JavaScript Standard Style

细则

- **使用两个空格**进行缩进。

  eslint: `indent`

```
function hello(name) {
  console.log('hi', name);
}
```

- 除需要转义的情况外，**字符串统一使用单引号。**

  eslint: `quotes`

```
console.log('hello there');
$("<div class='box'>");
```

- **不要定义未使用的变量。**

  eslint: `no-unused-vars`

```
function myFunction() {
  var result = something();   // ✗ avoid
}
```

- **关键字后面加空格。**

  eslint: `keyword-spacing`

```
if (condition) { ... }   // ✓ ok
if(condition) { ... }    // ✗ avoid
```

- **函数声明时括号与函数名间不加空格。**

  eslint: `space-before-function-paren`

```
function name (arg) { ... }   // ✗ avoid
function name(arg) { ... }    // ✓ ok

run(function () { ... });      // ✗ avoid
run(function() { ... });       // ✓ ok
```

- **始终使用** `===` 替代 `==`。
  例外： `obj == null` 可以用来检查 `null || undefined`。

  eslint: `eqeqeq`

```
if (name === 'John')   // ✓ ok
if (name == 'John')    // ✗ avoid
```

```
if (name !== 'John')   // ✓ ok
if (name != 'John')    // ✗ avoid
```

- **字符串拼接操作符 (Infix operators)** 之间要留空格。

  eslint: `space-infix-ops`

```
// ✓ ok
var x = 2;
var message = 'hello, ' + name + '!';
```

```
// ✗ avoid
var x=2;
var message = 'hello, '+name+'!';
```

- **逗号后面加空格。**

  eslint: `comma-spacing`

```
// ✓ ok
var list = [1, 2, 3, 4];
function greet(name, options) { ... }
```

```
// ✗ avoid
var list = [1,2,3,4];
function greet(name,options) { ... }
```

- **else 关键字要与花括号**保持在同一行。

  eslint: `brace-style`

```
// ✓ ok
if (condition) {
  // ...
} else {
  // ...
}
```

```
// ✗ avoid
if (condition) {
  // ...
}
else {
  // ...
}
```

- **多行 if 语句的**的括号不能省。

  eslint: `curly`

```
// ✓ ok
if (options.quiet !== true) console.log('done');
```

```
// ✓ ok
if (options.quiet !== true) {
  console.log('done');
}
```

```
// ✗ avoid
if (options.quiet !== true)
  console.log('done');
```

- **不要丢掉**异常处理中`err`参数。

  eslint: `handle-callback-err`

```
// ✓ ok
run(function(err) {
  if (err) throw err;
  window.alert('done');
});
```

```
// ✗ avoid
run(function(err) {
  window.alert('done');
});
```

- **使用浏览器全局变量时加上** `window.` 前缀。
  Exceptions are: `document`, `console` and `navigator`.

  eslint: `no-undef`

```
window.alert('hi');   // ✓ ok
```

- **不允许有连续多行空行。**

  eslint: `no-multiple-empty-lines`

```
// ✓ ok
var value = 'hello world';
console.log(value);
```

```
// ✗ avoid
var value = 'hello world';
```

console.log(value);

```
* **对于三元运算符** `?` 和 `:` 与他们所负责的代码处于同一行

  eslint: [`operator-linebreak`](http://eslint.org/docs/rules/operator-linebreak)

  ```js
  // ✓ ok
  var location = env.development ? 'localhost' : 'www.api.com';

  // ✓ ok
  var location = env.development
    ? 'localhost'
    : 'www.api.com';

  // ✗ avoid
  var location = env.development ?
    'localhost' :
    'www.api.com';
```

- **每个 var 关键字**单独声明一个变量。
```

eslint: `one-var`

```
// ✓ ok
var silent = true;
var verbose = true;

// ✗ avoid
var silent = true, verbose = true;

// ✗ avoid
var silent = true,
    verbose = true;
```

