JavaScript 教程

JavaScript 是 Web 的编程语言。

所有现代的 HTML 页面都使用 JavaScript。

JavaScript 非常容易学。

本教程将教你学习从初级到高级JavaScript知识。

JavaScript 在线实例

本教程包含了大量的 JavaScript 实例， 您可以点击 "尝试一下" 来在线查看实例。

实例

我的第一个 JavaScript 程序

这是一个段落

显示日期

尝试一下 »

在每个页面您可以点击 "尝试一下" 在线查看实例！！！

尝试每个实例，并且在线修改代码，查看不同的运行效果！！！

Note 如果能根据本站的实例一步一个脚印学习，你将会在很短的时间内学会 JavaScript。

为什么学习 JavaScript?

JavaScript web 开发人员必须学习的 3 门语言中的一门：

1. **HTML** 定义了网页的内容
2. **CSS** 描述了网页的布局
3. **JavaScript** 网页的行为

本教程是关于 JavaScript 及介绍 JavaScript 如何与 HTML 和 CSS 一起工作。

谁适合阅读本教程?

1. 如果您想学习 JavaScript，您可以学习本教程：

了解 JavaScript 是如何与 HTML 和 CSS 一起工作的。

2. 如果在此之前您已经使用过 JavaScript，您也可以阅读本教程：

JavaScript 一直在升级，所以我们需要时刻了解 JavaScript 的新技术。

阅读本教程前，您需要了解的知识：

阅读本教程，您需要有以下基础：

HTML 和 CSS 基础

如果您想学习这些基础知识，您可以在我们的首页找到相应的教程菜鸟教程。

JavaScript 实例

学习 100 多个 JavaScript 实例！

在实例页面中，您可以点击 "尝试一下" 来查看 JavaScript 在线实例。

- JavaScript 实例
- JavaScript 对象实例
- JavaScript 浏览器支持实例
- JavaScript HTML DOM 实例

JavaScript 测验

在菜鸟教程中测试您的 JavaScript 技能！

JavaScript 参考手册

在菜鸟教程中，我们为您提供完整的 JavaScript 对象、浏览器对象、HTML DOM 对象参考手册。

以下手册包含了每个对象、属性、方法的实例。

- JavaScript 内置对象
- Browser 对象
- HTML DOM 对象

HTML/CSS/JS 在线工具

HTML/CSS/JS 在线工具可以在线编辑 HTML、CSS、JS 代码，并实时查看效果，你也可以将优质代码保存分享：https://c.runoob.com/front-end/61

JavaScript 简介 □

□点我分享笔记

反馈/建议

□

□ JavaScript 教程 JavaScript 用法 □

JavaScript 简介

JavaScript 是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 是脚本语言

JavaScript 是一种轻量级的编程语言。

JavaScript 是可插入 HTML 页面的编程代码。

JavaScript 插入 HTML 页面后，可由所有的现代浏览器执行。

JavaScript 很容易学习。

您将学到什么

下面是您将在本教程中学到的主要内容。

JavaScript：直接写入 HTML 输出流

实例

```
document.write("<h1>这是一个标题</h1>");
document.write("<p>这是一个段落。</p>");
```

尝试一下 »

您只能在 HTML 输出中使用 document.write。如果您在文档加载后使用该方法，会覆盖整个文档。

JavaScript：对事件的反应

实例

```
<button type="button" onclick="alert('欢迎!')">点我!</button>
```

尝试一下 »

alert() 函数在 JavaScript 中并不常用，但它对于代码测试非常方便。

onclick 事件只是您即将在本教程中学到的众多事件之一。

JavaScript：改变 HTML 内容

使用 JavaScript 来处理 HTML 内容是非常强大的功能。

实例

```
x=document.getElementById("demo") //查找元素
x.innerHTML="Hello JavaScript"; //改变内容
```

尝试一下 »

您会经常看到 **document.getElementById("*some id*")**。这个方法是 HTML DOM 中定义的。

DOM (**D**ocument **O**bject **M**odel)（文档对象模型）是用于访问 HTML 元素的正式 W3C 标准。

您将在本教程的多个章节中学到有关 HTML DOM 的知识。

JavaScript：改变 HTML 图像

本例会动态地改变 HTML <image> 的来源（src）：

点亮灯泡

```
<script>
function changeImage()
{
element=document.getElementById('myimage')
if (element.src.match("bulbon"))
{
element.src="/images/pic_bulboff.gif";
}
else
{
element.src="/images/pic_bulbon.gif";
}
}
</script>
<img id="myimage" onclick="changeImage()" src="/images/pic_bulboff.gif" width="100" height="180">
```

点击以下灯泡查看效果：

点击灯泡就可以打开或关闭这盏灯

尝试一下 »

以上实例中代码 ***element.src.match("bulbon")*** *的作用意思是：检索* ***<img id="myimage" onclick="changeImage()" src="/images/pic_bulboff.gif" width="100" height="180">*** *里面 src 属性的值有没有包含* `bulbon` *这个字符串，如果存在字符串* ***bulbon*** *，图片* ***src*** *更新为* ***bulboff.gif*** *，若匹配不到* ***bulbon*** *字符串，* ***src*** *则更新为* ***bulbon.gif***

JavaScript 能够改变任意 HTML 元素的大多数属性，而不仅仅是图片。

JavaScript：改变 HTML 样式

改变 HTML 元素的样式，属于改变 HTML 属性的变种。

实例

```
x=document.getElementById("demo") //找到元素
x.style.color="#ff0000"; //改变样式
```

尝试一下 »

JavaScript：验证输入

JavaScript 常用于验证用户的输入。

实例

```
if isNaN(x) {
alert("不是数字");
}
```

尝试一下 »

以上实例只是普通的验证，如果要在生产环境中使用，需要严格判断，如果输入的空格，或者连续空格 isNaN 是判别不出来的。可以添加正则来判断（后续章节会说明）：

实例

```
if(isNaN(x)||x.replace(/(^\s*)|(\s*$)/g,"")==""){
alert("不是数字");
}
```

尝试一下 »

您知道吗？

JavaScript 与 Java 是两种完全不同的语言，无论在概念上还是设计上。
Java（由 Sun 发明）是更复杂的编程语言。

ECMA-262 是 JavaScript 标准的官方名称。

JavaScript 由 Brendan Eich 发明。它于 1995 年出现在 Netscape 中（该浏览器已停止更新），并于 1997 年被 ECMA（一个标准协会）采纳。

ECMAScript 版本

JavaScript 已经由 ECMA（欧洲电脑制造商协会）通过 ECMAScript 实现语言的标准化。

年份	名称	描述
1997	ECMAScript 1	第一个版本

1998	ECMAScript 2	版本变更
1999	ECMAScript 3	添加正则表达式 添加 try/catch
	ECMAScript 4	没有发布
2009	ECMAScript 5	添加 "strict mode"，严格模式 添加 JSON 支持
2011	ECMAScript 5.1	版本变更
2015	ECMAScript 6	添加类和模块
2016	ECMAScript 7	增加指数运算符 (**) 增加 Array.prototype.includes

ECMAScript 6 也称为 ECMAScript 2015。

ECMAScript 7 也称为 ECMAScript 2016。

□ JavaScript 教程 JavaScript 用法 □

□

1 篇笔记

#1

□写笔记

关于上面切换图片的例子，用三目运算比较简洁。

```
function changeImage(){

    var s = document.getElementById('myimage');

    s.src = s.src.match('bulboff')?"/images/pic_bulbon.gif":"/images/pic_bulboff.gif";

}
```

尝试一下 »

ZYiDa2个月前 (07-27)

反馈/建议

□

□ JavaScript 简介 JavaScript 输出 □

JavaScript 用法

HTML 中的脚本必须位于 <script> 与 </script> 标签之间。

脚本可被放置在 HTML 页面的 <body> 和 <head> 部分中。

<script> 标签

如需在 HTML 页面中插入 JavaScript，请使用 <script> 标签。

<script> 和 </script> 会告诉 JavaScript 在何处开始和结束。

<script> 和 </script> 之间的代码行包含了 JavaScript:

```
<script>
alert("我的第一个 JavaScript");
</script>
```

您无需理解上面的代码。只需明白，浏览器会解释并执行位于 <script> 和 </script>之间的 JavaScript 代码

> 那些老旧的实例可能会在 <script> 标签中使用 type="text/javascript"。现在已经不必这样做了。JavaScript 是所有现代浏览器以及 HTML5 中的默认脚本语言。

<body> 中的 JavaScript

在本例中，JavaScript 会在页面加载时向 HTML 的 <body> 写文本：

实例

```
<!DOCTYPE html>
<html>
<body>
.
.
<script>
document.write("<h1>这是一个标题</h1>");
document.write("<p>这是一个段落</p>");
</script>
.
.
</body>
</html>
```

尝试一下 »

JavaScript 函数和事件

上面例子中的 JavaScript 语句，会在页面加载时执行。

通常，我们需要在某个事件发生时执行代码，比如当用户点击按钮时。

如果我们把 JavaScript 代码放入函数中，就可以在事件发生时调用该函数。

您将在稍后的章节学到更多有关 JavaScript 函数和事件的知识。

在 <head> 或者 <body> 的JavaScript

您可以在 HTML 文档中放入不限数量的脚本。

脚本可位于 HTML 的 <body> 或 <head> 部分中，或者同时存在于两个部分中。

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。

<head> 中的 JavaScript 函数

在本例中，我们把一个 JavaScript 函数放置到 HTML 页面的 <head> 部分。

该函数会在点击按钮时被调用：

实例

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction()
{
```

```
document.getElementById("demo").innerHTML="我的第一个 JavaScript 函数";
}
</script>
</head>
<body>
<h1>我的 Web 页面</h1>
<p id="demo">一个段落</p>
<button type="button" onclick="myFunction()">尝试一下</button>
</body>
</html>
```

尝试一下 »

<body> 中的 JavaScript 函数

在本例中，我们把一个 JavaScript 函数放置到 HTML 页面的 <body> 部分。

该函数会在点击按钮时被调用：

实例

```
<!DOCTYPE html>
<html>
<body>
<h1>我的 Web 页面</h1>
<p id="demo">一个段落</p>
<button type="button" onclick="myFunction()">尝试一下</button>
<script>
function myFunction()
{
document.getElementById("demo").innerHTML="我的第一个 JavaScript 函数";
}
</script>
</body>
</html>
```

尝试一下 »

外部的 JavaScript

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：

实例

```
<!DOCTYPE html>
<html>
<body>
<script src="myScript.js"></script>
</body>
</html>
```

尝试一下 »

你可以将脚本放置于 <head> 或者 <body>中，放在 <script> 标签中的脚本与外部引用的脚本运行效果完全一致。

myScript.js 文件代码如下：

```
function myFunction()
{
document.getElementById("demo").innerHTML="我的第一个 JavaScript 函数";
}
```

外部脚本不能包含 <script> 标签。

JavaScript 输出

JavaScript 没有任何打印或者输出的函数。

JavaScript 显示数据

JavaScript 可以通过不同的方式来输出数据：

使用 **window.alert()** 弹出警告框。

使用 **document.write()** 方法将内容写到 HTML 文档中。

使用 **innerHTML** 写入到 HTML 元素。

使用 **console.log()** 写入到浏览器的控制台。

使用 window.alert()

你可以弹出警告框来显示数据：

实例

```
<!DOCTYPE html>
<html>
<body>

<h1>我的第一个页面</h1>
<p>我的第一个段落。</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

尝试一下 »

操作 HTML 元素

如需从 JavaScript 访问某个 HTML 元素，您可以使用 document.getElementById(*id*) 方法。

请使用 "id" 属性来标识 HTML 元素，并 innerHTML 来获取或插入元素内容：

实例

```
<!DOCTYPE html>
```

```
<html>
<body>

<h1>我的第一个 Web 页面</h1>

<p id="demo">我的第一个段落</p>

<script>
document.getElementById("demo").innerHTML = "段落已修改。";
</script>

</body>
</html>
```

尝试一下 »

以上 JavaScript 语句（在 <script> 标签中）可以在 web 浏览器中执行：

document.getElementById("demo") 是使用 id 属性来查找 HTML 元素的 JavaScript 代码 。

innerHTML = "段落已修改。" 是用于修改元素的 HTML 内容(innerHTML)的 JavaScript 代码。

在本教程中

在大多数情况下，在本教程中，我们将使用上面描述的方法来输出：

上面的例子直接把 id="demo" 的 <p> 元素写到 HTML 文档输出中：

写到 HTML 文档

出于测试目的，您可以将JavaScript直接写在HTML 文档中：

实例

```
<!DOCTYPE html>
<html>
<body>

<h1>我的第一个 Web 页面</h1>

<p>我的第一个段落。</p>

<script>
document.write(Date());
</script>

</body>
</html>
```

尝试一下 »

Note 请使用 document.write() 仅仅向文档输出写内容。

如果在文档已完成加载后执行 document.write，整个 HTML 页面将被覆盖。

实例

```
<!DOCTYPE html>
<html>
<body>

<h1>我的第一个 Web 页面</h1>

<p>我的第一个段落。</p>

<button onclick="myFunction()">点我</button>

<script>
function myFunction() {
    document.write(Date());
}
```

```
</script>

</body>
</html>
```

尝试一下 »

写到控制台

如果您的浏览器支持调试，你可以使用 **console.log()** 方法在浏览器中显示 JavaScript 值。

浏览器中使用 F12 来启用调试模式， 在调试窗口中点击 "Console" 菜单。

实例

```
<!DOCTYPE html>
<html>
<body>

<h1>我的第一个 Web 页面</h1>

<script>
a = 5;
b = 6;
c = a + b;
console.log(c);
</script>

</body>
</html>
```

尝试一下 »

实例 console 截图：

您知道吗?

Note 程序中调试是测试，查找及减少bug(错误)的过程。

JavaScript 用法 JavaScript 语句

5 篇笔记

写笔记

反馈/建议

javascript:void(0) 含义 JavaScript 调试

JavaScript 语法

JavaScript 是一个程序语言。语法规则定义了语言结构。

JavaScript 语法

JavaScript 是一个脚本语言。

它是一个轻量级，但功能强大的编程语言。

JavaScript 字面量

在编程语言中，一般固定值称为字面量，如 3.14。

数字（Number）字面量 可以是整数或者是小数，或者是科学计数(e)。

```
3.14

1001

123e5
```

尝试一下 »

字符串（String）字面量 可以使用单引号或双引号:

```
"John Doe"

'John Doe'
```

尝试一下 »

表达式字面量 用于计算：

```
5 + 6

5 * 10
```

尝试一下 »

数组（Array）字面量 定义一个数组：

```
[40, 100, 1, 5, 25, 10]
```

对象（Object）字面量 定义一个对象：

```
{firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}
```

函数（Function）字面量 定义一个函数：

```
function myFunction(a, b) { return a * b;}
```

JavaScript 变量

在编程语言中，变量用于存储数据值。

JavaScript 使用关键字 **var** 来定义变量， 使用等号来为变量赋值：

```
var x, length

x = 5

length = 6
```

尝试一下 »

变量可以通过变量名访问。在指令式语言中，变量通常是可变的。字面量是一个恒定的值。

Note 变量是一个**名称**。字面量是一个**值**。

JavaScript 操作符

JavaScript使用 **算术运算符** 来计算值:

```
(5 + 6) * 10
```

尝试一下 »

JavaScript使用**赋值运算符**给变量赋值：

```
x = 5
y = 6
z = (x + y) * 10
```

尝试一下 »

JavaScript语言有多种类型的运算符：

类型	实例	描述
赋值，算术和位运算符	= + - * /	在 JS 运算符中描述
条件，比较及逻辑运算符	== != < >	在 JS 比较运算符中描述

JavaScript 语句

在 HTML 中，JavaScript 语句向浏览器发出的命令。

语句是用分号分隔：

```
x = 5 + 6;
y = x * 10;
```

JavaScript 关键字

JavaScript 关键字用于标识要执行的操作。

和其他任何编程语言一样，JavaScript 保留了一些关键字为自己所用。

var 关键字告诉浏览器创建一个新的变量：

```
var x = 5 + 6;
var y = x * 10;
```

JavaScript 同样保留了一些关键字，这些关键字在当前的语言版本中并没有使用，但在以后 JavaScript 扩展中会用到。

JavaScript 关键字必须以字母、下划线（_）或美元符（$）开始。

后续的字符可以是字母、数字、下划线或美元符（数字是不允许作为首字符出现的，以便 JavaScript 可以轻易区分开关键字和数字）。

以下是 JavaScript 中最重要的保留字（按字母顺序）：

abstract	else	instanceof	super
boolean	enum	int	switch
break	export	interface	synchronized
byte	extends	let	this
case	false	long	throw
catch	final	native	throws
char	finally	new	transient
class	float	null	true
const	for	package	try
continue	function	private	typeof
debugger	goto	protected	var
default	if	public	void
delete	implements	return	volatile
do	import	short	while
double	in	static	with

JavaScript 注释

不是所有的 JavaScript 语句都是"命令"。双斜杠 // 后的内容将会被浏览器忽略：

```
// 我不会执行
```

JavaScript 数据类型

JavaScript 有多种数据类型：数字，字符串，数组，对象等等：

```
var length = 16;                               // Number 通过数字字面量赋值
var points = x * 10;                           // Number 通过表达式字面量赋值
var lastName = "Johnson";                      // String 通过字符串字面量赋值
var cars = ["Saab", "Volvo", "BMW"];           // Array  通过数组字面量赋值
var person = {firstName:"John", lastName:"Doe"};  // Object 通过对象字面量赋值
```

数据类型的概念

编程语言中，数据类型是一个非常重要的内容。

为了可以操作变量，了解数据类型的概念非常重要。

如果没有使用数据类型，以下实例将无法执行：

```
16 + "Volvo"
```

16 加上 "Volvo" 是如何计算呢? 以上会产生一个错误还是输出以下结果呢?

```
"16Volvo"
```

你可以在浏览器尝试执行以上代码查看效果。

在接下来的章节中你将学到更多关于数据类型的知识。

JavaScript 函数

JavaScript 语句可以写在函数内，函数可以重复引用：

引用一个函数 = 调用函数(执行函数内的语句)。

```
function myFunction(a, b) {
    return a * b;                                // 返回 a 乘以 b 的结果
}
```

JavaScript 字母大小写

JavaScript 对大小写是敏感的。

当编写 JavaScript 语句时，请留意是否关闭大小写切换键。

函数 **getElementById** 与 **getElementbyID** 是不同的。

同样，变量 **myVariable** 与 **MyVariable** 也是不同的。

JavaScript 字符集

JavaScript 使用 Unicode 字符集。

Unicode 覆盖了所有的字符，包含标点等字符。

如需进一步了解，请学习我们的 完整 Unicode 参考手册。

您知道吗?

