Neusoft

东软机密

文件编号：D05-PDS075

JavaScript 编码规范

版本：0.0.0-1.2.0

2009-6-30

东软集团股份有限公司 软件开发事业部

文件修改控制

修改编号	版本	修改条款及内容	修改日期
1	0.0.0-1.1.0	更换 LOGO	2005-7-29
2	0.0.0-1.1.1	1、修改公司标识为“东软集团股份有限公司”。 2、修改文件密级标识为“东软机密”。 3、按照《文件编写规范》修改各章节标题的字体和大小。	2008-8-25
3	0.0.0-1.2.0	根据公司要求，转换为OpenOffice格式	2009-06-19

目　录

1 介绍

这篇文档的目的是说明JavaScript的编码规范和它们的重要性.主要是为开发者所使用, 对大部分的项目管理者也有很大作用,使他们看到清楚连贯的源代码.项目管理者应该鼓励他们的团队成员在写代码的时候遵循编码规范.

1.1 适用范围

JavaScript编码规范是为client-side JavaScript所制定的, 但是也可适用于server-side JavaScript编程如Sybase DynaScript.

1.2 什么是编码规范

在写代码的时候所遵循特定的样式是为编码规范.当然编码规范不可能完全理想化和覆盖所有可能的方面.无论什么时候如果有什么怀疑的地方还需要靠经验和所有的知识进行决定.

1.3 为什么需要编码规范?

几乎没有一个软件在它的整个生命周期中只被它的原作者所维护. 因此应用编码规范变得很有用. 编码规范将使代码在最大程度上保持一致性. 一致性使代码易读易于理解. 易于开发易于维护.使在开发人员之间共享代码更方便.

长期的优点:

- ✓ 对不同的项目中的人员的配置更灵活，因为所有的人员都使用相同的语言进行交流和编码;
- ✓ 提高了软件的质量;
- ✓ 因为清楚的代码而节省了解决错误的时间;
- ✓ 因为可重用和清楚的代码而节省了开发时间.

1.4 感谢

这片文档表达了JavaScript编码规范. 尽管JavaScript和Java在大多数方面根本不相同, 但还是在某些方面存在着类似之处.因此本文以Sun微系统公司的Java编码规范作为基础. 以防止出现几份不一致的编码规范.

2 文件的构成

大多数顺序化(top-down)的程序文件都难于维护，甚至更难于修改. 为了便利的编写这些文件, 应遵守下面一些基本的规则:

- HTML文件或script libraries应该由多个段落组成，这些段落应该由一些空行分隔开并且每个段落都应该有一段可选的注释.
- 任何时候不要使文件长度超过2000行，否则文件内容会变得晦涩难读.

2.1 Script libraries

scripts放在哪? 实际上，可在HTML文档的任何地方插入<SCRIPT> 标记. 因为HTML 页面的加载方式是从上到下进行的, 当文档加载调用一个 script 函数或变量的时候，对应的函数或变量必须已经被显式的定义过了，否则导致错误的产生.

然而把HTML-tags 和JavaScript 代码用script libraries分开也是一种可取的方法. script library 是一个单独的文件，它只包含JavaScript 代码或函数. 可以引入或包含library 来调用library 中的函数. 尽管每种编程语言都有自己的一套包含或引入文件的方式, 但是都有可能以以下的HTML-tag包含进一个script library:

```
<SCRIPT LANGUAGE="JavaScript" SRC="cookies.js"></SCRIPT>
```

Script libraries 有很多优点, 如:

- 能够减少HTML, ASP, JSP 或Lotus Domino 文件的大小;
- 能够重用其他HTML文件中的函数;
- 易于与其他项目或应用程序共享script.

什么时候在HTML文件中使用library 代替直接进行编码?

- 一些常用的函数,如设置或取得cookies.
- 一大段JavaScript代码. 不要仅仅为了一行script而创建一个library , 但是推荐保存所有多行的scripts在library中.

