JavaScript 代码风格

1 前言

JavaScript 在百度一直有着广泛的应用，特别是在浏览器端的行为管理。本文档的目标是使 JavaScript 代码风格保持一致，容易被理解和被维护。

虽然本文档是针对 JavaScript 设计的，但是在使用各种 JavaScript 的预编译语言时(如 TypeScript 等)时，适用的部分也应尽量遵循本文档的约定。

2 代码风格

2.1 文件

[建议] JavaScript 文件使用无 BOM 的 UTF-8 编码。

解释：

UTF-8 编码具有更广泛的适应性。BOM 在使用程序或工具处理文件时可能造成不必要的干扰。

[建议] 在文件结尾处，保留一个空行。

2.2 结构

2.2.1 缩进

[强制] 使用 4 个空格作为一个缩进层级，不允许使用 2 个空格或 tab 字符。

[强制] switch 下的 case 和 default 必须增加一个缩进层级。

2.2.2 空格

[强制] 二元运算符两侧必须有一个空格，一元运算符与操作对象之间不允许有空格。

[强制] 用作代码块起始的左花括号{前必须有一个空格。

[强制] 关键字 if / else / for / while / function / switch / do / try / catch / finally 后，必须有一个空格。

[强制] 在对象创建时，属性中的:之后必须有空格，:之前不允许有空格。

[强制] 函数声明、具名函数表达式、函数调用中，函数名和(之间不允许有空格。

[强制] ,和;前不允许有空格。

[强制] 在函数调用、函数声明、括号表达式、属性访问、if / for / while / switch / catch 等语句中，()和[]内紧贴括号部分不允许有空格。

[强制] 单行声明的数组与对象，如果包含元素，{}和[]内紧贴括号部分不允许包含空格。

[强制] 行尾不得有多余的空格。

2.2.3 换行

[强制] 每个独立语句结束后必须换行。

[强制] 每行不得超过 120 个字符。

[强制] 运算符处换行时，运算符必须在新行的行首。

[强制] 在函数声明、函数表达式、函数调用、对象创建、数组创建、for 语句等场景中，不允许在,或;前换行。

[建议] 不同行为或逻辑的语句集，使用空行隔开，更易阅读。

[建议] 在语句的行长度超过 120 时，根据逻辑条件合理缩进。

[建议] 对于 if...else...、try...catch...finally 等语句，推荐使用在}号后添加一个换行的风格，使代码层次结构更清晰，阅读性更好：

```
if (condition) {
    // some statements;
}
else {
    // some statements;
}

try {
    // some statements;
}
catch (ex) {
    // some statements;
}
```

2.2.4 语句

[强制] 不得省略语句结束的分号。

[强制] 在 if / else / for / do / while 语句中，即使只有一行，也不得省略块{...}。

[强制] 函数定义结束不允许添加分号。

[强制] IIFE 必须在函数表达式外添加(，非 IIFE 不得在函数表达式外添加(

```
// good
var task = (function () {
   // Code
   return result;
})();

var func = function () {
};


// bad
var task = function () {
    // Code
    return result;
}();

var func = (function () {
});
```

2.3 命名

[强制] 变量使用 Camel 命名法。

var loadingModules = {};

[强制] 常量使用全部字母大写，单词间下划线分隔　的命名方式。

var HTML_ENTITY = {};

[强制] 函数使用 Camel 命名法。

function stringFormat(source) {

}

[强制] 函数的参数使用 Camel 命名法。

function hear(theBells) {

}

[强制] 类使用 Pascal 命名法。

function TextNode(options) {

}

[强制] 类的方法/属性使用 Camel 命名法。

function TextNode(value, engine) {

    this.value = value;

    this.engine = engine;

}

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

[强制] 枚举变量使用 Pascal 命名法，枚举的属性使用全部字母大写，单词间下划线分隔的命名方式。

```
var TargetState = {
    READING: 1,
    READED: 2,
    APPLIED: 3,
    READY: 4
};
```

[强制] 命名空间使用 Camel 命名法。

```
equipments.heavyWeapons = {};
```

[强制] 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。

```
function XMLParser() {
}
function insertHTML(element, html) {
}
var httpRequest = new HTTPRequest();
```

[强制] 类名使用名词。

```
function Engine(options) {
}
```

[建议] 函数名使用动宾短语。

```
function getStyle(element) {
}
```

[建议] boolean 类型的变量使用 is 或 has 开头。

```
var isReady = false;
var hasMoreCommands = false;
```

