JavaScript 编码风格规范

> 下面规则中未声明 value 的，均采用的 eslint 官网的默认值，详细参见官网Eslint

1. 生产环境中不能使用 console。
   级别：warn
   eslint: ***no-console***

2. 生产环境中不能使用 debugger。
   级别：warn
   eslint: ***no-debugger***

3. 避免存有声明后却未使用的变量。
   级别：warn
   eslint : ***no-unused-vars***
   options:

```
{
  'vars': 'all',// 允许声明未使用变量
  'args': 'none'// 对函数参数不检查
}
```

4. 在 try-catch-finally/if-else/function 等构造体中"{"禁止另起一行。
   级别：error
   eslint : ***brace-style***　　value:***stroustrup***
   options:

```
{
  "allowSingleLine": true  //允许单行内闭合一对{}
}
```

   ok √ :

```
function fn(){
  //code...
}
```

   avoid × :

```
function fn()
{
```

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

5. 指定必须使用单引号。
   级别：error
   eslint : ***quotes***　　value:***single***
   options:

```
{
 "avoidEscape": true,
 "allowTemplateLiterals": true
}
```

   ok √ :

```
let name = 'inspur'
```

   avoid × :

```
let name = "inspur"
```

6. 指定缩进长度为两个空格。
   级别：error
   eslint : ***indent***　　value:***2***
   options:

```
{
  //注意 以下数字不代表缩进长度，代表缩进级别
 "SwitchCase": 1, //指定 switch-case 语句的缩进级别
  "VariableDeclarator": 1, //指定 var 变量声明语句的缩进级别
  "outerIIFEBody": 1, //指定 IIFE（立即调用的函数表达式）的缩进级别
  "MemberExpression": 1, //指定多行属性链的缩进级别
  //通过传入一个对象来指定函数声明的缩进规则
  //指定函数声明中参数的缩进级别
  "FunctionDeclaration": { "parameters": 1, "body": 1 },
  "FunctionExpression": { "parameters": 1, "body": 1 },
  //通过传入一个对象来指定函数调用表达式的缩进规则
  "CallExpression": { "arguments": 1 },
  //指定数组中的元素的缩进级别
  "ArrayExpression": 1,
  //指定对象中的属性的缩进级别
  "ObjectExpression": 1,
  //指定 import 语句的缩进级别
  "ImportDeclaration": 1,
```

```
    //指定是否需要缩进嵌套在其他三元表达式中的三元表达式
    "flatTernaryExpressions": false,
    //指定注释是否需要需要与前一行或下一行的节点对齐
    "ignoreComments": false
 }
```

ok √ :

```
function fn(){
  console.log("缩进2个空格")
}
```

avoid × :

```
function fn(){
    console.log("缩进4个空格")
}
```

7. 禁止使用 var 声明变量，应使用 es6 中的 let\const 代替。
   级别：error
   eslint : ***no-var***

   ok √ :

   ```
   let name = 'inspur'
   //或者
   const name = 'inspur'
   ```

   avoid × :

   ```
   var name = 'inspur'
   ```

8. 对 null 进行比较运算时，必须用全等（===）或非全等（!==）。
   级别：error
   eslint : ***no-eq-null***

   ok √ :

   ```
   let a = null
   console.log(a === null)
   ```

avoid × :

```
let a = null
console.log(a == null)
```

9. 指定函数的"("前加空格，包括箭头函数。
   级别：error
   eslint : ***space-before-function-paren***　　value:***always***
   options:

```
 {
   anonymous: 'always', //匿名函数"("前不能含有空格
   named: 'always', //具名函数"("前不能含有空格
   asyncArrow: 'always' //箭头函数"("前必须含有空格
 }
```

   ok √ :

```
function foo () {
  //code
}

let bar = function () {
  //code
}

let fn = () => {
  //code
}
```

   avoid × :

```
function foo() {
  //code
}

let bar = function(){
  //code
}

let fn=()=>{
  //code
}
```

10. 关键字前后有其他内容时必须加空格。
    级别：error
    eslint : ***keyword-spacing***
    options:

```
 {
   "before": true,
   "after": true
  }
```

    ok √ :

```
if (true) {
  //code
}
```

    avoid × :

```
if(true) {
  //code
}
```

11. 始终使用 === 替代 ==。
    级别：error
    eslint : ***eqeqeq***　　value:***always***

    ok √ :

```
if (a === b) {
  //code
}
```

    avoid × :

```
if (a == b) {
  //code
}
```

12. 字符串拼接操作符 (Infix operators) 之间要留空格。
    级别：error
    eslint : ***space-infix-ops***

ok √ :

```
let name = 'inspur'
let message = 'hello, ' + name + '!'
```

avoid × :

```
let name = 'inspur'
let message = 'hello, '+name+'!'
```

13. 逗号后面要加空格。
级别：error
eslint : ***comma-spacing***
options:

```
 {
   "before": false, //逗号前不加空格
   "after": true //逗号后加空格
  }
```

ok √ :

```
function greet (name, options) {
  //code
}
```

avoid × :

```
function greet (name,options) {
  //code
}
```

14. 多行 if 语句的的括号不能省。
级别：error
eslint : ***curly***　　value:***multi-line***

ok √ :