Note JavaScript 中，常见的是驼峰法的命名规则，如 lastName (而不是lastname)。

□ javascript:void(0) 含义　　JavaScript 调试 □

□

5 篇笔记

□ **写笔记**

反馈/建议

□

□ JavaScript 输出　　JavaScript 注释 □

JavaScript 语句

JavaScript 语句向浏览器发出的命令。语句的作用是告诉浏览器该做什么。

JavaScript 语句

JavaScript 语句是发给浏览器的命令。

这些命令的作用是告诉浏览器要做的事情。

下面的 JavaScript 语句向 id="demo" 的 HTML 元素输出文本 "你好 Dolly"：

实例

```
document.getElementById("demo").innerHTML = "你好 Dolly";
```

尝试一下 »

分号 ;

分号用于分隔 JavaScript 语句。

通常我们在每条可执行的语句结尾添加分号。

使用分号的另一用处是在一行中编写多条语句。

实例:

```
a = 5;
b = 6;
c = a + b;
```

以上实例也可以这么写:

```
a = 5; b = 6; c = a + b;
```

尝试一下 »

您也可能看到不带有分号的案例。
在 JavaScript 中，用分号来结束语句是可选的。

JavaScript 代码

JavaScript 代码是 JavaScript 语句的序列。

浏览器按照编写顺序依次执行每条语句。

本例向网页输出一个标题和两个段落：

实例

```
document.getElementById("demo").innerHTML="你好 Dolly";
document.getElementById("myDIV").innerHTML="你最近怎么样?";
```

尝试一下 »

JavaScript 代码块

JavaScript 可以分批地组合起来。

代码块以左花括号开始，以右花括号结束。

代码块的作用是一并地执行语句序列。

本例向网页输出一个标题和两个段落：

实例

```
function myFunction()
{
document.getElementById("demo").innerHTML="你好Dolly";
document.getElementById("myDIV").innerHTML="你最近怎么样?";
}
```

尝试一下 »

您将在稍后的章节学到更多有关函数的知识。

JavaScript 语句标识符

JavaScript 语句通常以一个 **语句标识符** 为开始，并执行该语句。

语句标识符是保留关键字不能作为变量名使用。

下表列出了 JavaScript 语句标识符 (关键字)：

语句	描述
break	用于跳出循环。
catch	语句块，在 try 语句块执行出错时执行 catch 语句块。
continue	跳过循环中的一个迭代。
do ... while	执行一个语句块，在条件语句为 true 时继续执行该语句块。
for	在条件语句为 true 时，可以将代码块执行指定的次数。
for ... in	用于遍历数组或者对象的属性（对数组或者对象的属性进行循环操作）。
function	定义一个函数
if ... else	用于基于不同的条件来执行不同的动作。
return	退出函数
switch	用于基于不同的条件来执行不同的动作。
throw	抛出（生成）错误 。
try	实现错误处理，与 catch 一同使用。
var	声明一个变量。
while	当条件语句为 true 时，执行语句块。

空格

JavaScript 会忽略多余的空格。您可以向脚本添加空格，来提高其可读性。下面的两行代码是等效的：

```
var person="Hege";
var person = "Hege";
```

对代码行进行折行

您可以在文本字符串中使用反斜杠对代码行进行换行。下面的例子会正确地显示：

```
document.write("你好 \
世界!");
```

不过，您不能像这样折行：

```
document.write \
("你好世界!");
```

您知道吗？

提示：JavaScript 是脚本语言。浏览器会在读取代码时，逐行地执行脚本代码。而对于传统编程来说，会在执行前对所有代码进行编译。

JavaScript 注释

JavaScript 注释可用于提高代码的可读性。

JavaScript 注释

JavaScript 不会执行注释。

我们可以添加注释来对 JavaScript 进行解释，或者提高代码的可读性。

单行注释以 `//` 开头。

本例用单行注释来解释代码：

实例

```
// 输出标题：
document.getElementById("myH1").innerHTML="欢迎来到我的主页";
// 输出段落：
document.getElementById("myP").innerHTML="这是我的第一个段落。";
```

尝试一下 »

JavaScript 多行注释

多行注释以 `/*` 开始，以 `*/` 结尾。

下面的例子使用多行注释来解释代码：

实例

```
/*
下面的这些代码会输出
一个标题和一个段落
并将代表主页的开始
*/
document.getElementById("myH1").innerHTML="欢迎来到我的主页";
document.getElementById("myP").innerHTML="这是我的第一个段落。";
```

尝试一下 »

使用注释来阻止执行

在下面的例子中，注释用于阻止其中一条代码行的执行（可用于调试）：

实例

```
// document.getElementById("myH1").innerHTML="欢迎来到我的主页";
document.getElementById("myP").innerHTML="这是我的第一个段落。";
```

尝试一下 »

在下面的例子中，注释用于阻止代码块的执行（可用于调试）：

实例

```
/*
document.getElementById("myH1").innerHTML="欢迎来到我的主页";
document.getElementById("myP").innerHTML="这是我的第一个段落。";
*/
```

尝试一下 »

在行末使用注释

在下面的例子中，我们把注释放到代码行的结尾处：

实例

```
var x=5; // 声明 x 并把 5 赋值给它
var y=x+2; // 声明 y 并把 x+2 赋值给它
```

尝试一下 »

JavaScript 语句 JavaScript 变量

2 篇笔记

写笔记

#2

养成一个好的习惯就是写注释。
方便你二次阅读，维护你的代码。
如果后面项目转手，也方便别人来理解你的代码和维护你的代码。

Answer1年前 (2017-06-30)

#1

应用注释符号验证浏览器是否支持 JavaScript 脚本功能

如果用户不能确定浏览器是否支持JavaScript脚本，那么可以应用HTML提供的注释符号进行验证。HTML注释符号是以 `<--` 开始以 `-->` 结束的。如果在此注释符号内编写 JavaScrip t脚本，对于不支持 JavaScript 的浏览器，将会把编写的 JavaScript 脚本作为注释处理。
使用 JavaScript 脚本在页面中输出一个字符串，将 JavaScript 脚本编写在 HTML 注释中，如果浏览器支持 JavaScript 将输出此字符串，如果不支持将不输出此字符串，代码如下:

```
<script>
<!--
document.write("您的浏览器支持JavaScript脚本!");
//-->
</script>
```

注意：注释行结尾处的两条斜杠 `//` 是 JavaScript 注释符号。这可以避免 JavaScript 执行 `-->` 标签。

islander1年前 (2017-08-17)

JavaScript 变量

变量是用于存储信息的"容器"。

实例

```
var x=5;
var y=6;
var z=x+y;
```

尝试一下 »

就像代数那样

x=5

y=6

z=x+y

在代数中，我们使用字母（比如 x）来保存值（比如 5）。

通过上面的表达式 z=x+y，我们能够计算出 z 的值为 11。

在 JavaScript 中，这些字母被称为变量。

您可以把变量看做存储数据的容器。

JavaScript 变量

与代数一样，JavaScript 变量可用于存放值（比如 x=5）和表达式（比如 z=x+y）。

变量可以使用短名称（比如 x 和 y），也可以使用描述性更好的名称（比如 age, sum, totalvolume）。

变量必须以字母开头

变量也能以 $ 和 _ 符号开头（不过我们不推荐这么做）

变量名称对大小写敏感（y 和 Y 是不同的变量）

JavaScript 语句和 JavaScript 变量都对大小写敏感。

JavaScript 数据类型

JavaScript 变量还能保存其他数据类型，比如文本值 (name="Bill Gates")。

在 JavaScript 中，类似 "Bill Gates" 这样一条文本被称为字符串。

JavaScript 变量有很多种类型，但是现在，我们只关注数字和字符串。

当您向变量分配文本值时，应该用双引号或单引号包围这个值。

当您向变量赋的值是数值时，不要使用引号。如果您用引号包围数值，该值会被作为文本来处理。

实例

```
var pi=3.14;
var person="John Doe";
var answer='Yes I am!';
```

尝试一下 »

声明（创建） JavaScript 变量

在 JavaScript 中创建变量通常称为"声明"变量。

我们使用 var 关键词来声明变量：

```
var carname;
```

变量声明之后，该变量是空的（它没有值）。

如需向变量赋值，请使用等号：

```
carname="Volvo";
```

不过，您也可以在声明变量时对其赋值：

```
var carname="Volvo";
```

在下面的例子中，我们创建了名为 carname 的变量，并向其赋值 "Volvo"，然后把它放入 id="demo" 的 HTML 段落中：

实例

```
var carname="Volvo";
document.getElementById("demo").innerHTML=carname;
```

尝试一下 »

一个好的编程习惯是，在代码开始处，统一对需要的变量进行声明。

一条语句，多个变量

您可以在一条语句中声明很多变量。该语句以 var 开头，并使用逗号分隔变量即可：

```
var lastname="Doe", age=30, job="carpenter";
```

声明也可横跨多行：

```
var lastname="Doe",
age=30,
job="carpenter";
```

一条语句中声明的多个不可以赋同一个值:

```
var x,y,z=1;
```

x,y 为 undefined， z 为 1。

Value = undefined

在计算机程序中，经常会声明无值的变量。未使用值来声明的变量，其值实际上是 undefined。

在执行过以下语句后，变量 carname 的值将是 undefined：

```
var carname;
```

重新声明 JavaScript 变量

如果重新声明 JavaScript 变量，该变量的值不会丢失：

在以下两条语句执行后，变量 carname 的值依然是 "Volvo"：

```
var carname="Volvo";
var carname;
```

JavaScript 算数

您可以通过 JavaScript 变量来做算数，使用的是 = 和 + 这类运算符：

实例

```
y=5;
x=y+2;
```

尝试一下 »

您将在本教程稍后的章节学到更多有关 JavaScript 运算符的知识。

JavaScript 注释　　JavaScript 数据类型

2 篇笔记
#2

写笔记

介绍**JS**中的**let**变量：

```
let var1 [= value1] [, var2 [= value2]] [, ..., varN [= valueN]];
```

let允许你声明一个作用域被限制在块级中的变量、语句或者表达式。在Function中局部变量推荐使用let变量，避免变量名冲突。
作用域规则
let 声明的变量只在其声明的块或子块中可用，这一点，与var相似。二者之间最主要的区别在于var声明的变量的作用域是整个封闭函数。

```
function varTest() {
    var x = 1;
    if (true) {
        var x = 2;       // 同样的变量!
        console.log(x);  // 2
    }
    console.log(x);  // 2
}

function letTest() {
    let x = 1;
    if (true) {
        let x = 2;       // 不同的变量
        console.log(x);  // 2
    }
    console.log(x);  // 1
}
```

zhangxv 9个月前 (01-16)
#1

Javascript声明变量的时候，虽然用var关键字声明和不用关键字声明，很多时候运行并没有问题，但是这两种方式还是有区别的。可以正常运行的代码并不代表是合适的代码。

```
// num1为全局变量，num2为window的一个属性

var num1 = 1;

num2 = 2;

// delete num1;  无法删除

// delete num2;  删除

function model(){

var num1 = 1; // 本地变量

num2 = 2;     // window的属性

    // 匿名函数

    (function(){

        var num = 1; // 本地变量

        num1 = 2; // 继承作用域（闭包）

        num3 = 3; // window的属性

    }())

}
```

麦田里的蓝天 5个月前 (04-19)

反馈/建议

JavaScript 数据类型

值类型(基本类型)：字符串（String）、数字(Number)、布尔(Boolean)、对空（Null）、未定义（Undefined）、Symbol。

引用数据类型：对象(Object)、数组(Array)、函数(Function)。

注：Symbol 是 ES6 引入了一种新的原始数据类型，表示独一无二的值。

JavaScript 拥有动态类型

JavaScript 拥有动态类型。这意味着相同的变量可用作不同的类型：

实例

```
var x;               // x 为 undefined
var x = 5;           // 现在 x 为数字
var x = "John";      // 现在 x 为字符串
```

JavaScript 字符串

字符串是存储字符（比如 "Bill Gates"）的变量。

字符串可以是引号中的任意文本。您可以使用单引号或双引号：

实例

```
var carname="Volvo XC60";
var carname='Volvo XC60';
```

您可以在字符串中使用引号，只要不匹配包围字符串的引号即可：

实例

```
var answer="It's alright";
var answer="He is called 'Johnny'";
var answer='He is called "Johnny"';
```

尝试一下 »

您将在本教程的高级部分学到更多关于字符串的知识。

JavaScript 数字

JavaScript 只有一种数字类型。数字可以带小数点，也可以不带：

实例

```
var x1=34.00;      //使用小数点来写
var x2=34;         //不使用小数点来写
```

极大或极小的数字可以通过科学（指数）计数法来书写：

实例

```
var y=123e5;      // 12300000
var z=123e-5;     // 0.00123
```

尝试一下 »

您将在本教程的高级部分学到更多关于数字的知识。

JavaScript 布尔

布尔（逻辑）只能有两个值：true 或 false。

```
var x=true;
var y=false;
```

布尔常用在条件测试中。您将在本教程稍后的章节中学到更多关于条件测试的知识。

JavaScript 数组

下面的代码创建名为 cars 的数组：

```
var cars=new Array();
cars[0]="Saab";
cars[1]="Volvo";
cars[2]="BMW";
```

或者 (condensed array):

```
var cars=new Array("Saab","Volvo","BMW");
```

或者 (literal array):

实例

```
var cars=["Saab","Volvo","BMW"];
```

尝试一下 »

数组下标是基于零的，所以第一个项目是 [0]，第二个是 [1]，以此类推。

您将在本教程稍后的章节中学到更多关于数组的知识。

JavaScript 对象

对象由花括号分隔。在括号内部，对象的属性以名称和值对的形式 (name : value) 来定义。属性由逗号分隔：

```
var person={firstname:"John", lastname:"Doe", id:5566};
```

上面例子中的对象 (person) 有三个属性：firstname、lastname 以及 id。

空格和折行无关紧要。声明可横跨多行：

```
var person={
firstname : "John",
lastname  : "Doe",
id        :  5566
};
```

对象属性有两种寻址方式：

实例

```
name=person.lastname;
name=person["lastname"];
```

尝试一下 »

您将在本教程稍后的章节中学到更多关于对象的知识。

Undefined 和 Null

Undefined 这个值表示变量不含有值。

可以通过将变量的值设置为 null 来清空变量。

实例

```
cars=null;
person=null;
```

尝试一下 »

声明变量类型

当您声明新变量时，可以使用关键词 "new" 来声明其类型：

```
var carname=new String;
var x=      new Number;
var y=      new Boolean;
var cars=   new Array;
var person= new Object;
```

JavaScript 变量均为对象。当您声明一个变量时，就创建了一个新的对象。

JavaScript 对象

JavaScript 对象是拥有属性和方法的数据。

真实生活中的对象，属性和方法

真实生活中，一辆汽车是一个对象。

对象有它的属性，如重量和颜色等，方法有启动停止等:

对象	属性	方法
	car.name = Fiat car.model = 500 car.weight = 850kg car.color = white	car.start() car.drive() car.brake() car.stop()

所有汽车都有这些属性，但是每款车的属性都不尽相同。

所有汽车都拥有这些方法，但是它们被执行的时间都不尽相同。

JavaScript 对象

在 JavaScript中，几乎所有的事物都是对象。

Note 在 JavaScript 中，对象是非常重要的，当你理解了对象，就可以了解 JavaScript 。

你已经学习了 JavaScript 变量的赋值。

以下代码为变量 **car** 设置值为 "Fiat" :

```
var car = "Fiat";
```

对象也是一个变量，但对象可以包含多个值（多个变量）。

```
var car = {type:"Fiat", model:500, color:"white"};
```

在以上实例中，3 个值 ("Fiat", 500, "white") 赋予变量 car。

在以上实例中，3 个变量 (type, model, color) 赋予变量 car。

> **Note** JavaScript 对象是变量的容器。

对象定义

你可以使用字符来定义和创建 JavaScript 对象:

实例

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

尝试一下 »

定义 JavaScript 对象可以跨越多行，空格跟换行不是必须的：

实例

```
var person = {
    firstName:"John",
    lastName:"Doe",
    age:50,
    eyeColor:"blue"
};
```

尝试一下 »

对象属性

可以说 "JavaScript 对象是变量的容器"。

但是，我们通常认为 "JavaScript 对象是键值对的容器"。

键值对通常写法为 **name : value** (键与值以冒号分割)。

键值对在 JavaScript 对象通常称为 **对象属性**。

> **Note** JavaScript 对象是属性变量的容器。

对象键值对的写法类似于：

- PHP 中的关联数组
- Python 中的字典
- C 语言中的哈希表
- Java 中的哈希映射
- Ruby 和 Perl 中的哈希表

访问对象属性

你可以通过两种方式访问对象属性:

实例 1

```
person.lastName;
```

尝试一下 »

实例 2

```
person["lastName"];
```

尝试一下 »