[建议] Promise 对象用动宾短语的进行时表达。

```
var loadingData = ajax.get('url');
loadingData.then(callback);
```

2.4 注释

2.4.1 单行注释

[强制] 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。

2.4.2 多行注释

[建议] 避免使用 /*...*/ 这样的多行注释。有多行注释内容时，使用多个单行注释。

2.4.3 文档化注释

[强制] 为了便于代码阅读和自文档化，以下内容必须包含以 /**...*/ 形式的块注释中：文件、namespace、类、函数或方法、类属性、事件、全局变量、常量、AMD 模块
[强制] 文档注释前必须空一行

2.4.4 类型定义

[强制] 类型定义都是以{开始，以}结束。
[强制] 对于基本类型 {string}, {number}, {boolean}，首字母必须小写。

2.4.5 文件注释

[强制] 文件顶部必须包含文件注释，用 @file 标识文件说明。
[建议] 文件注释中可以用 @author 标识开发者信息。

2.4.6 命名空间注释

[建议] 命名空间使用 @namespace 标识。

2.4.7 类注释

[建议] 使用 @class 标记类或构造函数。
[建议] 使用 @extends 标记类的继承信息。
[强制] 使用包装方式扩展类成员时，必须通过@lends 进行重新指向。
[强制] 类的属性或方法等成员信息使用@public / @protected / @private 中的任意一个，指明可访问性。

2.4.8 函数/方法注释

[强制] 函数/方法注释必须包含函数说明，有参数和返回值时必须使用注释标识。
[强制] 参数和返回值注释必须包含类型信息和说明。
[建议] 当函数是内部函数，外部不可访问时，可以使用 @inner 标识。
[强制] 对 Object 中各项的描述，必须使用@param 标识。
[建议] 重写父类方法时，应当添加 @override 标识。如果重写的形参个数、类型、顺序和返回值类型均未发生变化，可省略 @param、@return，仅用 @override 标识，否则仍应作完整注释。

2.4.9 事件注释

[强制] 必须使用@event 标识事件，事件参数的标识与方法描述的参数标识相同。
[强制] 在会广播事件的函数前使用@fires 标识广播的事件，在广播事件代码前使用 @event 标识事件。
[建议] 对于事件对象的注释，使用 @param 标识，生成文档时可读性更好。

