前端编码规范

> 说明：此编码规范为测试版本，适用于前后段分离项目的前端。如有问题请及时沟通修正

目录

IDE

前端 IDE 推荐使用VS Code、Atom、Sublimetext和WebStrom;为了分离前端、稳定和快速编码。**建议**首选 VS Code。

书写规范

1. 为实现高性能代码执行、低版本浏览器兼容性、标准 Web 属性应用、无代码冲突和提高代码的可维护性等，需在一下至少前三个代码检测工具下编码。
    1. JavaScript Standard Style
    2. HTMLLint
    3. CSSLint
    4. TSLint
    5. LESSLint
    6. markdownlint

    > 对于*lint 监测出来的问题，不能通过修改对应 lint 的配置文件来避免你的代码出问题，请修改自己的代码以符合规约

2. 安装

```
# 需先安装nodeJS
$npm [-g] install eslint htmllint csslint
or  yarn [global] add eslint htmllint csslint
```

编码规范

1. HTML
    - HTMLLint
    - 标签内属性在前，事件在后

```
<div id="testId" onclick = "test()"></div>
```

- 模块注释

```
<!-- 文章列表列表模块-->
<div id="testId" onclick = "test()">
...
</div>
```

- 区块注释

```
<!--
@name: Drop Down Menu
@description: Style of top bar drop down menu.
@author: Ashu(Aaaaaashu@gmail.com)
-->
```

- HTML 推荐模板

```
<!DOCTYPE html>
    <html lang="zh-cmn-Hans">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
        <title>Style Guide</title>
        <meta name="description" content="不超过150个字符">
        <meta name="keywords" content="">
        <meta name="author" content="name, email@gmail.com">

        <!-- 为移动设备添加 viewport -->
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">

        <!-- iOS 图标 -->
        <link rel="apple-touch-icon-precomposed" href="/apple-touch-
icon-57x57-precomposed.png">

        <link rel="alternate" type="application/rss+xml" title="RSS"
href="/rss.xml" />
        <link rel="shortcut icon" href="path/to/favicon.ico">
    </head>
```

2. CSS

- 建议使用 Less 或 Sass

- CSSLint
- 命名
  - 禁用**hanyupinyin(汉语拼音)** (约定俗成的除外，如: youku , baidu , alibaba), 尽量使用专业英语
  - 驼峰式命名
  - 参数化命名，如黑色主题和白色主题，应写为 theme-black 和 theme-white
- 注释

```
/* 黑色主题 */
.theme-black {
  background-color: black;
}
```

- 内联外联
  - 尽可能地把样式分为外联样式，通过 class 引入，且有组织的分为全局样式和局部样式文件
- 响应式页布局

> **不要出现一个页面有两套 CSS 样式**

  - **在布局块状元素时，要么不设置高宽由内部元素自动撑开，要么定义 100%**
  - **flexbox 布局**（首选，注意各浏览器兼容性处理）
  - grid 布局
  - bootstrap12 栅格布局
  - 百分比布局
- 选择器必须是以某个前缀开头

```
.m-detail .info { sRules; }
.m-detail .current { sRules; }
.m-detail .news { sRules; }
```

- 前缀说明

前缀	说明	示例
g-	全局通用样式命名，前缀g全称为global，一旦修改将影响全站样式	g-mod
m-	模块命名方式	m-detail
ui-	组件命名方式	ui-selector
js-	所有用于纯交互的命名，不涉及任何样式规则。JSer拥有全部定义权限	js-switch

- 出于性能考量，在没有必要的情况下避免元素选择器叠加 Class 和 ID 同时使用。

- 声明顺序

  1. Positioning（定位）
  2. Box model（盒子模型）
  3. Typographic（排版）
  4. Visual（视觉效果，如动画颜色等）
  5. other（其他）

```
/* Positioning */
position: absolute;
top: 0;
right: 0;
bottom: 0;
left: 0;
z-index: 100;

/* Box model */
display: block;
box-sizing: border-box;
width: 100px;
height: 100px;
padding: 10px;
border: 1px solid #e5e5e5;
border-radius: 3px;
margin: 10px;
float: right;
overflow: hidden;

/* Typographic */
font: normal 13px "Helvetica Neue", sans-serif;
line-height: 1.5;
text-align: center;

/* Visual */
background-color: #f5f5f5;
color: #fff;
opacity: 0.8;

/* Other */
cursor: pointer;
```

