中软国际公司内部技术规范

JavaScript语言编程规范

中软国际科技服务有限公司

修订声明

参考: Google_JavaScript_编码规范指南，华为 JavaScript 编码规范等。

日期	修订版本	修订描述	作者
2014-08-15	1.0	整理创建初稿	徐庆阳+x00108368
2014-10-25	1.0	格式整理	徐庆阳+x00108368

目录

前言：

JavaScript 编程规范目的

1. 统一编程风格
2. 提高代码的可阅读性
3. 减少错误产生
4. 减少性能漏洞
5. 提高代码可靠性
6. 减少错误的编码设计
7. 作为代码检查的依据
8. 建立可维护的JavaScript语言编程规范

1. JavaScript 文件

1.1 文件引用

JavaScript 代码不应该被包含在HTML 文件中，除非这是段特定只属于此部分的代码。在HTML 中的JavaScript代码会明显增加文件大小，而且也不能对其进行缓存和压缩。filename.js 应尽量放到body 的后面。这样可以减少因为载入脚本而造成其他页面内容载入也被延迟的问题。禁止使用language 属性，必须使用type 属性。
示例：

```
<script type="text/javascript" src="filename.js"></script>
```

1.2 编码格式

分离于页面之外的js文件必须采用UTF-8无BOM格式编码，否则极其容易出现不可预知的乱码问题。维护已存在的js脚本文件必须把编码格式转换成规范要求的UTF-8无BOM格式。

2. 排版规则

2.1 缩进

每一级缩进采用4个空白符。不要使用Tab作为缩进方式，除非在编辑器里设置了自动将一个Tab转换为4个空白符。

2.2 空白

2.2.1 空行

适当的空行可以大大提高代码的可阅读性，可以使代码逻辑更清晰易懂。
相对独立的程序块之间、变量说明之后必须加空行。

示例：

```
if (a > b)
{
    doStart();
}
//此处是空行
return;
```

源程序中关系较为紧密的代码应尽可能相邻，便于程序阅读和查找。
示例：矩形的长与宽关系较密切，放在一起。

```
rect.length = 10;
rect.width = 5;
```

2.2.2 空格

在表达式中适当使用空格，会给代码的阅读带来方便。

关键字的后面如有括号，则最好在关键字和左括号'('之间留空白，如 for, if, while 等。而函数名和括号之间则不宜留空白，但若是匿名函数，则必须在 function 和左括号'('之间留空白，否则，编辑器会误认为函数名为 function。

在表达式中，二元运算符（除左括号'('，左方括号'['，作用域点'.'）和两个操作数之间最好留空白。一元运算符和其操作数之间不宜留空白。

逗号','的后面需要留空白，以显示明确的参数间隔，变量间隔等。

分号';'之后通常表明表达语句的结束，而应换行。在 for 的条件语句中，分号之后则应该留空白。

示例：

```
for (i = 0; i < 10; i++)
{
    doSomething();
}
a = function (e, d)
```

```
{
    return e + d;
};
```

2.3 换行

每行一个语句，除非这些语句有很密切的联系，否则每行只写一个语句。
示例：如下例子不符合规范。

```
var a; a = null;
```

2.3.1 较长的语句（>80 字符）

要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读。
示例：

```
if ((taskNo < maxActTaskNumber)
        && (statItemValid (statItem)))
{
    ... // program code
}
```

2.3.2 行结束

JavaScript 语句都使用英文分号(;)表示结束，不要使用换行符作为结束符，虽然大多数浏览器允许不写分号，但不便于阅读、维护。

3. 命名规范

约定类命名使用Pascal命名法，即用英文的大小写来分隔单词，所有单词的首字母大写；变量和方法都使用CamelI命名法，即第一个英文首字母小写，其余单词首字母大写。

3.1 变量命名

变量名使用意义完整的英文描述，采用 CamelI 命名法（全局变量除外）。
同时，变量的命名应遵循如下规则：
1） 全局变量应该使用全大写字母，单词间下划线间隔 var MAX_COUNT = 10
2） 不要将在循环中频繁使用的临时变量如 i, j 等用于其它用途
3） UI （用户界面）控制变量应在名称后加控制类型，例如： leftComboBox, TopScrollPane
4） 带有 "num" 或者 "count" 开头的变量名约定为数字
5） 能缩写的名称尽量使用缩写
6） 缩写词不要全部使用大写字母

7） 前面加 "is" 的变量名 应该为布尔值，同理可以为 "has", "can" 或者 "should"
8） 含有集合意义的属性命名，尽量包含其复数的意义，例如：customers