- **条件语句中赋值语句**使用括号包起来。这样使得代码更加清晰可读，而不会认为是将条件判断语句的全等号（`===`）错写成了等号（`=`）。

  eslint: `no-cond-assign`

```
// ✓ ok
while ((m = text.match(expr))) {
  // ...
}

// ✗ avoid
while (m = text.match(expr)) {
  // ...
}
```

- **单行代码块两边加空格。**

  eslint: `block-spacing`

```
  function foo() {return true;}    // ✗ avoid
  function foo() { return true; }  // ✓ ok
```

- **对于变量和函数名统一使用驼峰命名法。**

  eslint: `camelcase`

```
  function my_function() { }    // ✗ avoid
  function myFunction() { }     // ✓ ok

  var my_var = 'hello';           // ✗ avoid
  var myVar = 'hello';            // ✓ ok
```

- **不允许有多余的行末逗号。**

  eslint: `comma-dangle`

```
var obj = {
  message: 'hello',    // ✗ avoid
};
```

- **始终将逗号置于行末。**

  eslint: `comma-style`

```
var obj = {
  foo: 'foo'
  ,bar: 'bar'    // ✗ avoid
};

var obj = {
  foo: 'foo',
  bar: 'bar'    // ✓ ok
};
```

- **点号操作符须与属性需在同一行。**

  eslint: `dot-location`

```
console.
  log('hello');  // ✗ avoid

console
  .log('hello'); // ✓ ok
```

- **文件末尾留一空行。**

  elint: `eol-last`

- **函数调用时标识符与括号间不留间隔。**

  eslint: `func-call-spacing`

```
console.log ('hello'); // ✗ avoid
console.log('hello');  // ✓ ok
```

- **键值对当中冒号与值之间要留空白。**

  eslint: `key-spacing`

```
var obj = { 'key' : 'value' };     // ✗ avoid
var obj = { 'key' :'value' };      // ✗ avoid
var obj = { 'key':'value' };       // ✗ avoid
var obj = { 'key': 'value' };      // ✓ ok
```

- **构造函数要以大写字母开头。**

  eslint: `new-cap`

```
function animal() {}
var dog = new animal();    // ✗ avoid

function Animal() {}
var dog = new Animal();    // ✓ ok
```

- **无参的构造函数调用时要带上括号。**

  eslint: `new-parens`

```
function Animal() {}
var dog = new Animal;    // ✗ avoid
var dog = new Animal();  // ✓ ok
```

- **对象中定义了存值器，一定要对应的定义取值器。**

  eslint: `accessor-pairs`

```
var person = {
  set name(value) {    // ✗ avoid
    this.name = value;
  }
};

var person = {
  set name(value) {
    this.name = value;
  },
  get name() {         // ✓ ok
    return this.name;
  }
};
```

- **子类的构造器中一定要调用 `super`**

  eslint: `constructor-super`

```
class Dog {
  constructor() {
    super();   // ✗ avoid
  }
}

class Dog extends Mammal {
  constructor() {
    super();   // ✓ ok
  }
}
```

- 使用数组字面量而不是构造器。

  eslint: `no-array-constructor`

```
var nums = new Array(1, 2, 3);   // ✗ avoid
var nums = [1, 2, 3];            // ✓ ok
```

- 避免使用 `arguments.callee` 和 `arguments.caller`。

  eslint: `no-caller`

```
function foo(n) {
  if (n <= 0) return;

  arguments.callee(n - 1);   // ✗ avoid
}

function foo(n) {
  if (n <= 0) return;

  foo(n - 1);
}
```

- 避免对类名重新赋值。

  eslint: `no-class-assign`

```
class Dog {}
Dog = 'Fido';    // ✗ avoid
```

- 避免修改使用 `const` 声明的变量。

  eslint: `no-const-assign`

```
const score = 100;
score = 125;        // ✗ avoid
```

- **避免使用常量作为条件表达式的条件（循环语句除外）。**

  eslint: `no-constant-condition`

```
if (false) {    // ✗ avoid
  // ...
}

if (x === 0) {  // ✓ ok
  // ...
}

while (true) {  // ✓ ok
  // ...
}
```

