# Web编程风格

## 声明变量

**变量**：let；

**常量**：const。

## 字符串

**静态字符串**：用单引号(‘’)；

**动态字符串**：用反引号(``)。

**const a = 'foobar';**

**const b = `foo${a}bar`;**

**const c = 'foobar';**

## 赋值

### 1使用数组成员对变量赋值时，优先使用解构赋值。

**const arr = [1, 2, 3, 4];**

// good

**const [first, second] = arr;**

### 2函数的参数如果是对象的成员，优先使用解构赋值。

// good

**function getFullName(obj) {**

**const { firstName, lastName } = obj;**

**}**

// best

**function getFullName({ firstName, lastName }) {**

**}**

### 3如果函数返回多个值，优先使用对象的解构赋值，而不是数组的解构赋值。这样便于以后添加返回值，以及更改返回值的顺序。

// good

**function processInput(input) {**

**return { left, right, top, bottom };**

**}**

**const { left, right } = processInput(input);**

## 对象

### 1单行定义的对象，最后一个成员不以逗号结尾。多行定义的对象，最后一个成员以逗号结尾。

// good

**const a = { k1: v1, k2: v2 };**

**const b = {**

**k1: v1,**

**k2: v2,**

**};**

### 2对象尽量静态化，一旦定义，就不得随意添加新的属性。如果添加属性不可避免，要使用Object.assign方法。

// if reshape unavoidable

**const a = {};**

**Object.assign(a, { x: 3 });**

// good

**const a = { x: null };**

**a.x = 3;**

### 3如果对象的属性名是动态的，可以在创造对象的时候，使用属性表达式定义。

// good

**const obj = {**

**id: 5,**

**name: 'San Francisco',**

**[getKey('enabled')]: true,**

**};**

## 数组

### 1使用扩展运算符（...）拷贝数组。

// bad

**const len = items.length;**

**const itemsCopy = [];**

**let i;**

**for (i = 0; i < len; i++) {**

**itemsCopy[i] = items[i];**

}

// good

**const itemsCopy = [...items];**

### 2使用Array.from方法，将类似数组的对象转为数组。

**const foo = document.querySelectorAll('.foo');**

**const nodes = Array.from(foo);**

## 函数

### 1立即执行函数可以写成箭头函数的形式。

**(() => {**

**console.log('Welcome to the Internet.');**

**})();**

### 2那些需要使用函数表达式的场合，尽量用箭头函数代替。因为这样更简洁，而且绑定了this。

// good

**[1, 2, 3].map((x) => {**

**return x \* x;**

**});**

// best

**[1, 2, 3].map(x => x \* x);**

### 3箭头函数取代Function.prototype.bind，不应再用self/\_this/that绑定 this。

// bad

**const self = this;**

**const boundMethod = function(...params) {**

**return method.apply(self, params);**

**}**

// acceptable

**const boundMethod = method.bind(this);**

// best

**const boundMethod = (...params) => method.apply(this, params);**

简单的、单行的、不会复用的函数，建议采用箭头函数。

## 4使用默认值语法设置函数参数的默认值。

// bad

**function handleThings(opts) {**

**opts = opts || {};**

**}**

// good

**function handleThings(opts = {}) {**

**// ...**

**}**

## Map

注意区分Object和Map，只有模拟现实世界的实体对象时，才使用Object。如果只是需要key: value的数据结构，使用Map结构。因为Map有内建的遍历机制。

**let map = new Map(arr);**

**for (let key of map.keys()) {**

**console.log(key);**

**}**

**for (let value of map.values()) {**

**console.log(value);**

**}**

**for (let item of map.entries()) {**

**console.log(item[0], item[1]);**

**}**

## Class

### 1总是用Class，取代需要prototype的操作。

// bad

**function Queue(contents = []) {**

**this.\_queue = [...contents];**

**}**

**Queue.prototype.pop = function() {**

**const value = this.\_queue[0];**

**this.\_queue.splice(0, 1);**

**return value;**

**}**

// good

**class Queue {**

**constructor(contents = []) {**

**this.\_queue = [...contents];**

**}**

**pop() {**

**const value = this.\_queue[0];**

**this.\_queue.splice(0, 1);**

**return value;**

**}**

**}**

### 2使用extends实现继承，因为这样更简单，不会有破坏instanceof运算的危险。

// bad

**const inherits = require('inherits');**

**function PeekableQueue(contents) {**

**Queue.apply(this, contents);**

**}**

**inherits(PeekableQueue, Queue);**

**PeekableQueue.prototype.peek = function() {**

**return this.\_queue[0];**

**}**

// good

**class PeekableQueue extends Queue {**

**peek() {**

**return this.\_queue[0];**

**}**

**}**

## 模块

### 1首先，Module语法是JavaScript模块的标准写法，坚持使用这种写法。使用import取代require。

// bad

**const moduleA = require('moduleA');**

**const func1 = moduleA.func1;**

**const func2 = moduleA.func2;**

// good

**import { func1, func2 } from 'moduleA';**

### 2使用export取代module.exports。

// commonJS的写法

**var React = require('react');**

**var Breadcrumbs = React.createClass({**

**render() {**

**return <nav />;**

**}**

**});**

module.exports = Breadcrumbs;

// ES6的写法

**import React from 'react';**

**const Breadcrumbs = React.createClass({**

**render() {**

**return <nav />;**

**}**

**});**

**export default Breadcrumbs**

如果模块只有一个输出值，就使用export default，如果模块有多个输出值，就不使用export default，不要export default与普通的export同时使用。

### 3不要在模块输入中使用通配符。因为这样可以确保你的模块之中，有一个默认输出（export default）。

// bad

**import \* as myObject './importModule';**

// good

**import myObject from './importModule';**

### 4如果模块默认输出一个函数，函数名的首字母应该小写。

**function makeStyleGuide() {**

**}**

**export default makeStyleGuide;**

### 5如果模块默认输出一个对象，对象名的首字母应该大写。

**const StyleGuide = {**

**es6: {**

**}**

**};**

**export default StyleGuide;**