**JavaScript编程技术管理文档**

**前言：**

因为最近学习webgl常使用JavaScript语言，之前编写课程项目前端的时候也经常用到它，故选择为JavaScript编写一份编程技术规范。该份规范主要参考的是京东、阿里和百度的JavaScript编程规范，同时也结合参考了菜鸟教程等网站上的语言指南和实操中的经验。此外，因为作业题目是技术管理文档，所以除了主要的编程规范之外还写了一些和技术文档相关的东西。

1. **简介**

JavaScript是一种轻量级、解释型或者说即时编译型的[编程语言](https://baike.baidu.com/item/%E7%BC%96%E7%A8%8B%E8%AF%AD%E8%A8%80/9845131?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/JavaScript/_blank)，广泛应用于网页开发中，主要用于增强网页的[交互性](https://baike.baidu.com/item/%E4%BA%A4%E4%BA%92%E6%80%A7/10758528?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/JavaScript/_blank)和[动态性](https://baike.baidu.com/item/%E5%8A%A8%E6%80%81%E6%80%A7/53339475?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/JavaScript/_blank)。作为Web开发的核心技术之一，它与[HTML](https://baike.baidu.com/item/HTML/97049?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/JavaScript/_blank)和[CSS](https://baike.baidu.com/item/CSS/5457?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/JavaScript/_blank)共同协作，HTML负责定义网页的内容结构，CSS处理样式和布局，而JavaScript则控制网页的行为和交互。

作为互联网上最流行的脚本语言，这门语言可用于HTML和web，更可广泛用于服务器、PC、笔记本电脑、平板电脑和智能手机等设备。

JavaScript 包含一个对象标准库（例如，Array、Date和Math）和一套核心语言元素（例如，运算符、控制结构和语句）。可以通过用额外的对象实现各种目的对核心 JavaScript 进行扩展；例如：

客户端JavaScript通过提供控制浏览器及其文档对象模型（DOM）的对象对核心语言进行扩展。例如：客户端扩展允许应用程序将元素放在某个HTML表单中，并且支持响应用户事件，比如鼠标点击、表单输入和页面导航。

服务器端JavaScript则通过提供和在服务器上运行JavaScript有关的对象对核心语言进行扩展。例如：服务器端扩展支持应用程序和数据库通信，提供应用不同调用间的信息连续性，或者在服务器上执行文件操作。

这意味着，在浏览器中，JavaScript可以改变网页（DOM）的外观。同样地，在服务器上，Node.js JavaScript可以对在浏览器执行的代码发出的自定义请求做出响应。

1. **JavaScript用法**

HTML中的JavaScript脚本代码必须位于<script>与</script>标签之间，<script>和 </script>会告诉JavaScript在何处开始和结束。

JavaScript 脚本代码可被放置在HTML页面的<body>和<head>部分中。通常的做法是把函数放入<head>部分中，或者放在页面底部。这样就可以把它们安置到同一处位置，不会干扰页面的内容。

除了放在html里，也可以把脚本保存到外部文件中。外部文件通常包含被多个网页使用的代码。外部JavaScript文件的文件扩展名是.js。如需使用外部文件，请在<script>标签的“src”属性中设置该.js文件。

关于JavaScript的语法学习，包括注释、变量、数据类型、对象、函数、字符串、运算符、条件语句、循环等。

1. **编程规范**

**（1）代码风格**

*文件*

1. 【推荐】JavaScript 文件使用无BOM的UTF-8编码。
2. 【推荐】在文件结尾处，保留一个空行。

*结构*

1. 【强制】使用4个空格做为一个缩进层级，不允许使用2个空格 或tab字符。
2. 【强制】switch下的case和default必须增加一个缩进层级。

示例：

switch (variable) {

case '1':

// do...

break;

case '2':

// do...

break;

default:

// do...

