JavaScript规范

js采用ES6，由于兼容浏览器版本问题--谷歌最低v52，尽量避免使用过高版本的ES规范或者做好向下兼容，遵循最大限度的可复用，可读性强，便于维护的原则。

语言规范

变量声明

const 和 let 都是块级作用域，var 是函数级作用域
使用规范：

- 变量声明使用const和let，不使用var
- 常量声明，使用const
- 变量声明，使用let
- 将所有的 const 和 let 分组

```
// bad
let a;
const b;
let c;
const d;
let e;

// good
const b;
const d;
let a;
let c;
let e;
```

对象

- 使用对象方法的简写方式

```
// bad
const item = {
  value: 1,

  addValue: function (val) {
    return item.value + val;
  }
};

// good
const item = {
  value: 1,

  addValue(val) {
    return item.value + val;
  }
};
```

- 使用对象属性值的简写方式

```
const job = 'FrontEnd';

// bad
const item = {
  job: job,
};

// good
const item = {
  job,
};
```

- 对象属性值的简写方式要和声明式的方式分组

```
const job = 'FrontEnd';
const department = 'JDC';

// bad
const item = {
  sex: 'male',
  job,
  age: 25,
  department,
};

// good
const item = {
  job,
  department,
  sex: 'male',
  age: 25,
};
```

数组

- 使用拓展运算符 ... 复制数组（注意：浅拷贝）

```
// bad
const items = [];
const itemsCopy = [];
const len = items.length;
let i;

// bad
for (i = 0; i < len; i++) {
  itemsCopy[i] = items[i];
}

// good
itemsCopy = [...items];
```

解构赋值

- 当需要使用对象的多个属性时，请使用解构赋值

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

  return `${firstName} ${lastName}`;
}

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

  return `${firstName} ${lastName}`
}

// better
function getFullName ({ firstName, lastName }) {
  return `${firstName} ${lastName}`
}
```

- 当需要使用数组的多个值时，请同样使用解构赋值

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

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

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

- 函数需要回传多个值时，请使用对象的解构，而不是数组的解构

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

// 如果是数组解构，那么在调用时就需要考虑数据的顺序
const [top, xx, xxx, left] = doSomething();

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

// 此时不需要考虑数据的顺序
const { top, left } = doSomething()
```

字符串

- 字符串统一使用单引号的形式 "

```
// bad
const department = "TSA";

// good
const department = 'TSA';
```

- 字符串太长的时候，请不要使用字符串连接符换行 \，而是使用 +

```
const str = '全流量分析 全流量分析 全流量分析 ' +
  '全流量分析 全流量分析 全流量分析 ' +
  '全流量分析 全流量分析 全流量分析 ';
```

- 程序化生成字符串时，请使用模板字符串

```
const test = 'test';

// bad
const str = ['a', 'b', test].join();

// bad
const str = 'a' + 'b' + test;

// good
const str = `ab${test}`;
```

模块

- 使用标准的 ES6 模块语法 import 和 export

```
// bad
const util = require('./util');
module.exports = util;

// good
import Util from './util';
export default Util;

// better
import { Util } from './util';
export default Util
```

- 不要使用 import 的通配符 *，这样可以确保你只有一个默认的 export

```
// bad
import * as Util from './util';

// good
import Util from './util';
```

分号

- 语句末尾需要添加分号，以确保被编译器的正常解析。

禁用方法

- eval()：能够将字符串解析为js语句，可能受到xss攻击，尽量避免使用
- with() {}： 扩展上下文的作用域，造成作用域污染，尽量避免使用

代码规范

统一团队的编码规范，有助于代码的维护。

单行代码块

- 单行代码块中推荐加入空格，便于阅读和维护

```
// bad
a ? ((a > b) ? fn1() : fn2()) : fn1();
// good
a ? ( (a > b) ? fn1() : fn2() ) : fn1();
```

大括号风格

在编程过程中，大括号风格与缩进风格紧密联系，用来描述大括号相对代码块位置的方法有很多。在 JavaScript 中，主要有三种风格，如下：

1. One True Brace Style

```
if (foo) {
  bar();
} else {
  baz();
}
```

2. Stroustrup

```
if (foo) {
  bar();
}
else {
  baz();
}
```

3. Allman

```
if (foo)
{
  bar();
}
else
{
  baz();
}
```

- 推荐使用One True Brace Style 风格

变量命名

- 当命名变量时，主流分为驼峰式命名（variableName）和下划线命名（variable_name），推荐使用驼峰式命名。

拖尾逗号

使用拖尾逗号的好处是，简化了对象和数组添加或删除元素，我们只需要修改新增的行即可，并不会增加差异化的代码行数。

```
let foo = {
  name: 'foo',
  age: '22',
}
```

逗号风格

- 逗号分隔列表时， 推荐将逗号放置在当前行的末尾。

```
// 不推荐
let foo = 1
,
bar = 2;

let foo = 1
, bar = 2;

let foo = ['name'
          , 'age']
// 推荐
let foo = 1,
    bar = 2;

let foo = ['name',
            'age'];
```

逗号空格

- 为了提高代码的可读性，推荐在逗号后面使用空格，逗号前面不加空格。

```
// 不推荐
let foo = 1,bar = 2;
let foo = 1 , bar = 2;
let foo = 1 ,bar = 2;
// 推荐
let foo = 1, bar = 2;
```

缩进

- 约定使用 空格 来缩进，而且缩进使用两个空格。
- 通过配置 .editorconfig ，将 Tab 自动转换为空格。

对象字面量的键值缩进

- 约定对象字面量的键和值之间不能存在空格，且要求对象字面量的冒号和值之间存在一个空格。

```
// 不推荐
let obj = { 'foo' : 'haha' };
// 推荐
let obj = { 'foo': 'haha' };
```

构造函数首字母大写

- 约定构造函数的首字母要大小，以此来区分构造函数和普通函数。

```
// 不推荐
let fooItem = new foo();
// 推荐
let fooItem = new Foo();
```

链式赋值

- 链式赋值容易造成代码的可读性差，所以团队约定禁止使用链式赋值

```
// 不推荐
let a = b = c = 1;
// 推荐
let a = 1;
let b = 1;
let c = 1;
```

代码块空格

- 推荐代码块前要添加空格。

```
// 不推荐
if (a){
  b();
}

function a (){}
// 推荐
if (a) {
  b();
}

function a () {}
```

操作符的空格

- 推荐操作符前后都需要添加空格。

```
// 不推荐
let sum = 1+2;
// 推荐
let sum = 1 + 2;
```

注释

添加合理的注释能便于维护，方便代码review和重构

- 特殊逻辑的语句，添加单行注释

```
// 推荐
let list = hasPermission ? data : []; // 判断是否有权限获取数组
```

- 方法前书写方法注释，描述参数和返回值

```
 /**
  * @description: 方法描述
  * @param {type} 参数名
  * @return:
  */
```

- js文件前添加注释，描述作者和创建时间

```
/*
 * @Description: 项目描述
 * @Author: 作者名
 * @Date: 创建时间
 */
```