**烟草物流前端开发规范**

**引言**

**编写目的**

    一个软件的全部生命周期中，80%左右的时间是被用来对软件进行维护的。规范化的代码可以增强代码的可读性，从而有助于开发人员理解代码含义及迅速开发新代码。为了保证产品开发的一致性等，提高软件质量，改善软件的可读性和可维护性，软件编码必须进行规范。

    本文档是为规范编码而制定。

**约定说明**

1. 该规范以GitHub上较为通行的Standard JS(<https://github.com/standard>)为基础进行编制
2. 该规范语言标准基于ES6
3. 该规范会以代码检查工具，eslint配置方式加入到开发工具中，具体加入方式会另有文档予以告知阐明
4. 该规范中对于js中‘;’的处理，遵循了从最佳编码实践效果和代码可视性出发，采取了规避‘;’的做法。最后一章节，详细阐述了原因

**编程规约**

**使用两个空格进行缩进。**

eslint: [indent](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/quotes)

console.log('hello there')

$("<div class='box'>")

**不要定义未使用的变量。**

eslint: [no-unused-vars](http://eslint.org/docs/rules/no-unused-vars)

function myFunction () {

  var result = something()   // ✗ avoid

}

**关键字后面加空格。**

eslint: [keyword-spacing](http://eslint.org/docs/rules/keyword-spacing)

if (condition) { ... }   // ✓ ok

if(condition) { ... }    // ✗ avoid

**函数声明时括号与函数名间加空格。**

eslint: [space-before-function-paren](http://eslint.org/docs/rules/space-before-function-paren)

function name (arg) { ... }   // ✓ ok

function name(arg) { ... }    // ✗ avoid

run(function () { ... })      // ✓ ok

run(function() { ... })       // ✗ avoid

**孔丹：感觉这条并不是很合理，eslint工具是可配置的(**<http://blog.csdn.net/helpzp2008/article/details/51507428>**)，而且按照开发习惯以及github上一些其他的js规范，我觉得这里不需要空格(**<https://github.com/ecomfe/spec/blob/master/javascript-style-guide.md>**)**

孔令仪：

js代码规范目前每个公司都有每个公司的风格，部分规范无所谓对错之分，旨在提高代码可读性，可观性。

本篇幅是采纳目前使用比较广泛的两套标准（Standard JS，Airbnb，两套标准在vue脚手架中作为选择项供使用，基本代表了主流的编程规范）中的一套，

Standard JS 是JS开发者社区爱好者共同维护，Airbnb是Airbnb公司的内部规范，两者相比较，Standard JS 相对于 Airbnb，较为宽松。考虑到开发习惯的改变成本和开发效率，

故选择Standard JS

在函数名空格这部分，两个标准都建议留有空格，以下是Airbnb的规则

**始终使用 === 替代 ==。**

eslint: [eqeqeq](http://eslint.org/docs/rules/eqeqeq)

if (name === 'John')   // ✓ ok

if (name == 'John')    // ✗ avoid

if (name !== 'John')   // ✓ ok

if (name != 'John')    // ✗ avoid

例外： obj == null 可以用来检查 null || undefined。

贾瑞雪：疑问：===与==在判断上有区别？

孔令仪：解答：==   用于比较   判断 两者相等      ==在比较的时候可以转自动换数据类型 ===用于严格比较   判断两者严格相等   ===严格比较，不会进行自动转换，

要求进行比较的操作数必须类型一致，不一致时返回flase。

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

eslint: [space-infix-ops](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/brace-style)

// ✓ ok

if (condition) {

  // ...

} else {

  // ...

}

// ✗ avoid

if (condition)

{

  // ...

}

else

{

  // ...