对象方法

对象的方法定义了一个函数，并作为对象的属性存储。

对象方法通过添加 () 调用 (作为一个函数)。

该实例访问了 person 对象的 fullName() 方法:

实例

```
name = person.fullName();
```

尝试一下 »

如果你要访问 person 对象的 fullName 属性，它将作为一个定义函数的字符串返回：

实例

```
name = person.fullName;
```

尝试一下 »

Note JavaScript 对象是属性和方法的容器。

在随后的教程中你将学习到更多关于函数，属性和方法的知识。

访问对象方法

你可以使用以下语法创建对象方法：

```
methodName : function() { code lines }
```

你可以使用以下语法访问对象方法：

```
objectName.methodName()
```

通常 fullName() 是作为 person 对象的一个方法， fullName 是作为一个属性。

有多种方式可以创建，使用和修改 JavaScript 对象。

同样也有多种方式用来创建，使用和修改属性和方法。

Note 在随后的教程中，你将学习到更多关于对象的知识。

更多实例

创建 JavaScript 对象 I

创建 JavaScript 对象 II

访问对象属性 I

访问对象属性 II

函数属性作为一个方法访问

函数属性作为一个属性访问

JavaScript 数据类型 JavaScript 函数

JavaScript 函数

函数是由事件驱动的或者当它被调用时执行的可重复使用的代码块。

实例

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction()
{
alert("Hello World!");
}
</script>
</head>
<body>
<button onclick="myFunction()">Try it</button>
</body>
</html>
```

JavaScript 函数语法

函数就是包裹在花括号中的代码块，前面使用了关键词 function：

```
function functionname()
{
执行代码
}
```

当调用该函数时，会执行函数内的代码。

可以在某事件发生时直接调用函数（比如当用户点击按钮时），并且可由 JavaScript 在任何位置进行调用。

JavaScript 对大小写敏感。关键词 function 必须是小写的，并且必须以与函数名称相同的大小写来调用函数。

调用带参数的函数

在调用函数时，您可以向其传递值，这些值被称为参数。

这些参数可以在函数中使用。

您可以发送任意多的参数，由逗号 (,) 分隔：

```
myFunction(argument1,argument2)
```

当您声明函数时，请把参数作为变量来声明：

```
function myFunction(var1,var2)
{
代码
}
```

变量和参数必须以一致的顺序出现。第一个变量就是第一个被传递的参数的给定的值，以此类推。

实例

```
<p>点击这个按钮，来调用带参数的函数。</p>
<button onclick="myFunction('Harry Potter','Wizard')">点击这里</button>
<script>
function myFunction(name,job){
```

```
alert("Welcome " + name + ", the " + job);
}
</script>
```

尝试一下 »

上面的函数在按钮被点击时会提示 "Welcome Harry Potter, the Wizard"。

函数很灵活，您可以使用不同的参数来调用该函数，这样就会给出不同的消息：

实例

```
<button onclick="myFunction('Harry Potter','Wizard')">点击这里</button>
<button onclick="myFunction('Bob','Builder')">点击这里</button>
```

尝试一下 »

根据您点击的不同的按钮，上面的例子会提示 "Welcome Harry Potter, the Wizard" 或 "Welcome Bob, the Builder"。

带有返回值的函数

有时，我们会希望函数将值返回调用它的地方。

通过使用 return 语句就可以实现。

在使用 return 语句时，函数会停止执行，并返回指定的值。

语法

```
function myFunction()
{
    var x=5;
    return x;
}
```

上面的函数会返回值 5。

注意： 整个 JavaScript 并不会停止执行，仅仅是函数。JavaScript 将继续执行代码，从调用函数的地方。

函数调用将被返回值取代：

```
var myVar=myFunction();
```

myVar 变量的值是 5，也就是函数 "myFunction()" 所返回的值。

即使不把它保存为变量，您也可以使用返回值：

```
document.getElementById("demo").innerHTML=myFunction();
```

"demo" 元素的 innerHTML 将成为 5，也就是函数 "myFunction()" 所返回的值。

您可以使返回值基于传递到函数中的参数：

实例

计算两个数字的乘积，并返回结果：

```
function myFunction(a,b)
{
return a*b;
}
document.getElementById("demo").innerHTML=myFunction(4,3);
```

"demo" 元素的 innerHTML 将是：

```
12
```

尝试一下 »

在您仅仅希望退出函数时，也可使用 return 语句。返回值是可选的：

```
function myFunction(a,b)
{
if (a>b)
{
return;
}
x=a+b
}
```

如果 a 大于 b，则上面的代码将退出函数，并不会计算 a 和 b 的总和。

局部 JavaScript 变量

在 JavaScript 函数内部声明的变量（使用 var）是*局部*变量，所以只能在函数内部访问它。（该变量的作用域是局部的）。

您可以在不同的函数中使用名称相同的局部变量，因为只有声明过该变量的函数才能识别出该变量。

只要函数运行完毕，本地变量就会被删除。

全局 JavaScript 变量

在函数外声明的变量是*全局*变量，网页上的所有脚本和函数都能访问它。

JavaScript 变量的生存期

JavaScript 变量的生命期从它们被声明的时间开始。

局部变量会在函数运行以后被删除。

全局变量会在页面关闭后被删除。

向未声明的 JavaScript 变量分配值

如果您把值赋给尚未声明的变量，该变量将被自动作为 window 的一个属性。

这条语句：

```
carname="Volvo";
```

将声明 window 的一个属性 carname。

非严格模式下给未声明变量赋值创建的全局变量，是全局对象的可配置属性，可以删除。

```
var var1 = 1; // 不可配置全局属性

var2 = 2; // 没有使用 var 声明，可配置全局属性



console.log(this.var1); // 1

console.log(window.var1); // 1



delete var1; // false 无法删除

console.log(var1); //1



delete var2;

console.log(delete var2); // true

console.log(var2); // 已经删除 报错变量未定义
```

JavaScript 作用域

作用域可访问变量的集合。

JavaScript 作用域

在 JavaScript 中, 对象和函数同样也是变量。

在 JavaScript 中, 作用域为可访问变量，对象，函数的集合。

JavaScript 函数作用域: 作用域在函数内修改。

JavaScript 局部作用域

变量在函数内声明，变量为局部作用域。

局部变量：只能在函数内部访问。

实例

```
// 此处不能调用 carName 变量
function myFunction() {
var carName = "Volvo";
// 函数内可调用 carName 变量
}
```

尝试一下 »

因为局部变量只作用于函数内，所以不同的函数可以使用相同名称的变量。

局部变量在函数开始执行时创建，函数执行完后局部变量会自动销毁。

JavaScript 全局变量

变量在函数外定义，即为全局变量。

全局变量有 **全局作用域**: 网页中所有脚本和函数均可使用。

实例

```
var carName = " Volvo";
// 此处可调用 carName 变量
function myFunction() {
// 函数内可调用 carName 变量
}
```

尝试一下 »

如果变量在函数内没有声明（没有使用 var 关键字），该变量为全局变量。

以下实例中 carName 在函数内，但是为全局变量。

实例

```
// 此处可调用 carName 变量
function myFunction() {
carName = "Volvo";
// 此处可调用 carName 变量
```

```
}
```

尝试一下 »

JavaScript 变量生命周期

JavaScript 变量生命周期在它声明时初始化。

局部变量在函数执行完毕后销毁。

全局变量在页面关闭后销毁。

函数参数

函数参数只在函数内起作用，是局部变量。

HTML 中的全局变量

在 HTML 中, 全局变量是 window 对象: 所有数据变量都属于 window 对象。

实例

```
//此处可使用 window.carName
function myFunction() {
carName = "Volvo";
}
```

尝试一下 »

你知道吗?

Note 你的全局变量，或者函数，可以覆盖 window 对象的变量或者函数。
局部变量，包括 window 对象可以覆盖全局变量和函数。

JavaScript JSON　　JavaScript 事件

JavaScript 作用域　　JavaScript 字符串

JavaScript 事件

HTML 事件是发生在 HTML 元素上的事情。

当在 HTML 页面中使用 JavaScript 时， JavaScript 可以触发这些事件。

HTML 事件

HTML 事件可以是浏览器行为，也可以是用户行为。

以下是 HTML 事件的实例：

- HTML 页面完成加载
- HTML input 字段改变时
- HTML 按钮被点击

通常，当事件发生时，你可以做些事情。

在事件触发时 JavaScript 可以执行一些代码。

HTML 元素中可以添加事件属性，使用 JavaScript 代码来添加 HTML 元素。

单引号:

```
<some-HTML-element some-event='JavaScript 代码'>
```

双引号:

```
<some-HTML-element some-event="JavaScript 代码">
```

在以下实例中，按钮元素中添加了 onclick 属性 (并加上代码):

实例

```
<button onclick="getElementById('demo').innerHTML=Date()">现在的时间是?</button>
```

尝试一下 »

以上实例中，JavaScript 代码将修改 id="demo" 元素的内容。

在下一个实例中，代码将修改自身元素的内容 (使用 **this**.innerHTML):

实例

```
<button onclick="this.innerHTML=Date()">现在的时间是?</button>
```

尝试一下 »

Note JavaScript代码通常是几行代码。比较常见的是通过事件属性来调用：

实例

```
<button onclick="displayDate()">现在的时间是?</button>
```

尝试一下 »

常见的HTML事件

下面是一些常见的HTML事件的列表:

事件	描述
onchange	HTML 元素改变
onclick	用户点击 HTML 元素
onmouseover	用户在一个HTML元素上移动鼠标
onmouseout	用户从一个HTML元素上移开鼠标
onkeydown	用户按下键盘按键
onload	浏览器已完成页面的加载

更多事件列表: JavaScript 参考手册 - HTML DOM 事件。

JavaScript 可以做什么？

事件可以用于处理表单验证，用户输入，用户行为及浏览器动作:

- 页面加载时触发事件
- 页面关闭时触发事件
- 用户点击按钮执行动作
- 验证用户输入内容的合法性
- 等等 ...

可以使用多种方法来执行 JavaScript 事件代码：

- HTML 事件属性可以直接执行 JavaScript 代码
- HTML 事件属性可以调用 JavaScript 函数
- 你可以为 HTML 元素指定自己的事件处理程序
- 你可以阻止事件的发生。
- 等等 ...

> **Note** 在 HTML DOM 章节中你将会学到更多关于事件及事件处理程序的知识。

JavaScript 作用域 JavaScript 字符串

JavaScript 事件 javascript:void(0) 含义

JavaScript 字符串

JavaScript 字符串用于存储和处理文本。

JavaScript 字符串

字符串可以存储一系列字符，如 "John Doe"。

字符串可以是插入到引号中的任何字符。你可以使用单引号或双引号：

实例

```
var carname = "Volvo XC60";
var carname = 'Volvo XC60';
```

你可以使用索引位置来访问字符串中的每个字符：

实例

```
var character = carname[7];
```

字符串的索引从 0 开始，这意味着第一个字符索引值为 [0],第二个为 [1], 以此类推。

你可以在字符串中使用引号，字符串中的引号不要与字符串的引号相同:

实例

```
var answer = "It's alright";
var answer = "He is called 'Johnny'";
var answer = 'He is called "Johnny"';
```

你也可以在字符串添加转义字符来使用引号：

实例

```
var x = 'It\'s alright';
var y = "He is called \"Johnny\"";
```

尝试一下 »

字符串长度

可以使用内置属性 **length** 来计算字符串的长度：

实例

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

尝试一下 »

特殊字符

在 JavaScript 中，字符串写在单引号或双引号中。

因为这样，以下实例 JavaScript 无法解析：

```
"We are the so-called "Vikings" from the north."
```

字符串 "We are the so-called " 被截断。

如何解决以上的问题呢？可以使用反斜杠 (\) 来转义 "Vikings" 字符串中的双引号，如下:

```
"We are the so-called \"Vikings\" from the north."
```

反斜杠是一个**转义字符**。 转义字符将特殊字符转换为字符串字符：

转义字符 (\) 可以用于转义撇号，换行，引号，等其他特殊字符。

下表中列举了在字符串中可以使用转义字符转义的特殊字符：

代码	输出
\'	单引号
\"	双引号
\\	反斜杠
\n	换行
\r	回车
\t	tab(制表符)
\b	退格符

\f	换页符

字符串可以是对象

通常， JavaScript 字符串是原始值，可以使用字符创建： **var firstName = "John"**

但我们也可以使用 new 关键字将字符串定义为一个对象： **var firstName = new String("John")**

实例

```
var x = "John";
var y = new String("John");
typeof x // 返回 String
typeof y //  返回 Object
```

尝试一下 »

Note 不要创建 String 对象。它会拖慢执行速度，并可能产生其他副作用：

实例

```
var x = "John";
var y = new String("John");
(x === y) // 结果为 false，因为 x 是字符串，y 是对象
```

尝试一下 »

=== 为绝对相等，即数据类型与值都必须相等。

字符串属性和方法

原始值字符串，如 "John", 没有属性和方法(因为他们不是对象)。

原始值可以使用 JavaScript 的属性和方法，因为 JavaScript 在执行方法和属性时可以把原始值当作对象。

字符串方法我们将在下一章节中介绍。

字符串属性

属性	描述
constructor	返回创建字符串属性的函数
length	返回字符串的长度
prototype	允许您向对象添加属性和方法

字符串方法

更多方法实例可以参见：JavaScript String 对象。

方法	描述
charAt()	返回指定索引位置的字符
charCodeAt()	返回指定索引位置字符的 Unicode 值
concat()	连接两个或多个字符串，返回连接后的字符串
fromCharCode()	将 Unicode 转换为字符串
indexOf()	返回字符串中检索指定字符第一次出现的位置

lastIndexOf()	返回字符串中检索指定字符最后一次出现的位置
localeCompare()	用本地特定的顺序来比较两个字符串
match()	找到一个或多个正则表达式的匹配
replace()	替换与正则表达式匹配的子串
search()	检索与正则表达式相匹配的值
slice()	提取字符串的片断，并在新的字符串中返回被提取的部分
split()	把字符串分割为子字符串数组
substr()	从起始索引号提取字符串中指定数目的字符
substring()	提取字符串中两个指定的索引号之间的字符
toLocaleLowerCase()	根据主机的语言环境把字符串转换为小写，只有几种语言（如土耳其语）具有地方特有的大小写映射
toLocaleUpperCase()	根据主机的语言环境把字符串转换为大写，只有几种语言（如土耳其语）具有地方特有的大小写映射
toLowerCase()	把字符串转换为小写
toString()	返回字符串对象值
toUpperCase()	把字符串转换为大写
trim()	移除字符串首尾空白
valueOf()	返回某个字符串对象的原始值

□ JavaScript 事件　　javascript:void(0) 含义 □

3 篇笔记

□写笔记

反馈/建议

□ JavaScript 函数　　JavaScript 比较和逻辑运算符 □

JavaScript 运算符

运算符 = 用于赋值。

运算符 + 用于加值。

运算符 = 用于给 JavaScript 变量赋值。

算术运算符 + 用于把值加起来。

实例

指定变量值，并将值相加：

```
y=5;
z=2;
x=y+z;
```

在以上语句执行后，*x* 的值是：

```
7
```

尝试一下 »

JavaScript 算术运算符

y=5，下面的表格解释了这些算术运算符：

运算符	描述	例子	x 运算结果	y 运算结果	在线实例
+	加法	x=y+2	7	5	实例 »
-	减法	x=y-2	3	5	实例 »
*	乘法	x=y*2	10	5	实例 »
/	除法	x=y/2	2.5	5	实例 »
%	取模（余数）	x=y%2	1	5	实例 »
++	自增	x=++y	6	6	实例 »
		x=y++	5	6	实例 »
--	自减	x=--y	4	4	实例 »
		x=y--	5	4	实例 »

JavaScript 赋值运算符

赋值运算符用于给 JavaScript 变量赋值。

给定 **x=10** 和 **y=5**，下面的表格解释了赋值运算符：

运算符	例子	等同于	运算结果	在线实例
=	x=y		x=5	实例 »
+=	x+=y	x=x+y	x=15	实例 »
-=	x-=y	x=x-y	x=5	实例 »
*=	x*=y	x=x*y	x=50	实例 »
/=	x/=y	x=x/y	x=2	实例 »
%=	x%=y	x=x%y	x=0	实例 »

用于字符串的 + 运算符

+ 运算符用于把文本值或字符串变量加起来（连接起来）。

如需把两个或多个字符串变量连接起来，请使用 + 运算符。

实例

如需把两个或多个字符串变量连接起来，请使用 + 运算符：

```
txt1="What a very";
txt2="nice day";
txt3=txt1+txt2;
```

txt3 运算结果如下:

```
What a verynice day
```

尝试一下 »

要想在两个字符串之间增加空格，需要把空格插入一个字符串之中：

实例

```
txt1="What a very ";
txt2="nice day";
txt3=txt1+txt2;
```

在以上语句执行后，变量 *txt3*包含的值是：

```
What a very nice day
```

尝试一下 »

或者把空格插入表达式中：：

实例

```
txt1="What a very";
txt2="nice day";
txt3=txt1+" "+txt2;
```

在以上语句执行后，变量*txt3* 包含的值是：

```
What a very nice day
```

尝试一下 »

对字符串和数字进行加法运算

两个数字相加，返回数字相加的和，如果数字与字符串相加，返回字符串，如下实例：

实例

```
x=5+5;
y="5"+5;
z="Hello"+5;
```

x,*y*, 和 *z* 输出结果为:

```
10
55
Hello5
```

尝试一下 »

规则:如果把数字与字符串相加，结果将成为字符串！

JavaScript 比较 和 逻辑运算符

比较和逻辑运算符用于测试 *true* 或者 *false*。

比较运算符

比较运算符在逻辑语句中使用，以测定变量或值是否相等。

x=5，下面的表格解释了比较运算符：

运算符	描述	比较	返回值	实例
==	等于	x==8	*false*	实例 »
		x==5	*true*	实例 »
===	绝对等于（值和类型均相等）	x==="5"	*false*	实例 »
		x===5	*true*	实例 »
!=	不等于	x!=8	*true*	实例 »
!==	不绝对等于（值和类型有一个不相等，或两个都不相等）	x!=="5"	*true*	实例 »
		x!==5	*false*	实例 »
>	大于	x>8	*false*	实例 »
<	小于	x<8	*true*	实例 »
>=	大于或等于	x>=8	*false*	实例 »
<=	小于或等于	x<=8	*true*	实例 »

如何使用

可以在条件语句中使用比较运算符对值进行比较，然后根据结果来采取行动：

```
if (age<18) x="Too young";
```

您将在本教程的下一节中学习更多有关条件语句的知识。

逻辑运算符

逻辑运算符用于测定变量或值之间的逻辑。

给定 x=6 以及 y=3，下表解释了逻辑运算符：

运算符	描述	例子
&&	and	(x < 10 && y > 1) 为 true
||	or	(x==5 || y==5) 为 false
!	not	!(x==y) 为 true

条件运算符

JavaScript 还包含了基于某些条件对变量进行赋值的条件运算符。

语法

```
variablename=(condition)?value1:value2
```

例子

实例

如果变量 age 中的值小于 18，则向变量 voteable 赋值 "年龄太小"，否则赋值 "年龄已达到"。

```
voteable=(age<18)?"年龄太小":"年龄已达到";
```

尝试一下 »

JavaScript 运算符　　JavaScript If...Else 语句

4 篇笔记

写笔记

反馈/建议

JavaScript 比较和逻辑运算符　　JavaScript switch 语句

JavaScript if...Else 语句

条件语句用于基于不同的条件来执行不同的动作。

条件语句

通常在写代码时，您总是需要为不同的决定来执行不同的动作。您可以在代码中使用条件语句来完成该任务。

在 JavaScript 中，我们可使用以下条件语句：

- **if 语句** - 只有当指定条件为 true 时，使用该语句来执行代码
- **if...else 语句** - 当条件为 true 时执行代码，当条件为 false 时执行其他代码
- **if...else if....else 语句**- 使用该语句来选择多个代码块之一来执行
- **switch 语句** - 使用该语句来选择多个代码块之一来执行

if 语句

只有当指定条件为 true 时，该语句才会执行代码。

语法

```
if (condition)
{
    当条件为 true 时执行的代码
}
```

请使用小写的 **if**。使用大写字母（IF）会生成 JavaScript 错误！

实例

当时间小于 20:00 时，生成问候 "Good day"：

```
if (time<20)
{
x="Good day";
}
```

x的结果是：

```
Good day
```

尝试一下 »

请注意，在这个语法中，没有 ..else..。您已经告诉浏览器只有在指定条件为 true 时才执行代码。

if...else 语句

请使用 if....else 语句在条件为 true 时执行代码，在条件为 false 时执行其他代码。

语法

```
if (condition)
{
    当条件为 true 时执行的代码
}
else
{
    当条件不为 true 时执行的代码
}
```

实例

当时间小于 20:00 时，生成问候 "Good day"，否则生成问候 "Good evening"。

```
if (time<20)
{
x="Good day";
}
else
{
x="Good evening";
}
```

x 的结果是：

```
Good day
```

尝试一下 »

if...else if...else 语句

使用 if....else if...else 语句来选择多个代码块之一来执行。

语法

```
if (condition1)
{
    当条件 1 为 true 时执行的代码
```

```
}
else if (condition2)
{
    当条件 2 为 true 时执行的代码
}
else
{
  当条件 1 和 条件 2 都不为 true 时执行的代码
}
```

实例

如果时间小于 10:00，则生成问候 "Good morning"，如果时间大于 10:00 小于 20:00，则生成问候 "Good day"，否则生成问候 "Good evening"：

```
if (time<10)
{
document.write("<b>早上好</b>");
}
else if (time>=10 && time<16)
{
document.write("<b>今天好</b>");
}
else
{
document.write("<b>晚上好!</b>");
}
```

x 的结果是：

今天好

尝试一下 »

更多实例

随机链接

这个实例演示了一个链接，当您点击链接时，会带您到不同的地方去。每种机会都是 50% 的概率。

JavaScript 比较和逻辑运算符 JavaScript switch 语句

JavaScript If...Else 语句 JavaScript for 循环

JavaScript switch 语句

switch 语句用于基于不同的条件来执行不同的动作。

JavaScript switch 语句

请使用 switch 语句来选择要执行的多个代码块之一。

语法

```
switch(n)
{
case 1:
执行代码块 1
break;
case 2:
执行代码块 2
break;
default:
与 case 1 和 case 2 不同时执行的代码
}
```

工作原理：首先设置表达式 *n*（通常是一个变量）。随后表达式的值会与结构中的每个 case 的值做比较。如果存在匹配，则与该 case 关联的代码块会被执行。请使用 **break** 来阻止代码自动地向下一个 case 运行。

实例

显示今天的星期名称。请注意 Sunday=0, Monday=1, Tuesday=2, 等等：

```
var d=new Date().getDay();
switch (d)
{
case 0:x="今天是星期日";
break;
case 1:x="今天是星期一";
break;
case 2:x="今天是星期二";
break;
case 3:x="今天是星期三";
break;
case 4:x="今天是星期四";
break;
case 5:x="今天是星期五";
break;
case 6:x="今天是星期六";
break;
}
```

x 的运行结果：

```
今天是星期二
```

尝试一下 »

default 关键词

请使用 default 关键词来规定匹配不存在时做的事情：

实例

如果今天不是星期六或星期日，则会输出默认的消息：

```
var d=new Date().getDay();
switch (d)
{
case 6:x="今天是星期六";
break;
case 0:x="今天是星期日";
break;
default:
x="期待周末";
}
document.getElementById("demo").innerHTML=x;
```

x 的运行结果：

```
期待周末
```

尝试一下 »

JavaScript for 循环

循环可以将代码块执行指定的次数。

JavaScript 循环

如果您希望一遍又一遍地运行相同的代码，并且每次的值都不同，那么使用循环是很方便的。

我们可以这样输出数组的值：

一般写法：

```
document.write(cars[0] + "<br>");
document.write(cars[1] + "<br>");
document.write(cars[2] + "<br>");
document.write(cars[3] + "<br>");
document.write(cars[4] + "<br>");
document.write(cars[5] + "<br>");
```

使用for循环

```
for (var i=0;i<cars.length;i++)
{
document.write(cars[i] + "<br>");
}
```

尝试一下 »

不同类型的循环

JavaScript 支持不同类型的循环：

- **for** - 循环代码块一定的次数
- **for/in** - 循环遍历对象的属性
- **while** - 当指定的条件为 true 时循环指定的代码块
- **do/while** - 同样当指定的条件为 true 时循环指定的代码块

For 循环

for 循环是您在希望创建循环时常会用到的工具。

下面是 for 循环的语法：

```
for (语句 1; 语句 2; 语句 3)
{
    被执行的代码块
}
```

语句 1（代码块）开始前执行

语句 2 定义运行循环（代码块）的条件

语句 3 在循环（代码块）已被执行之后执行

实例

```
for (var i=0; i<5; i++)
{
x=x + "该数字为 " + i + "<br>";
}
```

尝试一下 »

从上面的例子中，您可以看到：

Statement 1 在循环开始之前设置变量 (var i=0)。

Statement 2 定义循环运行的条件（i 必须小于 5）。

Statement 3 在每次代码块已被执行后增加一个值 (i++)。

语句 1

通常我们会使用语句 1 初始化循环中所用的变量 (var i=0)。

语句 1 是可选的，也就是说不使用语句 1 也可以。

您可以在语句 1 中初始化任意（或者多个）值：

实例:

```
for (var i=0,len=cars.length; i<len; i++)
{
document.write(cars[i] + "<br>");
}
```

尝试一下 »

同时您还可以省略语句 1（比如在循环开始前已经设置了值时）：

实例:

```
var i=2,len=cars.length;
for (; i<len; i++)
{
document.write(cars[i] + "<br>");
}
```

尝试一下 »

语句 2

通常语句 2 用于评估初始变量的条件。

语句 2 同样是可选的。

如果语句 2 返回 true，则循环再次开始，如果返回 false，则循环将结束。

如果您省略了语句 2，那么必须在循环内提供 *break*。否则循环就无法停下来。这样有可能令浏览器崩溃。请在本教程稍后的章节阅读有关 break 的内容。

语句 3

通常语句 3 会增加初始变量的值。

语句 3 也是可选的。

语句 3 有多种用法。增量可以是负数 (i–)，或者更大 (i=i+15)。

语句 3 也可以省略（比如当循环内部有相应的代码时）：

实例：

```
var i=0,len=cars.length;
for (; i<len; )
{
document.write(cars[i] + "<br>");
i++;
}
```

尝试一下 »

For/In 循环

JavaScript for/in 语句循环遍历对象的属性：

实例

```
var person={fname:"John",lname:"Doe",age:25};
for (x in person) // x 为属性名
{
txt=txt + person[x];
}
```

尝试一下 »

您将在有关 JavaScript 对象的章节学到更多有关 for / in 循环的知识。

While 循环

我们将在下一章为您讲解 while 循环和 do/while 循环。

JavaScript switch 语句 JavaScript while 循环

JavaScript for 循环 JavaScript Break 和 Continue 语句

JavaScript while 循环

只要指定条件为 true，循环就可以一直执行代码块。

while 循环

while 循环会在指定条件为真时循环执行代码块。

语法

```
while (条件)
{
    需要执行的代码
}
```

实例

本例中的循环将继续运行，只要变量 i 小于 5：

实例

```
while (i<5)
{
x=x + "The number is " + i + "<br>";
i++;
}
```

尝试一下 »

如果您忘记增加条件中所用变量的值，该循环永远不会结束。这可能导致浏览器崩溃。

do/while 循环

do/while 循环是 while 循环的变体。该循环会在检查条件是否为真之前执行一次代码块，然后如果条件为真的话，就会重复这个循环。

语法

```
do
{
    需要执行的代码
}
while (条件);
```

实例

下面的例子使用 do/while 循环。该循环至少会执行一次，即使条件为 false 它也会执行一次，因为代码块会在条件被测试前执行：

实例

```
do
{
x=x + "The number is " + i + "<br>";
i++;
}
while (i<5);
```

尝试一下 »

别忘记增加条件中所用变量的值，否则循环永远不会结束！

比较 for 和 while

如果您已经阅读了前面那一章关于 for 循环的内容，您会发现 while 循环与 for 循环很像。

本例中的循环使用 **for 循环**来显示 cars 数组中的所有值：

实例

```
cars=["BMW","Volvo","Saab","Ford"];
var i=0;
for (;cars[i];)
{
document.write(cars[i] + "<br>");
i++;
```

```
}
```

尝试一下 »

本例中的循环使用 **while 循环**来显示 cars 数组中的所有值：

实例

```
cars=["BMW","Volvo","Saab","Ford"];
var i=0;
while (cars[i])
{
document.write(cars[i] + "<br>");
i++;
}
```

尝试一下 »

JavaScript for 循环　　JavaScript Break 和 Continue 语句

1 篇笔记

#1

写笔记

while 使用 length 属性循环数组

while 和 do/while 的区别 : do/while至少会执行一遍

```
var size=[1,2,3,4,5,6,7] ; //申明一个数组

var i=0;



//while循环

while( i < size.length ) {

    document.write(size[i] + " ");

    i++;

}



document.write("<br>---------------<br>");



//do…..while循环

j=0

do{

    document.write(size[j] + " ");

    j++;

}

while( j<size.length )
```

JavaScript Break 和 Continue 语句

break 语句用于跳出循环。

continue 用于跳过循环中的一个迭代。

Break 语句

我们已经在本教程之前的章节中见到过 break 语句。它用于跳出 switch() 语句。

break 语句可用于跳出循环。

continue 语句跳出循环后，会继续执行该循环之后的代码（如果有的话）：

实例

```
for (i=0;i<10;i++)
{
if (i==3)
{
break;
}
x=x + "The number is " + i + "<br>";
}
```

尝试一下 »

由于这个 if 语句只有一行代码，所以可以省略花括号：

```
for (i=0;i<10;i++)
{
if (i==3) break;
x=x + "The number is " + i + "<br>";
}
```

Continue 语句

continue 语句中断循环中的迭代，如果出现了指定的条件，然后继续循环中的下一个迭代。 该例子跳过了值 3：

实例

```
for (i=0;i<=10;i++)
{
if (i==3) continue;
x=x + "The number is " + i + "<br>";
```

```
}
```

尝试一下 »

JavaScript 标签

正如您在 switch 语句那一章中看到的，可以对 JavaScript 语句进行标记。

如需标记 JavaScript 语句，请在语句之前加上冒号：

```
label:
statements
```

break 和 continue 语句仅仅是能够跳出代码块的语句。

语法:

```
break labelname;
continue labelname;
```

continue 语句（带有或不带标签引用）只能用在循环中。

break 语句（不带标签引用），只能用在循环或 switch 中。

通过标签引用，break 语句可用于跳出任何 JavaScript 代码块：

实例

```
cars=["BMW","Volvo","Saab","Ford"];
list:
{
document.write(cars[0] + "<br>");
document.write(cars[1] + "<br>");
document.write(cars[2] + "<br>");
break list;
document.write(cars[3] + "<br>");
document.write(cars[4] + "<br>");
document.write(cars[5] + "<br>");
}
```

尝试一下 »

□ JavaScript while 循环　　JavaScript 错误 – Throw、Try 和 Catch □

关于 JavaScript 标签与 break 和 continue 一起使用的理解。
break 的作用是跳出代码块, 所以 break 可以使用与循环和 switch 等
continue 的作用是进入下一个迭代, 所以 continue 只能用于循环的代码块。
代码块: 基本上是｛｝大括号之间
然后：
1. 默认标签的情况（除了默认标签情况，其他时候必须要有名标签，否则会有惊喜）
当 break 和 continue 同时用于循环时，没有加标签，此时默认标签为当前"循环"的代码块。
当 break 用于 switch 时，默认标签为当前的 switch 代码块:
有名标签的情况

```
cars=["BMW","Volvo","Saab","Ford"];

list:

{

    document.write(cars[0] + "");
```

```
    document.write(cars[1] + "");
    document.write(cars[2] + "");
    break list;
    document.write(cars[3] + "");
    document.write(cars[4] + "");
    document.write(cars[5] + "");
}
```

上述break list;会跳出list的代码块。如果将break换成continue会有惊喜，违反了明确中的第二点，因为list只是个普通代码块，而不是循环。除非list写成如下形式 list:

```
for(var i=0; i<10; ++i)
{
    continue list;
}
```

有了标签，可以使用break和continue在多层循环的时候控制外层循环。
例如下面：

```
outerloop:
for (var i = 0; i < 10; i++)
{
    innerloop:
    for (var j = 0; j < 10; j++)
    {
        if (j > 3)
        {
            break;
        }
        if (i == 2)
        {
            break innerloop;
        }
        if (i == 4)
        {
            break outerloop;
```

```
        }
        document.write("i=" + i + " j=" + j + "");
    }
}
```

曹大宝2年前 (2017-03-28)

反馈/建议

JavaScript typeof, null, 和 undefined

JavaScript typeof, null, undefined, valueOf()。

typeof 操作符

你可以使用 typeof 操作符来检测变量的数据类型。

实例

```
typeof "John"                 // 返回 string
typeof 3.14                   // 返回 number
typeof false                  // 返回 boolean
typeof [1,2,3,4]              // 返回 object
typeof {name:'John', age:34} // 返回 object
```

尝试一下 »

Note 在JavaScript中，数组是一种特殊的对象类型。 因此 typeof [1,2,3,4] 返回 object。

null

在 JavaScript 中 null 表示 "什么都没有"。

null是一个只有一个值的特殊类型。表示一个空对象引用。

Note 用 typeof 检测 null 返回是object。

你可以设置为 null 来清空对象:

实例

```
var person = null;           // 值为 null(空), 但类型为对象
```

尝试一下 »

你可以设置为 undefined 来清空对象:

实例

```
var person = undefined;     // 值为 undefined, 类型为 undefined
```

尝试一下 »

undefined

在 JavaScript 中, **undefined** 是一个没有设置值的变量。

typeof 一个没有值的变量会返回 **undefined**。

实例

```
var person;                 // 值为 undefined(空), 类型是undefined
```

尝试一下 »

任何变量都可以通过设置值为 **undefined** 来清空。 类型为 **undefined**.

实例

```
person = undefined;         // 值为 undefined, 类型是undefined
```

尝试一下 »

undefined 和 null 的区别

实例

null 和 undefined 的值相等，但类型不等：

```
typeof undefined            // undefined
typeof null                 // object
null === undefined          // false
null == undefined           // true
```

尝试一下 »

JavaScript 正则表达式 JavaScript 类型转换

点我分享笔记

反馈/建议

JavaScript 类型转换

Number() 转换为数字， String() 转换为字符串， Boolean() 转化为布尔值。

JavaScript 数据类型

在 JavaScript 中有 5 种不同的数据类型：

- string
- number
- boolean
- object
- function

3 种对象类型：

- Object
- Date
- Array

2 个不包含任何值的数据类型：

- null
- undefined

typeof 操作符

你可以使用 **typeof** 操作符来查看 JavaScript 变量的数据类型。

实例