3.2 类命名

用Pascal命名规则，尽量谨慎的使用缩写，不要用下划线作类名单词连接符。类使用英文的大小写来分隔单词，所有单词的首字母大写。
示例：
Account
EventHandler
类/构造函数 可以使用扩展其基类的名称命名，这样可以正确、迅速的找到其基类的名称：
EventHandler
UIEventHandler
MouseEventHandler

3.2.1 类方法命名

普通方法使用意义完整的英文描述命名，采用 Camel 命名法。通常每个方法都是执行一个动作的，尽可能的采用动词和动宾结构。
示例：
obj.getSomeValue()
必须与new 共同使用的构造函数名应跟类名保持一致。当new 被省略时JavaScript 不会有任何编译错误或运行错误抛出。忘记加new 时会让不好的事情发生（比如被当成一般的函数），所以大写构造函数名是我们来尽量避免这种情况发生的唯一办法。

3.2.2 类属性命名

类的属性采用 camel 命名规则
示例：

```
function Account ()
{
    var _accountId = '';
    var databaseObject = '';
    this.showAccountId = function()
     {
         alert(this._ accountId);
     };
}
```

如果类属性设置为私有，则前面必须添加下划线。
示例：

```
this._somePrivateVariable = statement;
```

3.3 函数命名

函数名使用意义完整的英文描述，采用第一个单词的字母使用小写，剩余单词首字母大写其余字母小写的大小写混合法。

如：function disassociateIp(ip, instanceId)

4. 编码规范

4.1 变量

变量是程序的基础，规范的变量编码不仅能降低程序的出错率，更能增加代码的可读性，方便维护和扩展。

4.1.1 声明

所有的变量必须在使用前进行声明。JavaScript 并不强制必须这么做，但这么做可以让程序易于阅读，且也容易发现那些没声明的变量(它们会被编译成全局变量)。

最好把每个变量的声明语句单独放到一行，并加上注释说明，声明变量必须加上var关键字。

示例：

```
var currentEntry;      // 当前选择项
var level;             // 缩进程度
var size;              // 表格大小
```

4.1.2 初始化

变量必须在声明初始化以后才能使用，即便是 NULL 类型。

4.1.3 生命周期

变量应该尽量保持最小的生命周期

在函数的首部定义所有的变量。

尽量减少全局变量的使用，不要让局部变量覆盖全局变量。

4.1.4 浮点数

浮点变量必须指明小数点后一位（即使是 0）。

浮点变量必须指明实部，即使它们为零（使用 0. 开头）。

示例：

```
var floatNum = 0.0;
```

4.1.5 json 对象

Json 对象定义，应该补全双引号。

对于服务器返回的 JSON 数据，使用标准结构可以利用 Firefox 浏览器的 JSONView 插件方便查看（像查看 XML 那样树形显示），另外你如果使用 jQuery 做开发，最新版本 jQuery1.4+是对 JSON 格式有更高要求的，具体的可以自己查阅 jQuery 更新文档。

示例：

```
{name:"Tom"}
{'name':'Tom'}
```

应该改成：

```
{"name":"Tom"}
```

4.2 操作符

4.2.1 { }和[ ]

在 JavaScript 中，如需定义空对象和空数组，通常很自然地想到用 new Object() 和 new Array() 的方法。其实花括号'{}'和方括号'[]'可以直接用来定义一个空对象和一个空数组。这种书写方法可以使代码看起来简单易懂。

示例：

```
var aName = [];
var nameObj = {};
```

4.2.2 =,==和===

判断"逻辑等"在代码里太平常的不过事情了，但 JavaScript 与其他熟知的编程语言不同的是，除了可以使用两个等号'=='来作判断以外，还可以使用三个等号'==='来进行逻辑等判断。两者的不同是'=='作逻辑 等判断时，会先进行类型转换后再进行比较。'==='则不会。因而，'=='进行的判断结果可能产生偏差。'!='与'!=='的区别亦是如此。尽量使用'==='来进行逻辑等的判断，用'!=='进行逻辑不等的判断。

'='是赋值操作符,不能用于逻辑等。

示例：

```
var valueA = "1";
var valueB = 1;
```

```
if ( valueA == valueB)
 {
    alert("Equal");
 }
 else
 {
    alert("Not equal")
 }
 //output: "Equal"
 if ( valueA === valueB)
 {
    alert("Equal");
 }
 else
 {
    alert("Not equal")
 }
 //output: "Not equal"
```

