JavaScript规范

js规范采用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规范并且做好向下兼容，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-tsap

const 和 let 都是块级作用域，var 是函数级作用域，变量声明必须加上关键字
使用规范：

- 声明变量使用const和let，不推荐使用var，避免变量提升等问题，更规范更安全
- 常量声明，使用const
- 变量声明，使用let
- 将所有的 const 和 let 分组

```
// 不推荐
let a;
const max;
let c;
const min;
let e;

// 推荐
const MAX;
const MIN;
let a;
let c;
let e;
```

块内函数声明-bfc

不要在块内声明一个函数，虽然很多 JS 引擎都支持块内声明函数, 但它不属于 ECMAScript 规范 (见 ECMA-262, 第13和14条). 各个浏览器糟糕的实现相互不兼容, 有些也与未来 ECMAScript 草案相违背. ECMAScript 只允许在脚本的根语句或函数中声明函数.

```
/* 不推荐 不要在块内声明一个函数 */
if (x) {
  function foo() {}
}
```

如果确实需要在块中定义函数, 建议使用函数表达式来初始化变量:

```
if (x) {
  let foo = function() {};
}
```

对象

- 使用对象方法的简写方式

```
// 不推荐
let item = {
  value: 1,

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

// 推荐
let item = {
  value: 1,

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

- 使用对象属性值的简写方式

```
let job = 'FrontEnd';

// 不推荐
let item = {
  job: job,
};

// 推荐
let item = {
  job,
};
```

- 对象属性值的简写方式要和声明式的方式分组

```
let job = 'FrontEnd';
let department = 'TSA';

// 不推荐
let item = {
  sex: 'male',
  job,
  age: 25,
  department,
};

// 推荐
let 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）

```
// 不推荐
let items = [];
let itemsCopy = [];
let len = items.length;
let i;

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

// 推荐
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

```
// 不推荐
function getFullName (user) {
  let firstName = user.firstName;
  let lastName = user.lastName;

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

// 推荐
function getFullName (user) {
  let { firstName, lastName } = user;

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

// 推荐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

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

// 不推荐
let first = arr[0];
let second = arr[1];

// 推荐
let [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

```
// 不推荐
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();

// 推荐
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 "

```
// 不推荐
const department = "TSA";

// 推荐
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';

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

// 推荐
const str = 'a' + 'b' + test;

// 推荐
const str = `ab${test}`;
```

模块

- 使用标准的 ES6 模块语法 import 和 export

```
// 不推荐
let util = require('./util');
module.exports = util;

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

// 推荐better
import { Util } from './util';  // 对象解构
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

```
// 不推荐
import * as Util from './util';

// 推荐
import Util from './util';
```

分号-tsap

- 语句末尾需要添加分号，以确保代码被编译器的正常解析。

```
/* if,function,for,while,switch等代码块后不需要分号 */
if (x) {
        // code
}       // 末尾无需分号

function foo() {
        // code
}       // 末尾无需分号

for (init; condition; update) {
        // code
}       // 末尾无需分号

/* 并不是所有{}后面都需要省略分号 */
var foo = function() {}; // 函数表达式初始化变量，末尾添加分号
var oMenu = {
        title: 'xx',
        name: 'xxx',
        cnt: 10
}; // 对象声明和初始化，末尾添加分号
```

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

```
// 不推荐
a ? ((a > b) ? fn1() : fn2()) : fn1();
// 推荐
a ? ( (a > b) ? fn1() : fn2() ) : fn1();
```

大括号风格-tsap

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
- 不推荐省略if和else后面的代码块符号{}。当阅读或扩展程序时，可能会引起意想不到的错误。

```
/* 不推荐省略if和else后面的代码块符号 */
if (err) return err.msg;

if (score > 50)
        setFail();
else
        setSuccess();

/* 如果扩展代码可能会导致意料之外的错误，比如: */
if (err) system.log(err.code);return err.msg;  // err.msg始终会执行

/* 推荐 总是使用{} */
if (err) {
        return err.msg;
}

if (score > 50) {
        setFail();
} else {
        setSuccess();
}

/* 如果确实希望代码保持在一行，应该这样 */
if (err) { return err.msg; }
```