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

```
  console.log('done')
}
```

avoid × :

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

15. 不要丢掉异常处理中 err 参数。
    级别：error
    eslint : ***handle-callback-err*** value:***^(err|error)$***

    ok √ :

```
run(function (err) {
  if (err) throw err
  window.alert('done')
})
```

    avoid × :

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

16. 使用浏览器全局变量时加上 window. 前缀。
    级别：error
    eslint : ***no-undef***

    ok √ :

```
window.alert('hi')
```

    avoid × :

```
alert('hi')
```

17. 不允许有连续多行空行。
    级别：error
    eslint : ***no-multiple-empty-lines***
    options:

```
{
  "max": 1, //上一句代码与下一句代码之间的最大空行数
}
```

ok √ :

```
let value = 'hello world'
console.log(value)
```

avoid × :

```
let value = 'hello world'
//此处空出n多行

console.log(value)
```

18. 只允许在操作符之后换行。
    级别：error
    eslint : ***operator-linebreak*** value:***after***
    options:

```
{
  "overrides": {
    "?": "before", //只允许在"?"前换行
    ":": "before" //只允许在":"前换行
   }
}
```

ok √ :

```
answer = everything
? 42
: foo
//或者
answer = everything ? 42 : foo
```

avoid × :

```
answer = everything ?
42 :
foo
```

19. 每个 let / const 关键字单独声明一个变量。（本项目中已经禁用了 var）
    级别：error
    eslint : ***one-var***
    options:

```
{
  "initialized": "never" //不允许共用一个var表达式声明多个变量
}
```

    ok √ :

```
let silent = true
let verbose = true

const userInfo = {}
const item = {}
```

    avoid × :

```
let silent = true,verbose = true
```

20. 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（=）。
    级别：error
    eslint : ***no-cond-assign***

    ok √ :

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

    avoid × :

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

21. 单行代码块两边加空格。
    级别：error

eslint : ***block-spacing*** value:***always***

ok √ :

```
function foo() { return true }
```

avoid × :

```
function foo() {return true}
```

22. 对于变量和函数名统一使用驼峰命名法。
级别：error
eslint : ***camelcase***
options:

```
 {
   "properties": "never" //不检查对象内属性的名字
 }
```

ok √ :

```
function myFunction () {
  //code
}
let myVar = 'hello'
```

avoid × :

```
function my_function () {
  //code
}
let my_var = 'hello'
```

23. 不允许有多余的行末逗号。
级别：error
eslint : ***comma-dangle***
options:

```
 {
   "arrays": "never", //数组不允许末尾逗号
   "objects": "never", //对象不允许末尾逗号
```

```
    "imports": "never", //导入不允许末尾逗号
    "exports": "never", //导出不允许末尾逗号
    "functions": "never" //函数不允许末尾逗号
  }
```

ok √ :

```
let obj = {
  message: 'hello'
}
```

avoid × :

```
let obj = {
  message: 'hello',
}
```

24. 始终将逗号置于行末。
    级别：error
    eslint : ***comma-style***　　value:***last***

    ok √ :

```
let obj = {
  foo: 'foo',
  bar: 'bar'
}
```

    avoid × :

```
let obj = {
  foo: 'foo'
  ,bar: 'bar'
}
```

25. 点号操作符需与属性在同一行。
    级别：error
    eslint : ***dot-location***　　value:***property***

    ok √ :

```

```

```
console
       .log('hello')
```

avoid × :

```
console.
        log('hello')
```

26. 文件末尾不必留一空行。
    级别：never
    eslint : ***eol-last***

27. 函数调用时标识符与括号间不留间隔。
    级别：error
    eslint : ***func-call-spacing*** value:***never***

    ok √ :

```
console.log('hello')
```

    avoid × :

```
console.log ('hello')
```

28. 键值对当中冒号与值之间要留空白。
    级别：error
    eslint : ***key-spacing***
    options:

```
 {
   "beforeColon": false,
   "afterColon": true
 }
```

    ok √ :

```
let obj = { 'key': 'value' }
```

    avoid × :

```
let obj = { 'key' : 'value' }
let obj = { 'key' :'value' }
let obj = { 'key':'value' }
```