说明：valueA 和 valueB 两个变量的值显然是不相等的，起码 valueA 是个字符串，而 valueB 是一个数字。但用'=='进行判断时，程序却输出相等的字样。这是因为编译器对两个变量进行比较时，因为他们的类型不同，而自动地将 valueB 转换成字符串，而后再和 valueA 进行比较的。用'==='得到的判断结果正和预期的结果相符。

4.2.3 ? :

程序员往往试着在 ? 和 : 之间塞满了许多的代码，这是不规范的。

以下的是一些清晰的连接规则:

把条件放在括号内以使它和其他的代码相分离；

如果可能的话，动作可以用简单的函数；

把所做的动作，“?”，“:”放在不同的行，除非他们可以清楚的放在同一行。

示例:

```
(condition) ? funct1() : func2();

(condition)
? long statement
: another long statement;
```

4.2.4 巧用'+'

JavaScript 和其他编程语言不同的是，在 JavaScript 中，'+'除了表示数字值相加，字符串相连接以外，还可以作一元运算符用，把字符串转换为数字。因而如果使用不当，则可能与自增符'++'混淆而引起计算错误。

示例：

```
var valueA = 20;
var valueB = "10";
alert( valueA + valueB);          //ouput: 2010
alert( valueA + (+valueB));       //output: 30
alert( valueA ++valueB);          //Compile error
```

4.3 语句

4.3.1 简单语句

对于简单语句而言，需要提及的仍然是分号必要性，同时，一行最多有一个语句。如果一个赋值语句是用函数和对象来赋值，可能需要跨多行，一定切记要在赋值语句末加上分号。这是因为 JavaScript 中，所有表达式都可以当语句，遇换行符时会解析为表达式的结束，此时不规范的换行和分号的丢失，可能引入新的错误。

4.3.2 复杂语句

对于复合语句，if, for, while, do, switch, try … catch 等代码体，函数定义的函数体，对象的定义等都需要放在花括号'{}'里面。

- '{' 统一另起一行，放在行首，标志代码块的开始。
- '}' 应在一行开头，标志代码块的结束，同时需要和'{'所在行的开始对齐，以表明一个完整的复合语句段。这样可以极大地提高代码的可阅读性，控制逻辑能清晰地表现出来。
- 被包含的代码段应该再缩进 4 个空格。
- 即使被包含的代码段只有一句，也应该用花括号'{}'包含。尽管不用花括号代码也不会错，但如若需要增加语句的话，则较容易因花括号遗漏而引起的编译错误或逻辑错误。

虽然把左花括号{ 另起一行，放在行首会造出空间浪费，但发布版本的 js 文件都经过压缩后会消除掉。

示例：

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

```
{
    Statements;
}

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

不像别的复合语句,do 语句总是以;(分号)结尾。

```
switch (expression)
{
    case expression:
        statements;
        break;
    default:
        statements;
        break;
}
```

每一组 statements 应以 break,return,或者 throw 结尾。不要让它顺次往下执行。

4.3.3 return 语句

一条有返回值的return 语句不要使用“()”（括号）来括住返回值。如果返回表达式，则表达式应与return关键字在同一行，以避免误加分号错误。

return 关键字后若没有返回表达式，则返回 undefined。构造器的默认返回值为 this。

示例：

```
function F1()
{
   var valueA   = 1;
   var valueB   = 2;
   return valueA + valueB;
 }
 function F2()
 {
   var valueA   = 1;
   var valueB   = 2;
   return
       valueA + valueB;
 }
 alert( F1() );   //output: 3
 alert( F2() );   //output: undefined
```

4.3.4 for/for-in 循环

for-in 循环只用于object/map/hash 的遍历，而遍历数组通常用最普通的for 循环。另外，避免在for循环中使用关键字continue ,使用条件语句来做相关处理。

4.3.5 with 语句

不要使用with()。 使用 with 让你的代码在语义上变得不清晰。因为 with 的对象，可能会与局部变量产生冲突，从而改变你程序原本的用义。

4.3.6 不要使用生偏语法

JavaScript 作为一门动态脚本语言，灵活性既是优点也是缺点，众所周知，动态语言不同层次开发人员对实现同样一个功能写出来的代码在规范或语法上会存在较大的差别。不管怎么样，规范编码少搞怪，不把简单问题复杂化，不违反代码易读性原则才是大家应该做的。

4.4 函数

4.4.1 函数申明

函数也应在调用前进行声明，内部函数应在 var 声明内部变量的语句之后声明，可以清晰地表明内部变量和内部函数的作用域。此外，函数名紧接左括号'('，而右括号')'和后面的'{'之间要换行，以清楚地显示函数名以其参数部分，和函数体的开始。若函数为匿名 / 无名函数，则 function 关键字和左括号'('之间要留空格，否则可能误认为该函数的函数名为function。

示例：