```
typeof "John"                 // 返回 string
typeof 3.14                   // 返回 number
typeof NaN                    // 返回 number
typeof false                  // 返回 boolean
typeof [1,2,3,4]              // 返回 object
typeof {name:'John', age:34}  // 返回 object
typeof new Date()             // 返回 object
typeof function () {}         // 返回 function
typeof myCar                  // 返回 undefined (如果 myCar 没有声明)
typeof null                   // 返回 object
```

尝试一下 »

请注意：

- NaN 的数据类型是 number
- 数组(Array)的数据类型是 object
- 日期(Date)的数据类型为 object
- null 的数据类型是 object
- 未定义变量的数据类型为 undefined

如果对象是 JavaScript Array 或 JavaScript Date ，我们就无法通过 **typeof** 来判断他们的类型，因为都是 返回 Object。

constructor 属性

constructor 属性返回所有 JavaScript 变量的构造函数。

实例

```
"John".constructor                 // 返回函数 String()  { [native code] }
```

```
(3.14).constructor                 // 返回函数 Number()  { [native code] }
false.constructor                  // 返回函数 Boolean() { [native code] }
[1,2,3,4].constructor              // 返回函数 Array()   { [native code] }
{name:'John', age:34}.constructor  // 返回函数 Object()  { [native code] }
new Date().constructor             // 返回函数 Date()    { [native code] }
function () {}.constructor         // 返回函数 Function(){ [native code] }
```

尝试一下 »

你可以使用 constructor 属性来查看对象是否为数组 (包含字符串 "Array"):

实例

```
function isArray(myArray) {
    return myArray.constructor.toString().indexOf("Array") > -1;
}
```

尝试一下 »

你可以使用 constructor 属性来查看对象是否为日期 (包含字符串 "Date"):

实例

```
function isDate(myDate) {
    return myDate.constructor.toString().indexOf("Date") > -1;
}
```

尝试一下 »

JavaScript 类型转换

JavaScript 变量可以转换为新变量或其他数据类型：

通过使用 JavaScript 函数

通过 JavaScript 自身自动转换

将数字转换为字符串

全局方法 **String()** 可以将数字转换为字符串。

该方法可用于任何类型的数字，字母，变量，表达式：

实例

```
String(x)         // 将变量 x 转换为字符串并返回
String(123)       // 将数字 123 转换为字符串并返回
String(100 + 23)  // 将数字表达式转换为字符串并返回
```

尝试一下 »

Number 方法 **toString()** 也是有同样的效果。

实例

```
x.toString()
(123).toString()
(100 + 23).toString()
```

尝试一下 »

在 Number 方法 章节中，你可以找到更多数字转换为字符串的方法：

方法	描述
toExponential()	把对象的值转换为指数计数法。
toFixed()	把数字转换为字符串，结果的小数点后有指定位数的数字。

toPrecision()	把数字格式化为指定的长度。

将布尔值转换为字符串

全局方法 **String()** 可以将布尔值转换为字符串。

```
String(false)        // 返回 "false"
String(true)         // 返回 "true"
```

Boolean 方法 **toString()** 也有相同的效果。

```
false.toString()     // 返回 "false"
true.toString()      // 返回 "true"
```

将日期转换为字符串

Date() 返回字符串。

```
Date()      // 返回 Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)
```

全局方法 String() 可以将日期对象转换为字符串。

```
String(new Date())      // 返回 Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)
```

Date 方法 **toString()** 也有相同的效果。

实例

```
obj = new Date()
obj.toString()   // 返回 Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)
```

在 Date 方法 章节中，你可以查看更多关于日期转换为字符串的函数：

方法	描述
getDate()	从 Date 对象返回一个月中的某一天 (1 ~ 31)。
getDay()	从 Date 对象返回一周中的某一天 (0 ~ 6)。
getFullYear()	从 Date 对象以四位数字返回年份。
getHours()	返回 Date 对象的小时 (0 ~ 23)。
getMilliseconds()	返回 Date 对象的毫秒(0 ~ 999)。
getMinutes()	返回 Date 对象的分钟 (0 ~ 59)。
getMonth()	从 Date 对象返回月份 (0 ~ 11)。
getSeconds()	返回 Date 对象的秒数 (0 ~ 59)。
getTime()	返回 1970 年 1 月 1 日至今的毫秒数。

将字符串转换为数字

全局方法 **Number()** 可以将字符串转换为数字。

字符串包含数字(如 "3.14") 转换为数字 (如 3.14).

空字符串转换为 0。

其他的字符串会转换为 NaN (不是个数字)。

```
Number("3.14")    // 返回 3.14
Number(" ")       // 返回 0
Number("")        // 返回 0
```

```
Number("99 88")   // 返回 NaN
```

在 Number 方法 章节中，你可以查看到更多关于字符串转为数字的方法：

方法	描述
parseFloat()	解析一个字符串，并返回一个浮点数。
parseInt()	解析一个字符串，并返回一个整数。

一元运算符 +

Operator + 可用于将变量转换为数字：

实例

```
var y = "5";      // y 是一个字符串
var x = + y;      // x 是一个数字
```

尝试一下 »

如果变量不能转换，它仍然会是一个数字，但值为 NaN (不是一个数字):

实例

```
var y = "John";   // y 是一个字符串
var x = + y;      // x 是一个数字 (NaN)
```

尝试一下 »

将布尔值转换为数字

全局方法 **Number()** 可将布尔值转换为数字。

```
Number(false)     // 返回 0
Number(true)      // 返回 1
```

将日期转换为数字

全局方法 **Number()** 可将日期转换为数字。

```
d = new Date();
Number(d)          // 返回 1404568027739
```

日期方法 **getTime()** 也有相同的效果。

```
d = new Date();
d.getTime()        // 返回 1404568027739
```

自动转换类型

当 JavaScript 尝试操作一个 "错误" 的数据类型时，会自动转换为 "正确" 的数据类型。

以下输出结果不是你所期望的：

```
5 + null    // 返回 5         null 转换为 0
"5" + null  // 返回"5null"   null 转换为 "null"
"5" + 1     // 返回 "51"      1 转换为 "1"
"5" - 1     // 返回 4         "5" 转换为 5
```

自动转换为字符串

当你尝试输出一个对象或一个变量时 JavaScript 会自动调用变量的 toString() 方法：

```
document.getElementById("demo").innerHTML = myVar;

myVar = {name:"Fjohn"}  // toString 转换为 "[object Object]"
myVar = [1,2,3,4]       // toString 转换为 "1,2,3,4"
myVar = new Date()      // toString 转换为 "Fri Jul 18 2014 09:08:55 GMT+0200"
```

数字和布尔值也经常相互转换：

```
myVar = 123             // toString 转换为 "123"
myVar = true            // toString 转换为 "true"
myVar = false           // toString 转换为 "false"
```

下表展示了使用不同的数值转换为数字(Number), 字符串(String), 布尔值(Boolean):

原始值	转换为数字	转换为字符串	转换为布尔值	实例
false	0	"false"	false	尝试一下 »
true	1	"true"	true	尝试一下 »
0	0	"0"	false	尝试一下 »
1	1	"1"	true	尝试一下 »
"0"	0	"0"	**true**	尝试一下 »
"000"	0	"000"	**true**	尝试一下 »
"1"	1	"1"	true	尝试一下 »
NaN	NaN	"NaN"	false	尝试一下 »
Infinity	Infinity	"Infinity"	true	尝试一下 »
-Infinity	-Infinity	"-Infinity"	true	尝试一下 »
""	**0**	""	**false**	尝试一下 »
"20"	20	"20"	true	尝试一下 »
"Runoob"	NaN	"Runoob"	true	尝试一下 »
[ ]	**0**	""	true	尝试一下 »
[20]	**20**	"20"	true	尝试一下 »
[10,20]	NaN	"10,20"	true	尝试一下 »
["Runoob"]	NaN	"Runoob"	true	尝试一下 »
["Runoob","Google"]	NaN	"Runoob,Google"	true	尝试一下 »

function(){}	NaN	"function(){}"	true	尝试一下 »
{ }	NaN	"[object Object]"	true	尝试一下 »
null	0	"null"	false	尝试一下 »
undefined	NaN	"undefined"	false	尝试一下 »

JavaScript typeof, null, 和 undefined　　JavaScript JSON.parse()

检测数据类型：typeof 与 instanceof

typeof

typeof 用以获取一个变量或者表达式的类型，typeof 一般只能返回如下几个结果：

```
number,boolean,string,function（函数）,object（NULL,数组，对象）,undefined。
```

实例：

```
document.getElementById("demo").innerHTML =

typeof "john" + "<br>" +

typeof 3.14 + "<br>" +

typeof false + "<br>" +

typeof [1,2,3,4] + "<br>" +

typeof {name:'john', age:34};
```

尝试一下 »

我们可以使用 typeof 来获取一个变量是否存在，如 `if(typeof a!="undefined"){}`，而不要去使用 if(a) 因为如果 a 不存在（未声明）则会出错。
正因为 typeof 遇到 null,数组,对象时都会返回 object 类型，所以当我们要判断一个对象是否是数组时。
或者判断某个变量是否是某个对象的实例则要选择使用另一个关键语法 **instanceof**
。

instanceof

可通过 instanceof 操作符来判断对象的具体类型，语法格式:

```
var result = objectName instanceof objectType
```

返回布尔值，如果是指定类型返回 true，否则返回 false：
例：

```
arr = [1,2,3];

if(arr instanceof Array){

    document.write("arr 是一个数组");

} else {

    document.write("arr 不是一个数组");
```

```
}
```

尝试一下 »

夜尽11个月前 (11-16)

#1

undefined 与 null 的区别：

表面上 undefined 与 null 都是什么都没有的意思，但是实际上 undefined 是未定义（就是变量没有初始化），null 是一个变量初始化了，但是什么值都没给，只给了一个空对象；进一步说，undefined 与 null是值相等，类型不相等。

Endl2个月前 (08-14)

反馈/建议

JavaScript 正则表达式

正则表达式（英语：Regular Expression，在代码中常简写为regex、regexp或RE）使用单个字符串来描述、匹配一系列符合某个句法规则的字符串搜索模式。

搜索模式可用于文本搜索和文本替换。

什么是正则表达式？

正则表达式是由一个字符序列形成的搜索模式。

当你在文本中搜索数据时，你可以用搜索模式来描述你要查询的内容。

正则表达式可以是一个简单的字符，或一个更复杂的模式。

正则表达式可用于所有文本搜索和文本替换的操作。

语法

```
/正则表达式主体/修饰符(可选)
```

其中修饰符是可选的。

实例：

```
var patt = /runoob/i
```

实例解析：

/runoob/i 是一个正则表达式。

runoob 是一个**正则表达式主体** (用于检索)。

i 是一个**修饰符** (搜索不区分大小写)。

使用字符串方法

在 JavaScript 中，正则表达式通常用于两个字符串方法：search() 和 replace()。

search() 方法 用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的子字符串，并返回子串的起始位置。

replace() 方法 用于在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子串。

search() 方法使用正则表达式

实例

使用正则表达式搜索 "Runoob" 字符串，且不区分大小写：

```
var str = "Visit Runoob!";
var n = str.search(/Runoob/i);
```

输出结果为：

```
6
```

尝试一下 »

search() 方法使用字符串

search 方法可使用字符串作为参数。字符串参数会转换为正则表达式：

实例

检索字符串中 "Runoob" 的子串：

```
var str = "Visit Runoob!";
var n = str.search("Runoob");
```

尝试一下 »

replace() 方法使用正则表达式

实例

使用正则表达式且不区分大小写将字符串中的 Microsoft 替换为 Runoob：

```
var str = document.getElementById("demo").innerHTML;
var txt = str.replace(/microsoft/i,"Runoob");
```

结果输出为:

```
Visit Runoob!
```

尝试一下 »

replace() 方法使用字符串

replace() 方法将接收字符串作为参数：

```
var str = document.getElementById("demo").innerHTML;
var txt = str.replace("Microsoft","Runoob");
```

尝试一下 »

你注意到了吗？

Note 正则表达式参数可用在以上方法中 (替代字符串参数)。
正则表达式使得搜索功能更加强大(如实例中不区分大小写)。

正则表达式修饰符

修饰符 可以在全局搜索中不区分大小写:

修饰符	描述
i	执行对大小写不敏感的匹配。
g	执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。
m	执行多行匹配。

正则表达式模式

方括号用于查找某个范围内的字符：

表达式	描述
[abc]	查找方括号之间的任何字符。
[0-9]	查找任何从 0 至 9 的数字。
(x|y)	查找任何以 | 分隔的选项。

元字符是拥有特殊含义的字符：

元字符	描述
\d	查找数字。
\s	查找空白字符。
\b	匹配单词边界。
\uxxxx	查找以十六进制数 xxxx 规定的 Unicode 字符。

量词:

量词	描述
n+	匹配任何包含至少一个 *n* 的字符串。
n*	匹配任何包含零个或多个 *n* 的字符串。
n?	匹配任何包含零个或一个 *n* 的字符串。

使用 RegExp 对象

在 JavaScript 中，RegExp 对象是一个预定义了属性和方法的正则表达式对象。

使用 test()

test() 方法是一个正则表达式方法。

test() 方法用于检测一个字符串是否匹配某个模式，如果字符串中含有匹配的文本，则返回 true，否则返回 false。

以下实例用于搜索字符串中的字符 "e"：

实例

```
var patt = /e/;
patt.test("The best things in life are free!");
```

字符串中含有 "e"，所以该实例输出为：

```
true
```

尝试一下 »

你可以不用设置正则表达式的变量，以上两行代码可以合并为一行：

```
/e/.test("The best things in life are free!")
```

使用 exec()

exec() 方法是一个正则表达式方法。

exec() 方法用于检索字符串中的正则表达式的匹配。

该函数返回一个数组，其中存放匹配的结果。如果未找到匹配，则返回值为 null。

以下实例用于搜索字符串中的字母 "e":

Example 1

```
/e/.exec("The best things in life are free!");
```

字符串中含有 "e"，所以该实例输出为:

```
e
```

尝试一下 »

更多实例

JS 判断输入字符串是否为数字、字母、下划线组成

JS 判断输入字符串是否全部为字母

JS 判断输入字符串是否全部为数字

完整的 RegExp 参考手册

完整的 RegExp 对象参考手册，请参考我们的 JavaScript RegExp 参考手册。

该参考手册包含了所有 RegExp 对象的方法和属性。

□ JavaScript 闭包　　JavaScript typeof, null, 和 undefined □

1 篇笔记

#1

□写笔记

正则表达式表单验证实例：

```
/*是否带有小数*/
function    isDecimal(strValue )  {
   var  objRegExp= /^\d+\.\d+$/;
   return  objRegExp.test(strValue);
}

/*校验是否中文名称组成 */
function ischina(str) {
    var reg=/^[\u4E00-\u9FA5]{2,4}$/;    /*定义验证表达式*/
    return reg.test(str);     /*进行验证*/
}
```

```
/*校验是否全由8位数字组成 */

function isStudentNo(str) {

    var reg=/^[0-9]{8}$/;   /*定义验证表达式*/

    return reg.test(str);     /*进行验证*/

}



/*校验电话码格式 */

function isTelCode(str) {

    var reg= /^((0\d{2,3}-\d{7,8})|(1[3584]\d{9}))$/;

    return reg.test(str);

}



/*校验邮件地址是否合法 */

function IsEmail(str) {

    var reg=/^\w+@[a-zA-Z0-9]{2,10}(?:\.[a-z]{2,4}){1,3}$/;

    return reg.test(str);

}
```

尝试一下 »

qq10561254782年前 (2017-04-03)

反馈/建议

JavaScript 错误 - throw、try 和 catch

try 语句测试代码块的错误。

catch 语句处理错误。

throw 语句创建自定义错误。

JavaScript 错误

当 JavaScript 引擎执行 JavaScript 代码时，会发生各种错误。

可能是语法错误，通常是程序员造成的编码错误或错别字。

可能是拼写错误或语言中缺少的功能（可能由于浏览器差异）。

可能是由于来自服务器或用户的错误输出而导致的错误。

当然，也可能是由于许多其他不可预知的因素。

JavaScript 抛出（throw）错误

当错误发生时，当事情出问题时，JavaScript 引擎通常会停止，并生成一个错误消息。

描述这种情况的技术术语是：JavaScript 将**抛出**一个错误。

JavaScript try 和 catch

try 语句允许我们定义在执行时进行错误测试的代码块。

catch 语句允许我们定义当 try 代码块发生错误时，所执行的代码块。

JavaScript 语句 **try** 和 **catch** 是成对出现的。

语法

```
try {
//在这里运行代码
} catch(err) {
//在这里处理错误
}
```

实例

在下面的例子中，我们故意在 try 块的代码中写了一个错字。

catch 块会捕捉到 try 块中的错误，并执行代码来处理它。

实例

```
var txt="";
function message()
{
try {
adddlert("Welcome guest!");
} catch(err) {
txt="本页有一个错误。\n\n";
txt+="错误描述：" + err.message + "\n\n";
txt+="点击确定继续。\n\n";
alert(txt);
}
}
```

尝试一下 »

Throw 语句

throw 语句允许我们创建自定义错误。

正确的技术术语是：创建或**抛出异常**（exception）。

如果把 throw 与 try 和 catch 一起使用，那么您能够控制程序流，并生成自定义的错误消息。

语法

```
throw exception
```

异常可以是 JavaScript 字符串、数字、逻辑值或对象。

实例

本例检测输入变量的值。如果值是错误的，会抛出一个异常（错误）。catch 会捕捉到这个错误，并显示一段自定义的错误消息：

实例