- **正则中不要使用控制符。**

  eslint: `no-control-regex`

```
var pattern = /\x1f/;    // ✗ avoid
var pattern = /\x20/;    // ✓ ok
```

- **不要使用 `debugger`。**

  eslint: `no-debugger`

```
function sum(a, b) {
  debugger;      // ✗ avoid
  return a + b;
}
```

- **不要对变量使用 `delete` 操作。**

  eslint: `no-delete-var`

```
var name;
delete name;     // ✗ avoid
```

- **不要定义冗余的函数参数。**

  eslint: `no-dupe-args`

```
function sum(a, b, a) {  // ✗ avoid
  // ...
}

function sum(a, b, c) {  // ✓ ok
  // ...
}
```

- **类中不要定义冗余的属性。**

  eslint: `no-dupe-class-members`

```
class Dog {
  bark() {}
  bark() {}    // ✗ avoid
}
```

- **对象字面量中不要定义重复的属性。**

  eslint: `no-dupe-keys`

```
var user = {
  name: 'Jane Doe',
  name: 'John Doe'    // ✗ avoid
};
```

- **`switch` 语句中不要定义重复的 `case` 分支。**

  eslint: `no-duplicate-case`

```
switch (id) {
  case 1:
    // ...
  case 1:     // ✗ avoid
}
```

- **同一模块有多个导入时一次性写完。**

  eslint: `no-duplicate-imports`

```
import { myFunc1 } from 'module';
import { myFunc2 } from 'module';          // ✗ avoid

import { myFunc1, myFunc2 } from 'module'; // ✓ ok
```

- **正则中不要使用空字符。**

  eslint: `no-empty-character-class`

```
const myRegex = /^abc[]/;       // ✗ avoid
const myRegex = /^abc[a-z]/;    // ✓ ok
```

- **不要解构空值。**

  eslint: `no-empty-pattern`

```
const { a: {} } = foo;          // ✗ avoid
const { a: { b } } = foo;       // ✓ ok
```

- **不要使用 `eval()`。**

  eslint: `no-eval`

```
eval( "var result = user." + propName ); // ✗ avoid
var result = user[propName];             // ✓ ok
```

- **`catch` 中不要对错误重新赋值。**

  eslint: `no-ex-assign`

```
try {
  // ...
} catch (e) {
  e = 'new value';              // ✗ avoid
}

try {
  // ...
} catch (e) {
  const newVal = 'new value';   // ✓ ok
}
```

- **不要扩展原生对象。**

  eslint: `no-extend-native`

```
Object.prototype.age = 21;      // ✗ avoid
```

- **避免多余的函数上下文绑定。**

  eslint: `no-extra-bind`

```
const name = function() {
  getName();
}.bind(user);     // ✗ avoid

const name = function() {
  this.getName();
}.bind(user);     // ✓ ok
```

- **避免不必要的布尔转换。**

  eslint: `no-extra-boolean-cast`

```
const result = true;
if (!!result) {    // ✗ avoid
  // ...
}

const result = true;
if (result) {      // ✓ ok
  // ...
}
```

- **不要使用多余的括号包裹函数。**

  eslint: `no-extra-parens`

```
const myFunc = (function() { });    // ✗ avoid
const myFunc = function() { };      // ✓ ok
```

- **`switch` 一定要使用 `break` 来将条件分支正常中断。**

  eslint: `no-fallthrough`

```
switch (filter) {
  case 1:
    doSomething();    // ✗ avoid
  case 2:
    doSomethingElse();
}

switch (filter) {
  case 1:
    doSomething();
    break           // ✓ ok
  case 2:
    doSomethingElse();
}

switch (filter) {
  case 1:
    doSomething();
    // fallthrough  // ✓ ok
  case 2:
    doSomethingElse();
}
```

- **不要省去小数点前面的0。**

  eslint: `no-floating-decimal`

```
const discount = .5;      // ✗ avoid
const discount = 0.5;    // ✓ ok
```

