JavaScript 代码规范

本规范参考了js-standard-style

细则

- **使用两个空格**进行缩进。

```
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。**

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

- **关键字后面加空格。**

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

- **始终使用** `===` 替代 `==`。
  例外： `obj == null` 可以用来检查 `null || undefined`。 在数字和字符串数字比较时使用`==`

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

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

```
if (num != 3)   // ✓ ok
if (num !== 3)    // ✗ avoid
```

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

```
```

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

```
// X avoid
let x=2
let message = 'hello, '+name+'!'
```

- **逗号后面加空格。**

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

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

- **多行 if 语句的**的括号不能省。

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

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

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

- **不要丢掉**异常处理中`err`参数。

```
// ✓ ok
run(function (err) {
  if (err) throw err
```

```
  window.alert('done')
})
```

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

- **使用浏览器全局变量时加上** `window.` 前缀。
`document`、`console` 和 `navigator` 除外。

```
window.alert('hi')   // ✓ ok
```

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

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

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

// X avoid
let location = env.development ?
  'localhost' :
  'www.api.com'
```

- **每个 let, let, const 关键字**单独声明一个变量。

```
// ✓ ok
let silent = true
let verbose = true

// X avoid
let silent = true, verbose = true

// X avoid
let 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（`=`）。

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

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

- **单行代码块两边加空格。**

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

- **对于变量和函数名统一使用驼峰命名法。**

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

  let my_let = 'hello'           // X avoid
  let mylet = 'hello'            // ✓ ok
```

- **行末必须加逗号。**

```
  let obj = {
    message: 'hello',
    num: 3,   // ✓ ok
  }


  let myObj = {
    message: 'hello',
    num: 3    // X avoid
  }
```

- **始终将逗号置于行末。**

```
  let obj = {
    foo: 'foo'
    ,bar: 'bar'   // X avoid
  }

  let obj = {
```

```
    foo: 'foo',
    bar: 'bar'   // ✓ ok
  }
```

- **点号操作符须与属性需在同一行。**

```
  console.
    log('hello')  // X avoid

  console
    .log('hello') // ✓ ok
```

- **文件末尾留一空行。**
- **函数调用时标识符与括号间不留间隔。**

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

- **键值对当中冒号与值之间要留空白。**

```
let obj = { 'key' : 'value' }    // X avoid
let obj = { 'key' :'value' }     // X avoid
let obj = { 'key':'value' }      // X avoid
let obj = { 'key': 'value' }     // ✓ ok
```

- **构造函数必须要以大写字母开头。**

```
function animal () {}
let dog = new animal()    // X avoid

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

- **无参的构造函数调用时要带上括号。**

```
function Animal () {}
let dog = new Animal    // X avoid
let dog = new Animal()  // ✓ ok
```

- **对象中定义了存值器，一定要对应的定义取值器。**

```
let person = {
  set name (value) {    // X avoid
    this._name = value
  }
}

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

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

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

  arguments.callee(n - 1)   // X avoid
}

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

  foo(n - 1)
}
```

- **避免对类名重新赋值。**

```
class Dog {}
Dog = 'Fido'    // X avoid
```

- **避免使用常量作为条件表达式的条件（循环语句除外）。**

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

if (x === 0) {  // ✓ ok
  // ...
}

while (true) {  // ✓ ok
```

```
  // ...
}
```

- **正则中不要使用控制符。**

```
let pattern = /\x1f/    // ✗ avoid
let pattern = /\x20/    // ✓ ok
```

- **不要对变量使用 `delete` 操作。**

```
let name
delete name     // ✗ avoid
```

- **类中不要定义冗余的属性。**

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

- **对象字面量中不要定义重复的属性。**

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

- **`switch` 语句中不要定义重复的 `case` 分支。**

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

- **同一模块有多个导入时一次性写完。**

```
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。**

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

- **不要解构空值。**

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

- **`catch` 中不要对错误重新赋值。**

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

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

- **不要扩展原生对象。**

```
Object.prototype.age = 21      // X avoid
```

- **避免多余的函数上下文绑定。**

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

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

- **不要使用多余的括号包裹函数。**

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

- **不要省去小数点前面的0。**

```
const discount = .5      // X avoid
const discount = 0.5     // ✓ ok
```

- **避免对声明过的函数重新赋值。**

