

JavaScript 使用误区

本章节我们将讨论 JavaScript 的使用误区。

赋值运算符应用错误

在 JavaScript 程序中如果你在 if 条件语句中使用赋值运算符的等号 (=) 将会产生一个错误结果, 正确的方法是使用比较运算符的两个等号 (==)。

if 条件语句返回 **false** (是我们预期的)因为 x 不等于 10:

```
var x = 0;  
if (x == 10)
```

[尝试一下 »](#)

if 条件语句返回 **true** (不是我们预期的)因为条件语句执行为 x 赋值 10, 10 为 true:

```
var x = 0;  
if (x = 10)
```

[尝试一下 »](#)

if 条件语句返回 **false** (不是我们预期的)因为条件语句执行为 x 赋值 0, 0 为 false:

```
var x = 0;  
if (x = 0)
```

[尝试一下 »](#)

赋值语句返回变量的值。

比较运算符常见错误

在常规的比较中, 数据类型是被忽略的, 以下 if 条件语句返回 true :

```
var x = 10;  
var y = "10";  
if (x == y)
```

[尝试一下 »](#)

在严格的比较运算中, === 为恒等计算符, 同时检查表达式的值与类型, 以下 if 条件语句返回 false :

```
var x = 10;  
var y = "10";
```

```
if (x === y)
```

[尝试一下 »](#)

这种错误经常会在 switch 语句中出现，switch 语句会使用恒等运算符(===)进行比较：

以下实例会执行 alert 弹窗：

```
var x = 10;
switch(x) {
  case 10: alert("Hello");
}
```

[尝试一下 »](#)

以下实例由于类型不一致不会执行 alert 弹窗：

```
var x = 10;
switch(x) {
  case "10": alert("Hello");
}
```

[尝试一下 »](#)

加法与连接注意事项

加法是两个数字相加。

连接是两个字符串连接。

JavaScript 的加法和连接都使用 + 运算符。

接下来我们可以通过实例查看两个数字相加及数字与字符串连接的区别：

```
var x = 10 + 5;           // x 的结果为 15
var x = 10 + "5";         // x 的结果为 "105"
```

[尝试一下 »](#)

使用变量相加结果也不一致：

```
var x = 10;
var y = 5;
var z = x + y;             // z 的结果为 15

var x = 10;
var y = "5";
var z = x + y;             // z 的结果为 "105"
```

[尝试一下 »](#)

浮点型数据使用注意事项

JavaScript 中的所有数据都是以 64 位浮点型数据(float) 来存储。

所有的编程语言，包括 JavaScript，对浮点型数据的精确度都很难确定：

```
var x = 0.1;
var y = 0.2;
var z = x + y           // z 的结果为 0.3
if (z == 0.3)           // 返回 false
```

尝试一下 »

为解决以上问题，可以用整数的乘除法来解决：

实例

```
var z = (x * 10 + y * 10) / 10;           // z 的结果为 0.3
```

尝试一下 »

更多内容可以参考：[JavaScript 中精度问题以及解决方案](#)

JavaScript 字符串分行

JavaScript 允许我们在字符串中使用断行语句：

实例 1

```
var x =
"Hello World!";
```

尝试一下 »