```
var innerA = 1;
function outF()
{
    var innerA = 2;
    function _inF()
    {
        alert("valueA="+innerA);
    }
    _inF();
}
outF();                          //output: valueA=2
_inF();                          //error: innerF is not defined
```

说明：_inF() 函数仅在 outF() 函数的内部生效，局部变量 innerA 对内部函数的作用域生效。这样的编码方式使得变量和函数的作用域变得清晰。

4.4.2 函数参数

应该要明确定义函数固定数量的参数。

固定数量参数的函数内部不使用 arguments 去获取参数，因为这样，你定义的方法如果包含较多的脚本，就不能一眼看到这个方法接受些什么参数。

示例：如下例子不符合规范

```
var elemObj = function()
{
    return document.getElementById(arguments[0]);
}
```

应该改成：

```
var elemObj = function(elemID)
{
    return document.getElementById(elemID);
}
```

4.4.3 函数返回值

一个函数尽量返回统一的数据类型。(建议)

示例：

```
function getUserName(userID)
{
  if (data[userID])
  {
    return data[userID];
  }
  else
  {
    return false;
  }
}
```

应该改为：

```
function getUserName(userID)
{
  if (data[userID])
  {
    return data[userID];
  }
  else
  {
```

```
        return "";
    }
}
```

4.4.4 eval 是恶魔

eval 是 JavaScript 中最容易被滥用的方法。避免使用它。

在把变量 json 化时，请记得先把 JSON 字符串值用‘()’括号先括起来。

示例：

```
var jsonStr2 = '{"Name":"Tom","Sex":"Man"}';
var jsonObj2 = eval('(' + jsonStr2 + ')');
alert(jsonObj2.Name);
```

4.4.5 闭包

小心使用闭包，一定要记住闭包中局部变量是传引用，不是传值。闭包也许是 JS 中最有用的特性了。有一点需要牢记，闭包保留了一个指向它封闭作用域的指针，所以，在给 DOM 元素附加闭包时，很可能会产生循环引用，进一步导致内存泄漏。比如下面的代码：

```
function foo(element, a, b)
{
    element.onclick = function()
    {
        /* uses a and b */
    };
}
```

这里，即使没有使用 element，闭包也保留了 element, a 和 b 的引用。由于 element 也保留了对闭包的引用，这就产生了循环引用，这就不能被 GC 回收。这种情况下，可将代码重构为：

```
function foo(element, a, b)
{
    element.onclick = bar(a, b);
}
function bar(a, b)
{
    return function()
    {
        /* uses a and b */
    }
}
```

4.4.6 setTimeout/ setInterval

不要给 setTimeout 或者 setInterval 传递字符串参数。javascript 会把你传入的字符串参数解析执行

4.5 其他规范

4.5.1 禁止修改内置对象的原型

内置对象作为一套公共接口，具有约定俗成的行为方式，修改其原型，可能破坏接口语义。
错误示例：

```
Array.prototype.indexOf = function () {return -1}

// somewhere else
var o = [1, 1, 1, 1, 1, 2, 1, 1, 1];
alert(o.indexOf(2)); // 返回-1
```

4.5.2 降低与 XHTML 的耦合性

不要过于依赖 DOM 的一些内容特征来调用不同的脚本代码，而应该定义不同功能的方法，然后在 DOM 上调用，这样不管 DOM 是按钮还是链接，方法的调用都是一样的。

示例：

```
function myBtnClick(obj)
{
   if (/确定/.test(obj.innerHTML))
      alert('OK');
   else if (/取消/.test(obj.innerHTML))
      alert('Cancel');
   else
      alert('Other');
}

<a herf="javascript:;" onclick="myBtnClick(this)">确定</a>

<a herf="javascript:;" onclick="myBtnClick(this)">取消</a>
```

上面例子其实在一个函数内处理了两件事情，应该分成两个函数，像上面的写法，如果把链接换成按钮，比如改成这样：<input type="button" onclick="myBtnClick(this)" value="确定" />，

那么 myBtnClick 函数内部的 obj.innerHTML 就出问题了，因为此时应该 obj.value 才对，另外如果把按钮名称由中文 改为英文也会出问题，所以这种做法问题太多了。

4.5.3 变量与恒量相比

表达式或者变量与恒量相比时，总是将恒量放在等号/不等号的左边。假如你在等式中漏了一个等号，语法检查器会为你报错。能立刻找到数值而不是在你的表达式的末端找到它。
示例：