将library中具有相同主题的函数分组. 并给script library文件一个有条理的名字用来描述有哪些函数和代码在这个library中. 如果项目或应用程序使用不只一个library ， 则最好在不同的目录中保存这些library.

包含script library的最好的位置是在<HEAD> 和</HEAD>之间, 因为HTML 页面首先加载head部分.

2.2 开始的注释

所有的script文件都应该以下面详细列出的注释块的格式开头:主题, 作者, 创建日期, 文件名, 描述说明, 版权提示和文档的版本信息.如果有其他需要的话可以添加更多的标签

例1:

```
/*
* ============================================================
* C O O K I E S
* ============================================================
* AUTHOR : Joost H. van Rossum / Business Solutions
* CREATION DATE: 01-JAN-2001
* WORKFILE : cookies.js
* DESCRIPTION : You can use these functions to either get,
* set or delete cookies. These functions will
* only work with a browser that supports
* javascript 1.2
* COPYRIGHT : © 2001 Atos Origin All rights reserved
* ============================================================
* VERSION : 3
* DATE : 15-JAN-2001
* AUTHOR : J.M. de Vries / Business Solutions
* DESCRIPTION : Functions are now also working for JS 1.1
* ============================================================
* VERSION : 2
* DATE : 10-JAN-2001
* AUTHOR : Joost H. van Rossum / Business Solutions
* DESCRIPTION : Improved documentation
* ============================================================
*/
```

例2:

```
/*
* ============================================================
* S T R I N G S
* ============================================================
* AUTHOR : J.M. de Vries / Business Solutions
* CREATION DATE: 18-DEC-2000
* WORKFILE : stringScripts.js
* DESCRIPTION : Scripts concerning modification or validation
* of textstrings.
* COPYRIGHT : © 2001 Atos Origin All rights reserved
* ============================================================
* VERSION : 2
* DATE : 05-JAN-2001
* AUTHOR : Joost H. van Rossum / Business Solutions
* DESCRIPTION : Improved documentation
* ============================================================
*/
```

2.2.1 关键字扩充

大多数的版本控制系统在源文件中规定了一个选项来使用关键字扩充机制使文档自动更新. 这些关键字可以被用来代替手动书写注释的过程. 例如Visual Source Safe (VSS) 提供以下关键字:

关键字	描述
$History: $	把文件加到历史纪录
$Modtime: $	加入最后一次修改的日期和时间
$Revision: $	加入VSS版本号
$Workfile: $	加入文件名

举例提供一个VSS惯例的文档的历史纪录，注释文本部分如下:

```
/*
* $History: $
*/
```

实际显示的结果是:

```
/*
* $History: cookies.js $
*
* ***************** Version 25 *****************
* User: Jan.pietersen Date: 25-07-00 Time: 5:19p
* Updated in $/project/javascript
* checkCookie function added
*
* ***************** Version 24 *****************
* User: Jan.pietersen Date: 19-07-00 Time: 17:21
```

```
* Updated in $/project/javascript
* Bug in getCookie removed
*
* ***************** Version 23 *****************
* User: Piet.Jansen Date: 18-07-00 Time: 16:58
* Updated in $/project/javascript
* Improved documentation
*
* ***************** Version 22 *****************
* User: Piet.Jansen Date: 10-06-00 Time: 13:47
* Updated in $/project/javascript
* Functions are now also working for JavaScript1.1
*
*/
```

注意点: VSS中的关键字扩充机制对于所有的文件缺省选项是不可用的.只有VSS管理员才能够使关键字扩充机制可用.

3 一般的 script 布局

3.1 缩进

一个缩进单元为4个空格.

注意点: 一些编辑器使用8个空格的tabs键. 如果这样的话，要把tab键的空格数改为4个.