}

**多行 if 语句的的括号不能省。**

eslint: [curly](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-undef)

window.alert('hi')   // ✓ ok

**不允许有连续多行空行。**

eslint: [no-multiple-empty-lines](http://eslint.org/docs/rules/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**

季康: js编码中不建议使用三目运算,JS中三目运算坑相对较多

孔令仪：目前对于三元运算符，尚没有足够的共识来废除他，此处的规范可以降低语义释义的错误

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

// ✓ 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](http://eslint.org/docs/rules/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（=）。**

        季康：直接规定赋值语句直接写在条件语句外面可读性是不是更高

        孔令仪：此处规范为引导在特定情况下的规范性，具体使用何种方式又各人而定，不做提倡

        也不做禁止，只作规范引导

        并不是所有的情况下都适合把赋值语句写在条件语句之外，涉及到具体的作用域，

        例如java中常用的for循环

        贾瑞雪：不理解此句

        孔令仪：用括号区分赋值和判断

        以下例子说明：

        while ((m = text.match(expr))){} // 加上括号，可以毫无歧义的表达括号内为赋值语句

        如果不加括号

        while(m = text.match(expr)) 会在阅读代码时产生歧义，究竟是赋值语句，还是编码时疏忽，少写了=号

eslint: [no-cond-assign](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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。**

注：除构造函数外

eslint: [camelcase](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/comma-dangle)

  var obj = {

    message: 'hello',   // ✗ avoid

  }

**始终将逗号置于行末。**

eslint: [comma-style](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/dot-location)

  console.

    log('hello')  // ✗ avoid

  console

    .log('hello') // ✓ ok

**文件末尾留一空行。**

elint: [eol-last](http://eslint.org/docs/rules/eol-last)

**函数调用时标识符与括号间不留间隔。**

eslint: [func-call-spacing](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-class-assign)

class Dog {}

Dog = 'Fido'    // ✗ avoid

**避免修改使用 const 声明的变量。**

eslint: [no-const-assign](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-constant-condition)

if (false) {    // ✗ avoid

  // ...

}

if (x === 0) {  // ✓ ok

  // ...

}

while (true) {  // ✓ ok

  // ...

}

**正则中不要使用控制符。**

注:控制字符是ASCII范围0-31(16进制为X00~X1F)中特殊的不可见字符。这些字符在JavaScript字符串中很少使用，所以包含这些字符的正则表达式很可能是一个错误。

eslint: [no-control-regex](http://eslint.org/docs/rules/no-control-regex)

var pattern = /\x1f/    // ✗ avoid

var pattern = /\x20/    // ✓ ok

**不要使用 debugger。**

eslint: [no-debugger](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-delete-var)

var name

delete name     // ✗ avoid

**不要定义冗余的函数参数。**

eslint: [no-dupe-args](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-empty-character-class)

const myRegex = /^abc[]/      // ✗ avoid

const myRegex = /^abc[a-z]/   // ✓ ok

**不要解构空值。**

eslint: [no-empty-pattern](http://eslint.org/docs/rules/no-empty-pattern)

const { a: {} } = foo         // ✗ avoid

const { a: { b } } = foo      // ✓ ok

**不要使用 eval()。**

eslint: [no-eval](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-extend-native)

Object.prototype.age = 21     // ✗ avoid

**避免多余的函数上下文绑定。**

eslint: [no-extra-bind](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-global-assign)

window = {}     // ✗ avoid

**注意隐式的 eval()。**

eslint: [no-implied-eval](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-irregular-whitespace)

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

**禁止使用 \_\_iterator\_\_。**

eslint: [no-iterator](http://eslint.org/docs/rules/no-iterator)

Foo.prototype.\_\_iterator\_\_ = function () {}   // ✗ avoid

**外部变量不要与对象属性重名。**

eslint: [no-label-var](http://eslint.org/docs/rules/no-label-var)

var score = 100

function game () {

  score: 50         // ✗ avoid

}

**不要使用标签语句。**

eslint: [no-labels](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-mixed-spaces-and-tabs)

**除了缩进，不要使用多个空格。**

eslint: [no-multi-spaces](http://eslint.org/docs/rules/no-multi-spaces)

const id =    1234    // ✗ avoid

const id = 1234       // ✓ ok

**不要使用多行字符串。**

eslint: [no-multi-str](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-new)

new Character()                     // ✗ avoid

const character = new Character()   // ✓ ok

**禁止使用 Function 构造器。**

eslint: [no-new-func](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-new-object)

let config = new Object()   // ✗ avoid

**禁止使用 new require。**

eslint: [no-new-require](http://eslint.org/docs/rules/no-new-require)