2.4.10 常量注释

[强制] 常量必须使用 @const 标记，并包含说明和类型信息。

2.4.11 复杂类型注释

[建议] 对于类型未定义的复杂结构的注释，可以使用 @typedef 标识来定义。

2.4.12 AMD 模块注释

[强制] AMD 模块使用 @module 或 @exports 标识。

[强制] 对于已使用 @module 标识为 AMD 模块 的引用，在 namepaths 中必须增加 module: 作前缀。

[建议] 对于类定义的模块，可以使用 @alias 标识构建函数。

[建议] 多模块定义时，可以使用 @exports 标识各个模块

[建议] 对于 exports 为 Object 的模块，可以使用@namespace 标识。

[建议] 对于 exports 为类名的模块，使用@class 和@exports 标识。

2.4.13 细节注释

[建议] 细节注释遵循单行注释的格式。说明必须换行时，每行是一个单行注释的起始。

[强制] 有时我们会使用一些特殊标记进行说明。特殊标记必须使用单行注释的形式。下面列举了一些常用标记：

TODO: 有功能待实现。此时需要对将要实现的功能进行简单说明。

FIXME: 该处代码运行没问题，但可能由于时间赶或者其他原因，需要修正。此时需要对如何修正进行简单说明。

HACK: 为修正某些问题而写的不太好或者使用了某些诡异手段的代码。此时需要对思路或诡异手段进行描述。

XXX: 该处存在陷阱。此时需要对陷阱进行描述。

3 语言特性

3.1 变量

[强制] 变量在使用前必须通过 var 定义。

[强制] 每个 var 只能声明一个变量。

[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。

3.2 条件

[强制] 在 Equality Expression 中使用类型严格的 ===。仅当判断 null 或 undefined 时，允许使用 == null。

[建议] 尽可能使用简洁的表达式。

[建议] 按执行频率排列分支的顺序。

[建议] 对于相同变量或表达式的多值条件，用 switch 代替 if。

[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。

3.3 循环

[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。

[建议] 对有序集合进行遍历时，缓存 length。

[建议] 对循环内多次使用的不变值，在循环外用变量缓存。

[建议] 对有序集合进行顺序无关的遍历时，使用逆序遍历。

3.4 类型

3.4.1 类型检测

[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。

3.4.2 类型转换

[建议] 转换成 string 时，使用 + ''。

[建议] 转换成 number 时，通常使用 +。

[建议] string 转换成 number，要转换的字符串结尾包含非数字并期望忽略时，使用 parseInt。

[强制] 使用 parseInt 时，必须指定进制。

[建议] 转换成 boolean 时，使用 !!。

[建议] number 去除小数点，使用 Math.floor / Math.round / Math.ceil，不使用 parseInt。

3.5 字符串

[强制] 字符串开头和结束使用单引号 '。

[建议] 使用 数组 或 + 拼接字符串。

[建议] 复杂的数据到视图字符串的转换过程，选用一种模板引擎。

3.6 对象

[强制] 使用对象字面量 {} 创建新 Object。

```
// good
var obj = {};

// bad
var obj = new Object();
```

[强制] 对象创建时，如果一个对象的所有属性均可以不添加引号，则所有属性不得添加引号。

[强制] 对象创建时，如果任何一个属性需要添加引号，则所有属性必须添加 '。

```
// good
var info = {
    'name': 'someone',
    'age': 28,
    'more-info': '...'
};

// bad
var info = {
    name: 'someone',
    age: 28,
    'more-info': '...'
};
```

[强制] 不允许修改和扩展任何原生对象和宿主对象的原型。
[建议] 属性访问时，尽量使用 .。
[建议] for in 遍历对象时，使用 hasOwnProperty 过滤掉原型中的属性。

3.7 数组

[强制] 使用数组字面量 [] 创建新数组，除非想要创建的是指定长度的数组。
[强制] 遍历数组不使用 for in。
[建议] 不因为性能的原因自己实现数组排序功能，尽量使用数组的 sort 方法。
[建议] 清空数组使用 .length = 0。

3.8 函数

3.8.1 函数长度

[建议] 一个函数的长度控制在 50 行以内

3.8.2 参数设计

[建议] 一个函数的参数控制在 6 个以内。
[建议] 通过 options 参数传递非数据输入型参数。

```
/**
 * 移除某个元素
 *
 * @param {Node} element 需要移除的元素
 * @param {boolean} removeEventListeners 是否同时将所有注册在元素上的事件移除
 */
function removeElement(element, removeEventListeners) {
    element.parent.removeChild(element);
    if (removeEventListeners) {
        element.clearEventListeners();
    }
}
```

可以转换为下面的签名：

```
/**
 * 移除某个元素
 *
 * @param {Node} element 需要移除的元素
 * @param {Object} options 相关的逻辑配置
 * @param {boolean} options.removeEventListeners 是否同时将所有注册在元素上的事件移除
 */
function removeElement(element, options) {
    element.parent.removeChild(element);
    if (options.removeEventListeners) {
        element.clearEventListeners();
    }
}
```

3.8.3 闭包

[建议] 在适当的时候将闭包内大对象置为 null。

[建议] 使用 IIFE 避免 Lift 效应。

解释：

在引用函数外部变量时，函数执行时外部变量的值由运行时决定而非定义时，最典型的场景如下：

```
var tasks = [];
for (var i = 0; i < 5; i++) {
    tasks[tasks.length] = function () {
        console.log('Current cursor is at ' + i);
    };
}

var len = tasks.length;
while (len--) {
    tasks[len]();
}
```

以上代码对 tasks 中的函数的执行均会输出 Current cursor is at 5，往往不符合预期。

此现象称为 Lift 效应。解决的方式是通过额外加上一层闭包函数，将需要的外部变量作为参数传递来解除变量的绑定关系：

```
var tasks = [];
for (var i = 0; i < 5; i++) {
    // 注意有一层额外的闭包
    tasks[tasks.length] = (function (i) {
        return function () {
            console.log('Current cursor is at ' + i);
        };
    })(i);
}

var len = tasks.length;
while (len--) {
    tasks[len]();
}
```