- **避免对声明过的函数重新赋值。**

  eslint: `no-func-assign`

```
function myFunc() { }
myFunc = myOtherFunc;    // ✗ avoid
```

- **不要对全局只读对象重新赋值。**

  eslint: `no-global-assign`

```
window = {};     // ✗ avoid
```

- **注意隐式的 `eval()`。**

eslint: `no-implied-eval`

```
setTimeout("alert('Hello world')");                    // ✗ avoid
setTimeout(function() { alert('Hello world'); });      // ✓ ok
```

- **嵌套的代码块中禁止再定义函数。**

  eslint: `no-inner-declarations`

```
if (authenticated) {
  function setAuthUser() {}     // ✗ avoid
}
```

- **不要向 `RegExp` 构造器传入非法的正则表达式。**

  eslint: `no-invalid-regexp`

```
RegExp('[a-z');    // ✗ avoid
RegExp('[a-z]');   // ✓ ok
```

- **不要使用非法的空白符。**

  eslint: `no-irregular-whitespace`

```
function myFunc() /*<NBSP>*/{}    // ✗ avoid
```

- **禁止使用 `__iterator__`。**

  eslint: `no-iterator`

```
Foo.prototype.__iterator__ = function() {};    // ✗ avoid
```

- **外部变量不要与对象属性重名。**

  eslint: `no-label-var`

```
var score = 100;
function game() {
  score: 50          // ✗ avoid
}
```

- **不要使用标签语句。**

  eslint: `no-labels`

```
label:
  while (true) {
    break label;     // ✗ avoid
  }
```

- **不要书写不必要的嵌套代码块。**

  eslint: `no-lone-blocks`

```
function myFunc() {
  {                  // ✗ avoid
    myOtherFunc();
  }
}

function myFunc() {
  myOtherFunc();       // ✓ ok
}
```

- **不要混合使用空格与制表符作为缩进。**

  eslint: `no-mixed-spaces-and-tabs`

- **除了缩进，不要使用多个空格。**

  eslint: `no-multi-spaces`

```
const id =    1234;    // ✗ avoid
const id = 1234;       // ✓ ok
```

- **不要使用多行字符串。**

  eslint: `no-multi-str`

```
const message = 'Hello \
                world';     // ✗ avoid
```

- **`new` 创建对象实例后需要赋值给变量。**

  eslint: `no-new`

```
new Character();                     // ✗ avoid
const character = new Character();   // ✓ ok
```

- **禁止使用 `Function` 构造器。**

  eslint: `no-new-func`

```
var sum = new Function('a', 'b', 'return a + b');    // ✗ avoid
```

- 禁止使用 `Object` 构造器。

  eslint: `no-new-object`

```
let config = new Object();   // ✗ avoid
```

- 禁止使用 `new require`。

  eslint: `no-new-require`

```
const myModule = new require('my-module');    // ✗ avoid
```

- 禁止使用 `Symbol` 构造器。

  eslint: `no-new-symbol`

```
const foo = new Symbol('foo');   // ✗ avoid
```

- 禁止使用原始包装器。

  eslint: `no-new-wrappers`

```
const message = new String('hello');   // ✗ avoid
```

- 不要将全局对象的属性作为函数调用。

  eslint: `no-obj-calls`

```
const math = Math();   // ✗ avoid
```

- 不要使用八进制字面量。

  eslint: `no-octal`

```
const num = 042;     // ✗ avoid
const num = '042';   // ✓ ok
```

- 字符串字面量中也不要使用八进制转义字符。

  eslint: `no-octal-escape`

```
const copyright = 'Copyright \251';  // ✗ avoid
```

- 使用 `__dirname` 和 `__filename` 时尽量避免使用字符串拼接。

eslint: `no-path-concat`

```
const pathToFile = __dirname + '/app.js';            // ✗ avoid
const pathToFile = path.join(__dirname, 'app.js');   // ✓ ok
```