3.2 换行

超过100个字符长度的行总是无效的. 即使是在一台只有很低的分辨率(800*600)的终端上，也要尽可能的查看代码行是否会产生水平的滚动条. 当一个表达式不能够满足放在一行中的时候, 根据以下的原则将它拆分成几行:

- 操作符前换行;
- 逗号后换行;
- 尽量选择higher-level breaks而不用lower-level breaks: 尽量把有一定逻辑关系的代码段放在一起;
- 以相同的起始位置对齐每一行;
- 如果上述的规则有能使代码不能与其他的代码明确的分开来或为了使代码靠右对齐,使用8个空格的缩进代替4个空格的缩进.

一些在函数在换行时的例子:

```
var varname = someFunction(longExpression1, longExpression2,
                           longExpression3, longExpression4);

var varname = someFunction1(longExpressionA,
                           someFunction2(longExpressionB,
                                         longExpressionC));
```

下面是从算术表达式开始换行的两个例子. 应避免使用第一种形式，折行发生在括号内表达式的时候.

避免:

```
var varname1 = varname2 * (varname3 - varname4
             + varname5) - 4 * varname6;
```

正确:

```
var varname1 = varname2 * (varname3 - varname4 + varname5)
             - 4 * varname6;
```

对于if-else 语句的换行处理应使用8个空格,四个空格的缩进不能与if的内容很清楚的区分开. 例如:

避免:

```
if (((conditionA) && (conditionB))
    || ((conditionC) && (conditionD))
    || !((conditionE) && (conditionF))) {
    someFunction();
}
```

if中的代码不容易被看清楚. 使用以下几种缩进来代替:

正确:

```
if (((conditionA) && (conditionB))
       || ((conditionC) && (conditionD))
       || !((conditionE) && (conditionF))) {
    someFunction();
}

if (((conditionA) && (conditionB)) || ((conditionC) && (conditionD))
       || !((conditionE) && (conditionF))) {
    someFunction();
}
```

4 注释

有两种形式的注释可供使用: /*…*/和//…. 注释通常意味着把某段代码注释起来或对某段特

定实现的说明. 注释应该能够用来为代码提供一个概括性的说明和一些代码本身不能表示出的信息. 注释应该指包括阅读和理解程序的相关的内容.

注意点: 缺少注释会使代码难于理解，但过多的代码有时也会表现出代码的质量很糟糕.

4.1 实现注释的风格

JavaScript代码有四种实现注释的形式, 块注释, 单行注释,尾注释和行尾注释.

4.1.1 块注释

块注释应该在前面用一个空行与其他的代码分隔开来:

```
/*
* This is a block comment.
* Each line is preceded by an asterisks.
*/
```

4.1.2 单行注释

一行就能写完的短注释:

```
if (condition) {
    /* Short description about the following statments */
    statements;
}
else {
    /* Short description about the following statments */
    statements;
}
```

4.1.3 尾注释

能够与代码共存于一行的非常短的注释:

```
if (condition) {
    statements; /* Very short description */
}
else {
    statements; /* Very short description */
}
```

如果在一个代码块中需要不止一条短注释行,它们应该采用相同的缩进格式.

4.1.4 行尾注释

另一种实行短注释的方法是使用行尾注释:

```
if (condition) {
    statements; // Very short description
}
else {
    // Short description about the following statments
    statement1;
    statement2;
    statement3;
}
```

5 语句

5.1 简单语句

每个语句都应该以分号(;)结尾 并且每行都应该最多包含一条语句.例如:

正确:

```
i++;
i--;
varName1 = varName2 * 15;
```

避免:

```
i++; i--;
```

5.2 复合语句

复合语句由一对括号和括号内的语句所组成,如*if-else*, *while*和 *for* 语句. 左括号应该位于复合语句所在行的末尾.而右括号应该另起一行并与复合语句的开头对齐.

当括号用于复合语句的时候应该被成对使用,即使在只有一行语句的时候. 这使得当加入一条新语句的时候不会意外的忘了括号.

如果括号内的语句超出了一屏的范围, 那就长得难于理解.正常来说括号内的代码行数不应该超

过30行.为了避免出现以上的那种现象, 函数应该采用下面的这种写法.