29. 构造函数要以大写字母开头。
    级别：error
    eslint: ***new-cap***
    options:

```
{
  "newIsCap": true, //new 调用的函数，必须以大写开头
  "capIsNew": false //大写开头的函数，不必非要配合new调用
}
```

    ok √ :

```
function Animal () {}
let dog = new Animal()
```

    avoid × :

```
function animal () {}
let dog = new animal()
```

30. 无参的构造函数调用时要带上括号。
    级别：error
    eslint : ***new-parens***

    ok √ :

```
function Animal () {}
let dog = new Animal()
```

    avoid × :

```
function Animal () {}
let dog = new Animal
```

31. 对象中定义了存值器，一定要对应的定义取值器。
    级别：error

eslint : ***accessor-pairs***

ok √ :

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

avoid × :

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

32. 子类的构造器中一定要调用 super。
    级别：error
    eslint : ***constructor-super***

    ok √ :

    ```
    class Dog extends Mammal {
      constructor () {
        super()
        //code
      }
    }
    ```

    avoid × :

    ```
    class Dog extends Mammal {
      constructor () {
        //code
      }
    }
    ```

33. 使用数组字面量而不是构造器。
    级别：error

eslint : ***no-array-constructor***

ok √ :

```
let nums = [1, 2, 3]
```

avoid × :

```
let nums = new Array(1, 2, 3)
```

34. 避免使用 arguments.callee 和 arguments.caller。
    级别：error
    eslint : ***no-caller***

    ok √ :

    ```
    function foo (n) {
      if (n <= 0) return
      foo(n - 1)
    }
    ```

    avoid × :

    ```
    function foo (n) {
      if (n <= 0) return
      arguments.callee(n - 1)
    }
    ```

35. 避免对类名重新赋值。
    级别：error
    eslint : ***no-class-assign***

    avoid × :

    ```
    class Dog {}
    Dog = 'Fido'
    ```

36. 避免修改使用 const 声明的变量。
    级别：error
    eslint : ***no-const-assign***

    ok √ :

```
let num = 1
num++
```

```
const userInfo = {}
userInfo.name = 'Jack'
```

avoid × :

```
const score = 100
score = 125
```

```
for (const a in [1, 2, 3]) { // `a` is re-defined (not modified) on
each loop step.
  console.log(a);
}
```

37. 避免使用常量作为条件表达式的条件（循环语句除外）。
级别：error
eslint : ***no-constant-condition***
options:

```
 {
   "checkLoops": false //忽略检查循环语句
 }
```

ok √ :

```
if (x === 0) {
  // code
}
//或者
while (true) {
  // code
}
```

avoid × :

```
if (false) {
  // code
```

```
}
```

38. 正则中不要使用控制符。
    级别：error
    eslint : ***no-control-regex***

    ok √ :

    ```
    let pattern = /\x20/
    ```

    avoid × :

    ```
    let pattern = /\x1f/
    ```

39. 不要对变量使用 delete 操作。
    级别：error
    eslint : ***no-delete-var***

    avoid × :

    ```
    var name
    delete name
    ```

40. 不要定义重复的函数参数。
    级别：error
    eslint : ***no-dupe-args***

    ok √ :

    ```
    function sum (a, b, c) {
      // code
    }
    ```

    avoid × :

    ```
    function sum (a, b, a) {
      // code
    }
    ```

41. 类中不要定义重复的属性。
    级别：error

eslint : ***no-dupe-class-members***

ok √ :

```
class Dog {
  bark () {}
}
```

avoid × :

```
class Dog {
  bark () {}
  bark () {}
}
```

42. 对象字面量中不要定义重复的属性。
级别：error
eslint : ***no-dupe-keys***

ok √ :

```
let user = {
  name: 'Jane Doe'
}
```

avoid × :

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

43. switch 语句中不要定义重复的 case 分支。
级别：error
eslint : ***no-duplicate-case***

ok √ :

```
switch (id) {
  case 1:
    // code
}
```

avoid × :

```
switch (id) {
  case 1:
    // code
  case 1:
}
```

44. 同一模块有多个导入时一次性写完。
级别：error
eslint : ***no-duplicate-imports||import/no-duplicates***

ok √ :

```
import { myFunc1, myFunc2 } from 'module'
```

avoid × :

```
import { myFunc1 } from 'module'
import { myFunc2 } from 'module'
```

45. 正则中不要使用空字符。
级别：error
eslint : ***no-empty-character-class***

ok √ :

```
const myRegex = /^abc[a-z]/
```

avoid × :

```
const myRegex = /^abc[]/
```

46. 不要解构空值。
级别：error
eslint : ***no-empty-pattern***

ok √ :

```
const { a: { b } } = foo
```

avoid × :

```
const { a: {} } = foo
```

47. 不要使用 eval()。
级别：error
eslint : ***no-eval***

ok √ :

```
let result = user[propName]
```

avoid × :

```
eval( "let result = user." + propName )
```

48. catch 中不要对错误重新赋值。
级别：error
eslint : ***no-ex-assign***

ok √ :

```
try {
  // code
} catch (e) {
  const newVal = 'new value'
}
```

avoid × :

```
try {
  // code
} catch (e) {
  e = 'new value'
}
```

49. 不要扩展原生对象。
级别：error
eslint : ***no-extend-native***
该规则接受一个exceptions选项，该选项可用于指定允许扩展名的内建列表。

avoid × :

```
Object.prototype.age = 21
Object.defineProperty(Array.prototype, "times", { value: 999 });
```