}

1. 【允许】在单行短代码中，为紧凑显示可省略部分空格（如三元表达式）。

示例：

const result = condition ? value1 : value2; //允许紧凑写法

1. 【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。

示例：

// good

function funcName() {

}

// bad

function funcName () {

}

1. 【强制】运算符处换行时，运算符必须在新行的行首。

示例：

// good

if (user.isAuthenticated()

&& user.isInRole('admin')

&& user.hasAuthority('add-admin')

|| user.hasAuthority('delete-admin')

) {

// Code

}

var result = number1 + number2 + number3

+ number4 + number5;

// bad

if (user.isAuthenticated() &&

user.isInRole('admin') &&

user.hasAuthority('add-admin') ||

user.hasAuthority('delete-admin')) {

// Code

}

var result = number1 + number2 + number3 +

number4 + number5;

1. 【推荐】在语句的行长度超过120时，根据逻辑条件合理缩进。
2. 【强制】函数定义结束不允许添加分号

示例：

// good

function funcName() {

}

// bad

function funcName() {

};

// 如果是函数表达式，分号是不允许省略的。

var funcName = function () {

};

1. 【允许】在编辑器配置统一的情况下，允许使用Tab键缩进，但需团队内严格约定。
2. 【允许】对于极短的条件语句，可省略换行。

*命名*

1. 【强制】常量使用全部字母大写，单词间下划线分割的命名方式。

示例：

var loadingModules = {};

1. 【允许】在临时变量中使用单字母命名。

示例：

for (let i = 0; i < 10; i++) { /\* ... \*/ }

1. 【强制】命名空间使用Camel命名法。

示例：

equipments.heavyWeapons = {};

1. 【强制】类名使用名词。

示例：

function Engine(options) {

}

1. 【推荐】boolean类型的变量使用is或has开头。

示例：

var isReady = false;

var hasMoreCommands = false;

1. 【允许】个人项目中，简单常量可以使用Camel命名法（如非全局常量）。

示例：

const maxItems = 20; // 个人项目中允许

*注释*

1. 【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. 【推荐】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。
3. 【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：

解释：

文件

namespace

类

函数或方法

类属性

事件

全局变量

常量

AMD 模块

1. 【强制】文件顶部必须包含文件注释，用@file标识文件说明。

示例：

/\*\*

\* @file Describe the file

\*/

1. 【推荐】命名空间使用@namespace标识。

示例：

/\*\*

\* @namespace

\*/

var util = {};

1. 【强制】AMD模块使用@module或@exports标识。
2. 【允许】在快速原型或临时调试代码中，使用行尾注释。（标注临时）

**（2）语言特性**

*变量*

1. 【强制】变量、函数在使用前必须先定义。

不通过var定义变量将导致变量污染全局环境。

示例：

// good

var name = 'MyName';

// bad

name = 'MyName';

1. 【强制】每个var只能声明一个变量。

示例：

// good

var hangModules = [];

var missModules = [];

var visited = {};

// bad

var hangModules = [],

missModules = [],

visited = {};

1. 【允许】在函数式编程场景中，使用var声明无修改的变量（需明确意图）。

示例：

var list = [1, 2, 3]; // 声明后无修改，允许var

*条件*

1. 【强制】在Equality Expression中使用类型严格的===。仅当判断null undefined时，允许使用== null。

示例：

// good

if (age === 30) {

// ......

}

// bad

if (age == 30) {

// ......

}

1. 【推荐】尽可能使用简洁的表达式。
2. 【允许】在非严格类型校验场景中，使用==判断。

*循环*

1. 【推荐】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。
2. 【推荐】对有序集合进行遍历时，缓存length。

*类型*

1. 【推荐】转换成string时，使用+‘’。

示例：

// good

num + '';

// bad

new String(num);

num.toString();

String(num);

1. 【强制】使用parseInt时，必须指定进制。

示例：

// good

parseInt(str, 10);

// bad

parseInt(str);

*字符串*

1. 【强制】字符串开头和结束使用单引号‘。
2. 【允许】在 HTML 字符串中使用双引号，避免转义（需与外部单引号配合）。

示例：

const html = '<div class="container">Content</div>'; // 允许双引号包裹HTML

*对象*

1. 【强制】使用对象字面量{}创建新object。

示例：

// good

var obj = {};

// bad

var obj = new Object();

1. 【推荐】属性访问时，尽量使用.。

示例：

info.age;

info['more-info'];

1. 【允许】在动态属性名场景中，使用[]访问，比如属性名由变量生成。

示例：

const prop = 'user' + id; const value = obj[prop]; // 允许动态属性访问

1. 【允许】在第三方兼容场景中，扩展原生对象原型。

示例：

// 仅为兼容旧库，需注释说明 if (!Array.prototype.includes) { Array.prototype.includes = function() { /\* ... \*/ }; }

*数组*

1. 【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。

示例：

// good

var arr = [];

// bad

var arr = new Array();

1. 【推荐】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。

*函数*

1. 【允许】在简单场景中，使用赋值表达式替代默认参数（如非严格类型场景）。

示例：

function log(message) { message = message || 'Default message'; // 允许简单默认值 }

1. 【推荐】一个函数的长度控制在50行以内。
2. 【推荐】空函数不使用new Function()的形式。

示例：

var emptyFunction = function () {};

