前端工程师纳米学位样式指南 - JavaScript

点此查看文档英文版（pdf，GitHub.io）

简介

该指南是你在项目进行过程中所需遵守的官方指南。优达学城的评估人员会根据该指南为你的项目打分。在前端网页开发的世界中，有很“最佳”样式供你选择。因此，为了减少学生在项目过程中因选择何种样式所产生的困惑，我们强烈建议所有学生在其项目中遵循这个样式指南。

一般格式规则

末尾空白

删除行尾空格。

行尾空格属于多余的符号，会使 diff 更加难以阅读。

不推荐：

```
var name = "John Smith";__
```

推荐：

```
var name = "John Smith";
```

如果使用 Sublime Text，你可在用户设置（User Settings）JSON 文件（可在文本编辑器的菜单中找到）中添加以下代码，每当你以此方法储存文件时，去除行尾空格操作便会自动完成：

```
"trim_trailing_white_space_on_save": true
```

缩进

整个文件中的缩进应保持前后一致，使用 Tab、2个空格或4个空格都可以，但需保持前后一致。

一般元规则

编码

使用 UTF-8（无 BOM）。

确保你的编辑器将没有字节顺序标记的 UTF-8 用作字符编码。

注释

用注释解释代码的覆盖范围、目的和作用以及使用和选择各解决方案的原因。

你可以选择使用 JSDoc，即编写代码注释的文件生成器和标准，对你的 JavaScript 功能进行记录，其优点包括为你的注释提供技术参照和能够针对文件生成网页的命令行 jsdoc 工具。 JSDoc 会为你提供记录代码的多种注释，但我们只推荐你使用以下种类：

- @constructor：用于记录类别，即用新关键词调用的函数。
- @description：用于描述你的函数，该标签还可以使你在需要时添加 HTML 标记。
- @param：用于描述函数参数的名称、类别和说明。
- @returns：记录函数返回值的类型和说明。

该实例说明了如何记录类构造器（注意注释区开头使用的 /**，这个非常重要）：

```
/**
* @description Represents a book
* @constructor
* @param {string} title - The title of the book
* @param {string} author - The author of the book
*/
function Book(title, author) {
        ...
}
```

以下函数含有能返回值的参数，注意，这里的参数作用一目了然，因此并未对其进行说明。

```
/**
* @description Adds two numbers
* @param {number} a
* @param {number} b
* @returns {number} Sum of a and b
*/
function sum(a, b) {
        return a + b;
}
```

你也可以使用更多你想要编写的注释。

任务项

用 TODO: 标注待办事项和任务项：

仅用关键词 TODO 标注待办事项，不要使用 @@ 等其他格式的字样。在任务项前加冒号，如：TODO: action item。

推荐：

```
// TODO: add other fruits
```

JavaScript 语言规则

变量

使用 var 声明变量。

如果你未能设置变量，其会被置于全局语境中并可能清除现有值。若没有任何声明，变量的存在范围将难以确定。

常量

如果要将某个值设为不可改变的常量，应用大写字母对其命名，如 CONSTANT_VALUE。不要使用关键词 const，因为部分浏览器不支持此用法。

分号

始终使用分号。

依靠隐式插入会造成难以排除的细微问题。分号应放在函数表达式的末尾，而不是函数声明的末尾。

不推荐：

```
var foo = function() {
        return true // Missing semicolon
} // Missing semicolon
function foo() {
        return true;
}; // Extra semicolon
```

推荐：

```
var foo = function() {
        return true;
};
function foo() {
        return true;
}
```

基本类型包装对象

基本类型无需使用包装对象，此外，包装对象还具有潜在危险，但可以使用类型转换。

不推荐：

```
var x = new Boolean(0);
```

```
if (x) {
    alert('hi');      // Shows 'hi' because typeof x is truthy object
}
```

推荐：

```
var x = Boolean(false);
if (x) {
    alert('hi');      // Show 'hi' because typeof x is a falsey boolean
}
```

闭包

进行该操作时要小心谨慎。

创建闭包的能力可能是 JavaScript 中最有用但最常被忽略的能力。需要记住的是，闭包会储存对其封闭范围的指示器，因此，将 DOM 元素与闭包相连会生成循环引用，从而导致内存的泄露。

不推荐：

```
function foo(element, a, b) {
    element.onclick = function() { /* uses a and b */ }
}
```

推荐：

```
function foo(element, a, b) {
    element.onclick = bar(a, b);
}
function bar(a, b) {
        return function() { /* uses a and b */ }
}
```

for 、for-in 和 forEach

数组

在迭代数组时，相比 `for-in` 循环， `forEach` 或 `for` 循环更具优势。

不推荐：

```
myArray = ['a', 1, 'etc'];
for (var indexNum in myArray) {
   console.log(myArray[indexNum]);
}
var starWars = {
   "creatures": [
          {
          "name": "bantha",
          "face": "furry"
          },
          {
          "name": "loth-cat",
          "face": "toothy"
          }
          ]
};
for (var i in starWars.creatures) {
   console.log(starWars.creatures[i].name);
   console.log(starWars.creatures[i].face);
};
```