正确:

```
function doSomething() {
    statement1;
    statement2;
    ……
    ……
    statement19;
    statement20;
}

function doSomeMoreThings() {
    statement1;
    statement2;
    ……
    ……
    statement19;
    statement20;
}
if (condition1) {
    doSomething();
    doSomeMoreThings();
}
```

避免:

```
if (condition1) {
    statement1;
    statement2;
    statement3;
    statement4;
    ……
    ……
    statement43;
    statement44;
    statement45;
}
```

5.3 返回语句

返回语句中的返回值不应该用括号括起来,除非明显的以某种特定的方式返回值. 如:

正确:

```
return true;
return myString.length;
return (size ? size : defaultSize);
```

避免:

```
return (true);
return (myString.length);
return size ? size : defaultSize;
```

5.4 If, if-else, if else-if else 语句

if-else 语句应遵循下面的格式:

```
if (condition1) {
    statements;
}

if (condition1) {
    statements;
}
else {
    statements;
}

if (condition1) {
    statements;
}
else if (condition2) {
    statements;
}
else {
    statements;
}
```

5.4.1 圆括号

使用 *if-else* 语句的时候用括号括起来使语句更易于阅读,把条件集中到一起.

```
if (condition) {
    statements;
}

if ((condition1) || (condition2)) {
    statements;
}

if ((condition1) && ((condition2) || (condition3))) {
    statements;
}
```

如果在*if-else* 中存在着大量的条件语句如 *AND* 和 *OR* 等条件,则会使条件明显地难于阅读.因此推荐把部分的条件作为一个*boolean*型的变量保存起来,并在*if-else*中使用这个*boolean*型的变量.

正确:

```
var variableName = ((condition2) || (condition3));
if (((condition) && (variableName)) || (condition4)) {
    statements;
}
```

避免:

```
if (((condition) && ((condition2) || (condition3))) || (condition4)) {
    statements;
}
```

5.5 For 语句

*for*语句应遵循下面的格式:

```
for (initialisation; condition; update) {
    statements;
}
```

正确:

```
for (var i = 0; i < yourArray.length; i++) {
```

```
    document.write(yourArray[i] + '<BR>');
}
```

避免:

```
for (var i = 0; i < yourArray.length; i++)
    document.write(yourArray[i] + '<BR>');

for (var i = 0; i < myArray.length; i ++) document.write(myArray[i]);
```

可以在*for* 语句的初始化部分,条件部分和计数部分中使用逗号操作符用到一个或多个语句. 但是强烈推荐不要使用这种形式以避免产生复杂的代码.如果使用这种形式的话,那么使用的语句和变量不要超过3个.

5.6 While 语句

while 语句应遵循下面的格式:

```
while (condition) {
    statements;
}
```

5.7 Do-while 语句

do-while 语句应遵循下面的格式:

```
do {
    statements;
} while (condition);
```

5.8 Switch 语句

如果可能的话,使用*switch* 语句而不是使用复杂的*if-else*语句. *switch* 语句提供与*if-else*语句同样的作用, 但是它更易于阅读. *switch*语句应遵循下面的格式:

```
switch(character) {
    case 'A': {
        statements;
        break;
    }
```

```
        case 'B': {
            statements;
            break;
        }
        case 'C': {
            statements;
            break;
        }
        default: {
            statements;
            break;
        }
    }
```

尽管default case中的break语句是多余的,但是这样做能够防止在以后的时候添加其它的case语句产生没有进行中断处理所产生的错误.

5.8.1 不中断

虽然可以不使用break语句以使case能够继续执行下去.但是这有可能在调试代码或查找错误时变得困难重重. 因此一个不中断的注释应该被添加到没有使用break语句的case条件后面:

```
    switch(character) {
        case 'X': {
            statements;
            /* Falls through */
        }
        case 'Y': {
            statements;
            break;
        }
        case 'Z': {
            statements;
            break;
        }
        default: {
            statements;
            break;
        }
    }
```

5.9 函数

script library中的各个函数之间用空行分开,他们应该有下列的格式:

```
function functionName1() {
    statements;
}
function functionName2(argument1, argument2) {
    statements;
}
```

一个函数不应该超过30行. 超过30行的函数应该被拆分成各个独立的代码段,并放到两个或多个函数中.