50. 避免多余的函数上下文绑定。
    级别：error
    eslint : ***no-extra-bind***

    ok √ :

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

    avoid × :

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

51. 避免不必要的布尔转换。
    级别：error
    eslint : ***no-extra-boolean-cast***

    ok √ :

```
const result = true
if (result) {
  // code
}
```

    avoid × :

```
const result = true
if (!!result) {
  // code
}
```

52. 不要使用多余的括号包裹函数。
    级别：error
    eslint : ***no-extra-parens***　　value:***functions***

    ok √ :

```
const myFunc = function () { }
```

avoid × :

```
const myFunc = (function () { })
```

53. switch 一定要使用 break 来将条件分支正常中断。
级别：error
eslint : ***no-fallthrough***

ok √ :

```
switch (filter) {
  case 1:
    doSomething()
    break
  case 2:
    doSomethingElse()
}
```

avoid × :

```
switch (filter) {
  case 1:
    doSomething()
  case 2:
    doSomethingElse()
}
```

54. 不要省去小数点前面的 0。
级别：error
eslint : ***no-floating-decimal***

ok √ :

```
const discount = 0.5
```

avoid × :

```
const discount = .5
```

55. 避免对声明过的函数重新赋值。
    级别：error
    eslint : ***no-func-assign***

    avoid × :

```
function myFunc () { }
myFunc = myOtherFunc
```

56. 不要对全局只读对象重新赋值。
    级别：error
    eslint : ***no-global-assign***

    avoid × :

```
window = {}
```

57. 注意隐式的 eval()。
    级别：error
    eslint : ***no-implied-eval***

    ok √ :

```
setTimeout(function () { alert('Hello world') })
```

    avoid × :

```
setTimeout("alert('Hello world')")
```

58. 嵌套的代码块中禁止再定义函数。
    级别：error
    eslint : ***no-inner-declarations*** value:***functions***

    avoid × :

```
if (authenticated) {
  function setAuthUser () {}
}
```

59. 不要向 RegExp 构造器传入非法的正则表达式。
    级别：error
    eslint : ***no-invalid-regexp***

ok √ :

```
RegExp('[a-z]')
```

avoid × :

```
RegExp('[a-z')
```

60. 不要使用非法的空白符。
级别：error
eslint : ***no-irregular-whitespace***

avoid × :

```
function thing() /*<NBSP>*/{
  return 'test';
}
```

61. 禁止使用 no-iterator。
级别：error
eslint : ***no-iterator***

avoid × :

```
Foo.prototype.__iterator__ = function() {
    return new FooIterator(this);
};

foo.__iterator__ = function () {};

foo["__iterator__"] = function () {};
```

62. 外部变量不要与对象属性重名。
级别：error
eslint : ***no-label-var***

avoid × :

```
let score = 100
function game () {
  score: while (true) {
    score -= 10
    if (score > 0) continue score
```

```
    break
  }
}
```

63. 不要使用标签语句。
    级别：error
    eslint : ***no-labels***
    options:

```
{
  "allowLoop": false,
  "allowSwitch": false
}
```

    avoid × :

```
label:
      while (true) {
        break label
      }
```

64. 不要书写不必要的嵌套代码块。
    级别：error
    eslint : ***no-lone-blocks***

    ok √ :

```
function myFunc () {
    myOtherFunc()
}
```

    avoid × :

```
function myFunc () {
  {
    myOtherFunc()
  }
}
```

65. 不要混合使用空格与制表符作为缩进。
    级别：error
    eslint : ***no-mixed-spaces-and-tabs***

66. 除了缩进，不要使用多个空格。
    级别：error
    eslint : ***no-multi-spaces***

    ok √ :

    ```
    const id = 1234
    ```

    avoid × :

    ```
    const id =    1234
    ```

67. 不要使用多行字符串。
    级别：error
    eslint : ***no-multi-str***

    avoid × :

    ```
    const message = 'Hello \
                            world'
    ```

68. new 创建对象实例后需要赋值给变量。
    级别：error
    eslint : ***no-new***

    ok √ :

    ```
    const character = new Character()
    ```

    avoid × :

    ```
    new Character()
    ```

69. 禁止使用 Function 构造器。
    级别：error
    eslint : ***no-new-func***

    avoid × :

    ```
    let sum = new Function('a', 'b', 'return a + b')
    ```

70. 禁止使用 Object 构造器。
    级别：error
    eslint : ***no-new-object***

    avoid × :

```
let config = new Object()
```

71. 禁止使用 new require。
    级别：error
    eslint : ***no-new-require***

    avoid × :

