ECMAScript 6 入门

作者：阮一峰

目录

其他

- 源码
- 修订历史
- 反馈意见

编程风格

上一章 下一章

本章探讨如何将 ES6 的新语法，运用到编码实践之中，与传统的 JavaScript 语法结合在一起，写出合理的、易于阅读和维护的代码。

多家公司和组织已经公开了它们的风格规范，下面的内容主要参考了 Airbnb 公司的 JavaScript 风格规范。

---

1. 块级作用域

（1）let 取代 var

ES6 提出了两个新的声明变量的命令：`let` 和 `const`。其中，`let` 完全可以取代 `var`，因为两者语义相同，而且 `let` 没有副作用。

```
'use strict';

if (true) {
  let x = 'hello';
}

for (let i = 0; i < 10; i++) {
  console.log(i);
}
```

上面代码如果用 `var` 替代 `let`，实际上就声明了两个全局变量，这显然不是本意。变量应该只在其声明的代码块内有效，`var` 命令做不到这一点。

`var` 命令存在变量提升效用，`let` 命令没有这个问题。

```
'use strict';

if (true) {
  console.log(x); // ReferenceError
  let x = 'hello';
}
```

上面代码如果使用 `var` 替代 `let`，`console.log` 那一行就不会报错，而是会输出 `undefined`，因为变量声明提升到代码块的头部。这违反了变量先声明后使用的原则。

所以，建议不再使用 `var` 命令，而是使用 `let` 命令取代。

（2）全局常量和线程安全

在 `let` 和 `const` 之间，建议优先使用 `const`，尤其是在全局环境，不应该设置变量，只应设置常量。

`const` 优于 `let` 有几个原因。一个是 `const` 可以提醒阅读程序的人，这个变量不应该改变；另一个是 `const` 比较符合函数式编程思想，运算不改变值，只是新建值，而且这样也有利于将来的分布式运算；最后一个原因是 JavaScript 编译器会对 `const` 进行优化，所以多使用 `const`，有利于提高程序的运行效率，也就是说 `let` 和 `const` 的本质区别，其实是编译器内部的处理不同。

```
// bad
var a = 1, b = 2, c = 3;

// good
const a = 1;
const b = 2;
const c = 3;
```

```
// best
const [a, b, c] = [1, 2, 3];
```

`const` 声明常量还有两个好处，一是阅读代码的人立刻会意识到不应该修改这个值，二是防止了无意间修改变量值所导致的错误。

所有的函数都应该设置为常量。

长远来看，JavaScript 可能会有多线程的实现（比如 Intel 公司的 River Trail 那一类的项目），这时 `let` 表示的变量，只应出现在单线程运行的代码中，不能是多线程共享的，这样有利于保证线程安全。

---

2. 字符串

静态字符串一律使用单引号或反引号，不使用双引号。动态字符串使用反引号。

```
// bad
const a = "foobar";
const b = 'foo' + a + 'bar';

// acceptable
const c = `foobar`;

// good
const a = 'foobar';
const b = `foo${a}bar`;
```

---

3. 解构赋值

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];

// bad
const first = arr[0];
const second = arr[1];

// good
const [first, second] = arr;
```

函数的参数如果是对象的成员，优先使用解构赋值。

```
// bad
function getFullName(user) {
  const firstName = user.firstName;
  const lastName = user.lastName;
}

// good
function getFullName(obj) {
  const { firstName, lastName } = obj;
}

// best
function getFullName({ firstName, lastName }) {
}
```

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。

```
// bad
function processInput(input) {
  return [left, right, top, bottom];
}

// good
function processInput(input) {
  return { left, right, top, bottom };
}

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

---

4. 对象

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。

```
// bad
const a = { k1: v1, k2: v2, };
const b = {
  k1: v1,
  k2: v2
};

// good
const a = { k1: v1, k2: v2 };
const b = {
  k1: v1,
  k2: v2,
};
```

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。

```
// bad
const a = {};
a.x = 3;

// if reshape unavoidable
const a = {};
Object.assign(a, { x: 3 });

// good
const a = { x: null };
a.x = 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。