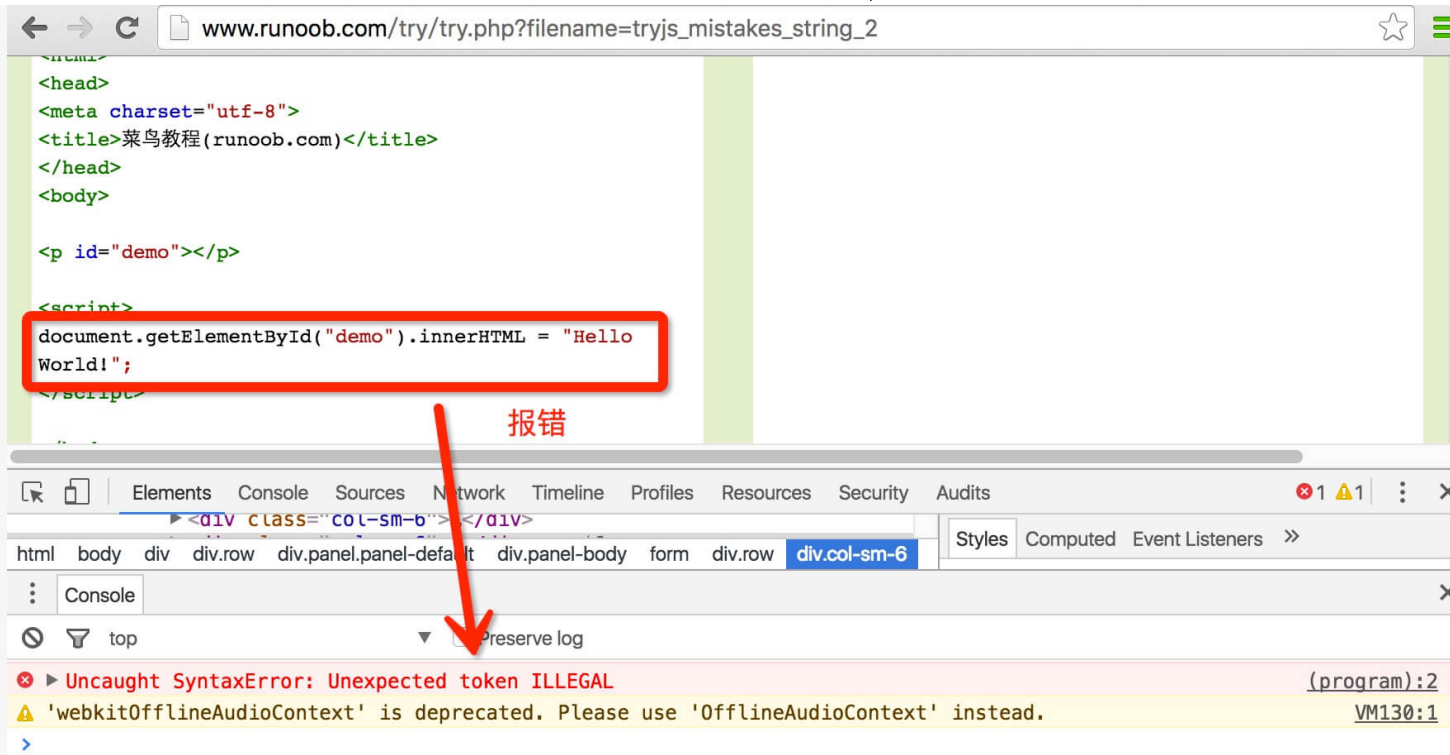
但是，在字符串中直接使用回车换行是会报错的：

实例 2

```
var x = "Hello
World!";
```

尝试一下 »

我们可以在选择开发工具或按下 F12 来查看错误信息：



报错

```
<script>
document.getElementById("demo").innerHTML = "Hello
World!";
</script>
```

Uncaught SyntaxError: Unexpected token ILLEGAL (program):2

'webkitOfflineAudioContext' is deprecated. Please use 'OfflineAudioContext' instead. VM130:1

字符串断行需要使用反斜杠(\)，如下所示:

实例 3

```
var x = "Hello \
World!";
```

尝试一下 »

错误的使用分号

以下实例中，if 语句失去方法体，原 if 语句的方法体作为独立的代码块被执行，导致错误的输出结果。

由于分号使用错误，if 语句中的代码块就一定会执行：

```
if (x == 19);
{
    // code block
}
```

尝试一下 »

Return 语句使用注意事项

JavaScript 默认是在代码的最后一行自动结束。

以下两个实例返回结果是一样的(一个有分号一个没有):

实例 1

```
function myFunction(a) {  
  var power = 10  
  return a * power  
}
```

[尝试一下 »](#)

实例 2

```
function myFunction(a) {  
  var power = 10;  
  return a * power;  
}
```

[尝试一下 »](#)

JavaScript 也可以使用多行来结束一个语句。

以下实例返回相同的结果：

实例 3

```
function myFunction(a) {  
  var  
  power = 10;  
  return a * power;  
}
```

[尝试一下 »](#)

但是，以下实例结果会返回 **undefined**：

实例 4

```
function myFunction(a) {  
  var  
  power = 10;  
  return  
  a * power;  
}
```

[尝试一下 »](#)

为什么会有这样的结果呢？因为在 JavaScript 中，实例 4 的代码与下面的代码一致：

```
function myFunction(a) {  
  var  
  power = 10;  
  return;      // 分号结束，返回 undefined
```

```
a * power;  
}
```