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

72. 禁止使用 Symbol 构造器。
    级别：error
    eslint : ***no-new-symbol***

    ok √ :

```
var foo = Symbol('foo');
```

    avoid × :

```
const foo = new Symbol('foo')
```

73. 禁止使用原始包装器。
    级别：error
    eslint : ***no-new-wrappers***

    ok √ :

```
const message = 'hello'
```

    avoid × :

```
const message = new String('hello')
```

74. 不要将全局对象的属性作为函数调用。
    级别：error
    eslint : ***no-obj-calls***

    avoid × :

```
const math = Math()
```

75. 不要使用八进制字面量。
    级别：error
    eslint : ***no-octal***

    ok √ :

```
const decimal = 34
const octalString = '042'
```

    avoid × :

```
const octal = 042
```

76. 字符串字面量中也不要使用八进制转义字符。
    级别：error
    eslint : ***no-octal-escape***

    avoid × :

```
const copyright = 'Copyright \251'
```

77. 使用 dirname 和 filename 时避免使用字符串拼接。
    级别：error
    eslint : ***no-path-concat***

    ok √ :

```
const pathToFile = path.join(__dirname, 'app.js')
```

    avoid × :

```
const pathToFile = __dirname + '/app.js'
```

78. 使用 getPrototypeOf 来替代 proto。
    级别：error
    eslint : ***no-proto***

    ok √ :

```
const foo = Object.getPrototypeOf(obj)
```

    avoid × :

```
const foo = obj.__proto__
```

79. 不要重复声明变量。
    级别：error
    eslint : ***no-redeclare***

    ok √ :

```
let name = 'John'
name = 'Jane'
```

    avoid × :

```
let name = 'John'
let name = 'Jane'
```

80. 正则中避免使用多个空格。
    级别：error
    eslint : ***no-regex-spaces***

    ok √ :

```
const regexp = /test {3}value/
const regexp = /test value/
```

    avoid × :

```
const regexp = /test   value/
```

81. return 语句中的赋值必需有括号包裹。
    级别：error
    eslint : ***no-return-assign*** value:***except-parens***

    ok √ :

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

    avoid × :

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

82. 避免将变量赋值给自己。
    级别：error
    eslint : ***no-self-assign***

    avoid × :

    ```
    name = name
    ```

83. 避免将变量与自己进行比较操作。
    级别：error
    eslint : ***no-self-compare***

    avoid × :

    ```
    if (score === score) {}
    ```

84. 避免使用逗号操作符。
    级别：error
    eslint : ***no-sequences***

    avoid × :

    ```
    if (doSomething(), !!test) {}
    ```

85. 不要随意更改关键字的值。
    级别：error

eslint : ***no-shadow-restricted-names***

avoid × :

```
let undefined = 'value'
```

86. 禁止使用稀疏数组（Sparse arrays）。
级别：error
eslint : ***no-sparse-arrays***

avoid × :

```
let fruits = ['apple',, 'orange']
```

87. 不要使用制表符。
级别：error
eslint : ***no-tabs***

ok √ :

```
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
```

88. 正确使用 ES6 中的字符串模板。
级别：error
eslint : ***no-template-curly-in-string***

ok √ :

```
const message = `Hello ${name}`
```

avoid × :

```
const message = 'Hello ${name}'
```

89. 使用 this 前请确保 super() 已调用。
级别：error
eslint : ***no-this-before-super***

ok √ :

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

avoid × :

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

90. 用 throw 抛错时，抛出 Error 对象而不是字符串。
级别：error
eslint : ***no-throw-literal***

ok √ :

```
throw new Error('error')
```

avoid × :

```
throw 'error'
```

91. 行末不留空格。
级别：error
eslint : ***no-trailing-spaces***

92. 不要使用 undefined 来初始化变量。
    级别：error
    eslint : ***no-undef-init***

    ok √ :

```
let name
name = 'value'
```

    avoid × :

```
let name = undefined
```

93. 循环语句中注意避免更新循环变量。
    级别：error
    eslint : ***no-unmodified-loop-condition***

    ok √ :

```
for (let i = 0; i < items.length; i++) {...}
```

    avoid × :

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

94. 如果有更好的实现，尽量不要使用三元表达式。
    级别：error
    eslint : ***no-unneeded-ternary***
    options:

```
{
  "defaultAssignment": false //禁止将条件表达式作为默认分配模式
}
```

    ok √ :

```
let score = val || 0
```

    avoid × :

```
let score = val ? val : 0
```

95. return，throw，continue 和 break 后不要再跟代码。
    级别：error
    eslint : ***no-unreachable***

    avoid × :

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

96. finally 代码块中不要再改变程序执行流程。
    级别：error
    eslint : ***no-unsafe-finally***

    avoid × :