```
// bad
const obj = {
  id: 5,
  name: 'San Francisco',
};
obj[getKey('enabled')] = true;

// good
const obj = {
  id: 5,
  name: 'San Francisco',
  [getKey('enabled')]: true,
};
```

上面代码中，对象 `obj` 的最后一个属性名，需要计算得到。这时最好采用属性表达式，在新建 `obj` 的时候，将该属性与其他属性定义在一起。这样一来，所有属性就在一个地方定义了。

另外，对象的属性和方法，尽量采用简洁表达法，这样易于描述和书写。

```
var ref = 'some value';

// bad
const atom = {
  ref: ref,

  value: 1,

  addValue: function (value) {
    return atom.value + value;
  },
};

// good
const atom = {
  ref,

  value: 1,

  addValue(value) {
    return atom.value + value;
  },
};
```

---

5. 数组

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];
```

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);
```

---

6. 函数

立即执行函数可以写成箭头函数的形式。

```
(() => {
  console.log('Welcome to the Internet.');
```

```
})();
```

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。

```
// bad
[1, 2, 3].map(function (x) {
  return x * x;
});

// good
[1, 2, 3].map((x) => {
  return x * x;
});

// best
[1, 2, 3].map(x => x * x);
```

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。如果函数体较为复杂，行数较多，还是应该采用传统的函数写法。

所有配置项都应该集中在一个对象，放在最后一个参数，布尔值不可以直接作为参数。

```
// bad
function divide(a, b, option = false ) {
}

// good
function divide(a, b, { option = false } = {}) {
}
```

不要在函数体内使用 arguments 变量，使用 rest 运算符（...）代替。因为 rest 运算符显式表明你想要获取参数，而且 arguments 是一个类似数组的对象，而 rest 运算符可以提供一个真正的数组。

```
// bad
function concatenateAll() {
  const args = Array.prototype.slice.call(arguments);
  return args.join('');
}

// good
function concatenateAll(...args) {
  return args.join('');
}
```

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 = {}) {
  // ...
}
```

---

7. Map 结构

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]);
}
```

---

8. Class

总是用 Class，取代需要 prototype 的操作。因为 Class 的写法更简洁，更易于理解。

```
// 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;
  }
}
```

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];
  }
}
```

---

9. 模块

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';
```

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';

class Breadcrumbs extends React.Component {
  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` ， `export default` 与普通的 `export` 不要同时使用。

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 from './importModule';

// good
import myObject from './importModule';
```

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;
```

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;
```

---

10. ESLint 的使用

ESLint 是一个语法规则和代码风格的检查工具，可以用来保证写出语法正确、风格统一的代码。

首先，安装 ESLint。

```
$ npm i -g eslint
```

然后，安装 Airbnb 语法规则，以及 import、a11y、react 插件。

```
$ npm i -g eslint-config-airbnb
$ npm i -g eslint-plugin-import eslint-plugin-jsx-a11y eslint-plugin-react
```

最后，在项目的根目录下新建一个 `.eslintrc` 文件，配置 ESLint。

```
{
  "extends": "eslint-config-airbnb"
}
```

现在就可以检查，当前项目的代码是否符合预设的规则。

`index.js` 文件的代码如下。

```
var unusued = 'I have no purpose!';

function greet() {
    var message = 'Hello, World!';
    alert(message);
}

greet();
```

使用 ESLint 检查这个文件，就会报出错误。

```
$ eslint index.js
index.js
  1:1  error  Unexpected var, use let or const instead          no-var
  1:5  error  unusued is defined but never used                 no-unused-vars
  4:5  error  Expected indentation of 2 characters but found 4  indent
  4:5  error  Unexpected var, use let or const in
```

```
5:5  error  Expected indentation of 2 characters but found 4  indent

✖ 5 problems (5 errors, 0 warnings)
```

上面代码说明，原文件有五个错误，其中两个是不应该使用 `var` 命令，而要使用 `let` 或 `const`；一个是定义了变量，却没有使用；另外两个是行首缩进为 4 个空格，而不是规定的 2 个空格。

