JS代码风格规范

1. 变量声明

使用 `const` 与 `let` 替代 var 声明变量

```
// 可变量用let
let a = 1;

// 引用类型使用const保持引用本身不会变
const obj = {
    "name": "Kangkang"
};
```

2. 对象

使用对象属性值与方法的简写

```
const value = 'Hello World';

// bad
const obj = {
    value: value,

    addValue: function(value) {
        obj.value += value;
    }
}

// good
const obj = {
    value,

    addValue(value) {
        obj.value += value;
    }
}
```

另外，注意对属性值的分组，以加强可读性

3. 数组

使用扩展运算符 `...` 复制数组

```

```

```
const items = [1, 2, 3, 4, 5, 6, 7];

// bad
const itemsCopy = [];
for(let i = 0; i < items.length; i++) {
    itemsCopy[i] = items[i];
}

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

统一使用 `Array.from` 来将类数组对象转为数组而不是扩展运算符 `...`

```
const foo = document.querySelectorAll('.foo');
const nodes = Array.from(foo);
```

4. 解构

使用解构存取和使用多属性对象

```
// 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}`;
}

// best
function getFullname({ firstName, lastName }) {
    return `${firstName} ${lastName}`;
}
```

对数组也可使用解构赋值

```
const arr = [1, 2, 3, 4, 5];
const [first, second] = arr;
```

使用对象解构来回传多个值

```
function getPosition(point) {
    // ... process ...
    return { x, y, z };
}
// 调用时选取所需值
const { x, y } = getPosition(point);
```

6. 字符串

字符串统一使用单引号，此外程序化生成字符串时，使用模板字符串代替字符串连接。

```
// bad
function hello(name) {
    return "Hello " + name;
}

// good
function hello(name) {
    return `Hello ${name}`;
}
```

7. 函数

优先使用函数声明而不是函数表达式，函数声明更容易在调用栈中被识别

```
// bad
const foo = function () {
    // ...
};

// good
function foo() {
    // ...
}
```

尽量使用默认参数来替代修改参数的行为

```
// bad
function handle(opts) {
    opts = opts || {};
    // or
    if (opts === void 0) {
        opts = {};
    }
}
```

```
// good
function handle(opts = {}) {
    // ...
}
```

8. 类型转换

```
// 1. 转为字符串
const id = String(this.id);

// 2. 转为数字
const num = +inputNum;
// 此外可以利用~~截整
const floatNum = 3.14;
const num = ~~floatNum; // 3

// 3. 转为布尔量
const flag = !!input;
```

对于部分函数会返回-1表示未查找等概念，对-1的处理见下

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

let index = arr.indexOf(5);
if ( ~index ) {
    // 表示 如果找到(即 不等于-1)
}
if ( !~index ) {
    // 表示 没有找到(即 等于-1)
}
```

9. 空白

1. 流程控制中的空格

```
// if 语句
if (flag) {
    // ...
}

if (flag1) {
    // ...
} else if (flag2) {
    // ...
} else {
    // ...
```

```
}

// switch 语句
switch () {
    case 0:
        // ...
        break;
    default:
        // ...
}

// for 语句
for (let i of arr) {
    // ...
}

for (let i = 0; i < arr.length; i++) {
    // ...
}

// while 语句
while (flag) {
    // ...
}

do {
    // ...
}
while ();

// 三元运算符
const goal = (flag) ? a : b;
```

2. 函数中的空格

```
function hello(name) {
    // ...
}

const foo = function (arg) {
    // ...
};

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

[1, 2, 3].reduce( (total, n) => {
    return total + n;
}, 0);
```

```
$target.click( () => {
    // ...
});

const func = () => {
    // ...
};
```

3. import 与 export中

```
import { targetA, targetB, targetC as C } from "../target";
export { func as target };
```

4. 函数调用时

```
if (!!flag) {
    // ...
}

=>

// 同样的还有 ~~input, !input, $input, +input等
// 即左括号后有除了字母、''外的别的特殊符号时
if ( !!flag ) {
    // ...
}

// 函数调用同理
func(a, b);
func( $targetA, $targetB );
// [], {}写在多行时候不用
func( [a, b] );
func([
    a, b
]);
func( {name: "kangkang"} );
func({
    name: "kangkang"
});
//当函数调用参数名过长时候可以用空格
func(longlonglongargment);
func( longlonglongargument );
```