变量命名

- 当命名变量时，主流分为驼峰式命名和下划线命名，推荐使用驼峰式命名。

细分约定-bfc：

- 所有变量名应是有意义的英文，不要使用拼音或单个字母+数字；
- 变量命名采用小驼峰法(第一个单词首字母小写)；
- 常量所有单词大写，并且每个单词间加下划线；
- 类命名必须是大驼峰法(所有单词第一个字母均大写)；
- 类的私有变量属性成员， 建议使用混合式命名，并在前面加下划线。

```
/* 推荐 */
var sHtml = '<div>动态HTML</div>'; // 变量见名知意，小驼峰
const PI = 3.14;                    // 常量大写
function Car() {                    // 类名大驼峰
        var _maxSpeed = 200;          // 私有变量属性，下划线开头
        this.start = function() {}
        this.stop = function() {}
}

/* 不推荐 */
var a1 = '<div>动态HTML</div>';    // 变量名字母+数字，无意义
var pi = 3.14;                      // 常量名小写，不易辨识
function car() {                    // 类名全小写，不易辨识
        var maxSpeed = 200;           // 私有变量，不易辨识
        this.start = function() {}
        this.stop = function() {}
}
```

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-tsap

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
- 单独一行：//(双斜线)与文字之间保留一个空格
- 代码后面添加注释：//(双斜线)与代码之间保留两个个空格，并且//(双斜线)与文字之间保留一个空格

```
/* 推荐 单行注释 */
// 以下为初始化代码：1)单独在一行
var maxCount = 10;  // 设置最大量；2)在代码后面注释
setTitle();         // 设置标题；3) 与上面的注释对齐
setName();          // 设置名称；3) 临近的多处注释推荐对齐
```

- 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
 */
```

方法和文件的注释，推荐使用idea的自定义注释代码块，方便快捷，戳这里查看

过长的单行予以换行-bfc

换行应选择在操作符和标点符号之后

```
/* 推荐 逻辑运算符在换行首，更利于代码阅读 */
if (oUser.nAge < 30
    && oUser.bIsChecked === true
    || oUser.sName === 'admin') {
    // code
}
```

循环的使用-bfc

在循环中，尽量使用变量先获取到相关数值，在放入循环中进行判断，否则非常影响性能。

```
/* 不推荐 每次循环都要重新计算length值 */
for (var i = 0; i < $('#id').children().length; i++) {
    // code
}

/* 推荐 先获取len再使用 */
for (var i = 0, len = $('#id').children().length; i < len; i++) {
    // code
}
```

代码间隔-bfc

- 功能模块之间，间隔一行就够了，不需要多行空白。
- 控制器间最多间隔两行

```
/* 不推荐 功能模块之间使用过多换行，太随意 */
// 功能模块1
init();
setTitle();
setName();



// 功能模块2
createElement();
getDate();
getData();

/* 推荐 模块间使用一行空白，必要时添加注释说明 */
// 功能模块1
init();
setTitle();
setName();

// 功能模块2
createElement();
getDate();
getData();
```

代码结构化-bfc

以Angular的controller代码为示例，展示常见的代码结构

```
/* 不推荐 代码零散 */
app.controller('appPage', function($scope){
        $scope.add = function() {
                // code
        }
        $scope.edit = function() {
                // code
        }
        $scope.delete = function() {
                // code
        }

        $scope.submitLoading = false;
        $scope.dataLoading = false;

        $scope.getParam = function() {
                // code
        }
        $scope.getData = function() {
                // code
        }

        // code
});
```

以功能模块来对函数进行归类

```
/* 推荐  */
app.controller('appPage', function($scope){
        // 菜单栏model层
        $scope.menuModel = {
                loading: false,
                add: function() {
                        // code
                },
                edit: function() {
                        // code
                },
                delete: function() {
                        // code
                }
        }

        // 数据列表model层
        $scope.mainModel = {
                loading: false,
                getParam = function() {
                        // code
                },
                getData: function() {
                        // code
                }
        }

        // code
});
```

另外：

1. 在HTML里直接使用的变量，须在controller里声明和初始化；
2. 代码封装，逻辑处理->service，功能组件->directive（比如时间组件、上传控件、图表绘制），或两者结合。