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

97. 关系运算符的左值不要做取反操作。
    级别：error
    eslint : ***no-unsafe-negation|no-negated-in-lhs***

    avoid × :

```
if (!key in obj) {}
```

98. 避免不必要的 .call() 和 .apply()。
    级别：error
    eslint : ***no-useless-call***

    ok √ :

```
// The `this` binding is different.
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]);
obj.foo.call(otherObj, 1, 2, 3);
obj.foo.apply(otherObj, [1, 2, 3]);

// The argument list is variadic.
foo.apply(undefined, args);
foo.apply(null, args);
obj.foo.apply(obj, args);
```

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);`
obj.foo.call(obj, 1, 2, 3);
obj.foo.apply(obj, [1, 2, 3]);
```

99. 避免使用不必要的计算值作对象属性。
    级别：error
    eslint : ***no-useless-computed-key***

    ok √ :

    ```
    const user = { name: 'John Doe' }
    ```

    avoid × :

    ```
    const user = { ['name']: 'John Doe' }
    ```

100. 禁止多余的构造器。
    级别：error
    eslint : ***no-useless-constructor***

    avoid × :

    ```
    class Car {
      constructor () {
      }
    }
    ```

```

```

101. 禁止不必要的转义。
级别：error
eslint : ***no-useless-escape***

avoid × :

```
let message = 'Hell\o'
```

102. import, export 和解构操作中，禁止赋值到同名变量。
级别：error
eslint : ***no-useless-rename***

ok √ :

```
import { config } from './config'
```

avoid × :

```
import { config as config } from './config'
```

103. 属性前面不要加空格。
级别：error
eslint : ***no-whitespace-before-property***

ok √ :

```
user.name
```

avoid × :

```
user .name
user. name
```

104. 禁止使用 with。
级别：error
eslint : ***no-with***

avoid × :

```

```

```
with (val) {...}
```

105. 对象属性换行时注意统一代码风格。

级别：error

eslint : ***object-property-newline***

options:

```
{
  "allowMultiplePropertiesPerLine": true //允许对象在同一行
}
```

ok √ :

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

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

avoid × :

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

106. 代码块中避免多余留白。

级别：error

eslint : ***padded-blocks***

options:

```
{
  "blocks": "never",
  "switches": "never",
  "classes": "never"
}
```

ok √ :

```
if (user) {
  const name = getName()
}
```

avoid × :

```
if (user) {

  const name = getName()
}
```

107. 展开运算符与它的表达式间不要留空白。
级别：error
eslint : ***rest-spread-spacing*** value:***never***

ok √ :

```
fn(...args)
```

avoid × :

```
fn(... args)
```

108. 遇到分号时空格要后留前不留。
级别：error
eslint : ***semi-spacing***
options:

```
{
  "before": false,
  "after": true
}
```

ok √ :

```
for (let i = 0; i < items.length; i++) {...}
```

avoid × :

```
for (let i = 0 ;i < items.length ;i++) {...}
```

109. 代码块首尾留空格。
级别：error
eslint : ***space-before-blocks***　　value:***always***

ok √ :

```
if (admin) {...}
```

avoid × :

```
if (admin){...}
```

110. 圆括号间不留空格。
级别：error
eslint : ***space-in-parens***　　value:***never***

ok √ :

```
getName(name)
```

avoid × :

```
getName( name )
```

111. 一元运算符后面跟一个空格。
级别：error
eslint : ***space-unary-ops***
options:

```
{
  "words": true,
  "nonwords": false
}
```

ok √ :

```
typeof !admin
```

avoid × :

```
typeof!admin
```

112. 注释首尾留空格。
级别：error
eslint : ***spaced-comment*** value:***always***

ok √ :

```
// comment
/* comment */
```

avoid × :

```
//comment
/*comment*/
```

113. 模板字符串中变量前后不加空格。
级别：error
eslint : ***template-curly-spacing*** value:***never***

ok √ :

```
const message = `Hello, ${name}`
```

avoid × :

```
const message = `Hello, ${ name }`
```

114. 检查 NaN 的正确姿势是使用 isNaN()。
级别：error
eslint : ***use-isnan***

ok √ :

```
if (isNaN(price)) { }
```

avoid × :

```
if (price === NaN) { }
```

115. 用合法的字符串跟 typeof 进行比较操作。
级别：error
eslint : ***valid-typeof***
options:

```
{
  "requireStringLiterals": true
}
```

ok √ :

```
typeof name === 'undefined'
```

avoid × :

```
typeof name === 'undefimed'
```

116. 自调用匿名函数 (IIFEs) 使用括号包裹。
级别：error
eslint : ***wrap-iife***　　value:***any***
options:

```
{
  "functionPrototypeMethods": true
}
```

ok √ :

```
const getName = (function () { }())
const getName = (function () { })()
```

avoid × :