```
function myFunction() {
var message, x;
message = document.getElementById("message");
message.innerHTML = "";
x = document.getElementById("demo").value;
try {
if(x == "") throw "值为空";
if(isNaN(x)) throw "不是数字";
x = Number(x);
if(x < 5) throw "太小";
if(x > 10) throw "太大";
}
catch(err) {
message.innerHTML = "错误: " + err;
}
}
```

尝试一下 »

请注意，如果 getElementById 函数出错，上面的例子也会抛出一个错误。

JavaScript Break 和 Continue 语句　　JavaScript 表单验证

JavaScript 语法　　JavaScript 函数定义

JavaScript 调试

在编写 JavaScript 时，如果没有调试工具将是一件很痛苦的事情。

JavaScript 调试

没有调试工具是很难去编写 JavaScript 程序的。

你的代码可能包含语法错误，逻辑错误，如果没有调试工具，这些错误比较难于发现。

通常，如果 JavaScript 出现错误，是不会有提示信息，这样你就无法找到代码错误的位置。

> **Note** 通常，你在编写一个新的 JavaScript 代码过程中都会发生错误。

JavaScript 调试工具

在程序代码中寻找错误叫做代码调试。

调试很难，但幸运的是，很多浏览器都内置了调试工具。

内置的调试工具可以开始或关闭，严重的错误信息会发送给用户。

有了调试工具，我们就可以设置断点 (代码停止执行的位置), 且可以在代码执行时检测变量。

浏览器启用调试工具一般是按下 F12 键，并在调试菜单中选择 "Console" 。

console.log() 方法

如果浏览器支持调试，你可以使用 console.log() 方法在调试窗口上打印 JavaScript 值：

实例

```
a = 5;
b = 6;
c = a + b;
console.log(c);
```

尝试一下 »

设置断点

在调试窗口中，你可以设置 JavaScript 代码的断点。

在每个断点上，都会停止执行 JavaScript 代码，以便于我们检查 JavaScript 变量的值。

在检查完毕后，可以重新执行代码（如播放按钮）。

debugger 关键字

debugger 关键字用于停止执行 JavaScript，并调用调试函数。

这个关键字与在调试工具中设置断点的效果是一样的。

如果没有调试可用，debugger 语句将无法工作。

开启 debugger ，代码在第三行前停止执行。

实例

```
var x = 15 * 5;
debugger;
document.getElementbyId("demo").innerHTML = x;
```

尝试一下 »

主要浏览器的调试工具

通常，浏览器启用调试工具一般是按下 F12 键，并在调试菜单中选择 "Console" 。

各浏览器的步骤如下:

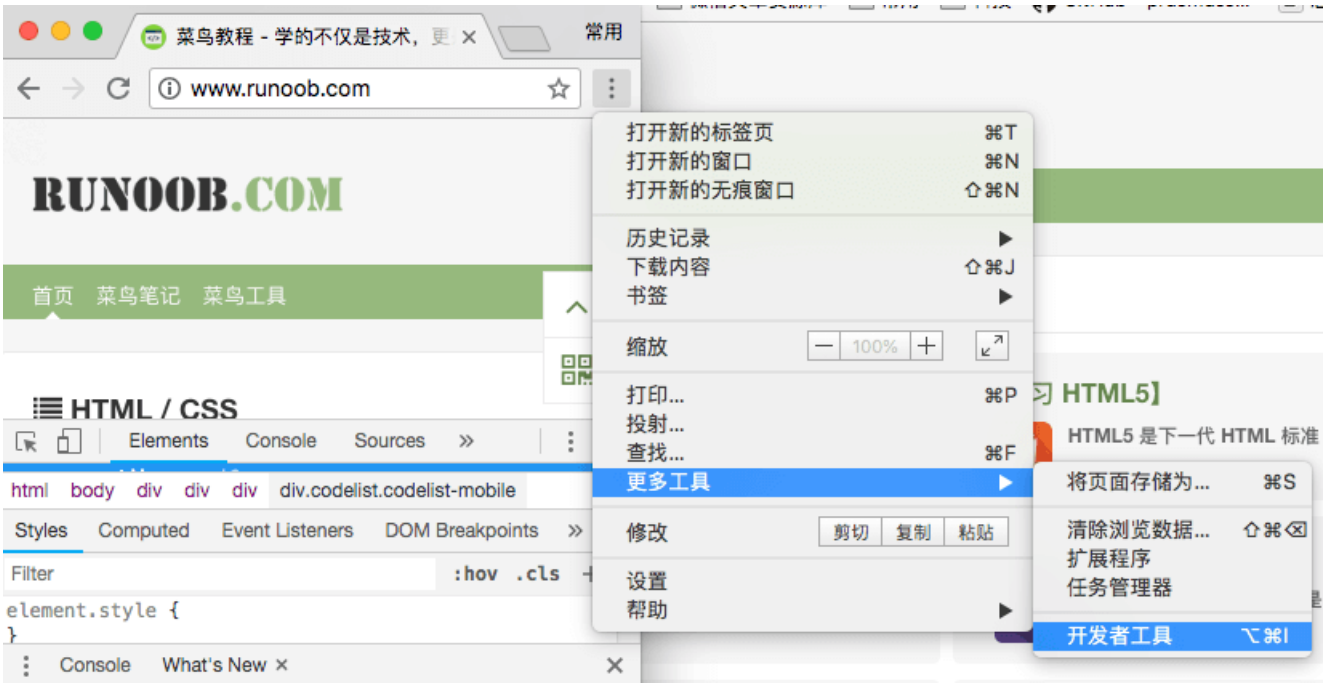
Chrome 浏览器

打开浏览器。

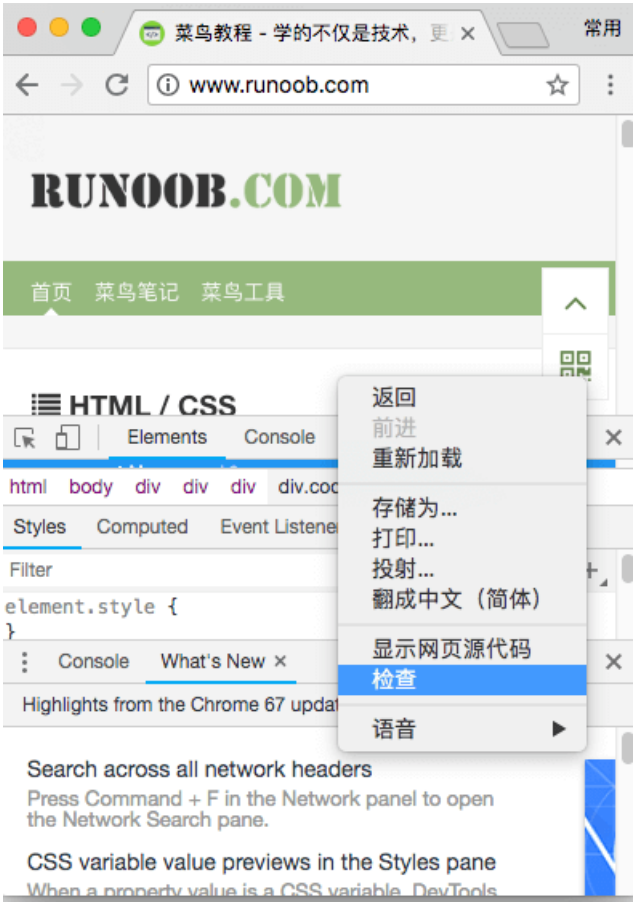
在菜单中选择 **"更多工具"**。

在 **"更多工具"** 中选择 **"开发者工具"**。

最后，选择 Console。



或者你可以右击鼠标选择 **"检查"**，如下图：



Firefox 浏览器

打开浏览器。

右击鼠标，选择 **"查看元素"**。

Safari

打开浏览器。

右击鼠标，选择检查元素。

在底部弹出的窗口中选择"控制台"。

Internet Explorer 浏览器。

打开浏览器。

在菜单中选择工具。

在工具中选择开发者工具。

最后，选择 Console。

Opera

打开浏览器。

点击左上角，选择**"开发者工具"**,选择**"WEB检查器"**。

更简单的方式是：右击鼠标，选择**"查看元素"**

JavaScript 语法

JavaScript 函数定义

点我分享笔记

反馈/建议

JavaScript 变量提升

JavaScript 中，函数及变量的声明都将被提升到函数的最顶部。

JavaScript 中，变量可以在使用后声明，也就是变量可以先使用再声明。

以下两个实例将获得相同的结果：

实例 1

```
x = 5; // 变量 x 设置为 5

elem = document.getElementById("demo"); // 查找元素
elem.innerHTML = x;                     // 在元素中显示 x

var x; // 声明 x
```

尝试一下 »

实例 2

```
var x; // 声明 x
x = 5; // 变量 x 设置为 5

elem = document.getElementById("demo"); // 查找元素
elem.innerHTML = x;                     // 在元素中显示 x
```

尝试一下 »

要理解以上实例就需要理解 "hoisting(变量提升)"。

变量提升：函数声明和变量声明总是会被解释器悄悄地被"提升"到方法体的最顶部。

JavaScript 初始化不会提升

JavaScript 只有声明的变量会提升，初始化的不会。

以下两个实例结果结果不相同：

实例 1

```
var x = 5; // 初始化 x
var y = 7; // 初始化 y

elem = document.getElementById("demo"); // 查找元素
elem.innerHTML = x + " " + y;           // 显示 x 和 y
```

尝试一下 »

实例 2

```
var x = 5; // 初始化 x

elem = document.getElementById("demo"); // 查找元素
elem.innerHTML = x + " " + y;           // 显示 x 和 y

var y = 7; // 初始化 y
```

尝试一下 »

实例 2 的 y 输出了 **undefined**，这是因为变量声明 (var y) 提升了，但是初始化(y = 7) 并不会提升，所以 y 变量是一个未定义的变量。

实例 2 类似以下代码:

```
var x = 5; // 初始化 x

var y;     // 声明 y



elem = document.getElementById("demo"); // 查找元素

elem.innerHTML = x + " " + y;           // 显示 x 和 y



y = 7;    // 设置 y 为 7
```

在头部声明你的变量

对于大多数程序员来说并不知道 JavaScript 变量提升。

如果程序员不能很好的理解变量提升，他们写的程序就容易出现一些问题。

为了避免这些问题，通常我们在每个作用域开始前声明这些变量，这也是正常的 JavaScript 解析步骤，易于我们理解。

Note JavaScript 严格模式(strict mode)不允许使用未声明的变量。
在下一个章节中我们将会学习到 **"严格模式(strict mode)"** 。

□ JavaScript 代码规范　　JavaScript 严格模式(use strict) □

其实主要理解 js 的解析机制就行。
遇到 script 标签的话 js 就进行预解析，将变量 var 和 function 声明提升，但不会执行 function，然后就进入上下文执行，上下文执行还是执行预解析同样操作，直到没有 var 和 function，就开始执行上下文。如:

```
a=5;

show();

var a;

function show(){};
```

预解析：

```
function show(){};

var a;

a=5;

show();
```

需要注意都是函数声明提升直接把整个函数提到执行环境的最顶端。

mowen4个月前 (06-07)
#1

除了以上的函数声明方式外，还可以使用匿名函数的方式。
声明：

```
var 变量名称=function(形参列表){

  //函数体

}
```

调用：

```
变量名称(实参列表)
```

注意：使用匿名函数的方式不存在函数提升，因为函数名称使用变量表示的，只存在变量提升。例：

```
var getName=function(){

  console.log(2);

}



function getName(){

  console.log(1);

}



getName();

//结果为2
```

可能会有人觉得最后输出的结果是 1。但是 getName 是一个变量，因此这个变量的声明也将提升到顶部，而变量的赋值依然保留在原来的位置。需要注意的是，函数优先，虽然函数声明和变量声明都会被提升，但是函数会首先被提升，然后才是变量。

```
//函数、变量声明提升后

function getName(){    //函数声明提升到顶部

  console.log(1);

}



var getName;    //变量声明提升

getName = function(){    //变量赋值依然保留在原来的位置

  console.log(2);

}
```

```
getName();    // 最终输出：2
```

菜菜1个月前 (08-18)

反馈/建议

JavaScript 严格模式(use strict)

JavaScript 严格模式（strict mode）即在严格的条件下运行。

使用 "use strict" 指令

"use strict" 指令在 JavaScript 1.8.5 (ECMAScript5) 中新增。

它不是一条语句，但是是一个字面量表达式，在 JavaScript 旧版本中会被忽略。

"use strict" 的目的是指定代码在严格条件下执行。

严格模式下你不能使用未声明的变量。

> **Note** 支持严格模式的浏览器:
> Internet Explorer 10 +、 Firefox 4+ Chrome 13+、 Safari 5.1+、 Opera 12+。

严格模式声明

严格模式通过在脚本或函数的头部添加 "use strict"; 表达式来声明。

实例中我们可以在浏览器按下 **F12 (或点击"工具>更多工具>开发者工具")** 开启调试模式，查看报错信息。

Gif 图演示如下：

实例

```
"use strict";
x = 3.14;       // 报错 (x 未定义)
```

尝试一下 »

实例

```
"use strict";
myFunction();

function myFunction() {
    y = 3.14;   // 报错 (y 未定义)
}
```

尝试一下 »

在函数内部声明是局部作用域 (只在函数内使用严格模式):

实例

```
x = 3.14;       // 不报错
myFunction();

function myFunction() {
   "use strict";
    y = 3.14;   // 报错 (y 未定义)
}
```

尝试一下 »

为什么使用严格模式:

消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;

消除代码运行的一些不安全之处，保证代码运行的安全;

提高编译器效率，增加运行速度;

为未来新版本的Javascript做好铺垫。

"严格模式"体现了Javascript更合理、更安全、更严谨的发展方向，包括IE 10在内的主流浏览器，都已经支持它，许多大项目已经开始全面拥抱它。另一方面，同样的代码，在"严格模式"中，可能会有不一样的运行结果；一些在"正常模式"下可以运行的语句，在"严格模式"下将不能运行。掌握这些内容，有助于更细致深入地理解Javascript，让你变成一个更好的程序员。

严格模式的限制

不允许使用未声明的变量:

```
"use strict";
x = 3.14;                // 报错 (x 未定义)
```

尝试一下 »

Note 对象也是一个变量。

```
"use strict";
x = {p1:10, p2:20};      // 报错 (x 未定义)
```

尝试一下 »

不允许删除变量或对象。

```
"use strict";
```

```
var x = 3.14;
delete x;                // 报错
```

尝试一下 »

不允许删除函数。

```
"use strict";
function x(p1, p2) {};
delete x;                // 报错
```

尝试一下 »

不允许变量重名:

```
"use strict";
function x(p1, p1) {};   // 报错
```

尝试一下 »

不允许使用八进制:

```
"use strict";
var x = 010;             // 报错
```

尝试一下 »

不允许使用转义字符:

```
"use strict";
var x = \010;            // 报错
```

尝试一下 »

不允许对只读属性赋值:

```
"use strict";
var obj = {};
Object.defineProperty(obj, "x", {value:0, writable:false});

obj.x = 3.14;            // 报错
```

尝试一下 »

不允许对一个使用getter方法读取的属性进行赋值

```
"use strict";
var obj = {get x() {return 0} };

obj.x = 3.14;            // 报错
```

尝试一下 »

不允许删除一个不允许删除的属性：

```
"use strict";
delete Object.prototype; // 报错
```

尝试一下 »

变量名不能使用 "eval" 字符串:

```
"use strict";
var eval = 3.14;         // 报错
```

尝试一下 »

变量名不能使用 "arguments" 字符串:

```
"use strict";
var arguments = 3.14;    // 报错
```

尝试一下 »

不允许使用以下这种语句:

```
"use strict";
with (Math){x = cos(2)}; // 报错
```

尝试一下 »

由于一些安全原因，在作用域 eval() 创建的变量不能被调用：

```
"use strict";
eval ("var x = 2");
alert (x);               // 报错
```

尝试一下 »

禁止this关键字指向全局对象。

```
function f(){
    return !this;
}
// 返回false，因为"this"指向全局对象，"!this"就是false

function f(){
    "use strict";
    return !this;
}
// 返回true，因为严格模式下，this的值为undefined，所以"!this"为true。
```

因此，使用构造函数时，如果忘了加new，this不再指向全局对象，而是报错。

```
function f(){
    "use strict";
    this.a = 1;
};
f();// 报错，this未定义
```

保留关键字

为了向将来Javascript的新版本过渡，严格模式新增了一些保留关键字：

implements

interface

let

package

private

protected

public

static

yield