```
if ( null    ==    errorNum ) {
      doSomething();
}
```

4.5.4 没有含义的数字

尽量避免裸裸的数字，他们应该使用常量来代替。

4.5.5 页面卸载、局部刷新调整时，要注销相应的定时器和延时器

定时器和延时器容易造成JS 对DOM的引用，在JS对象本身没有被释放的情况下容易引起DOM对象的泄漏。在页面局部刷新时必须清理响应的定时器，否则会造成多个定时器同时执行的业务问题和每个定时器对应闭包所引用的内存无法释放的问题。
错误代码：

```
var intervalID = setInterval(fn, timeout);
```

正确代码：

```
var intervalID = setInterval(fn, timeout);
…
//清理函数中要对定时器进行销毁
if (intervalID)
{
      clearInterval(intervalID);
}
```

4.5.6 页面卸载时要注销绑定的事件监听

说明：通常事件监听都是与DOM对象挂钩的，当事件监听没有被注销时，可能会造成JS对象和DOM对象之间存在引用关系，当JS对象或者DOM对象没有释放时，也会造成另外一方无法释放，尤其在大量使用闭包的情况下，情况会比较复杂，容易引入事件回调闭包造成JS对象和DOM的循环引用，使用闭包造成的循环引用在IE、firefox上会造成内存泄漏，原因是由于IE和Firefox都使用引用计数方式的垃圾回收算法。

动态删除的DOM对象在其销毁时也要注销绑定的事件监听。
错误示例：没有事件的注销动作，在IE或者firefox上会造成循环引用 button 这个DOM对象和 onButtonClick 这个闭包的循环引用。

```
$(function() {
    var btn = $('<button id="btn">toggle target</button>');
    btn.appendTo($(document.body));
    btn.on('click', function onButtonClick() {
        $('#target').toggle();
    });
    btn = null;
});
```

推荐做法：

```
$(function() {
    var btn = $('<button id="btn">toggle target</button>');
    btn.appendTo($(document.body));
    btn.on('click', function() {
        $('#target').toggle();
    });
    btn = null;
});
$(window).on('unload', function() {
    //添加销毁动作，并注销事件。
    $('#btn').off();
});
```

4.5.7 业务事件在业务销毁时注销

错误示例：

```
//注册API事件回调
$.addEventListener('com.huawei.oms.fm.event.alarmSound', modifySound);
```

推荐做法：

```
//注册API事件回调
$.addEventListener('com.huawei.oms.fm.event.alarmSound', modifySound);
//在销毁时注销业务事件
$.removeEventListener('com.huawei.oms.fm.event.alarmSound', modifySound)
```

4.5.8 避免嵌入式的赋值

有时候在某些地方我们可以看到嵌入式赋值的语句，那些结构不是一个少冗余，可读性强的好方法。
示例：

```
while (a !== (b = c + d))
{
    doSomething();
}
```

++和--操作符类似于赋值语句。因此，出于许多的目的，在使用函数的时候会产生副作用。使用嵌入式赋值提高运行时性能是可能的。

无论怎样，程序员在使用嵌入式赋值语句时需要考虑在增长的速度和减少的可维护性两者间加以权衡。

示例:

```
a = b + c;
d = a + r;
```

不要写成:

```
d = (a = b + c) + r;
```

虽然后者可以节省一个周期。但在长远来看，随着程序的维护费用渐渐增长，程序的编写者对代码渐渐遗 忘，就会减少在成熟期的最优化所得。

4.5.9 删除不再使用的代码段

当代码调整或重构后，之前编写的不再使用的代码应该及时删除，如果认为这些代码还有一定利用价值可以把它们剪切到临时文件中。留在项目中不仅增加了文件体 积，这对团队其它成员甚至自己都起到一定干扰作用，怕将来自己看回代码都搞不懂这方法是干什么的，是否有使用过。当然可以用文档注释标签 @deprecated把这个方法标识为不推荐的。

4.6 陷阱和技巧

4.6.1 parseInt

parseInt方法必须指定基数。如果不指定parseInt的第二个参数，如果前缀为“0x”，按16进制转换；对前缀为“0”的字符串，在chrome和firefox下，默认使用10进制转换，但在IE浏览器下按照8进制转换。

错误示例：

