维护性高的方案（比如用简单的循环语句代替 forEach）。

不推荐

```
(function(log){
    'use strict';
    var arr = [10, 3, 7, 9, 100, 20], sum = 0, i;

    for(i = 0; i < arr.length; i++) {
        sum += arr[i];
    }
```

```
    log('The sum of array ' + arr + ' is: ' + sum);

}(window.console.log));
```

推荐

```
(function(log){
    'use strict';

    var arr = [10, 3, 7, 9, 100, 20];

    var sum = arr.reduce(function(prevValue, currentValue) {
        return prevValue + currentValue;
    }, 0);

    log('The sum of array ' + arr + ' is: ' + sum);

}(window.console.log));
```

3.16. 数组和对象字面量

不推荐

```
// Length is 3.
var a1 = new Array(x1, x2, x3);
// Length is 2.

var a2 = new Array(x1, x2);
// If x1 is a number and it is a natural number the length will be x1.
// If x1 is a number but not a natural number this will throw an exception.
// Otherwise the array will have one element with x1 as its value.

var a3 = new Array(x1);

// Length is 0.

var a4 = new Array();
```

正因如此，如果将代码传参从两个变为一个，那数组很有可能发生意料不到的长度变化。为避免此类怪异状况，请总是采用更多可读的数组字面量。

推荐

```
var a = [x1, x2, x3];
var a2 = [x1, x2];
var a3 = [x1];
var a4 = [];
```

对象构造器不会有类似的问题，但是为了可读性和统一性，我们应该使用对象字面量。

不推荐

```
var o = new Object();
var o2 = new Object();
o2.a = 0;
o2.b = 1;
o2.c = 2;
o2['strange key'] = 3;
```

推荐

```
var o = {};
var o2 = {
    a: 0,
    b: 1,
    c: 2,
    'strange key': 3
};
```

3.17. 修改内建对象的原型链

修改内建的诸如 Object.prototype 和 Array.prototype 是被严厉禁止的。修改其它的内建对象比如 Function.prototype，虽危害没那么大，但始终还是会导致在开发过程中难以 debug 的问题，应当也要避免。

3.18. 字符串

统一使用单引号(‘)，不使用双引号(“)。这在创建 HTML 字符串非常有好处：

```
var msg = 'This is some HTML <div class="makes-sense"></div>';
```

3.19. 三元条件判断

不推荐

```
if(x === 10) {
    return 'valid';
} else {
    return 'invalid';
}
```

推荐

```
return x === 10 ? 'valid' : 'invalid';
```

4. CSS 和 Sass (SCSS) 规范

4.1. ID and class naming

ID 和 class(类)名总是使用可以反应元素目的和用途的名称，或其他通用名称。代替表象和晦涩难懂的名称。

应该首选具体和反映元素目的的名称，因为这些是最可以理解的，而且发生变化的可能性最小。

通用名称只是多个元素的备用名，他们兄弟元素之间是一样的，没有特别意义。
区分他们，使他们具有特殊意义，通常需要为“帮手”。

尽管 class(类)名和 ID 的语义化对于计算机解析来说没有什么实际的意义，
语义化的名称 通常是正确的选择，因为它们所代表的信息含义，不包含表现的限制。
不推荐

```
.fw-800 {
  font-weight: 800;
}

.red {
  color: red;
}
```

推荐

```
.heavy {
  font-weight: 800;
}

.important {
  color: red;
}
```

4.2. 合理的避免使用 ID

一般情况下 ID 不应该被应用于样式。ID 的样式不能被复用并且每个页面中你只能使用一次 ID。使用 ID 唯一有效的是确定网页或整个站点中的位置。尽管如此，你应该始终考虑使用 class，而不是 id，除非只使用一次。
不推荐

```
#content .title {
  font-size: 2em;
```

```
}
```
推荐
```
.content .title {
  font-size: 2em;
}
```
另一个反对使用 ID 的观点是含有 ID 选择器权重很高。

4.3. CSS 选择器中避免标签名

不推荐

```
div.content > header.content-header > h2.title {
    font-size: 2em;
}
```
推荐
```
.content > .content-header > .title {
    font-size: 2em;
}
```

4.4. 尽可能的精确

很多前端开发人员写选择器链的时候不使用 直接子选择器（注：直接子选择器和后代选择器的区别）。
有时，这可能会导致疼痛的设计问题并且有时候可能会很耗性能。
```
<div class='con'>
  <div>
    <p>span1
      <div>
        <p>span2</p>
      </div>
    </p>
  </div>
</div>
```
要解决 con 类下的 p 下不同的样式，这将需要更精确的选择器再次重写他们的样式。
不推荐

```
.con div p{
    color:red;
}
```