留言

性的前提下尽量用常用的那10%来代替。比如根据我的经验，在同一个项目中prototypal inheritance和class based inheritance就应该最多只允许用一种，这样从排列组合的角度上就砍掉了很多可能发生的很tricky的问题。实际上更多的情况是这两个可以都不用，而只使用简单的mixin pattern，因为大部分真正的项目几乎没有情况必须要区分哪个是object自己的property哪个是从base继承来的，只要这个object上面有我需要用到的property就好了。所以这样一来我只需要用Object.assign或...来做mixin，而不用担心那一大堆额外语法功能衍生出来的各种问题，自然就可以大大降低代码分析阅读和debug的复杂度。类似的情况还有很多，总之es6、es7提供了这么多新功能，很多就是和以前的理念不兼容的，所以要么就以旧语法为基础稍微借鉴新语法当中简单实用的部分，要么就以新语法为基础抛弃一切可以抛弃的旧功能，切记不要两头都想占全，毕竟不是所有功能都真的有用，就如同goto，很多编程语言都有这个功能但几乎所有项目都会避免使用它。

1 ^ | v • Reply • Share ›

Benedict Jin • 8 months ago

Airbnb 的规范的确很棒，之前给 Airbnb 开源项目 Superset 做二次开发的时候，也顺便学习了下哈哈

https://yuzhouwan.com/posts... Superset 二次开发）

^ | v • Reply • Share ›

daeming • a year ago

误解了

^ | v • Reply • Share ›

ruanyf Mod ➚ daeming • a year ago

foo.apply(this, params) 啊

^ | v • Reply • Share ›

daeming ➚ ruanyf • a year ago

抱歉，是我理解错误了，书的内容正确

^ | v • Reply • Share ›

xzfd1010 • a year ago

“提供”应为“提高”吧

^ | v • Reply • Share ›

ruanyf Mod ➚ xzfd1010 • a year ago

谢谢指出，已经改正。

^ | v • Reply • Share ›

Tonedony Jine • a year ago

写的真的是太好了，简直完美

^ | v • Reply • Share ›

Yan frankly • a year ago

great!

zhaotoday • 2 years ago

感觉参考：https://github.com/feross/s... 会更好点，比较明显的区别就是不写分号。

Reply • Share ›

伍峰杰 → zhaotoday • 2 years ago

标准语法不是应该推荐都写分号吗？？

Reply • Share ›

fanjunzhi • 3 years ago

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。好像eslint会报警告

Reply • Share ›

伍峰杰 → fanjunzhi • 2 years ago

确实 默认的eslint 会报warning

Reply • Share ›

DanoR • 3 years ago

V8 5.0已经支持非严格模式下使用let了...今天放出更新的Node6.0已经将V8更新到6.0...望更新

Reply • Share ›

Mu Yue → DanoR • 3 years ago

Node6.0的module,可以用import了么

1 Reply • Share ›

maochunguang → Mu Yue • 3 years ago

我用6.0测试了,不支持import和export

Reply • Share ›

ruanyf Mod → DanoR • 3 years ago

谢谢指出，已经删除了这句话。

Reply • Share ›

Aki • 3 years ago

"只有模拟实体对象时，才使用Object"
什么情况叫做模拟实体对象？

Reply • Share ›

ruanyf Mod → Aki • 3 years ago

就是现实中存在的对象。

Reply • Share ›

qdladoooo • 3 years ago

一个滞后于时代的标准。唯一流行的拷贝泛型孤例，最终转向早已不被推崇的面向对象泛型。更适合大型项目，但又有些像xhtml之于html，减少了语言的容错能力。考虑各种语法糖，我个人感觉好过ES3，但不是最好ES6.