3.8.4 空函数

[建议] 空函数不使用 new Function() 的形式。

[建议] 对于性能有高要求的场合，建议存在一个空函数的常量，供多处使用共享。

3.9 面向对象

[强制] 类的继承方案，实现时需要修正 constructor。

[建议] 声明类时，保证 constructor 的正确性。

[建议] 属性在构造函数中声明，方法在原型中声明。

[强制] 自定义事件的 事件名 必须全小写。

[强制] 自定义事件只能有一个 event 参数。如果事件需要传递较多信息，应仔细设计事件对象。

[建议] 设计自定义事件时，应考虑禁止默认行为。

3.10 动态特性

3.10.1 eval

[强制] 避免使用直接 eval 函数。

[建议] 尽量避免使用 eval 函数。

3.10.2 动态执行代码

[建议] 使用 new Function 执行动态代码。

3.10.3 with

[建议] 尽量不要使用 with。

3.10.4 delete

[建议] 减少 delete 的使用。

[建议] 处理 delete 可能产生的异常。

3.10.5 对象属性

[建议] 避免修改外部传入的对象。

[建议] 具备强类型的设计。

4 浏览器环境

4.1 模块化

4.1.1 AMD

[强制] 使用 AMD 作为模块定义。

[强制] 模块 id 必须符合标准：

模块 id 必须符合以下约束条件：

类型为 string，并且是由/分割的一系列 terms 来组成。例如：this/is/a/module。

term 应该符合[a-zA-Z0-9_-]+规则。

不应该有.js 后缀。

跟文件的路径保持一致。

4.1.2 define

[建议] 定义模块时不要指明 id 和 dependencies。推荐使用 define(factory) 的形式进行模块定义

[建议] 使用 return 来返回模块定义。

4.1.3 require

[强制] 全局运行环境中，require 必须以 async require 形式调用。

[强制] 模块定义中只允许使用 local require，不允许使用 global require。

[强制] Package 在实现时，内部模块的 require 必须使用 relative id。

[建议] 不会被调用的依赖模块，在 factory 开始处统一 require。

4.2 DOM

4.2.1 元素获取

[建议] 对于单个元素，尽可能使用 document.getElementById 获取，避免使用 document.all。

[建议] 对于多个元素的集合，尽可能使用 context.getElementsByTagName 获取。其中 context 可以为 document 或其他元素。指定 tagName 参数为 * 可以获得所有子元素。

[建议] 遍历元素集合时，尽量缓存集合长度。如需多次操作同一集合，则应将集合转为数组。

[建议] 获取元素的直接子元素时使用 children。避免使用 childNodes，除非预期是需要包含文本、注释和属性类型的节点。

4.2.2 样式获取

[建议] 获取元素实际样式信息时，应使用 getComputedStyle 或 currentStyle。

通过 style 只能获得内联定义或通过 JavaScript 直接设置的样式。通过 CSS class 设置的元素样式无法直接通过 style 获取。

4.2.3 样式设置

[建议] 尽可能通过为元素添加预定义的 className 来改变元素样式，避免直接操作 style 设置。

[强制] 通过 style 对象设置元素样式时，对于带单位非 0 值的属性，不允许省略单位。

4.2.4 DOM 操作

[建议] 操作 DOM 时，尽量减少页面 reflow。

页面 reflow 是非常耗时的行为，非常容易导致性能瓶颈。下面一些场景会触发浏览器的 reflow：

DOM 元素的添加、修改（内容）、删除。

应用新的样式或者修改任何影响元素布局的属性。

Resize 浏览器窗口、滚动页面。

读取元素的某些属性（offsetLeft、offsetTop、offsetHeight、offsetWidth、scrollTop/Left/Width/Height、clientTop/Left/Width/Height、getComputedStyle()、currentStyle(in IE)）。

[建议] 尽量减少 DOM 操作。

应在循环体中拼接 HTML 字符串，循环结束后写父元素的 innerHTML。

4.2.5 DOM 事件

[建议] 优先使用 addEventListener / attachEvent 绑定事件，避免直接在 HTML 属性中或 DOM 的 expando 属性绑定事件处理。

解释：

expando 属性绑定事件容易导致互相覆盖。

[建议] 使用 addEventListener 时第三个参数使用 false。

解释：

标准浏览器中的 addEventListener 可以通过第三个参数指定两种时间触发模型：冒泡和捕获。而 IE 的 attachEvent 仅支持冒泡的事件触发。所以为了保持一致性，通常 addEventListener 的第三个参数都为 false。

[建议] 在没有事件自动管理的框架支持下，应持有监听器函数的引用，在适当时候（元素释放、页面卸载等）移除添加的监听器。