解析

如果是一个不完整的语句，如下所示：

```
var
```

JavaScript 将尝试读取第二行的语句：

```
power = 10;
```

但是由于这样的语句是完整的：

```
return
```

JavaScript 将自动关闭语句：

```
return;
```

在 JavaScript 中，分号是可选的。

由于 return 是一个完整的语句，所以 JavaScript 将关闭 return 语句。



注意：不用对 return 语句进行断行。

数组中使用名字来索引

许多程序语言都允许使用名字来作为数组的索引。

使用名字来作为索引的数组称为关联数组(或哈希)。

JavaScript 不支持使用名字来索引数组，只允许使用数字索引。

实例

```
var person = [];  
person[0] = "John";  
person[1] = "Doe";  
person[2] = 46;  
var x = person.length;      // person.length 返回 3  
var y = person[0];          // person[0] 返回 "John"
```

尝试一下 »

在 JavaScript 中，**对象** 使用 **名字作为索引**。

如果你使用名字作为索引，当访问数组时，JavaScript 会把数组重新定义为标准对象。

执行这样操作后，数组的方法及属性将不能再使用，否则会产生错误：

实例

```
var person = [];  
person["firstName"] = "John";  
person["lastName"] = "Doe";  
person["age"] = 46;  
var x = person.length;           // person.length 返回 0  
var y = person[0];               // person[0] 返回 undefined
```

尝试一下 »

定义数组元素，最后不能添加逗号

添加逗号虽然语法没有问题，但是在不同的浏览器可能得到不同的结果。

```
var colors = [5, 6, 7,]; //这样数组的长度可能为3 也可能为4。
```

正确的定义方式：

```
points = [40, 100, 1, 5, 25, 10];
```

定义对象，最后不能添加逗号

错误的定义方式：

```
websites = {site:"菜鸟教程", url:"www.runoob.com", like:460,}
```

正确的定义方式：

```
websites = {site:"菜鸟教程", url:"www.runoob.com", like:460}
```

Undefined 不是 Null

在 JavaScript 中，**null** 用于对象，**undefined** 用于变量，属性和方法。

对象只有被定义才有可能为 **null**，否则为 **undefined**。

如果我们想测试对象是否存在，在对象还没定义时将会抛出一个错误。

错误的使用方式：

```
if (myObj !== null && typeof myObj !== "undefined")
```

正确的方式是我们需要先使用 `typeof` 来检测对象是否已定义：

```
if (typeof myObj !== "undefined" && myObj !== null)
```

程序块作用域

在每个代码块中 JavaScript 不会创建一个新的作用域，一般各个代码块的作用域都是全局的。

以下代码的变量 `i` 返回 10，而不是 `undefined`：

实例

```
for (var i = 0; i < 10; i++) {  
    // some code  
}  
return i;
```

[尝试一下 »](#)[← JavaScript 严格模式\(use strict\)](#)[JavaScript 表单 →](#)**2 篇笔记**[✎ 写笔记](#)

虽然在 JavaScript 中，分号是可选的。

但是要注意 **return** 的用法：

这样的语句是完整的：

```
return
```

执行时 JavaScript 将自动关闭语句：

```
return;
```

由于 `return` 是一个完整的语句，所以 JavaScript 将关闭 `return` 语句。

所以不用对 `return` 语句进行断行。如下实例：

```
return  
ture;  
//JavaScript会解析成：  
return ; true;  
//而代码本意是这样的：  
return true;
```

拱猪的白菜 2年前 (2017-09-05)



JavaScript 中，分号是可选的,在缺少了分号就无法正确解析代码的时候，JavaScript 才会填补分号。

```
var a
a
=
3
console.log(a)
```

JavaScript 将其解析为：

```
var a;a=3;console.log(a);
```

语句的分隔规则会导致一些意想不到的情形，这段代码写成了两行，看起来是两条独立的语句：

```
var y=x+f
(a+b).toString()
```

但第二行的圆括号却和第一行的f组成了一个函数调用，JavaScript会把这段代码看做：

```
var y=x+f(a+b).toString();
```

而这段代码的本意并不是这样。为了能让上述代码解析为两条不同的语句，必须手动填写行尾的显式分号。

子不语 1年前 (2017-09-28)