```
const getName = function () { }()
```

117. yield _ 中的 _ 前后都要有空格。
级别：error
eslint : ***yield-star-spacing*** value:***both***

ok √ :

```
yield * increment()
```

avoid × :

```
yield* increment()
```

118. 请书写优雅的条件语句（avoid Yoda conditions）。
级别：error
eslint : ***yoda*** value:***never***

ok √ :

```
if (age === 42) { }
```

avoid × :

```
if (42 === age) { }
```

119. 禁止行尾分号
级别：error
eslint : ***semi*** value:***never***

ok √ :

```
window.alert('hi')
```

avoid × :

```
window.alert('hi');
```

120. 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。

级别：error
eslint : ***no-unexpected-multiline***

ok √ :

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

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

;`hello`.indexOf('o')
```

avoid × :

```
(function () {
  window.alert('ok')
}())

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

`hello`.indexOf('o')
```

121. 箭头函数前后留空格。
级别：error
eslint : ***arrow-spacing***
options:

```
{
  "before": true,
  "after": true
}
```

ok √ :

```
() => {};
(a) => {};
() => {'\n'};
```

avoid × :

```
() =>{};
(a)=>{};
```

```
()=> {'\n'};
```

122. 状态机*前后留空格。
级别：error
eslint : ***generator-star-spacing***
options:

```
{
  "before": true,
  "after": true
}
```

ok √ :

```
function * generator () {}
```

avoid × :

```
function *generator() {}
function* generator() {}
```

123. 禁止与-0 进行比较。
级别：error
eslint : ***no-compare-neg-zero***

ok √ :

```
if (x === -0) {
  // doSomething()...
}
```

avoid × :

```
if (x === 0) {
  // doSomething()...
}
```

124. 在复杂的逻辑判断中对应组别的判断逻辑用括号括起来。
级别：error
eslint : ***no-mixed-operators***
options:

```
{
  "groups": [
          ["==", "!=", "===", "!==", ">", ">=", "<", "<="],
          ["&&", "||"],
          ["in", "instanceof"]
      ],
  "allowSamePrecedence": true
}
```

ok √ :

```
let foo = a || b || c;
let foo = a && b && c;
let foo = (a && b < 0) || c > 0 || d + 1 === 0;
let foo = a && (b < 0 || c > 0 || d + 1 === 0);
let foo = a + (b * c);
let foo = (a + b) * c;
```

avoid × :

```
let foo = a && b < 0 || c > 0 || d + 1 === 0;
let foo = a + b * c;
```

125. 不能 return await 函数。

级别：error

eslint : ***no-return-await***

ok √ :

```
async function foo() {
  return bar();
}

async function foo() {
  await bar();
  return;
}

async function foo() {
  const x = await bar();
  return x;
}
```

avoid × :

```
async function foo() {
  return await bar();
}
```

126. 避免对程序没有影响的未使用的表达式。

级别：error

eslint : ***no-unused-expressions***

options:

```
{
  "allowShortCircuit": true, //允许短路判断逻辑
  "allowTernary": true,  //允许三元表达式
  "allowTaggedTemplates": true  //允许使用模板标签
}
```

avoid × :

```
0

if(0) 0

{0}

f(0), {}

a && b()

a, b()

c = a, b;

a() && function namedFunctionInExpressionContext () {f();}

(function anIncompleteIIFE () {});
```

127. 不要使用未声明的变量。

级别：error

eslint : ***no-use-before-define***

options:

```
{
  "functions": false, //检查函数
  "classes": false,  //检查类
  "variables": false  //检查变量声明
}
```

avoid × :

```
alert(a);
var a = 10;

f();
function f() {}

function g() {
    return b;
}
var b = 1;

{
    alert(c);
    let c = 1;
}
```

128. 避免没有返回任何内容的 return。
级别：error
eslint : ***no-useless-return***

avoid × :

```
function test() {
  return
}
```

129. 大括号内需要有空格（{}除外）。
级别：error
eslint : ***object-curly-spacing*** value:***always***

ok √ :

```
var obj = {}
var obj = { 'foo': 'bar' }
var obj = { 'foo': { 'bar': 'baz' }, 'qux': 'quxx' }
var obj = {
  'foo': 'bar'
}
var { x } = y;
import { foo } from 'bar'
```

avoid × :

```
var obj = {}
var obj = {'foo': 'bar'}
var obj = {'foo': {'bar': 'baz'}, 'qux': 'quxx'}
var obj = {
  'foo': 'bar'
}
var obj = {'foo': 'bar'
}
var obj = {
  'foo':'bar'
}
var {x} = y;
import {foo} from 'bar'
```

