|  |  |
| --- | --- |
| Document number 文档编号 | Confidentiality level 密级 |
|  | 内部公开 |
| Document version 文档版本 | Total 12 pages 共 12 页 |
| V1.00 |

JavaScript语言编码规范

|  |  |  |  |
| --- | --- | --- | --- |
| Prepared by  拟制 |  | Date  日期 | 2016-09-06 |
| Reviewed by  评审人 |  | Date  日期 | yyyy-mm-dd |
| Approved by  批准 |  | Date  日期 | yyyy-mm-dd |

Revision Record 修订记录

| Date  日期 | | Revision Version  修订版本 | Sec No.  修改章节 | | Change Description  修改描述 | | Author  作者 | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| yyyy-mm-dd | Vx.xx | | |  | |  | |  |
|  |  | | |  | |  | |  |
|  |  | | |  | |  | |  |
|  |  | | |  | |  | |  |
|  |  | | |  | |  | |  |

[1 命名规范 1](#_Toc460918212)

[1.1 基本原则 1](#_Toc460918213)

[1.2 Prototype/Method命名规范 1](#_Toc460918214)

[1.2.1 原型Prototype 1](#_Toc460918215)

[1.2.2 方法Method/函数Function 1](#_Toc460918216)

[1.3 字段命名规范 2](#_Toc460918217)

[1.3.1 常量 2](#_Toc460918218)

[1.3.2 变量 2](#_Toc460918219)

[1.3.3 集合 2](#_Toc460918220)

[1.3.4 特殊值 2](#_Toc460918221)

[1.4 异常 3](#_Toc460918222)

[1.5 文件命名规范 3](#_Toc460918223)

[1.6 命名约定表 3](#_Toc460918224)

[2 代码组织与风格 4](#_Toc460918225)

[2.1 整体风格 4](#_Toc460918226)

[2.1.1 缩排 4](#_Toc460918227)

[2.2 文件布局 5](#_Toc460918228)

[3 注释格式规范 5](#_Toc460918229)

[4 编程规范 5](#_Toc460918230)

[4.1 原型 5](#_Toc460918231)

[4.1.1 toString() 5](#_Toc460918232)

[4.2 方法 6](#_Toc460918233)

[4.2.1 基本原则 6](#_Toc460918234)

[4.2.2 参数 6](#_Toc460918235)

[4.3 表达式与语句 7](#_Toc460918236)

[4.3.1 基本原则 7](#_Toc460918237)

[4.3.2 控制语句 7](#_Toc460918238)

[4.3.3 循环语句 7](#_Toc460918239)

[4.4 错误处理 7](#_Toc460918240)

[4.4.1 基本原则 7](#_Toc460918241)

[4.4.2 可预见错误 8](#_Toc460918242)

[4.4.3 不可预见错误 8](#_Toc460918243)

[4.5 测试与Bug跟踪 8](#_Toc460918244)

[4.5.1 基本原则 8](#_Toc460918245)

[4.5.2 跟踪和缺陷处理 9](#_Toc460918246)

# 命名规范

## 基本原则

规范的命名能使程序更易阅读，从而更易于理解。它们也可以提供一些标识功能方面的信息，有助于更好的理解代码和应用。

* 使用可以准确说明变量/函数/原型(prototype)的完整英文描述符。例如， firstName，listAllUsers 或 CorporateCustomer等，严禁使用汉语拼音、不相关单词及汉字进行命名。
* 采用大小写混合，提高名字的可读性。一般应该采用小写字母，但是原型(prototype)命名时任意单词的首字母大写。
* 尽量少用缩写，但如果一定要使用，当使用公共缩写和习惯缩写等，如实现（implement）可缩写成impl，经理（manager）可缩写成mgr等，严禁滥用缩写。
* 避免使用长名字（最好不超过 25 个字母）。
* 避免使用相似或者仅在大小写上有区别的名字。
* 避免使用数字，但可用2代替to，用4代替for等，如：go2Jsp。此外，若元素id包含数字则除外

## Prototype/Method命名规范

### 原型Prototype

所有单词首字母大写。使用能确切反应该原型含义、功能等的词。一般采用名词。

### 方法Method/函数Function

方法的命名应采用完整的、能够清楚表达该方法功能的英文描述符，首字母小写，其他单词的首字母大写。方法名称的第一个单词常常采用一个有强烈动作色彩的动词。

取值类使用get前缀，设值类使用set前缀，判断类使用is(has)前缀。如下例所示：

getName()

setSarry()

isLogon()