5.9.1 函数命名

函数名应该应该以一个小写字母开头. 紧接着的第二个单词的第一个字母大写. 函数的名字必须对函数的行为进行描述并以一个动词开头.

JavaScript简单的把函数分为以下几种类型:

- 返回true 或者false的函数;
- 返回特定值的函数 (字符串,数字,等.);
- 赋值的函数 (字符串, 数字,等等.);
- 执行一些语句作用的函数.

返回true 或者false的函数应以“is”开头:

```
isVisible();
isEmpty();
isCorrectDate();
```

返回特定值的函数应以“get”开头:

```
getCookie();
getHeight();
```

赋值的函数应以 “set”开头:

```
setCookie();
setHeight();
```

执行一些语句作用的函数应该有一个能够描述函数功能的名字:

```
stripSpaces(txtstring);
replaceString(txtString, findText, replaceText);
```

5.9.2 函数文档

每个函数必须以如下所示的注释块作为函数的开始头:

- 这个函数的功能和用途;
- 函数是否需要参数或需要什么参数;
- 函数的返回值;
- 函数中已知的不完善的地方;
- 作者(与library的作者不同的时候);
- 代码改变时候的版本和历史信息.

```
/*
* ------------------------------------------------------------------
* FUNCTION : getLeft()
* ------------------------------------------------------------------
* DESCRIPTION : Returns a string containing a specified number of
* characters from the left side of a string.
* PARAMETERS :
* textString : Required - String from which the leftmost characters
* are returned
* len : Required - Number indicating how many characters to
* return
* OUTPUT : Left part of string
* ------------------------------------------------------------------
*/
function getLeft(textString, len) {
    if (len <= 0) {
        return "";
    }
    else if (len > String(textString).length) {
        return textString;
    }
    else {
        return String(textString).substring(0, len);
    }
}
```

5.10 变量

尽管JavaScript并不需要使用*var*语句定义变量, 但还是推荐使用这个语句在任何你定义变量的地方. 一个变量的名字应该以小写字母开头.紧接着的第二个单词的第一个字母应该大写.变量的名字有特定的意义以表示它的内容.

正确:

```
var clientName;
clientName = “John Smith”;

var clientAddress = “Street 22B”;
```

避免:

```
/* It is not clear where the variable has been declared */
myStringVariable1 = “some text”;
myStringVariable1 = myStringVariable1 + “some more text”;

/* It does not describes the possible value of the variable */
var b;

/* Second word should start with a capital */
var codeexample;
```

注意点: 字符串应该使用双引号而不是单引号:

正确:

```
myStringVariable1 = “some sentence with text”;
myStringVariable2 = “a sentence with \”quotation marks\””;
```

避免:

```
myStringVariable1 = ‘some sentence with text’;
myStringVariable2 = ‘a sentence with ”quotation marks”’;
```

6 空白空间

6.1 空行

按照代码的逻辑关系用空行分段改善了代码的可读性. 一个空行应该被应用在如下的场合:

- 函数之间;
- 函数中的本地变量和第一条语句之间;
- 一个注释块的前面;
- 一个代码块中适当的地方.

6.2 空格

空格应该被应用在如下的场合:

- 左括号前面;
- 左大括号前面,函数除外;
- 参数列表中的分号之后面;
- 在每个操作符的前面和后面，不包括点操作符 “.”;
- *for* 语句的每个表达式之间.

正确:

```
for (i = 0; i < 10; i++) {
    doSomething(argument1, argument2);
    a = b * c;
    textString = “my name is “ + name + ”.”;
}
```

7 独立于浏览器的 JavaScript

使用JavaScript最困难的地方在于生成相对于浏览器独立的script,尤其是对于动态的HTML scripts来说. 每一种浏览器都有各自的JavaScript的属性和版本. 看下表了解各种浏览器中所支持的JavaScript的版本.

