JS规范

- 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 >")
```

- **不要定义未使用的变量。**

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

- **关键字后面加空格。**

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

- **函数声明时括号与函数名间加空格。**

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

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

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

```
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。**

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

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

- **逗号后面加空格。**

```
// ✓ 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。**

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

```
// ✗ avoid
if (condition)
{
```

```
  // ...
}
else
{
  // ...
}
```

- **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')
}
```

```
// ✗ 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')
})
```

```
// ✗ 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.

```
window.alert('hi')  // ✓ ok
```

- **不允许有连续多行空行。**

```
// ✓ 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。**

```
// ✓ 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。**

```
// ✓ 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（=）。**

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

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

- **单行代码块两边加空格。**

```
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。**

```
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。**

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

- **始终将逗号置于行末。**

```
  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。**

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

- **键值对当中冒号与值之间要留空白。**

```
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。**

```
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。**

```
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。**

```
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。**

```
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。**

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

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

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

- **避免修改使用 const 声明的变量。**

```
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）。**

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

if (x === 0) {  // ✓ ok
  // ...
}

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

- **正则中不要使用控制符。**

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

- **不要使用 debugger。**

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

- **不要对变量使用 delete 操作。**

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

function sum (a, b, c) {  // ✓ ok
```

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

- **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。**

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

- **不要解构空值。**

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

- **不要使用 eval()。**

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

- **catch 中不要对错误重新赋值。**

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

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

- **不要扩展原生对象。**

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

- **避免多余的函数上下文绑定。**

```
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。**

```
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。**

```
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。**

```
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。**

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

- **避免对声明过的函数重新赋值。**

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

- **不要对全局只读对象重新赋值。**

```
window = {}      // ✗ avoid
```

- **注意隐式的 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。**

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

- **不要向 RegExp 构造器传入非法的正则表达式。**

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

- **禁止使用 iterator。**

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

- **外部变量不要与对象属性重名。**

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

- **不要使用标签语句。**

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

- **不要书写不必要的嵌套代码块。**

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

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

- **不要混合使用空格与制表符作为缩进。**
- **除了缩进，不要使用多个空格。**

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

- **不要使用多行字符串。**

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

- **new 创建对象实例后需要赋值给变量。**

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

- **禁止使用 Function 构造器。**

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

- **禁止使用 Object 构造器。**

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

- **禁止使用 new require。**

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

- **禁止使用 Symbol 构造器。**

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

- **禁止使用原始包装器。**

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

- **不要将全局对象的属性作为函数调用。**

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

- **不要使用八进制字面量。**

```
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。**

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

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

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

- **使用 getPrototypeOf 来替代 proto。**

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

- **不要重复声明变量。**

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

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

- **正则中避免使用多个空格。**

```
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。**

```
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。**

```
name = name   // ✗ avoid
```

- **避免将变量与自己进行比较操作。**

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

- **避免使用逗号操作符。**

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

- **不要随意更改关键字的值。**

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

- **禁止使用稀疏数组（Sparse arrays）。**

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

- **不要使用制表符。**
- **正确使用 ES6 中的字符串模板。**

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

- **使用 this 前请确保 super() 已调用。**

```
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。**

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

- **行末不留空格。**
- **不要使用 undefined 来初始化变量。**

```
let name = undefined    // ✗ avoid

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

- **循环语句中注意更新循环变量。**

```
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。**

```
let score = val ? val : 0     // ✗ 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')     // ✗ avoid
}
```

- **finally 代码块中不要再改变程序执行流程。**

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

- **关系运算符的左值不要做取反操作。**

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

- **避免不必要的 .call() 和 .apply()。**

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

- **避免使用不必要的计算值作对象属性。**

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

- **禁止多余的构造器。**

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

- **禁止不必要的转义。**

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

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

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

- **属性前面不要加空格。**

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

- **禁止使用 with。**

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

- **对象属性换行时注意统一代码风格。**

```
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'
}
```

- **代码块中避免多余留白。**

```
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。**

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

- **代码块首尾留空格。**

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

- **圆括号间不留空格。**

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

- **一元运算符后面跟一个空格。**

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

- **注释首尾留空格。**

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

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

- **模板字符串中变量前后不加空格。**

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

- **检查 NaN 的正确姿势是使用 isNaN()。**

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

- **用合法的字符串跟 typeof 进行比较操作。**

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

- **自调用匿名函数 (IIFEs) 使用括号包裹。**

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

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

- **yield * 中的 * 前后都要有空格。**

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

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

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

关于分号

- **不要使用分号。**

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

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

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

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

```
// ✓ ok
;`hello`.indexOf('o')

// ✗ avoid
`hello`.indexOf('o')
```

> 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。

- **比如。**

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

建议的写法是

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