Reply • Share ›

louisGan • 3 years ago
这不是js，这是不强制缩进的python
Reply • Share ›

Andy • 3 years ago
对象尽量静态化
为什么要这么做？
Reply • Share ›

Folyd • 3 years ago
推荐也看一下Airbnb的Javascript编程规范
Reply • Share ›

yanxiong huang • 3 years ago
js 越来越像dart
Reply • Share ›

NferZhuang • 3 years ago
> 所有的函数都应该设置为常量。
>
> let表示的变量，只应出现在单线程运行的代码中，不能是多线程共享的，这样有利于保证线程安全。

这两个知识点，希望能够展开来讲，毕竟不是谁看到这里都能明白的。
至少也应该给一个参考链接，以便读者能够了解到相关知识。
Reply • Share ›

jiangtao ➚ NferZhuang • 3 years ago
追加 对此很困惑
Reply • Share ›

ruanyf Mod ➚ jiangtao • 3 years ago
这主要针对 Intel 的 River trail项目等多线程环境，但是没有应用场景，我把它改掉吧。
Reply • Share ›

Riophae Lee • 3 years ago
eslint-config-airbnb 我觉得算得上是检验前端水平的基本标准之一
Reply • Share ›

陈安 • 3 years ago
Class应该是不能像C#那样写成partial分在不同文件的吧。
还是prototype好，可以分几个文件定义一个类。
Reply • Share ›

disqus_7H8C6PW50p ➚ 陈安 • 3 years ago
你可以使用mixin来聚合多个文件。
36 Reply • Share ›

linhan ➚ 陈安 • 3 years ago

其实我觉得讨论 原型链 不好 而class 好 好像在说中文不好 英文才好的 概念一样 两种不同的语言思维 , 作者说要用class 的理由真的很无理,我没有其他意思 但是不能接受 说prototype 是一种不好的语言习惯 ,

因为 var a= function (){}

a.staticValue =1 ;//这个就是静态变量

a.staticFunction =function(){} //这个是静态方法

var dynamic= new a();

dynamic.prototype.dynamicValue_1 = 222;//这个不就是 成员变量

dynamic.prototype.dynamicFunction_1 =function(){

// 成员函数

}

我觉得这样用完全没有问题 ,少了static 还有很多的标识符, 我觉得说class好的人 ,像不想改变他以前用过的编程语言习惯...这样有点过分 就是为什么要js 适应他们的语言习惯 为啥java c ++ ,Python不改变,根本就没有所谓的什么 好的语言语法习惯....说到底就是 ES委员会有很多主流java c++的委员会成员 硬生生的强加语言习惯...

1 ^ | v • Reply • Share ›

张宁 ➚ linhan • 3 years ago

所以，，，你只是在试图说明中文比英文好？😢

^ | v • Reply • Share ›

linhan ➚ 张宁 • 3 years ago

注意我没说那个语言好....正确来说 就是因为某些人觉得别的语言好 才会将语法硬生生搬到其他语言身上... 我只是说语言各有特色 没必要迎合某些人.... js本身就有类的写法....

^ | v • Reply • Share ›

nd888915 ➚ linhan • 3 years ago

我也这么觉得

^ | v • Reply • Share ›

linhan ➚ nd888915 • 3 years ago

^^

^ | v • Reply • Share ›

dou4cc • 3 years ago

慎用箭头函数，个人觉得箭头函数主要用于惰性求值和漂亮的闭包。

^ | v • Reply • Share ›

yebo qin ➚ dou4cc • 3 years ago

不明白怎么个惰性求值法了

^ | v • Reply • Share ›

Harden Zhang • 3 years ago
尼玛，越来越像python了，简直一毛一样！
Reply • Share ›

黄晓落 • 4 years ago
如果所有匿名函数都使用箭头, 那么这个匿名不是一行就能写完的时候该怎么办??
Reply • Share ›

Jiangshui Yu ➚ 黄晓落 • 4 years ago
http://es6.ruanyifeng.com/#... 认真看下
Reply • Share ›

mobai~ • 4 years ago
赞,布道好文章
Reply • Share ›

Jarvis Chen • 4 years ago
var 变量提升的特点不知道是好处还是坏处
Reply • Share ›

Jare Guo • 4 years ago
静态字符串一律使用单引号或反引号，不使用双引号。动态字符串使用反引号。

这句话写错了
Reply • Share ›

bumfo • 4 years ago
React 的直接 html 的寫法也被 es6 支持了？
Reply • Share ›

Load more comments