浏览器	JavaScript 版本
Netscape 2	1.0
Netscape 3	1.1
Netscape 4 (<=4.05)	1.2
Netscape 4 (>4.05)	1.3
Netscape 6	1.4
MS Internet Explorer 2	None
MS Internet Explorer 3	1.0
MS Internet Explorer 4	1.2
MS Internet Explorer 5	1.3
MS Internet Explorer 5.5	1.4
America Online's Embedded Browser 3	1.0
America Online's Embedded Browser 4	1.2
America Online's Embedded Browser 5	1.3
Opera 4	1.2

Opera 5	1.3

8 Script library 例子

```
<!-- Hide from older browsers
    /*
    * ============================================================
    * S T R I N G S
    * ============================================================
    * AUTHOR : Joost H. van Rossum / Business Solutions
    * CREATION DATE: 01-JAN-2001
    * WORKFILE : stringScripts.js
    * DESCRIPTION : Scripts for manipulating and validating
    * textstrings.
    * COPYRIGHT : © 2001 Atos Origin All rights reserved
    * ============================================================
    * VERSION : 2
    * DATE : 10-JAN-2001
    * AUTHOR : Joost H. van Rossum / Business Solutions
    * DESCRIPTION : Documentation improvement, no code changes.
    * ============================================================
    */
    /*
    * ------------------------------------------------------------------------
    * FUNCTION : getLeft()
    * ------------------------------------------------------------------------
    * DESCRIPTION : Returns a string containing a specified number of
    * characters from the left side of a string.
    * PARAMETERS :
    * textString : Required - String from which the leftmost characters
    * are returned
    * len : Required - Number indicating how many characters to
    * return
    * OUTPUT : Left part of string
    * ------------------------------------------------------------------------
    */
    function getLeft(textString, len) {
        if (len <= 0) {
            return "";
        }
        else if (len > String(textString).length) {
            return textString;
        }
        else {
            return String(textString).substring(0, len);
        }
    }
    /*
    * ------------------------------------------------------------------------
    * FUNCTION : isEmpty()
    * ------------------------------------------------------------------------
```

```
    * DESCRIPTION : Returns a boolean that indicates whether the string
    * is empty or filled.
    * PARAMETERS :
    * textString : Required - The string to be validated
    * OUTPUT : True or false
    * -----------------------------------------------------------------------
    */
    function isEmpty(textString) {
        if (textString == null) {
            return true;
        }
        else if (textString.length > 0) {
            return false;
        }
        else {
            return true;
        }
    }
    /*
    * -----------------------------------------------------------------------
    * FUNCTION : trimLeft()
    * -----------------------------------------------------------------------
    * DESCRIPTION : Removes all spaces at the beginning of the string,
    * but it also removes tabs, line feeds, etc. at the
    * beginning of the string.
    * PARAMETERS :
    * textString : Required - The string to be trimmed left
    * OUTPUT : Left trimmed string
    * -----------------------------------------------------------------------
    */
    function trimLeft(textString) {
        var whiteSpace = new String(" \t\n\r");
        var newTextString = new String(textString);
        if (whiteSpace.indexOf(newTextString.charAt(0)) != -1) {
            var j = 0;
            while (whiteSpace.indexOf(newTextString.charAt(j)) != -1) {
                j++;
                newTextString = newTextString.substring(2);
            }
        }
        return newTextString;
    }

// Stop hiding -->
```

9 参考书目

- *Writing Robust Java Code* by Scott W. Ambler (AmbySoft), 2000
- *Java Code Conventions* by Scott Hommel (Sun), 1999
- *A Coding Style Guide for Java* by Achut Reddy (Sun), 1998
- *Client-Side JavaScript Guide 1.3* by Netscape, 1999