```
function myFunc () { }
myFunc = myOtherFunc    // X avoid
```

- **不要对全局只读对象重新赋值。**

```
window = {}     // X avoid
```

- **注意隐式的 `eval()`。**

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

- **嵌套的代码块中禁止再定义函数。**

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

- **不要向 `RegExp` 构造器传入非法的正则表达式。**

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

- **不要使用非法的空白符。**

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

- **禁止使用 `__iterator__`。**

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

- **外部变量不要与对象属性重名。**

```
let score = 100
function game () {
  score: while (true) {      // X avoid
    score -= 10
    if (score > 0) continue score
    break
  }
}
```

- **不要混合使用空格与制表符作为缩进。**
- **不要使用多行字符串。**

```
const message = 'Hello \
                world'     // X avoid
```

- **禁止使用 `new require`。**

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

- **禁止使用 `Symbol` 构造器。**

```
const foo = new Symbol('foo')   // X avoid
```

- **禁止使用原始包装器。**

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

- **不要使用八进制字面量。**

```
const octal = 042         // X avoid
const decimal = 34        // ✓ ok
const octalString = '042' // ✓ ok
```

- **字符串字面量中也不要使用八进制转义字符。**

```
const copyright = 'Copyright \251'  // X avoid
```

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

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

- **正则中避免使用多个空格。**

```
const regexp = /test   value/   // X 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。**

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

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

- **避免将变量赋值给自己。**

```
name = name   // X avoid
```

- **不要随意更改关键字的值。**

```
let undefined = 'value'     // X avoid
```

- **禁止使用稀疏数组（Sparse arrays）。**

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

- **不要使用制表符。**
- **用 `throw` 抛错时，抛出 `Error` 对象而不是字符串。**

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

- **行末不留空格。**
- **循环语句中注意更新循环变量。**

```
for (let i = 0; i < items.length; j++) {...}    // X 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。**

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

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

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

- **`finally` 代码块中不要再改变程序执行流程。**

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

- **关系运算符的左值不要做取反操作。**

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

- **避免不必要的 `.call()` 和 `.apply()`。**

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

- **避免使用不必要的计算值作对象属性。**

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

- **禁止不必要的转义。**

```
let message = 'Hell\o'  // X avoid
```

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

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

- **属性前面不要加空格。**

```
user .name      // X avoid
user.name       // ✓ ok
```

- **对象属性换行时注意统一代码风格。**

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

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

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

- **展开运算符与它的表达式间不要留空白。**

```
fn(... args)    // X avoid
fn(...args)     // ✓ ok
```

- **遇到分号时空格要后留前不留。**

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

- **代码块首尾留空格。**

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

- **圆括号间不留空格。**

```
getName( name )     // X avoid
getName(name)       // ✓ ok
```

- **一元运算符后面跟一个空格。**

```
typeof!admin        // X avoid
typeof !admin        // ✓ ok
```

- **注释首尾留空格。**

```
//comment           // X avoid
// comment          // ✓ ok

/*comment*/         // X avoid
/* comment */       // ✓ ok
```

- **模板字符串中变量前后不加空格。**

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

- **检查 `NaN` 的正确姿势是使用 `Number.isNaN()`。**

```
if (price === NaN) { }      // X avoid
if (Number.isNaN(price)) { }       // ✓ ok
```

- **自调用匿名函数 (IIFEs) 使用括号包裹。**

```
const getName = function () { }()     // X avoid

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

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

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

关于分号

- 不要使用分号

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

- 不要使用 (, [, or ` 等作为一行的开始。在没有分号的情况下代码压缩后会导致报错，而坚持这一规范则可避免出错。

```
// ✓ ok
;(function () {
  window.alert('ok')
}())

// X avoid
(function () {
  window.alert('ok')
}())
```

```
// ✓ ok
;[1, 2, 3].forEach(bar)

// X avoid
[1, 2, 3].forEach(bar)
```

```
// ✓ ok
;`hello`.indexOf('o')

// X avoid
`hello`.indexOf('o')
```

备注：上面的写法只能说聪明过头了。

相比更加可读易懂的代码，那些看似投巧的写法是不可取的。

譬如：

```
;[1, 2, 3].forEach(bar)
```

建议的写法是：

```
let nums = [1, 2, 3]
nums.forEach(bar)
```