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

**禁止使用 Symbol 构造器。**

eslint: [no-new-symbol](http://eslint.org/docs/rules/no-new-symbol)

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

**禁止使用原始包装器。**

eslint: [no-new-wrappers](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-obj-calls)

const math = Math()   // ✗ avoid

**不要使用八进制字面量。**

eslint: [no-octal](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-proto)

const foo = obj.\_\_proto\_\_               // ✗ avoid

const foo = Object.getPrototypeOf(obj)  // ✓ ok

**不要重复声明变量。**

eslint: [no-redeclare](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-self-assign)

name = name   // ✗ avoid

**避免将变量与自己进行比较操作。**

esint: [no-self-compare](http://eslint.org/docs/rules/no-self-compare)

if (score === score) {}   // ✗ avoid

**避免使用逗号操作符。**

eslint: [no-sequences](http://eslint.org/docs/rules/no-sequences)

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

**不要随意更改关键字的值。**

eslint: [no-shadow-restricted-names](http://eslint.org/docs/rules/no-shadow-restricted-names)

let undefined = 'value'     // ✗ avoid

**禁止使用稀疏数组（Sparse arrays）。**

eslint: [no-sparse-arrays](http://eslint.org/docs/rules/no-sparse-arrays)

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

**不要使用制表符。**

eslint: [no-tabs](http://eslint.org/docs/rules/no-tabs)

制表符是指转义制表符“/t”

var a /t= 2;

/\*\*

\* /t/t it's a test function

\*/

function test(){}

var x = 1; // /t test // ✗ avoid

var a = 2;

/\*\*

\* it's a test function

\*/

function test(){}

var x = 1; // test //✓ ok

**正确使用 ES6 中的字符串模板。**

eslint: [no-template-curly-in-string](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-throw-literal)

throw 'error'               // ✗ avoid

throw new Error('error')    // ✓ ok

**行末不留空格。**

eslint: [no-trailing-spaces](http://eslint.org/docs/rules/no-trailing-spaces)

**不要使用 undefined 来初始化变量。**

eslint: [no-undef-init](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-unsafe-negation)

if (!key in obj) {}       // ✗ avoid

if (!(key in obj)) {}       // ✓ ok

**避免不必要的 .call() 和 .apply()。**

eslint: [no-useless-call](http://eslint.org/docs/rules/no-useless-call)

sum.call(null, 1, 2, 3)   // ✗ avoid

// These are same as `foo(1, 2, 3)

foo.call(undefined, 1, 2, 3)

foo.apply(undefined, [1, 2, 3])

foo.call(null, 1, 2, 3)

foo.apply(null, [1, 2, 3])

// These are same as `obj.foo(1, 2, 3);` // ✗ avoid

obj.foo.call(obj, 1, 2, 3)

obj.foo.apply(obj, [1, 2, 3])

//✓ ok

foo.call(obj, 1, 2, 3)

foo.apply(obj, [1, 2, 3])

obj.foo.call(null, 1, 2, 3)

obj.foo.apply(null, [1, 2, 3])贾瑞雪：违背了规则“不要使用分号”

**避免使用不必要的计算值作对象属性。**

eslint: [no-useless-computed-key](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-useless-constructor)

class Car {

  constructor () {      // ✗ avoid

  }

}

class A {

    constructor () {

        doSomething(); // ✓ ok

    }

}

**禁止不必要的转义。**

eslint: [no-useless-escape](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/no-whitespace-before-property)

user .name      // ✗ avoid

user.name       // ✓ ok

**禁止使用 with。**

eslint: [no-with](http://eslint.org/docs/rules/no-with)

with 语句是运行缓慢的代码块，尤其是在已设置了属性值时。大多数情况下，如果可能，最好避免使用它

with (val) {...}    // ✗ avoid

**对象属性换行时注意统一代码风格。**

eslint: [object-property-newline](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/space-before-blocks)

if (admin){...}     // ✗ avoid

if (admin) {...}    // ✓ ok

**圆括号间不留空格。**

eslint: [space-in-parens](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/space-unary-ops)