130. promise 只能被 errors 对象 reject。
级别：error
eslint : ***prefer-promise-reject-errors***
该规则采用一个可选的对象参数：allowEmptyReject: true

ok √ :

```
Promise.reject()

Promise.reject(new Error(123))

let p = new Promise((resolve, reject) => {
  reject(new Error('错误'))
})
```

avoid × :

```
Promise.reject(123)

let p = new Promise((resolve, reject) => {
  reject('错误')
})
```

131. 创建 symbol 时，不能为空，括号内必须有描述。
级别：error
eslint : ***symbol-description***

ok √ :

```
let foo = Symbol("some description");
```

```
let someString = "some description";
let bar = Symbol(someString);
```

avoid × :

```
let foo = Symbol();
```

132. 模板标签函数与模板文字之间不含空格。
级别：error
eslint : ***template-tag-spacing*** value:***never***

ok √ :

```
let name = `inspur${sayhello()}`
```

avoid × :

```
let name = `inspur ${sayhello()}`
```

133. 文件不能以 U + FEFF 开头。
级别：error
eslint : ***unicode-bom*** value:***never***

ok √ :

```
let abc
```

avoid × :

```
U+FEFF
let abc
```

134. 支持对 ES2015+ (ES6+) import/export 语法的校验。
级别：error
eslint : ***import/export***

135. import 必须放到文件头部，且绝对导入放在相对导入之前。
级别：error
eslint : ***import/first***

ok √ :

```
import Vue from 'vue'
import ToDoList from './ToDoList'
let list = []
```

avoid × :

```
import ToDoList from './ToDoList'
let list = []
import Vue from 'vue'
```

136. 禁止使用导出名称作为默认导出标识符。
级别：error
eslint : ***no-named-default***

137. 禁止使用 import 引入 webpack的loader。
级别：error
eslint : ***import/no-webpack-loader-syntax***
推荐使用require引入webpack的loader

138. 禁止使用已经废弃的api。
级别：error
eslint : ***node/no-deprecated-api***

ok √ :

```
let a = Buffer.from('10')
```

avoid × :

```
let a = new Buffer('10')
```

139. 禁止在 nodejs 中使用 process.exit作为异常抛出。
级别：error
eslint : ***node/process-exit-as-throw***

ok √ :

```
throw new Error('api错误')
```

avoid × :

```
process.exit(1)
```

140. 创建新的 promise 时参数名称必须一致。
级别：error
eslint : ***promise/param-names***

ok √ :

```
const p1 = new Promise(resolve => {
  setTimeout(() => {
    resolve()
  }, 1000)
})

const p2 = new Promise(resolve => {
  setTimeout(() => {
    resolve()
  }, 2000)
})
```

avoid × :

```
const p1 = new Promise(resolve => {
  setTimeout(() => {
    resolve()
  }, 1000)
})

const p2 = new Promise(handle => {
  setTimeout(() => {
    handle()
  }, 2000)
})
```

141. 在数组括号内强制实现一致的间距。
级别：error
eslint : ***standard/array-bracket-even-spacing*** value:***either***

ok √ :

```
let arr = []
arr = ['foo', 'bar', 'baz']
arr = [['foo'], 'bar', 'baz']
arr = [
  'foo',
  'bar',
```

```
  'baz'
]
```

avoid × :

```
var arr = ['foo', 'bar'];
arr = ['foo', 'bar' ];
arr = [ ['foo'], 'bar' ];
arr = ['foo',
  'bar'
];
```

142. 在计算的属性括号内强制执行一致的间距，不能含有空格。
级别：error
eslint : ***standard/computed-property-even-spacing*** value:***even***

ok √ :

```
var obj = { prop: "value" };
var a = "prop";
var x = obj[a]; // computed property in object member expression

var a = "prop";
var obj = {
  [a]: "value" // computed property key in object literal (ECMAScript
6)
};
```

avoid × :

```
obj[foo ]
obj[ 'foo']
var x = {[ b ]: a}
obj[foo[ bar ]]
```

143. 禁止在 callback 中直接传值，须先用变量接管，再传变量。
级别：error
eslint : ***standard/no-callback-literal***

ok √ :

```
let value = 122
callback(value)
```

```
let obj = { name: 123 }
callback(obj)
```

avoid × :

```
callback(123)
callback({ name: 123 })
```

144. 在花括号内强制实现一致的间距。

级别：error

eslint : ***standard/object-curly-even-spacing*** value:***either***

ok √ :

```
// simple object literals
var obj = { foo: "bar" };

// nested object literals
var obj = { foo: { zoo: "bar" } };

// destructuring assignment (EcmaScript 6)
var { x, y } = y;

// import/export declarations (EcmaScript 6)
import { foo } from "bar";
export { foo };
```

avoid × :

```
var obj = { 'foo': 'bar' };
var obj = {'foo': 'bar' };
var obj = { baz: {'foo': 'qux'}, bar};
var obj = {baz: { 'foo': 'qux'}, bar};
var {x } = y;
import { foo } from 'bar';
```