JavaScript编码规范

# 变量

## 所有的赋值都用const或者let，避免使用var

原因：var是全局作用域，会污染全局，且有变量提升；const/let是文件或块级作用域，没有变量提升；如果变量不需要重新赋值，则优先使用const

eslint：prefer-const，no-const-assign，no-var

# 对象

## 使用字面量来创建对象，避免new Object()

const item = {};

原因：没有本质上的区别，对象字面量的形式更简洁、更方便阅读，可以在一个地方定义多个属性

eslint：no-new-object

## 当创建一个带有动态属性名的对象时，用计算后属性名

function getKey(k) {

return `a key named ${k}`;

}

const obj = {

id: 5,

name: 'San Francisco',

[getKey('enabled')]: true,

};

## 用对象方法简写、属性缩写，且简写部分排在前面

const value = 1;

const atom = {

value,

addValue(value) {

return atom.value + value;

},

name: 'San Francisco',

};

原因：更简洁，哪些是简写形式，一目了然

eslint：object-shorthand、object-shorthand

## 只对那些无效的标示使用引号

const good = {

foo: 3,

bar: 4,

'data-blah': 5,

};

原因：更容易阅读，方便语法高亮，并且方便JS引擎做优化

eslint：quote-props

## 不直接调用Object.prototype上的方法

// 不好

console.log(object.hasOwnProperty(key));

// 推荐

console.log(Object.prototype.hasOwnProperty.call(object, key));

原因：对象中有自定义的同名属性，此时直接调用就会报错，比如： { hasOwnProperty: false }，或者是一个空对象Object.create(null)

eslint：no-prototype-builtins

## 对象浅拷贝时，推荐使用扩展运算符，而不是Object.assign。获取对象指定的几个属性时，用对象的rest解构运算符

const original = { a: 1, b: 2 };

// 浅拷贝

const copy = { ...original, c: 3 }; // copy => { a: 1, b: 2, c: 3 }

// rest 赋值运算符

const { a, ...noA } = copy; // noA => { b: 2, c: 3 }

# 数组

## 用字面量赋值，避免new Array()

原因：1.单参数陷阱，2.Array可能被重定义

例外：给构造函数传入指定的一个数值来创建稀疏数组

eslint：no-array-constructor

## 用Array#push 代替直接向数组中添加一个值

## 用扩展运算符做数组浅拷贝，类似上面的对象浅拷贝

## 用扩展运算符而不是Array.from来将一个可迭代的对象转换成数组

## 用 Array.from 去将一个类数组对象转成一个数组

## 用 Array.from 而不是扩展运算符去做map遍历。 因为这样可以避免创建一个临时数组。

## 在数组方法的回调函数中使用 return 语句。 如果函数体由一条返回一个表达式的语句组成， 并且这个表达式没有副作用， 这个时候可以忽略return

原因：部分数组方法回调要求有返回值

eslint：array-callback-return

## 如果一个数组有很多行，在数组的 [ 后和 ] 前断行

# 解构

## 用解构赋值来获取对象或数组的某个或多个属性值

原因：新的语法，更简洁

eslint：prefer-destructuring

## 多个返回值用对象的解构，而不是数据解构

原因：不用考虑返回值的顺序

# 字符串

## 对string用单引号 ''

原因：风格统一

eslint：quotes

## 超过100个字符的字符串不应该用string串联成多行

原因：不是很友好，且不易被搜索