typeof!admin        // ✗ avoid

typeof !admin        // ✓ ok

**注释首尾留空格。**

eslint: [spaced-comment](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/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](http://eslint.org/docs/rules/yield-star-spacing)

yield\* increment()    // ✗ avoid

yield \* increment()   // ✓ ok

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

eslint: [yoda](http://eslint.org/docs/rules/yoda)

if (42 === age) { }    // ✗ avoid

if (age === 42) { }    // ✓ ok

**不要使用分号。** 贾瑞雪：是否有必要要求“不使用分号”？如果有必要的话，对一些易混淆的语法，要再深入普及。

                            孔令仪：唯一混淆的语法在于以[，(，+，\*，/，-，,，. 或者其他只会单独出现在两块内容间的二元操作符，

                            这类操作需要在开头处加；防止压缩后语法解释出错。

eslint: [semi](http://eslint.org/docs/rules/semi)

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。**

eslint: [no-unexpected-multiline](http://eslint.org/docs/rules/no-unexpected-multiline)

// ✓ 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')

备注：上面的写法只能是饮鸩止渴，虽可以避免错误但可读易懂性变差。

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：

var nums = [1, 2, 3]

nums.forEach(bar)

孙德峰：对于带小数的数字在js中加减乘除的算法，建议提供统一的工具方法，转化成整数后进行计算，避免小数计算中数量错误问题

孔令仪：原则上，前端只负责展示，所有计算都有后端进行处理，后端给什么，前端展示什么。该建议会公共讨论。

**拓展阅读**

季康:拓展阅读建议增加ES6常用语法特性,可以参考<https://segmentfault.com/a/1190000005742091>

孔令仪：es6语法特性为基础类语法内容，暂不放入开发规范中，分号的扩展阅读在于向大家分享解释为什么不用分号

**关于分号**

当前主流的代码压缩方案都是基于词法（AST-based）进行的，所以在处理无分号的代码时完全没有压力（何况 JavaScript 中分号本来就不是强制的）。

 自动化插入分号的做法(JavaScript 中，行尾的分号一种自动插入机制)是安全可依赖的，而且其产出的代码能够在所有浏览器里很好地运行。 Closure compiler, yuicompressor, packer 还有 jsmin 都能正确地对这样的代码进行压缩处理。并没有任何性能相关的问题。

先弄清楚 JS 是怎样断句的（还有就是哪些地方看起来断了其实并没有），明白了这个后就可以随心写出漂亮的代码了。

一般来说， \n 表示语句结束，除非：

1. 该语句有未闭合的括号， 数组字面量， 对象字面量 或者其他不能正常结束一条语句的情况（譬如，以 . 或 , 结尾）
2. 该语句是 -- 或者 ++ （它会将后面的内容进行自增或减）
3. 该语句是 for()，while()，do，if() 或者 else 并且没有 {
4. 下一行以 [，(，+，\*，/，-，,，. 或者其他只会单独出现在两块内容间的二元操作符。

第一条很容易理解。即使在 JSLint 中，也允许在JSON和带括号的结构中使用中出现’/n’，以及使用‘,’作为 var跨越多行结尾的语句的结尾。

第二条有点奇葩。 还想不出谁会（除了这里用作讨论外）写出

i

++

j

 这样的代码来，顺便说一下，这种写法最后解析的结果是 i; ++j，而不是 i++; j。

第三条也容易理解。

if (x)

y()

等价于

if (x)

 { y() }

解释器会向下寻找到代码块或一条语句为止。

“;”是一个有效的JavaScript语句，所以if(x);等价于if(x){}，“如果x，什么都不做”。这更常用于循环，循环检查也是更新函数。不常见，但不是闻所未闻的。

第四条就是常见的 “看，说过要加分号！” 的情形。但这些情况可以通过在语句前面加上分号来解决，如果你确定该语句跟前面的没关系的话。举个例子，假如你想这样：

foo();

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

那么完全可以这样来写：

foo()

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

后者的好处是分号比较瞩目，一旦习惯后便再也不会看到以 ( 和 [ 开头又不带分号的语句了。

为了增加可读性，可以写成

foo()

let a = [1,2,3]

a.forEach(bar)