以上会把 span1 和 span2 均设置成红色
推荐

```
div.con > div > p {
  color:red;
}
```

4.5. 缩写属性

不推荐

```
border-top-style: none;
font-family: palatino, georgia, serif;
font-size: 100%;
line-height: 1.6;
padding-bottom: 2em;
padding-left: 1em;
padding-right: 1em;
padding-top: 0;
```

推荐

```
border-top: 0;
font: 100%/1.6 palatino, georgia, serif;
padding: 0 1em 2em;
```

4.6. 0 和 单位

不推荐

```
padding-bottom: 0px;
margin: 0em;
```

推荐

```
padding-bottom: 0;

margin: 0;
```

4.7. 十六进制表示法

在可能的情况下，使用 3 个字符的十六进制表示法。
颜色值允许这样表示，
3 个字符的十六进制表示法更简短。
始终使用小写的十六进制数字。

不推荐

```
color: #FF33AA;
```

推荐

```
color: #f3a;
```

4.8. ID 和 Class（类） 名的分隔符

使用连字符（中划线）分隔 ID 和 Class（类）名中的单词。为了增强课理解性，在选择器中不要使用除了连字符（中划线）以为的任何字符（包括没有）来连接单词和缩写。

另外，作为该标准，预设属性选择器能识别连字符（中划线）作为单词[attribute|=value]的分隔符，
所以最好的坚持使用连字符作为分隔符。
不推荐

```
.demoimage {}
.error_status {}
```

推荐

```
#video-id {}
.ads-sample {}
```

4.9. 声明顺序

这是一个选择器内书写 CSS 属性顺序的大致轮廓。这是为了保证更好的可读性和可扫描重要。
作为最佳实践，我们应该遵循以下顺序（应该按照下表的顺序）：

- 结构性属性：
  1. display

2. position, left, top, right etc.
3. overflow, float, clear etc.
4. margin, padding

- 表现性属性：
  1. background, border etc.
  2. font, text

不推荐

```
.box {
  font-family: 'Arial', sans-serif;
  border: 3px solid #ddd;
  left: 30%;
  position: absolute;
  text-transform: uppercase;
  background-color: #eee;
  right: 30%;
  display: block;
  font-size: 1.5rem;
  overflow: hidden;
  padding: 1em;
  margin: 1em;
}
```

推荐

```
.box {
  display: block;
  position: absolute;
  left: 30%;
  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;
}
```

4.10. 声明结束

为了保证一致性和可扩展性，每个声明应该用分号结束，每个声明换行。

不推荐

```
.test {
  display: block;
  height: 100px
```

```
}
```
推荐
```
.test {
  display: block;
  height: 100px;
}
```

4.11. 属性名结束

属性名的冒号后使用一个空格。出于一致性的原因，
属性和值（但属性和冒号之间没有空格）的之间始终使用一个空格。
不推荐

```
h3 {
  font-weight:bold;
}
```
推荐
```
h3 {
  font-weight: bold;
}
```

4.12. 规则分隔

规则之间始终有一个空行（双换行符）分隔。
推荐

```
html {
  background: #fff;
}

body {
  margin: auto;
  width: 50%;
}
```

4.13. CSS 引号

属性选择器或属性值用双引号（” ”），而不是单引号（’ ’）括起来。
URI 值（url()）不要使用引号。

不推荐

```
@import url('//cdn.com/foundation.css');

html {
  font-family: 'open sans', arial, sans-serif;
}

body:after {
content: 'pause';
}
```

推荐

```
@import url(//cdn.com/foundation.css);

html {
  font-family: "open sans", arial, sans-serif;
}

body:after {
  content: "pause";
}
```

4.14. 选择器嵌套 (SCSS)

在 Sass 中你可以嵌套选择器，这可以使代码变得更清洁和可读。嵌套所有的选择器，但尽量避免嵌套没有任何内容的选择器。
如果你需要指定一些子元素的样式属性，而父元素将不什么样式属性，
可以使用常规的 CSS 选择器链。
这将防止您的脚本看起来过于复杂。
不推荐

```
.content {
  display: block;
}

.content > .news-article > .title {
  font-size: 1.2em;
}
```

不推荐

```
.content {
  display: block;
  > .news-article {
```

```
    > .title {
      font-size: 1.2em;
    }
  }
}
```

推荐

```
.content {
  display: block;
  > .news-article > .title {
    font-size: 1.2em;
  }
}
```

4.15. 嵌套中引入 空行 (SCSS)

嵌套选择器和 CSS 属性之间空一行。

不推荐

```
.content {
  display: block;
  > .news-article {
    background-color: #eee;
    > .title {
      font-size: 1.2em;
    }
    > .article-footnote {
      font-size: 0.8em;
    }
  }
}
```

推荐

```
.content {
   display: block;

   > .news-article {
     background-color: #eee;

     > .title {
        font-size: 1.2em;
     }

     > .article-footnote {
         font-size: 0.8em;
     }
  }
```

```
}
```