- 使用 `getPrototypeOf` 来替代 `__proto__`。

  eslint: `no-proto`

```
const foo = obj.__proto__;               // ✗ avoid
const foo = Object.getPrototypeOf(obj);  // ✓ ok
```

- **不要重复声明变量。**

  eslint: `no-redeclare`

```
let name = 'John';
let name = 'Jane';     // ✗ avoid

let name = 'John';
name = 'Jane';         // ✓ ok
```

- **正则中避免使用多个空格。**

  eslint: `no-regex-spaces`

```
const regexp = /test   value/;   // ✗ avoid

const regexp = /test {3}value/;  // ✓ ok
const regexp = /test value/;     // ✓ ok
```

- **return 语句中的赋值必需有括号包裹。**

  eslint: `no-return-assign`

```
function sum(a, b) {
  return result = a + b;     // ✗ avoid
}

function sum(a, b) {
  return (result = a + b);   // ✓ ok
}
```

- **避免将变量赋值给自己。**

  eslint: `no-self-assign`

```
name = name;    // ✗ avoid
```

- **避免将变量与自己进行比较操作。**

  esint: `no-self-compare`

```
if (score === score) {}    // ✗ avoid
```

- **避免使用逗号操作符。**

  eslint: `no-sequences`

```
if (doSomething(), !!test) {}    // ✗ avoid
```

- **不要随意更改关键字的值。**

  eslint: `no-shadow-restricted-names`

```
let undefined = 'value';      // ✗ avoid
```

- **禁止使用稀疏数组（Sparse arrays）。**

  eslint: `no-sparse-arrays`

```
let fruits = ['apple',, 'orange'];        // ✗ avoid
```

- **不要使用制表符。**

  eslint: `no-tabs`

- **正确使用 ES6 中的字符串模板。**

  eslint: `no-template-curly-in-string`

```
const message = 'Hello ${name}';    // ✗ avoid
const message = `Hello ${name}`;    // ✓ ok
```

- **使用 `this` 前请确保 `super()` 已调用。**

  eslint: `no-this-before-super`

```
class Dog extends Animal {
  constructor() {
    this.legs = 4;     // ✗ avoid
    super();
  }
}
```

- 用 `throw` 抛错时，抛出 `Error` 对象而不是字符串。

  eslint: `no-throw-literal`

```
throw 'error';               // ✗ avoid
throw new Error('error');    // ✓ ok
```

- 行末不留空格。

  eslint: `no-trailing-spaces`

- 不要使用 `undefined` 来初始化变量。

  eslint: `no-undef-init`

```
let name = undefined;    // ✗ avoid

let name;
name = 'value';          // ✓ ok
```

- 循环语句中注意更新循环变量。

  eslint: `no-unmodified-loop-condition`

```
for (let i = 0; i < items.length; j++) {...}    // ✗ avoid
for (let i = 0; i < items.length; i++) {...}    // ✓ ok
```

- 如果有更好的实现，尽量不要使用三元表达式。

  eslint: `no-unneeded-ternary`

```
let score = val ? val : 0;     // ✗ avoid
let score = val || 0;          // ✓ ok
```

- `return`，`throw`，`continue` 和 `break` 后不要再跟代码。

  eslint: `no-unreachable`

```
function doSomething() {
  return true;
  console.log('never called');     // ✗ avoid
}
```

- `finally` 代码块中不要再改变程序执行流程。

  eslint: no-unsafe-finally

```
try {
  // ...
} catch (e) {
  // ...
} finally {
  return 42;     // ✗ avoid
}
```

- 关系运算符的左值不要做取反操作。

  eslint: no-unsafe-negation

```
if (!key in obj) {}       // ✗ avoid
```

- 避免不必要的 `.call()` 和 `.apply()`。

  eslint: no-useless-call

```
sum.call(null, 1, 2, 3);   // ✗ avoid
```

- 避免使用不必要的计算值作对象属性。

  eslint: no-useless-computed-key

```
const user = { ['name']: 'John Doe' };   // ✗ avoid
const user = { name: 'John Doe' };       // ✓ ok
```