1. 【允许】在回调函数中，为简洁性省略参数括号。

示例：

numbers.map(num => num \* 2); // 允许单参数省略括号

*面向对象*

1. 【强制】类的继承方案，实现时需要修正constructor。

示例：

/\*\*

\* 构建类之间的继承关系

\*

\* @param {Function} subClass 子类函数

\* @param {Function} superClass 父类函数

\*/

function inherits(subClass, superClass) {

var F = new Function();

F.prototype = superClass.prototype;

subClass.prototype = new F();

subClass.prototype.constructor = subClass;

}

1. 【推荐】属性在构造函数中声明，方法在原型中声明。
2. 【推荐】设计自定义事件时，应考虑禁止默认行为。

*动态特性*

1. 【强制】避免使用直接eval函数。
2. 【推荐】减少delete的使用。
3. 【推荐】处理delete可能产生的异常。

示例：

try {

delete o.x;

}

catch (deleteError) {

o.x = null;

}

**（3）浏览器环境**

*模块化*

1. 【强制】使用AMD作为模块定义。
2. 【强制】模块id必须符合标准。
3. 【允许】在小型脚本中，使用 CommonJS 规范（require/module.exports）替代 AMD。

示例：

*const fs = require('fs'); // 小型脚本中允许CommonJS*

1. 【推荐】定义模块的时候不要指明id和dependencies。

示例：

推荐使用define（factory）的形式进行模块定义。

define(

function (require) {

}

);

1. 【推荐】使用return来返回模块定义。

示例：

define(

function (require) {

var exports = {};

// ...

return exports;

}

);

1. 【强制】全局运行环境中，require必须以async require形式调用。

示例：

// good

require(['foo'], function (foo) {

});

// bad

var foo = require('foo');

*DOM*

1. 【推荐】对于单个元素，尽可能使用document.getElementByTd获取，避免使用document.all。
2. 【推荐】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。
3. 【推荐】操作DOM时，尽量减少页面reflow。
4. 【推荐】尽量减少DOM操作。
5. 【允许】在临时调试工具中，使用document.write输出内容（需后续清理）。

示例：

document.write('<div>Debug info</div>'); // 临时调试允许

1. **协作规范**
2. 【强制】所有代码在合并到主分支之前必须经过至少一位团队成员的审查。审查内容包括但不限于代码是否符合上述文档规范、逻辑是否正确、是否存在潜在的性能问题等。
3. 【强制】采用合理的分支管理策略，如 Gitflow 工作流。开发新功能时必须基于 develop 分支创建新的功能分支，修复 bug 时基于 master 分支创建 hotfix 分支，严禁在主分支上直接进行开发和修改。
4. 【强制】代码提交时必须填写清晰、有意义的提交信息。提交信息应简要描述本次提交的目的和主要改动内容，格式可以参考 “[功能模块] 描述本次提交做了什么，例如修复了某个 bug 或添加了某个功能”。
5. 【推荐】团队定期组织代码分享会，成员可以分享自己在开发过程中遇到的问题、解决方案以及学到的新技术、新技巧等，促进团队整体技术水平的提升。
6. 【推荐】当代码发生重大变更或新功能添加时，及时更新相关的技术文档，包括但不限于接口文档、使用说明文档等，以保证文档与代码的一致性，方便其他成员理解和维护。
7. 【推荐】定期对项目代码进行备份，可使用云存储或外部存储设备等方式，以防代码丢失或损坏。同时，在每次重大版本发布后，对代码进行一次完整的归档备份。
8. 【允许】成员可以根据自己的习惯和需求，配置个性化的开发环境，但需保证开发环境的基本要求与项目一致，且不会对项目的协作开发造成影响。
9. **常见问题**
10. 函数返回undefined：

解决方案：确保函数中有明确的return语句。

1. 比较两个对象是否相等：

解决方案：检查对象属性引用是否相同。——>递归检查所有属性值。

1. 优化循环性能：

解决方案：缓存数组长度/使用更高效的循环结构。

1. 防止内存泄露：

解决方案：一般的泄露场景检查——未清除的定时器或回调/未移除的DOM事件监听器/闭包导致的变量引用。

1. 安全的从url获取查询参数：

使用URLSearchParamsAPI，示例如下：

const params = new URLSearchParams(window.location.search);