3. JavaScript

- JavaScript Standard Style 编码风格
- 尽可能多的使用 ECMAScript 规范
- 语义化命名
  - 缩写规范

```
navigation   =>  nav
header       =>  hd
description  =>  desc
button => btn
previous => prev
```

- 应以功能或内容命名，不以表现形式命名；命名-缩写规范使用业界熟知的或者约定俗成的。**常量**采用全部大写，单词间下划线分隔的命名方式。**变量**采用驼峰命名法。**私有属性**、**变量**和**方法**以下划线 _ 开头。由多个单词组成的 **缩写词**，在命名中，根据当前命名法和出现的位置，所有字母的大小写与首字母的大小写保持一致。**枚举变量** 使用 Pascal 命名法。（与骆驼命名法类似。只不过骆驼命名法是首字母小写，而帕斯卡命名法是首字母大写）

```
let routerJump = ""; // 驼峰命名法
const PI = 3.14; // math.PI
let _name = "tom"; // 私有属性
function _getMyIdea() {
  return "my idea";
}
const ENVE_CIRCLE = "circle"; // 枚举变量
function XMLParser() {} // 多个单词组成的 缩写词
let reEvaluate = "重新评估";
```

- Promise 使用避免回调地狱

```
function getProductionInfo(id) {
  return new Promise((resolve, reject) => {
    let results = AjaxProcess();
    if (
      results.status == "success" &&
      results.data &&
      results.data.length > 0
    ) {
      resove(results.data);
    } else {
      reject(result.err);
    }
  });
}
```

- 命名
  - **禁用汉语拼音，使用英语**
  - 属性驼峰式命名

```
let tempObj = {};
tempObj["name"] = "myName";
tempObj["baseAttribute"] = "baseAttribute";
tempObj = null;
```

- 方法/函数

> 动词+名词 使用动宾短语和驼峰命名法，参数也使用驼峰命名法，可选参数以 **opt** 开头；如果是私有函数应为 _+动词+名词； 方法/函数如果是类，应使用名词，且要符合 Pascal 命名法，类的方法/属性和事件使用驼峰命名法。

```
/**
 * 获取姓名
 * @param {int} id - 用户id.
 * @returns {string} name - 用户姓名.
 */
function getName(id) {
  let name;
  // processing....
  return name;
}
/**
 * 获取姓名
 * @param {int} id - 用户id.
 * @returns {string} name - 用户姓名.
 */
function _getName(id) {
  let name;
  // processing....
  return name;
}
```

- 事件命名

> on+ 动作状态分词

```
<div onStarted="started()" />
```

- Bool 类型

> is/has+名词

```
// 是否前进
let isGo = true;
// 是否含有Name
let hasName = true;
```

- 操作符始终写在前一行，以免分号的隐式插入产生预想不到的问题

```
var y = a ? longButSimpleOperandB : longButSimpleOperandC;
if (a === "string" || s === "object") {
}
```

- 注释
  - 单行注释: 在代码上面注释，必须独占一行。// 后跟一个空格，缩进与下一行被注释说明的代码一致。

```
// 有事要喊出来
this.sayOut();
```

  - 后缀注释: 在一段语句后面后缀进行注释。 // 前后都跟一个空格，用于对某个语句的说明。

```
this.getBaseData(); // 基础数据请求完成之后执行的方法
```