推荐：

```
mySimpleArray = ['a', 1, 'etc'];
mySimpleArray.forEach(function(val) {
   console.log(val);
});
var starWars = {
   "creatures": [
```

```
        {
        "name": "bantha",
        "face": "furry"
        },
        {
        "name": "loth-cat",
        "face": "toothy"
        }
        ]
};
starWars.creatures.forEach(function(creature){
    console.log(creature.name);
    console.log(creature.face)
});
// or
myArray = ['a', 1, 'etc'];
for (var indexCount = 0; indexCount < myArray.length; indexCount++) {
    console.log(myArray[indexCount]);
};
```

对象

for-in 循环用于在对象中循环关键词。这样很容易出错，因为， for-in 不会从 0 循环至 length - 1，而是循环对象及其原型链中现存的所有关键词。

如果可以的话，对数据进行整理，以避免迭代对象。如果不可行，将 for-in 循环的内容包裹在条件语句中，以避免迭代原型链。

不推荐：

```
myObj = {'firstName':'Ada','secondName':'Lovelace'};
for (var key in myObj) {
    console.log(myObj[key]);
}
```

推荐：

```
myObj = {'firstName':'Ada','lastName':'Lovelace'};
for (var key in myObj) {
        if (myObj.hasOwnProperty(key)) {
    console.log(myObj[key]);
        }
}
```

多行字符串字面量

不要使用。

编译期间无法妥善删除各行开头的空格，斜杠后的空白会引发棘手的问题，虽然大部分脚本引擎支持该操作，但这并不属于规格中的一部分。

不推荐：

```
var myString = 'A rather long string of English text, an error message \
        actually that just keeps going and going -- an error \
        message that is really really long.';
```

推荐：

```
var myString = 'A rather long string of English text, an error message' +
        'actually that just keeps going and going -- an error' +
        'message that is really really long.';
```

数组和对象字面量

使用数组和对象字面量，而不是数组和对象构造函数。

不推荐：

```
var myArray = new Array(x1, x2, x3);
var myObject = new Object();
myObject.a = 0;
```

推荐：

```
var myArray = [x1, x2, x3];
var myObject = {
        a: 0
};
```

JavaScript 样式规则

命名

总体来说，函数名称为 functionNames，变量名称为 variableNames，类名称为 ClassNames，方法名称为 methodNames，常量值名称为 CONSTANT_VALUES，文件名称为 filenames。

代码格式

由于分号的隐式插入，大括号应与其内容放置在同一行。

推荐：

```
if (something) {
        // Do something
} else {
        // Do something else
}
```

只有在单行数组和对象初始器可以在写同一行时方可使用这两项。左括号前和右括号后都不应有空格。

推荐：

```
var array = [1, 2, 3];
var object = {a: 1, b: 2, c: 3};
```

多行数组和对象初始器需进行单行缩进，与代码块一样，其括号与内容应位于同一行。

推荐：

```
var array = [
        'Joe <joe@email.com>',
        'Sal <sal@email.com>',
        'Murr <murr@email.com>',
        'Q <q@email.com>'
];
var object = {
        id: 'foo',
        class: 'foo-important',
        name: 'notification'
};
```

插入语

仅在需要时使用。

仅在语法和语义需要时少量、概括性地使用。

字符串

为了保持连贯性，应使用单引号（'）而不是双引号（"）。这在创建含有 HTML 的字符串时尤其有帮助。

推荐：

```
var element = '<button class="btn">Click Me</button>';
```

**此规则较为明显的一个例外是在 JSON 对象中： JSON 要求使用双引号。

技巧提示

真假布尔表达式

以下为假的布尔表达式：

- Null
- undefined
- '' the empty string

- 0 the number

注意区分，以下为真的表达式：

- '0' the string
- [] the empty array
- {} the empty object

三元条件运算符

不强制规定，但建议使用三元条件运算符编写简洁代码，避免使用以下代码：

不推荐：

```
if (val) {
        return foo();
} else {
        return bar();
}
```

你可以这样编写：

推荐：

```
return val ? foo() : bar();
```

&& 和 ||

这些二元布尔运算符为简化形式，会计算至最后一个已计算项。|| 被称为默认的运算符，这是因为不应以下列方式编写代码：

不推荐：

```
function foo(name) {
        var theName;
        if (name) {
        theName = name;
        } else {
        theName = 'John';
```

```
        }
}
```

而应这样编写：

推荐：

```
function foo(name) {
        var theName = name || 'John';
}
```

&& 也被用于缩减代码。例如，不用这样编写：

不推荐：

```
if (node) {
        if (node.kids) {
    console.log(node.kids);
        }
}
```

而这样编写代码：

推荐：

```
if (node && node.kids) {
  console.log(node.kids);
}
```