- 禁止多余的构造器。

  eslint: no-useless-constructor

```
class Car {
  constructor() {      // ✗ avoid
  }
}
```

- 禁止不必要的转义。

eslint: `no-useless-escape`

```
let message = 'Hell\o';  // ✗ avoid
```

- **import, export 和解构操作中，禁止赋值到同名变量。**

  eslint: `no-useless-rename`

```
import { config as config } from './config';      // ✗ avoid
import { config } from './config';                // ✓ ok
```

- **属性前面不要加空格。**

  eslint: `no-whitespace-before-property`

```
user .name;       // ✗ avoid
user.name;        // ✓ ok
```

- **禁止使用 `with`。**

  eslint: `no-with`

```
with (val) {...}    // ✗ avoid
```

- **对象属性换行时注意统一代码风格。**

  eslint: `object-property-newline`

```
const user = {
  name: 'Jane Doe', age: 30,
  username: 'jdoe86'            // ✗ avoid
};

const user = { name: 'Jane Doe', age: 30, username: 'jdoe86' };    // ✓
ok

const user = {
  name: 'Jane Doe',
  age: 30,
  username: 'jdoe86'
};                                                                 // ✓
ok
```

- **代码块中避免多余留白。**

  eslint: `padded-blocks`

```
if (user) {
                                // ✗ avoid
  const name = getName();

}

if (user) {
  const name = getName();     // ✓ ok
}
```

- **展开运算符与它的表达式间不要留空白。**

  eslint: `rest-spread-spacing`

```
fn(... args);     // ✗ avoid
fn(...args);      // ✓ ok
```

- **遇到分号时空格要后留前不留。**

  eslint: `semi-spacing`

```
for (let i = 0 ;i < items.length ;i++) {...}     // ✗ avoid
for (let i = 0; i < items.length; i++) {...}     // ✓ ok
```

- **代码块首尾留空格。**

  eslint: `space-before-blocks`

```
if (admin){...}      // ✗ avoid
if (admin) {...}     // ✓ ok
```

- **圆括号间不留空格。**

  eslint: `space-in-parens`

```
getName( name );      // ✗ avoid
getName(name);        // ✓ ok
```

- **一元运算符后面跟一个空格。**

  eslint: `space-unary-ops`

```
typeof!admin;         // ✗ avoid
typeof !admin;         // ✓ ok
```

- **注释首尾留空格。**

eslint: `spaced-comment`

```
//comment           // ✗ avoid
// comment          // ✓ ok

/*comment*/         // ✗ avoid
/* comment */       // ✓ ok
```

- **模板字符串中变量前后不加空格。**

  eslint: `template-curly-spacing`

```
const message = `Hello, ${ name }`;     // ✗ avoid
const message = `Hello, ${name}`;       // ✓ ok
```

- **检查 `NaN` 的正确姿势是使用 `isNaN()`。**

  eslint: `use-isnan`

```
if (price === NaN) { }      // ✗ avoid
if (isNaN(price)) { }       // ✓ ok
```

- **用合法的字符串跟 `typeof` 进行比较操作。**

  eslint: `valid-typeof`

```
typeof name === 'undefimed';      // ✗ avoid
typeof name === 'undefined';      // ✓ ok
```

- **自调用匿名函数 (IIFEs) 使用括号包裹。**

  eslint: `wrap-iife`

```
const getName = function() { }();     // ✗ avoid

const getName = (function() { }());   // ✓ ok
const getName = (function() { })();   // ✓ ok
```

- **`yield *` 中的 `*` 前后都要有空格。**

  eslint: `yield-star-spacing`

```
yield* increment();     // ✗ avoid
yield * increment();    // ✓ ok
```

- **请书写优雅的条件语句（avoid Yoda conditions）。**

eslint: `yoda`

```
if (42 === age) { }    // ✗ avoid
if (age === 42) { }    // ✓ ok
```

- **要使用分号。**

  eslint: `semi`

```
window.alert('hi')   // ✗ avoid
window.alert('hi');  // ✓ ok
```