  - 方法注释采用JSDoc式注释

```
/**
 * Book类，代表一个书本.
 * @constructor
 * @param {string} title - 书本的标题.
 * @param {string} author - 书本的作者.
 */
function Book(title, author) {
  this.title = title;
  this.author = author;
}
Book.prototype = {
  /**
   * 获取书本的标题
   * @returns {string|*}
   */
  getTitle: function() {
    return this.title;
  },
  /**
   * 设置书本的页数
   * @param pageNum {number} 页数
   */
  setPageNum: function(pageNum) {
    this.pageNum = pageNum;
```

```
    }
};
```

- 文件注释

文件注释置于文件顶部！ 用于告诉不熟悉这段代码的读者这个文件中包含哪些东西。 应该提供文件的大体内容, 它的作者, 依赖关系和兼容性信息。如下:

```
/**
 * @fileoverview Description of file, its uses and information
 * about its dependencies.
 * @author user@meizu.com (Firstname Lastname)
 * Copyright 2015 Meizu Inc. All Rights Reserved.
 */
```

- **建议**

多进行注释，对于自行定义的一些对象参数必须进行注释说明如下：

```
{
        name: null, // 姓名
        mobile: null, // 手机号码
        idCard: null, // 身份证号
        monthlyIncome: '', // 月收入
}
```

- 特性

[强制] 任何使用 this 的地方都要说名此 this 代表谁

[强制] 每个 var 只能声明一个变量。一个 var 声明多个变量，容易导致较长的行长度，并且在修改时容易造成逗号和分号的混淆

```
var hangModules = [];
var missModules = [];
var visited = {};
```

[强制] 变量必须 即用即声明，不得在函数或其它形式的代码块起始位置统一声明所有变量。变量声明与使用的距离越远，出现的跨度越大，代码的阅读与维护成本越高

[强制] 在 判断中使用类型严格的 ===

[建议] 按执行频率排列判断或代码分支的顺序

[建议] 如果函数或全局中的 else 块后没有任何语句，可以删除 else

```
function getName() {
  if (name) {
    return name;
  }

  return "unnamed";
}
```

[建议] 不要在循环体中包含函数表达式，事先将函数提取到循环体外。因为循环体中的函数表达式，运行过程中会生成循环次数个函数对象

```
function clicker() {
  // ......
}
for (var i = 0, len = elements.length; i < len; i++) {
  var element = elements[i];
  addListener(element, "click", clicker);
}
```

[建议] 对循环内多次使用的不变值，在循环外用变量缓存

```
var width = wrap.offsetWidth + "px";
for (var i = 0, len = elements.length; i < len; i++) {
  var element = elements[i];
  element.style.width = width;
  // ......
}
```

[建议] 类型检测优先使用 typeof。对象类型检测使用 instanceof。null 或 undefined 的检测使用 == null

```
// string
typeof variable === "string";

// number
typeof variable === "number";

// boolean
typeof variable === "boolean";

// Function
typeof variable === "function";

// Object
typeof variable === "object";
```

```
// RegExp
variable instanceof RegExp;

// Array
variable instanceof Array;

// null
variable === null;

// null or undefined
variable == null;

// undefined
typeof variable === "undefined";
```

[强制] 字符串开头和结束使用单引号 '。输入单引号不需要按住 shift，方便输入。实际使用中，字符串经常用来拼接 HTML。为方便 HTML 中包含双引号而不需要转义写法

[强制] 不允许修改和扩展任何原生对象和宿主对象的原型。**如需**扩展，写在公共类中

[建议] 属性访问时，尽量使用 .