方法参数建议顺序：(被操作者，操作内容，操作标志，其他……)。如下例所示：

function replace (sourceStr, //源字串

oldStr, //被替换字串

newStr){ //替换为字串

}

## 字段命名规范

### 常量

采用完整的英文大写单词，在词与词之间用下划线连接，如：DEFAULT\_VALUE

### 变量

对不易清楚识别出其类型的变量应使用类型缩写作其前缀，如字符串使用strXXX，boolean使用isXXX，hasXXX等等。除第一个单词外其余单词首字母大写。

对私有实例变量可使用\_前缀，但在其存取方法中则应该将其前缀去掉。

### 集合

一个集合，例如数组或xml节点集合，应采用复数命名来表示变量中存放的是一组对象。命名应采用完整的英文描述符，名字中所有非首单词的第一个字母应大写，适当使用集合缩写前缀。如下例所示：

集合变量命名示例

var rowNodes = xmlDom.selectNodes(“/\*/data/row”);

var aryUsers = []; //用户列表

### 特殊值

我们在程序里经常会用到一些量，它是有特定的含义的。例如，现在我们写一个薪金统计程序，公司员工有50人，我们在程序里就会用50这个数去进行各种各样的运算。在这里，50就是“特殊值”。当别的程序员在程序里看到50这个数，将很难知道它的含义，造成理解上的困难。

在程序里出现“特殊值”会降低程序的可读性、可维护性和可扩展性，故规定不得出现此类“特殊值”。避免的方法是把“特殊值”定义为一个常量。注意这个常量的命名应该能表达该数的意义，并且应该全部大写，且单词之间使用下划线。例如上面50这个数，我们可以定义一个名为NUM\_OF\_EMPLOYEES的常量。这样，别的程序员在读程序的时候就可以很容易的理解了。

## 异常

通常情况下不需要自定义异常原型，仅使用内建error对象。

异常原型由表示该异常类型的单词和Exception组成，如ResponseException。

异常实例一般使用e、ex等，在多个异常时使用该异常名或简写加E，Ex等组成，例如：SQLEx、ActionEx。

## 文件命名规范

外部js文件一律小写命名。

## 命名约定表

|  |  |  |
| --- | --- | --- |
| **操作项** | **命名约定** | **示例** |
| 参数 | 使用传递值/对象的完整的英文描述符。 | userID |
| 字段/属性 | 字段采用完整的英文描述，第一个字母小写，任何中间单词的首字母大写。 | firstName |
| 布尔型的获取成员函数 | 所有的布尔型获取函数必须用单词 is（has）做前缀。 | isString()  hasMoney() |
| 原型 | 采用完整的英文描述符，所有单词的第一个字母大写。 | Customer |
| 异常原型名 | 由表示该异常类型等的单词和Exception组成 | SQLException  ActionException |
| 异常实例名 | 通常采用字母 e 、ex表示异常。  多个异常时使用异常名或其简写加E、Ex等构成 | e  SQLEx |
| 静态常量字段（常量） | 全部采用大写字母，单词之间用下划线分隔。采用静态常量获取成员函数。 | DEFAULT\_NAME |
| 局部变量 | 采用完整的英文描述符，第一个字母小写，但不要隐藏已有字段。例如，如果有一个字段叫 firstName，不要让一个局部变量叫 firstName。 | strName,totalMoney |
| 循环计数器 | 通常采用字母 i，j，k 或者 counter，index | i,j,k,count,index |
| 成员函数 | 采用完整的英文描述说明成员函数功能，第一个单词尽可能采用一个生动的动词，除第一个单词外，每个单词第一个字母小写。 | openFile()  addUser() |
| 获取成员函数 | 被访问字段名的前面加上前缀 get。 | getUserName() |
| 设置成员函数 | 被访问字段名的前面加上前缀 set。 | setUserName () |

# 代码组织与风格

代码组织和风格的基本原则是：利于个人开发，便于相互交流。

因个人习惯和编辑器等可以设置而形成自己独特的风格，但必须前后一致，并符合本规范的基本要求和原则。

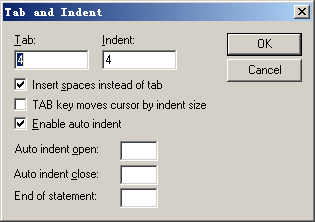
本章所涉及到的内容一般都可在editplus工具编辑环境中进行相应设置。

## 整体风格

整体风格可以参考Java篇代码组织与结构的整体风格章节。

### 缩排

代码中以TAB（4个字符）缩进，在编辑器中请将TAB设置为相同的长度，否则在不同编辑器或设置下会导致TAB长度不等而影响整个程序代码的格式。在editplus中通过Document－Tab Indent中设置，设置参数如下图所示：