```
parseInt('011')    //chrome下返回11；IE下返回9
```

推荐做法：

```
parseInt('011',  10)
```

4.6.2 布尔表达式

建议：在需要判断某变量不是0，空串或者false时，使用if(x)。JavaScript是弱类型的，在布尔表达式中，会转换类型为Boolean类型进行判断。

0

-0
null
“” 空串
false
undefined
NaN
对于上面7个值，if (x)返回false。

‘0’ 字符串0
[] 空数组
{} 空对象
对于上面3个值，返回的是true。

错误示例：
if (x != null) {...} //尽量不使用，可能判断条件不完整
推荐做法：
if (x) {...}

4.6.3 DOM 节点 ID

DOM节点的ID命名使用英文字母、_和数字，避免使用.< >等其他特殊字符。DOM节点的ID常会用作CSS等的选择器，特殊字符可能会与选择器的语法冲突。
错误示例：

```
<div id="eview.div"></div>
```

4.6.4 遍历 NodeList

NodeList是通过给节点迭代器加一个过滤器来实现的，这表示获取他的属性，如length的时间复杂度为O(n)，通过length来遍历整个列表需要O(n^2)。
错误示例：

```
var paragraphs = document.getElementsByTagName('p');
for (var i = 0; i < paragraphs.length; i++)
{
    doSomething(paragraphs[i]);
}
```

推荐做法：

```
var paragraphs = document.getElementsByTagName('p');
for (var i = 0, paragraph; paragraph = paragraphs[i]; i++)
{
    doSomething(paragraph);
}
```

或者：

```
var parentNode = document.getElementById('foo');
```

```
for (var child = parentNode.firstChild; child; child = child.nextSibling)
{
    doSomething(child);
}
```

4.6.5 遗漏的参数

函数的可选参数或者新增参数设置默认值。说明：如果你需要在已经被调用的函数中增加一个参数来处理一个特殊情况下的调用，请给这个函数中的这个参数设置默认值，以防万一在众多脚本中的众多调用中的一个忘记更新。

推荐做法：

```
// country是新增的参数
function addressFunction(country)
{
    country = country || “US”; //如果没有传入country,给出默认值“US”
    //剩下代码
}
```

4.6.6 HTML id 冲突

尽量不要使用Html中的Id作为JavaScript的变量名。在JavaScript中函数和属性共享同一个名字空间。所以，当在HTML中的一个id和函数或属性有相同的名字时，你会得到难以跟踪的逻辑错误。

错误示例：

```
<html>
<head></head>
<body>
    <ul>
        <li id="length">1</li>
        <li id="thisLength">2</li>
        <li id="thatLength">3</li>
    </ul>
</body>
<script>
var listitems = document.getElementsByTagName('li');
var liCount = listitems.length; //IE下返回的是<li id="length">1</li>这个节点而不是所有<li>的数量
var thisLength = document.getElementById('thisLength');
thatLength = document.getElementById('thatLength');
//IE下会出现“对象不支持此属性和方法”的错误，IE8 beta2下首次加载页面会出错，刷新页面则不会
//在IE中thisLength和thatLength直接表示以其为id值的DOM节点，
//所以赋值时会出错，当有var声明时，IE会把其当着变量，这个时候就正常了。
</script>
</html>
```

5. 注释规则

5.1 单行注释

有三种使用单行注释的方法：
1. 单独成行，解释该注释后面一行代码的意思，'//'后注释内容前总是存在一个空行，注释跟代码处于同一缩进下；
2. 在一行代码后进行注释，代码与注释间至少有一级缩进。注释不应该超出行最大长度，否则将注释挪到代码前单独成行；
3. 用于注释掉大段的代码
例子如下：

```
if (condition)
{
    // 注释前空行，与代码同级缩进
    behavior();
}

var result = "Hello" + " World";    // 至少间隔一级缩进

// 多个单行注释用来注释掉大段代码
// if (condition)
// {
//     behavior();
//     anotherBehavior();
// }
```

5.2 多行注释

多行注释就是/* 这里是注释内容 */，可以跨越多行：

```
/*
 * Java版本
 * 的多行注释
 * 最少有三行，/*，内容以及 */各一行
 */
```

多行注释总是出现在需要解释的代码前，与单行注释一样，需要与前面的代码通过一个空隔开，并与要解释的代码处于同一级缩进：

```
if (condition)
{
    /*
     * 与代码处于同一级缩进，并与前面代码通过空行隔开
     */
    behavior();
}
```

注意多行注释不能出现在同行代码后：

```
var result = "Hello" + " World"; /* 这样注释是不对的 */
```

5.3 一般规则

- 当代码不清晰时使用注释，用它们来阐明代码的意图，而不要在做无谓的注释。

```
// 这是一个坏例子，注释无助于理解count意思
// 初始化count
var count = 10;