[建议] 一个函数的长度控制在 50 行以内。将过多的逻辑单元混在一个大函数中，易导致难以维护。一个清晰易懂的函数应该完成单一的逻辑单元。复杂的操作应进一步抽取，通过函数的调用来体现流程。特定算法等不可分割的逻辑允许例外

```
function syncViewStateOnUserAction() {
  if (x.checked) {
    y.checked = true;
    z.value = "";
  } else {
    y.checked = false;
  }

  if (a.value) {
    warning.innerText = "";
    submitButton.disabled = false;
  } else {
    warning.innerText = "Please enter it";
    submitButton.disabled = true;
  }
}

// 直接阅读该函数会难以明确其主线逻辑，因此下方是一种更合理的表达方式：

function syncViewStateOnUserAction() {
  syncXStateToView();
  checkAAvailability();
}

function syncXStateToView() {
```

```
    y.checked = x.checked;

    if (x.checked) {
      z.value = "";
    }
  }

  function checkAAvailability() {
    if (a.value) {
      clearWarnignForA();
    } else {
      displayWarningForAMissing();
    }
  }
```

[建议] 一个函数的参数控制在 6 个以内

[建议] 属性在构造函数中声明，方法在原型中声明

```
function TextNode(value, engine) {
  this.value = value;
  this.engine = engine;
}

TextNode.prototype.clone = function() {
  return this;
};
```

- 程序最多有一个全局变量用来组件之间通信，但不能将后台请求的数据通过全局变量来传递
- 对程序片段里面的变量使用完销毁

```
let temp = {};
// temp =  ....
temp = null;
```

- **尽可能抛弃 JQuery**， 组件开发中不能使用 JQuery

接口规则

1. 参数规则

```
// 参数都用Object格式传输
const params= {
    a: 'a',
    b: 1,
    c: true
```

```
    }
    // 发起请求
    HttpClient.get('restURl', params).then(...)
```

2. 返回数据规则

```
    // 请求返回的参数格式
    {
        code: 200, // 状态码
        data: [] | {}, // 请求成功后返回的数据
        message: '状态码为非成功状态下需要显示的提示内容'
    }
```

3. Promise

```
 /**
        * JQuery Ajax Post请求
        * @constructor
        * @param {string} url - 请求目标Url.
        * @param {Objcet} params - 学生对象.
        * @param {int} params.age - 学生年龄.
        * @param {string} params.name -学生姓名 .
        * @param {string} proxyUrl - 请求代理地址.
        * @returns {Promise} Promise 请求返回Promise
        */
function Ajax4Post(url, params, proxyUrl){
    return new Promise((resolve, reject)=> {
        let requestUrl = window.location.hostname === url.hostname ? url:
`${proxyUrl}? ${url}`;
        let result = {};
        $.ajax({
            url: requestUrl,
            params: params,
            data: 'json'
        }).done((data) = > {
            if(data && data.length >= 0) {
                result['status'] = 'success';
                result['data'] = data;
                resolve(results);
            } else {
                result['status'] = 'failed'
                result['message'] = 'error reason...';
                reject(result);
            }
        }).fail((err)=> {
            result['status'] = 'failed';
            result['message'] = err;
            reject(result);
        })
```

```
    })
}
```

4. callback

> 推荐使用 Promise

```
/**
        * JQuery Ajax Post请求
        * @constructor
        * @param {string} url - 请求目标Url.
        * @param {Objcet} params - 学生对象.
        * @param {int} params.age - 学生年龄.
        * @param {string} params.name -学生姓名 .
         * @param {Function} callback -回调函数 ， 成功返回{status: 'sucess',
data: []}; 失败返回{status:'failed', message: errorObj}.
        * @param {string} proxyUrl - 请求代理地址.
        */
function Ajax4Post(url, params, callback, proxyUrl){
        let requestUrl = window.location.hostname === url.hostname ? url:
`${proxyUrl}? ${url}`;
        let result = {};
        $.ajax({
            url: requestUrl,
            params: params,
            data: 'json'
        }).done((data) = > {
            result['status'] = 'success';
            result['data'] = data;
            callback? callback(result): null;
            // callback? callback.bind(this, result): null;
        }).fail((err)=> {
            result['status'] = 'failed';
            result['message'] = err;
            callback? callback(result): null;
            // callback? callback.bind(this, result): null;
        })
    })
}
```

5. 参数 JSON 化

> 如 1 中的 params 所示

参考

- 参考地址

  javascript-style-guide