所有脚本代码全部左起空4格位开始，如下：

//不要这样顶格写

function test{

…

}

//无论外部.js文件或者页面内嵌脚本，一律左起空四格再写

function test(){

…

}

## 文件布局

外部js文件一般放置在与HTML页面文件平级的js目录下。

内嵌js，一般应放在HTML标签<head>内，特殊情况也可在<body>内

# 注释格式规范

基本同JAVA篇的注释格式规范章节。

# 编程规范

javascript是一种基于对象的、解释型的脚本语言，它本身不如JAVA那样严格，因此编程规范上可以相对简化一些。

## 原型

### toString()

每个原型都应该定义toString()方法以便输出有用信息。

toString()方法示例：

/ \*\*

\* 函数说明：输出Tree基本信息

\*/

function Tree(){

…

}

Tree.prototype.toString = function(){

return “[Tree]”;

}

## 方法

### 基本原则

避免在一个较长的方法里提供多个出口：

//不要使用这种方式，当处理程序段很长时将很难找到出口点

if(condition){

return A;

}else{

return B;

}

//建议使用如下方式

var result = null;

if(condition){

result = A;

}else{

result = B;

}

return result;

### 参数

避免过多的参数，一般以5个为限。

如果确实需要输入多个参数，可以考虑将参数类型定义成一个原型或object，以成员去描述各个参数。

考虑变更函数参数的因素，也应考虑如上处理，避免函数发生变更（个数、顺序）时需要在所有使用到的地方进行修改。

## 表达式与语句

### 基本原则

表达式和语句应清晰、简洁，易于阅读和理解，避免使用晦涩难懂的语句。

使用圆括号明确表达式执行优先级。

### 控制语句

判断中如有常量，则应将常量置与判断式的右侧。如：

if ( true == isAdmin())...

if ( null == user)...

boolean类型判断语句尽量明确条件比较值true/false

//不建议使用

if (isCond)...

if (!isCond)...

//尽量使用

if(true == isCond)…

if (false == isCond)…

if(true!= isCond)…

if…else if…else语句必须使用{}将每个判断条件后的执行语句括起来。

### 循环语句

* 循环中必须有终止循环的条件或语句，避免死循环。
* 当多层循环嵌套时，计数器变量注意不要有冲突。
* 注意循环条件在执行循环过程中是否会发生变化，如果会则必须把循环条件的值在执行循环前获取而不要在每次循环去执行。
* 考虑运行效率问题也应把循环条件值放在循环执行前获取。

## 错误处理

### 基本原则

* 通常的法则是系统在正常状态并且用户正常操作下，不应产生任何异常。
* 对可预见的错误不进行捕捉。
* 对不可预见或者难以解决（通常由于浏览器BUG造成）错误进行try{…}catch(e){..}捕捉处理。

### 可预见错误

对可预见的错误不进行捕捉处理，而是在错误发生前通过条件判断避免发生，如：

//若不对div1是否为null进行检查，则在其为null时会抛出缺少对象错误

document.getElementById(“div1”).style.width = 100;

//预先对对象进行检查

var objDiv1 = document.getElementById(“div1”);

if(objDiv1!=null){

objDiv1.style.width = 100;

}

### 不可预见错误

对不可预见或者因浏览器、脚本解析器BUG造成的难以解决的错误需要进行捕捉处理，如：

try{

xmlhttp.open(“GET”,url,NOT\_ASYNC);

}catch(e){

alert(e.description);

}

对捕捉到的错误一般情况必须给出反馈处理，例如alert。

## 测试与Bug跟踪

### 基本原则

* 测试不通过的代码不得提交到CVS库或者发布。
* 对于将来可能会引起错误，或者需求变更可能会带来很大改动量的编程方式应尽量预先避免，不能因为图一时省事而不考虑后果。

### 跟踪和缺陷处理

* 当系统出现Bug时应由该Bug负责人（代码负责人）尽快修改。
* Bug的处理根据其优先级高低和级别高低先后处理。
* Bug级别和优先级别参见《测试手册》。
* 禁止隐瞒Bug。