// 好例子，解释了count的用处
// 改变count的值会变更打印次数
var count = 10;
```

- 不容易理解的代码总是需要注释。
- 容易被其他开发人员误认为是错误的，这些代码需要加上注释，比如说一些技巧性的用法：

```
while (element && (element = element[axis])) // 注意：这里是赋值不是比较
{
    // 这里赋值 = 容易被认为是少写了一个=号的==，所以需要注释
    behavior();
}
```

- 特定浏览器专用的用法需要被注释

5.4 函数注释

函数声明时要在开头说明其实现功能、各参数、返回值意义，复杂逻辑要在声明时说明其实现思想，并在关键步骤做出注释。调用方法时也要指出其目的。

示例：

```
/**
 * 交换数组两个值
 * @param {Array} array 操作原数组对象
 * @param {Number} fromIndex 交换对象的数组下标
 * @param {Number} toIndex  被交换对象的数组下标
 * @return {Array} 新数组
 */
 function arraySwap(array,  fromIndex,  toIndex )
{
    var temp  =  array[fromIndex];
    array[fromIndex]  =  array[toIndex];
    array[toIndex]  =  temp;
    return  array;
 }
```

5.5 失效代码

当一段代码不再有效，但这段代码对于阅读者理解程序有重大意义，可以将其注释掉

示例：

```
// [操作者] 年月日代码失效原因 START
// ……
// ……
// [操作者] 年月日代码失效原因 END
```

5.6 版权信息

示例：

```
/**
 * Title: 页面信息（如：新闻建立页面）
 * Description: 描述信息（如：建立和修改新闻信息）
 * Copyright: Copyright (c) 2011-2012 huawei All Rights Reserved
 * Nameplace: AddNewDetailInfo
 * @author : 作者姓名
 * @version : 1.00
 */
```

6. 浏览器兼容问题

6.1 为了获得最大的可移植性和兼容性，尽量依赖标准方法

说明：尤其对于开放式的系统，我们不能约束系统使用者的运行环境，要充分考虑兼容性问题

常见问题	问题描述	解决办法
HTML 对象获取问题	FireFox/IE8:document.getElementById("idName"); ie(非 IE8):document.idname 或者 document.getElementById("idName")	统一使用 document.getElementById("idName") 或 jquery:$(selector)
const 问题	说明:Firefox 下,可以使用 const 关键字或 var 关键字来定义常量; IE 下,只能使用 var 关键字来定义常量.	统一使用 var 关键字来定义常量.
event.x 与 event.y 问题	IE 下,event 对象有 x,y 属性,但是没有 pageX,pageY 属性; Firefox 下,event 对象有 pageX,pageY 属性,但是没有 x,y 属性.	使用 mX(mX=event.x?event.x:event.pageX;)来代替 IE 下的 event.x 或者 Firefox 下的 event.pageX
window.location.href 问题	IE 或者 Firefox2.0.x 下,可以使用 window.location 或 window.location.href; Firefox1.5.x 下,只能使用 window.location	使用 window.location 来代替 window.location.href
访问 frame 对象	IE:使用 window.frameId 或者 window.frameName 来访问这个 frame 对象.frameId 和 frameName 可以同名。 Firefox:只能使用 window.frameName 来访问这个 frame 对象	另外，在 IE 和 Firefox 中都可以使用 window.document.getElementById("frameId")来访问这个 frame 对象
切换 frame 内容	在 IE 和 Firefox 中都可以使用 window.document.getElementById("testFrame").src="xxx.html" 或 window.frameName.location="xxx.html" 来切换 frame 的内容	使用 window.frameName 或 window.document.getElementById("frameId")
frame 参数回传父窗口	如果需要将 frame 中的参数传回父窗口(注意不是 opener,而是 parentframe)，可以在 frame 中使用	例 如 :parent.document.form1.filena

	parent 来访问父窗口	me.value="Aqing";
模态和非模态窗口问题	IE 下，可以通过 showModalDialog 和 showModelessDialog 打开模态和非模态窗口;Firefox 下则不能	直接使用 window.open(pageURL,name,parameters)方式打开新窗口。 如果需要将子窗口中的参数传递回父窗口，可以在子窗口中使用 window.opener 来访问父窗口. 如 :var parWin=window.opener;parWin.document.getElementById("Aqing").value="Aqing";
firefox 与 IE 的父元素(parentElement)的区别	IE:obj.parentElement firefox:obj.parentNode	因为 firefox 与 IE 都支持 DOM,因此使用 obj.parentNode 是不错选择,或者使用 jquery:$(selector).parent()
document.formName.item("itemName")问题	IE 下，可以使用 document.formName.item("itemName") 或 document.formName.elements["elementName"]; Firefox 下，只能使用 document.formName.elements["elementName"]	统一使用 document.formName.elements["elementName"]
集合类对象问题	IE 下，可以使用()或[]获取集合类对象; Firefox 下，只能使用[]获取集合类对象	统一使用[]获取集合类对象
自定义属性问题	IE 下,可以使用获取常规属性的方法来获取自定义属性，也可以使用 getAttribute()获取自定义属性; Firefox 下，只能使用 getAttribute()获取自定义属性。	统一通过 getAttribute()获取自定义属性，或使用 Jquery:$(selector).attr();
input.type 属性问题	IE 下 input.type 属性为只读；但是 Firefox 下 input.type 属性为读写。	不修改 input.type 属性。如果必须要修改，可以先隐藏原来的 input，然后在同样的位置再插入一个新的 input 元素
event.srcElement 问题	IE 下,even 对象有 srcElement 属性,但是没有 target 属性; FF 下,even 对象有 target 属性,但是没有 srcElement 属性。	使用 srcObj=event.srcElement?event.srcElement:event.target;
body 载入问题	Firefox 的 body 对象在 body 标签没有被浏览器完全读入之前就存在; 而 IE 的 body 对象则必须在 body 标签被浏览器完全读入之后才存在。	[注]这个问题尚未实际验证，待验证后再来修改。 [注]经验证，IE6、Opera9 以及 FireFox2 中不存在上述问题，单纯的 JS 脚本可以访问在脚本之前已经载入的所有对象和元素，即使这个元素还没有载入完成。 这个地方最好的解决办法是定义一下 body 加载完之后在去执行的 JS 方

		法或者脚本。 1、$(function(){...}); 2 、 $(document).ready(){function(){ ...}};
事件委托方法	IE 下，使用 document.body.onload=inject;其中 function inject()在这之前已被实现；在 Firefox 下，使用 document.body.onload=inject();	统一使用 document.body.onload=newFunction('inject()'); 或者 document.body.onload=function(){/*这里是代码*/}, 或使用 jquery: $(function(){}). [注意]Function 和 function 的区别

Table 操作问题	IE、FF 以及其它浏览器对于 table 标签的操作都各不相同，在 ie 中不允许对 table 和 tr 的 innerHTML 赋值，使用 js 增加一个 tr 时，使用 appendChild 方法也不管用。	//向 table 追加一个空行:
		varrow=otable.insertRow(-1);
		varcell=document.createElement("td");cell.innerHTML="";
		cell.className="XXXX";
		row.appendChild(cell);
		[注]由于俺很少使用 JS 直接操作表格，这个问题没有遇见过。建议使用 JS 框架集来操作 table，如 JQuery
		$(selector).append()
		对 table 的操作最简单快捷的方式是将可能要分开的行分出来单独做一个表格出来，这样便于操作

对象宽高赋值问题	FireFox 中类似 obj.style.height=imgObj.height 的语句无效，应使用$(selector).height();	应使用 $(selector).height();
js 获取对象		建议统一使用 $(selector);
js 获取对象的高宽		$(selector).width(); $(selector).height(); $(selector).innerWidth(); $(selector).innerHeight(); $(selector).outerWidth(boolean); $(selector).outerHeight(boolean);
js 获取对象的属性		$(selector).attr(“attribute”) //这里 id 与 name 属性比较特殊可以直接将 jquery 对象转换成 dom 对象获取，效率更高 --$(selector)[0].id, $(selector)[0].name

js 设置对象的属性		$(selector).attr({ Id:id, Name:name, "class":className, Title:title, …. "data-"+attribute : ?? //自定义属性前缀为"data-" ….});
js 获取对象的样式		获取对象的样式 $(selector).css(“background”)
js 设置对象的样式		$(selector).css({ Position: "absolute", Background: "#333333", MarginLeft: "3px", "z-index": 11234 })

js 获取对象的物理位置		var offset = $(selector).offset();
		var offLeft = offset.left;//相对页面 X 轴位移
		var offTop = offset.top;//相对页面 Y 轴位移
		var position = $(selector).position();
		var positionLeft = position.left;//相对定位的 X 轴位移
		var positionTop = position.top;//相对定位的 Y 轴位移
		可以这样理解：
		offset 可以理解为相对于根目录的坐标位置
		position 是相对于他父级的坐标位置

js 事件源位移		var e = event || window.event;
		var x = e.pageX;
		var y = e.pageY;
		以上几种位移可以实现任何拖拽,滚动等动画.