```
"use strict";
var public = 1500;      // 报错
```

尝试一下 »

Note "use strict" 指令只允许出现在脚本或函数的开头。

JavaScript 变量提升 JavaScript 使用误区

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 »

Note 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 语句会使用恒等计算符(===)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 »

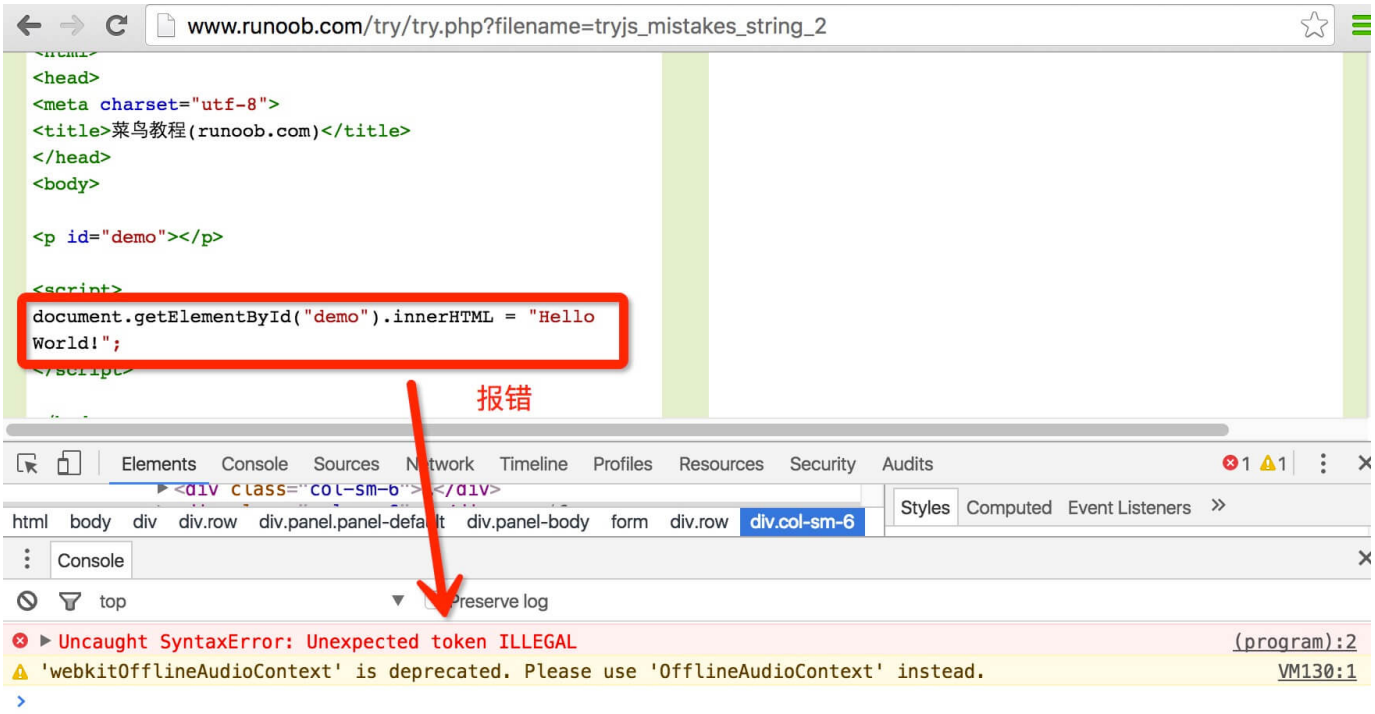
但是，在字符串中直接使用回车换行是会报错的：

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：



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。

> **Note** **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 □

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1年前 (2017-09-05)

#1

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)

反馈/建议

JavaScript 表单

JavaScript 表单验证

HTML 表单验证可以通过 JavaScript 来完成。

以下实例代码用于判断表单字段(fname)值是否存在，如果存在，则弹出信息，否则阻止表单提交：

JavaScript 实例

```
function validateForm() {
var x = document.forms["myForm"]["fname"].value;
if (x == null || x == "") {
alert("需要输入名字。");
return false;
}
}
```

以上 JavaScript 代码可以通过 HTML 代码来调用：

HTML 表单实例

```
<form name="myForm" action="demo_form.php" onsubmit="return validateForm()" method="post">
名字: <input type="text" name="fname">
<input type="submit" value="提交">
</form>
```

尝试一下 »

JavaScript 验证输入的数字

JavaScript 常用于对输入数字的验证：

请输入 1 到 10 之间的数字：

提交

尝试一下 »

HTML 表单自动验证

HTML 表单验证也可以通过浏览器来自动完成。

如果表单字段 (fname) 的值为空, **required** 属性会阻止表单提交：

实例

```
<form action="demo_form.php" method="post">
<input type="text" name="fname" required="required">
<input type="submit" value="提交">
</form>
```

尝试一下 »

Internet Explorer 9 及更早 IE 浏览器不支持表单自动验证。

数据验证

数据验证用于确保用户输入的数据是有效的。

典型的数据验证有：

必需字段是否有输入?

用户是否输入了合法的数据?

在数字字段是否输入了文本?

大多数情况下，数据验证用于确保用户正确输入数据。

数据验证可以使用不同方法来定义，并通过多种方式来调用。

服务端数据验证是在数据提交到服务器上后再验证。

客户端数据验证 side validation是在数据发送到服务器前，在浏览器上完成验证。

HTML 约束验证

HTML5 新增了 HTML 表单的验证方式：约束验证（constraint validation）。

约束验证是表单被提交时浏览器用来实现验证的一种算法。

HTML 约束验证基于：

HTML 输入属性

CSS 伪类选择器

DOM 属性和方法

约束验证 HTML 输入属性

属性	描述
disabled	规定输入的元素不可用
max	规定输入元素的最大值
min	规定输入元素的最小值
pattern	规定输入元素值的模式
required	规定输入元素字段是必需的
type	规定输入元素的类型

完整列表，请查看 HTML 输入属性。

约束验证 CSS 伪类选择器

选择器	描述
:disabled	选取属性为 "disabled" 属性的 input 元素
:invalid	选取无效的 input 元素
:optional	选择没有"required"属性的 input 元素
:required	选择有"required"属性的 input 元素
:valid	选取有效值的 input 元素

完整列表，请查看 CSS 伪类。

□ JavaScript 使用误区　　JavaScript 验证 API □


```
function validateForm(form) {

    var x = form.name.value;
```

```
    if (x == null || x == "") {
        alert("输入不能为空! ");
        return false;
    }
}
```

所有表单调用时都使用：

```
onsubmit="return validateForm(this)"
```

经验证好使。

机械挨踢男5个月前 (04-21)

#1

`onsubmit="return validateForm()"` 为什么不是 `onsubmit="validateForm()"` ？？

onsubmit="validateForm()" 能够调用 **validateForm()** 对表单进行验证，但是在验证不通过的情况下，并不能阻止表单提交。

onsubmit="return validateForm()" 当验证不通过时，返回 false，可以阻止表单提交。

为何？

原来 onsubmit 属性就像是 <form> 这个 html 对象的一个方法名，其值（一字符串）就是其方法体，默认返回 true；

```
onsubmit="return validateForm()"
```

相当于:

```
Form.prototype.onsubmit = function() {
    return validateForm()
};
```

这样复写了 onsubmit 的默认方法（默认返回 true），根据 validateForm() 的结果返回 true 或 false，当验证不通过时，返回 false，onsubmit="return false;" 阻止表单提交。

汉森3周前 (09-11)

反馈/建议

JavaScript 表单验证

JavaScript 表单验证

JavaScript 可用来在数据被送往服务器前对 HTML 表单中的这些输入数据进行验证。

表单数据经常需要使用 JavaScript 来验证其正确性：

验证表单数据是否为空？

验证输入是否是一个正确的email地址？

验证日期是否输入正确？

验证表单输入内容是否为数字型？

必填（或必选）项目

下面的函数用来检查用户是否已填写表单中的必填（或必选）项目。假如必填或必选项为空，那么警告框会弹出，并且函数的返回值为 false，否则函数的返回值则为 true（意味着数据没有问题）：

```
function validateForm()
{
var x=document.forms["myForm"]["fname"].value;
if (x==null || x=="")
{
alert("姓必须填写");
return false;
}
}
```

以上函数在 form 表单提交时被调用:

实例

```
<form name="myForm" action="demo-form.php" onsubmit="return validateForm()" method="post">
姓: <input type="text" name="fname">
<input type="submit" value="提交">
</form>
```

尝试一下 »

E-mail 验证

下面的函数检查输入的数据是否符合电子邮件地址的基本语法。

意思就是说，输入的数据必须包含 @ 符号和点号(.)。同时，@ 不可以是邮件地址的首字符，并且 @ 之后需有至少一个点号：

```
function validateForm(){
var x=document.forms["myForm"]["email"].value;
var atpos=x.indexOf("@");
var dotpos=x.lastIndexOf(".");
if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length){
alert("不是一个有效的 e-mail 地址");
return false;
}
}
```

下面是连同 HTML 表单的完整代码：

实例

```
<form name="myForm" action="demo-form.php" onsubmit="return validateForm();" method="post">
Email: <input type="text" name="email">
<input type="submit" value="提交">
</form>
```

尝试一下 »

JavaScript 验证 API

约束验证 DOM 方法

Property	Description
checkValidity()	如果 input 元素中的数据是合法的返回 true，否则返回 false。

setCustomValidity()	设置 input 元素的 validationMessage 属性，用于自定义错误提示信息的方法。
	使用 setCustomValidity 设置了自定义提示后，validity.customError 就会变成true，则 checkValidity 总是会返回false。如果要重新判断需要取消自定义提示，方式如下：
	`setCustomValidity('')`
	`setCustomValidity(null)`
	`setCustomValidity(undefined)`

以下实例如果输入信息不合法，则返回错误信息：

checkValidity() 方法

```
<input id="id1" type="number" min="100" max="300" required>
<button onclick="myFunction()">验证</button>
<p id="demo"></p>
<script>
function myFunction() {
var inpObj = document.getElementById("id1");
if (inpObj.checkValidity() == false) {
document.getElementById("demo").innerHTML = inpObj.validationMessage;
}
}
</script>
```

尝试一下 »

约束验证 DOM 属性

属性	描述
validity	布尔属性值，返回 input 输入值是否合法
validationMessage	浏览器错误提示信息
willValidate	指定 input 是否需要验证

Validity 属性

input 元素的 **validity 属性**包含一系列关于 validity 数据属性:

属性	描述
customError	设置为 true, 如果设置了自定义的 validity 信息。
patternMismatch	设置为 true, 如果元素的值不匹配它的模式属性。
rangeOverflow	设置为 true, 如果元素的值大于设置的最大值。
rangeUnderflow	设置为 true, 如果元素的值小于它的最小值。
stepMismatch	设置为 true, 如果元素的值不是按照规定的 step 属性设置。
tooLong	设置为 true, 如果元素的值超过了 maxLength 属性设置的长度。
typeMismatch	设置为 true, 如果元素的值不是预期相匹配的类型。
valueMissing	设置为 true，如果元素 (required 属性) 没有值。
valid	设置为 true，如果元素的值是合法的。

实例

如果输入的值大于 100，显示一个信息：

rangeOverflow 属性

```
<input id="id1" type="number" max="100">
<button onclick="myFunction()">验证</button>
<p id="demo"></p>
<script>
function myFunction() {
var txt = "";
if (document.getElementById("id1").validity.rangeOverflow) {
txt = "输入的值太大了";
}
document.getElementById("demo").innerHTML = txt;
}
</script>
```

尝试一下 »

如果输入的值小于 100，显示一个信息：

rangeUnderflow 属性

```
<input id="id1" type="number" min="100" required>
<button onclick="myFunction()">OK</button>
<p id="demo"></p>
<script>
function myFunction() {
var txt = "";
var inpObj = document.getElementById("id1");
if(!isNumeric(inpObj.value)) {
txt = "你输入的不是数字";
} else if (inpObj.validity.rangeUnderflow) {
txt = "输入的值太小了";
} else {
txt = "输入正确";
}
document.getElementById("demo").innerHTML = txt;
}
// 判断输入是否为数字
function isNumeric(n) {
return !isNaN(parseFloat(n)) && isFinite(n);
}
```

```
</script>
```

尝试一下 »

JavaScript 表单 JavaScript HTML DOM 集合(Collection)

1 篇笔记

#1

写笔记

setCustomValidity的用法：

```
var inpObj = document.getElementById("id1");
inpObj.setCustomValidity(''); // 取消自定义提示的方式
if (inpObj.checkValidity() == false) {
    if(inpObj.value==""){
        inpObj.setCustomValidity("不能为空！");
    }else if(inpObj.value<100 || inpObj.value>300){
        inpObj.setCustomValidity("请重新输入数值（100~300之间）!");
    }
    document.getElementById("demo").innerHTML = inpObj.validationMessage;
} else {
    document.getElementById("demo").innerHTML = "输入正确";
}
```

尝试一下 »

不爱李磊的韩梅梅1年前 (2017-05-24)

反馈/建议

JavaScript 总结 JavaScript HTML DOM EventListener

JavaScript 保留关键字

在 JavaScript 中，一些标识符是保留关键字，不能用作变量名或函数名。

JavaScript 标准

所有的现代浏览器完全支持 ECMAScript 3（ES3，JavaScript 的第三版，从 1999 年开始）。

ECMAScript 4（ES4）未通过。

ECMAScript 5（ES5，2009 年发布），是 JavaScript 最新的官方版本。

随着时间的推移，我们开始看到，所有的现代浏览器已经完全支持 ES5。

JavaScript 保留关键字

Javascript 的保留关键字不可以用作变量、标签或者函数名。有些保留关键字是作为 Javascript 以后扩展使用。

abstract	arguments	boolean	break	byte
case	catch	char	class*	const
continue	debugger	default	delete	do
double	else	enum*	eval	export*
extends*	false	final	finally	float
for	function	goto	if	implements
import*	in	instanceof	int	interface
let	long	native	new	null
package	private	protected	public	return
short	static	super*	switch	synchronized
this	throw	throws	transient	true
try	typeof	var	void	volatile
while	with	yield		

* 标记的关键字是 ECMAScript5 中新添加的。

JavaScript 对象、属性和方法

您也应该避免使用 JavaScript 内置的对象、属性和方法的名称作为 Javascript 的变量或函数名：

Array	Date	eval	function	hasOwnProperty
Infinity	isFinite	isNaN	isPrototypeOf	length
Math	NaN	name	Number	Object
prototype	String	toString	undefined	valueOf

Java 保留关键字

JavaScript 经常与 Java 一起使用。您应该避免使用一些 Java 对象和属性作为 JavaScript 标识符：

getClass	java	JavaArray	javaClass	JavaObject	JavaPackage

Windows 保留关键字

JavaScript 可以在 HTML 外部使用。它可在许多其他应用程序中作为编程语言使用。

在 HTML 中，您必须（为了可移植性，您也应该这么做）避免使用 HTML 和 Windows 对象和属性的名称作为 Javascript 的变量及函数名：

alert	all	anchor	anchors	area
assign	blur	button	checkbox	clearInterval
clearTimeout	clientInformation	close	closed	confirm
constructor	crypto	decodeURI	decodeURIComponent	defaultStatus
document	element	elements	embed	embeds
encodeURI	encodeURIComponent	escape	event	fileUpload
focus	form	forms	frame	innerHeight
innerWidth	layer	layers	link	location
mimeTypes	navigate	navigator	frames	frameRate
hidden	history	image	images	offscreenBuffering
open	opener	option	outerHeight	outerWidth
packages	pageXOffset	pageYOffset	parent	parseFloat
parseInt	password	pkcs11	plugin	prompt
propertyIsEnum	radio	reset	screenX	screenY
scroll	secure	select	self	setInterval
setTimeout	status	submit	taint	text
textarea	top	unescape	untaint	window

HTML 事件句柄

除此之外，您还应该避免使用 HTML 事件句柄的名称作为 Javascript 的变量及函数名。

实例：

onblur	onclick	onerror	onfocus
onkeydown	onkeypress	onkeyup	onmouseover
onload	onmouseup	onmousedown	onsubmit

非标准 JavaScript

除了保留关键字，在 JavaScript 实现中也有一些非标准的关键字。

一个实例是 **const** 关键字，用于定义变量。 一些 JavaScript 引擎把 const 当作 var 的同义词。另一些引擎则把 const 当作只读变量的定义。

Const 是 JavaScript 的扩展。JavaScript 引擎支持它用在 Firefox 和 Chrome 中。但是它并不是 JavaScript 标准 ES3 或 ES5 的组成部分。**建议：不要使用它。**

□ JavaScript 总结　　JavaScript HTML DOM EventListener □

JavaScript JSON

JSON 是用于存储和传输数据的格式。

JSON 通常用于服务端向网页传递数据 。

什么是 JSON?

JSON 英文全称 **J**ava**S**cript **O**bject **N**otation

JSON 是一种轻量级的数据交换格式。

JSON是独立的语言 *

JSON 易于理解。

Note * JSON 使用 JavaScript 语法，但是 JSON 格式仅仅是一个文本。
文本可以被任何编程语言读取及作为数据格式传递。

JSON 实例

以下 JSON 语法定义了 sites 对象: 3 条网站信息（对象）的数组:

JSON 实例

```
{"sites":[
{"name":"Runoob", "url":"www.runoob.com"},
{"name":"Google", "url":"www.google.com"},
{"name":"Taobao", "url":"www.taobao.com"}
]}
```

JSON 格式化后为 JavaScript 对象

JSON 格式在语法上与创建 JavaScript 对象代码是相同的。

由于它们很相似，所以 JavaScript 程序可以很容易的将 JSON 数据转换为 JavaScript 对象。

JSON 语法规则

数据为 键/值 对。

数据由逗号分隔。

大括号保存对象

方括号保存数组

JSON 数据 - 一个名称对应一个值

JSON 数据格式为 键/值 对，就像 JavaScript 对象属性。

键/值对包括字段名称（在双引号中），后面一个冒号，然后是值：

```
"name":"Runoob"
```

JSON 对象

JSON 对象保存在大括号内。

就像在 JavaScript 中, 对象可以保存多个 键/值 对：

```
{"name":"Runoob", "url":"www.runoob.com"}
```

JSON 数组

JSON 数组保存在中括号内。

就像在 JavaScript 中, 数组可以包含对象：

```
"sites":[
{"name":"Runoob", "url":"www.runoob.com"},
{"name":"Google", "url":"www.google.com"},
{"name":"Taobao", "url":"www.taobao.com"}
]
```

在以上实例中，对象 "sites" 是一个数组，包含了三个对象。

每个对象为站点的信息（网站名和网站地址）。

JSON 字符串转换为 JavaScript 对象

通常我们从服务器中读取 JSON 数据，并在网页中显示数据。

简单起见，我们网页中直接设置 JSON 字符串 (你还可以阅读我们的 JSON 教程):

首先，创建 JavaScript 字符串，字符串为 JSON 格式的数据：

```
var text = '{ "sites" : [' +
'{ "name":"Runoob" , "url":"www.runoob.com" },' +
'{ "name":"Google" , "url":"www.google.com" },' +
'{ "name":"Taobao" , "url":"www.taobao.com" } ]}';
```

然后，使用 JavaScript 内置函数 JSON.parse() 将字符串转换为 JavaScript 对象:

```
var obj = JSON.parse(text);
```

最后，在你的页面中使用新的 JavaScript 对象：

实例

```
var text = '{ "sites" : [' +
'{ "name":"Runoob" , "url":"www.runoob.com" },' +
'{ "name":"Google" , "url":"www.google.com" },' +
'{ "name":"Taobao" , "url":"www.taobao.com" } ]}';
obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.sites[1].name + " " + obj.sites[1].url;
```

尝试一下 »

相关函数

函数	描述
JSON.parse()	用于将一个 JSON 字符串转换为 JavaScript 对象。
JSON.stringify()	用于将 JavaScript 值转换为 JSON 字符串。

更多 JSON 信息，你可以阅读我们的 JSON 教程。

javascript:void(0) 含义

我们经常会使用到 javascript:void(0) 这样的代码，那么在 JavaScript 中 javascript:void(0) 代表的是什么意思呢？

javascript:void(0) 中最关键的是 void 关键字， void 是 JavaScript 中非常重要的关键字，该操作符指定要计算一个表达式但是不返回值。

语法格式如下：

```
<head>

<script type="text/javascript">

<!--

void func()

javascript:void func()


或者


void(func())

javascript:void(func())

//-->

</script>

</head>
```

下面的代码创建了一个超级链接，当用户点击以后不会发生任何事。

实例

```
<a href="javascript:void(0)">单击此处什么也不会发生</a>
```

尝试一下 »

当用户链接时，void(0) 计算为 0，但 Javascript 上没有任何效果。

以下实例中，在用户点击链接后显示警告信息：

实例

```
<head>
<script type="text/javascript">
<!--
//-->
</script>
</head>
<body>
<a href="javascript:void(alert('Warning!!!'))">点我!</a>
</body>
```

尝试一下 »

以下实例中参数 a 将返回 undefined :

实例

```
<head>
<script type="text/javascript">
<!--
function getValue(){
  var a,b,c;
  a = void ( b = 5, c = 7 );
  document.write('a = ' + a + ' b = ' + b +' c = ' + c );
}
//-->
</script>
</head>
```

尝试一下 »

href="#"与href="javascript:void(0)"的区别

包含了一个位置信息，默认的锚是**#top** 也就是网页的上端。

而javascript:void(0), 仅仅表示一个死链接。

在页面很长的时候会使用 **#** 来定位页面的具体位置，格式为：**# + id**。

如果你要定义一个死链接请使用 javascript:void(0) 。

实例

```
<a href="javascript:void(0);">点我没有反应的!</a>
<a href="#pos">点我定位到指定位置!</a>
<br>
...
<br>
<p id="pos">尾部定位点</p>
```

尝试一下 »

JavaScript 字符串 JavaScript 语法

1 篇笔记

#1

写笔记

void()仅仅是代表不返回任何值，但是括号内的表达式还是要运行，如

```
void(alert("Wornning!"))
```

hehe11个月前 (11-01)

反馈/建议

JavaScript 代码规范

所有的 JavaScript 项目适用同一种规范。

JavaScript 代码规范

代码规范通常包括以下几个方面:

变量和函数的命名规则

空格，缩进，注释的使用规则。

其他常用规范.....

规范的代码可以更易于阅读与维护。

代码规范一般在开发前规定，可以跟你的团队成员来协商设置。

变量名

变量名推荐使用驼峰法来命名(**camelCase**):

```
firstName = "John";
lastName = "Doe";

price = 19.90;
tax = 0.20;

fullPrice = price + (price * tax);
```

空格与运算符

通常运算符 (= + - * /) 前后需要添加空格:

实例:

```
var x = y + z;
var values = ["Volvo", "Saab", "Fiat"];
```

代码缩进

通常使用 4 个空格符号来缩进代码块：

函数:

```
function toCelsius(fahrenheit) {
    return (5 / 9) * (fahrenheit - 32);
}
```

Note

不推荐使用 TAB 键来缩进，因为不同编辑器 TAB 键的解析不一样。

语句规则

简单语句的通用规则:

一条语句通常以分号作为结束符。

实例:

```
var values = ["Volvo", "Saab", "Fiat"];

var person = {
    firstName: "John",
    lastName: "Doe",
    age: 50,
    eyeColor: "blue"
};
```

复杂语句的通用规则:

将左花括号放在第一行的结尾。

左花括号前添加一空格。

将右花括号独立放在一行。

不要以分号结束一个复杂的声明。

函数:

```
function toCelsius(fahrenheit) {
    return (5 / 9) * (fahrenheit - 32);
}
```

循环:

```
for (i = 0; i < 5; i++) {
    x += i;
}
```

条件语句:

```
if (time < 20) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

对象规则

对象定义的规则:

将左花括号与类名放在同一行。

冒号与属性值间有个空格。

字符串使用双引号，数字不需要。

最后一个属性-值对后面不要添加逗号。

将右花括号独立放在一行，并以分号作为结束符号。

实例:

```
var person = {
    firstName: "John",
    lastName: "Doe",
    age: 50,
    eyeColor: "blue"
};
```

短的对象代码可以直接写成一行:

实例:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

每行代码字符小于 80

为了便于阅读每行字符建议小于数 80 个。

如果一个 JavaScript 语句超过了 80 个字符，建议在 运算符或者逗号后换行。

实例:

```
document.getElementById("demo").innerHTML =
    "Hello Runoob.";
```

尝试一下 »

命名规则

一般很多代码语言的命名规则都是类似的，例如:

变量和函数为小驼峰法标识, 即除第一个单词之外，其他单词首字母大写（ **lowerCamelCase**）

全局变量为大写 (**UPPERCASE**)

常量 (如 PI) 为大写 (**UPPERCASE**)

变量命名你是否使用这几种规则： **hyp-hens**, **camelCase**, 或 **under_scores**？

HTML 和 CSS 的横杠(-)字符:

HTML5 属性可以以 data- (如：data-quantity, data-price) 作为前缀。

CSS 使用 - 来连接属性名 (font-size)。

- 通常在 JavaScript 中被认为是减法，所以不允许使用。

下划线:

很多程序员比较喜欢使用下划线(如：date_of_birth), 特别是在 SQL 数据库中。

PHP 语言通常都使用下划线。

帕斯卡拼写法(PascalCase):

帕斯卡拼写法(PascalCase) 在 C 语言中语言较多。

驼峰法：

JavaScript 中通常推荐使用驼峰法，jQuery 及其他 JavaScript 库都使用驼峰法。

Note 变量名不要以 $ 作为开始标记，会与很多 JavaScript 库冲突。

HTML 载入外部 JavaScript 文件

使用简洁的格式载入 JavaScript 文件 (type 属性不是必须的):

```
<script src="myscript.js">
```

使用 JavaScript 访问 HTML 元素

一个糟糕的 HTML 格式可能会导致 JavaScript 执行错误。

以下两个 JavaScript 语句会输出不同结果:

实例

```
var obj = getElementById("Demo")

var obj = getElementById("demo")
```

尝试一下 »

HTML 与 JavaScript 尽量使用相同的命名规则。

访问 HTML(5) 代码规范。

文件扩展名

HTML 文件后缀可以是 **.html** (或 **.htm**)。

CSS 文件后缀是 **.css** 。

JavaScript 文件后缀是 **.js** 。

使用小写文件名

大多 Web 服务器 (Apache, Unix) 对大小写敏感： london.jpg 不能通过 London.jpg 访问。

其他 Web 服务器 (Microsoft, IIS) 对大小写不敏感： london.jpg 可以通过 London.jpg 或 london.jpg 访问。

你必须保持统一的风格，我们建议统一使用小写的文件名。

JavaScript JSON.stringify()　　JavaScript 变量提升

点我分享笔记

反馈/建议

JavaScript 调试　　JavaScript 函数参数

JavaScript 函数定义

JavaScript 使用关键字 **function** 定义函数。

函数可以通过声明定义，也可以是一个表达式。

函数声明

在之前的教程中，你已经了解了函数声明的语法：

```
function functionName(parameters) {
  执行的代码
}
```

函数声明后不会立即执行，会在我们需要的时候调用到。

实例

```
function myFunction(a, b) {
    return a * b;
```

```
}
```

尝试一下 »

Note 分号是用来分隔可执行JavaScript语句。
由于函数声明不是一个可执行语句，所以不以分号结束。

函数表达式

JavaScript 函数可以通过一个表达式定义。

函数表达式可以存储在变量中：

实例

```
var x = function (a, b) {return a * b};
```

尝试一下 »

在函数表达式存储在变量后，变量也可作为一个函数使用：

实例

```
var x = function (a, b) {return a * b};
var z = x(4, 3);
```

尝试一下 »

以上函数实际上是一个 **匿名函数** (函数没有名称)。

函数存储在变量中，不需要函数名称，通常通过变量名来调用。

Note 上述函数以分号结尾，因为它是一个执行语句。

Function() 构造函数

在以上实例中，我们了解到函数通过关键字 **function** 定义。

函数同样可以通过内置的 JavaScript 函数构造器（Function()）定义。

实例

```
var myFunction = new Function("a", "b", "return a * b");

var x = myFunction(4, 3);
```

尝试一下 »

实际上，你不必使用构造函数。上面实例可以写成：

实例

```
var myFunction = function (a, b) {return a * b}

var x = myFunction(4, 3);
```

尝试一下 »

Note 在 JavaScript 中，很多时候，你需要避免使用 **new** 关键字。

函数提升（Hoisting）

在之前的教程中我们已经了解了 "hoisting(提升)"。

提升（Hoisting）是 JavaScript 默认将当前作用域提升到前面去的的行为。

提升（Hoisting）应用在变量的声明与函数的声明。

因此，函数可以在声明之前调用：

```
myFunction(5);

function myFunction(y) {
    return y * y;
}
```

使用表达式定义函数时无法提升。

自调用函数

函数表达式可以 "自调用"。

自调用表达式会自动调用。

如果表达式后面紧跟 () ，则会自动调用。

不能自调用声明的函数。

通过添加括号，来说明它是一个函数表达式：

实例

```
(function () {
    var x = "Hello!!";      // 我将调用自己
})();
```

尝试一下 »

以上函数实际上是一个 **匿名自我调用的函数** (没有函数名)。

函数可作为一个值使用

JavaScript 函数作为一个值使用：

实例

```
function myFunction(a, b) {
    return a * b;
}

var x = myFunction(4, 3);
```

尝试一下 »

JavaScript 函数可作为表达式使用：

实例

```
function myFunction(a, b) {
    return a * b;
}

var x = myFunction(4, 3) * 2;
```

尝试一下 »

函数是对象

在 JavaScript 中使用 **typeof** 操作符判断函数类型将返回 "function" 。

但是JavaScript 函数描述为一个对象更加准确。

JavaScript 函数有 **属性** 和 **方法**。

arguments.length 属性返回函数调用过程接收到的参数个数：

实例

```
function myFunction(a, b) {
    return arguments.length;
}
```

尝试一下 »

toString() 方法将函数作为一个字符串返回:

实例

```
function myFunction(a, b) {
    return a * b;
}

var txt = myFunction.toString();
```

尝试一下 »

Note 函数定义作为对象的属性，称之为对象方法。
函数如果用于创建新的对象，称之为对象的构造函数。

JavaScript 调试 JavaScript 函数参数

1 篇笔记

#1

写笔记

匿名函数自动调用表达式：

```
(function(){})()
```

例如：

```
<p id="demo"></p>



<script>

(function(){document.getElementById("demo").innerHTML = "hello kity";})()

</script>
```

尝试一下 »

珍惜**5**个月前 **(05-04)**

反馈/建议

JavaScript 函数参数

JavaScript 函数对参数的值没有进行任何的检查。

函数显式参数(Parameters)与隐式参数(Arguments)

在先前的教程中，我们已经学习了函数的显式参数:

```
functionName(parameter1, parameter2, parameter3) {
// 要执行的代码.....
}
```

函数显式参数在函数定义时列出。

函数隐式参数在函数调用时传递给函数真正的值。

参数规则

JavaScript 函数定义时显式参数没有指定数据类型。

JavaScript 函数对隐式参数没有进行类型检测。

JavaScript 函数对隐式参数的个数没有进行检测。

默认参数

如果函数在调用时未提供隐式参数，参数会默认设置为： **undefined**

有时这是可以接受的，但是建议最好为参数设置一个默认值：

实例

```
function myFunction(x, y) {
if (y === undefined) {
y = 0;
}
}
```

尝试一下 »

或者，更简单的方式：

实例

```
function myFunction(x, y) {
y = y || 0;
}
```

尝试一下 »

Note 如果y已经定义， y || 返回 y, 因为 y 是 true, 否则返回 0, 因为 undefined 为 false。

如果函数调用时设置了过多的参数，参数将无法被引用，因为无法找到对应的参数名。 只能使用 arguments 对象来调用。

Arguments 对象

JavaScript 函数有个内置的对象 arguments 对象。

argument 对象包含了函数调用的参数数组。

通过这种方式你可以很方便的找到最大的一个参数的值：

实例

```
x = findMax(1, 123, 500, 115, 44, 88);
function findMax() {
var i, max = arguments[0];
if(arguments.length < 2) return max;
for (i = 0; i < arguments.length; i++) {
if (arguments[i] > max) {
max = arguments[i];
}
}
return max;
}
```

尝试一下 »

或者创建一个函数用来统计所有数值的和：

实例

```
x = sumAll(1, 123, 500, 115, 44, 88);
function sumAll() {
var i, sum = 0;
for (i = 0; i < arguments.length; i++) {
sum += arguments[i];
}
return sum;
}
```

尝试一下 »

通过值传递参数

在函数中调用的参数是函数的隐式参数。

JavaScript 隐式参数通过值来传递：函数仅仅只是获取值。

如果函数修改参数的值，不会修改显式参数的初始值（在函数外定义）。

隐式参数的改变在函数外是不可见的。

通过对象传递参数

在JavaScript中，可以引用对象的值。

因此我们在函数内部修改对象的属性就会修改其初始的值。

修改对象属性可作用于函数外部（全局变量）。

修改对象属性在函数外是可见的。

JavaScript 函数定义　　JavaScript 函数调用

JavaScript 函数参数 JavaScript 闭包

JavaScript 函数调用

JavaScript 函数有 4 种调用方式。

每种方式的不同在于 **this** 的初始化。

this 关键字

一般而言，在Javascript中，this指向函数执行时的当前对象。

> **Note** 注意 **this** 是保留关键字，你不能修改 **this** 的值。

调用 JavaScript 函数

在之前的章节中我们已经学会了如何创建函数。

函数中的代码在函数被调用后执行。

作为一个函数调用

实例

```
function myFunction(a, b) {
return a * b;
}
myFunction(10, 2); // myFunction(10, 2) 返回 20
```

尝试一下 »

以上函数不属于任何对象。但是在 JavaScript 中它始终是默认的全局对象。

在 HTML 中默认的全局对象是 HTML 页面本身，所以函数是属于 HTML 页面。

在浏览器中的页面对象是浏览器窗口(window 对象)。以上函数会自动变为 window 对象的函数。

myFunction() 和 window.myFunction() 是一样的：

实例

```
function myFunction(a, b) {
return a * b;
}
window.myFunction(10, 2); // window.myFunction(10, 2) 返回 20
```

尝试一下 »

> **Note** 这是调用 JavaScript 函数常用的方法， 但不是良好的编程习惯
> 全局变量，方法或函数容易造成命名冲突的bug。

全局对象

当函数没有被自身的对象调用时 **this** 的值就会变成全局对象。

在 web 浏览器中全局对象是浏览器窗口（window 对象）。

该实例返回 **this** 的值是 window 对象:

实例

```
function myFunction() {
return this;
}
myFunction(); // 返回 window 对象
```

尝试一下 »

Note 函数作为全局对象调用，会使 **this** 的值成为全局对象。
使用 window 对象作为一个变量容易造成程序崩溃。

函数作为方法调用

在 JavaScript 中你可以将函数定义为对象的方法。

以下实例创建了一个对象 (**myObject**), 对象有两个属性 (**firstName** 和 **lastName**), 及一个方法 (**fullName**):

实例

```
var myObject = {
firstName:"John",
lastName: "Doe",
fullName: function () {
return this.firstName + " " + this.lastName;
}
}
myObject.fullName(); // 返回 "John Doe"
```

尝试一下 »

fullName 方法是一个函数。函数属于对象。 **myObject** 是函数的所有者。

this对象，拥有 JavaScript 代码。实例中 **this** 的值为 **myObject** 对象。

测试以下！修改 **fullName** 方法并返回 **this** 值:

实例

```
var myObject = {
firstName:"John",
lastName: "Doe",
fullName: function () {
return this;
}
}
myObject.fullName(); // 返回 [object Object] (所有者对象)
```

尝试一下 »

Note 函数作为对象方法调用，会使得 **this** 的值成为对象本身。

使用构造函数调用函数

如果函数调用前使用了 **new** 关键字, 则是调用了构造函数。

这看起来就像创建了新的函数，但实际上 JavaScript 函数是重新创建的对象：

实例

```
// 构造函数:
function myFunction(arg1, arg2) {
this.firstName = arg1;
this.lastName = arg2;
}
// This creates a new object
var x = new myFunction("John","Doe");
x.firstName; // 返回 "John"
```

尝试一下 »

构造函数的调用会创建一个新的对象。新对象会继承构造函数的属性和方法。

Note 构造函数中 **this** 关键字没有任何的值。
this 的值在函数调用实例化对象(new object)时创建。

作为函数方法调用函数

在 JavaScript 中, 函数是对象。JavaScript 函数有它的属性和方法。

call() 和 **apply()** 是预定义的函数方法。 两个方法可用于调用函数，两个方法的第一个参数必须是对象本身。

实例

```
function myFunction(a, b) {
return a * b;
}
myObject = myFunction.call(myObject, 10, 2); // 返回 20
```

尝试一下 »

实例

```
function myFunction(a, b) {
return a * b;
}
myArray = [10, 2];
myObject = myFunction.apply(myObject, myArray); // 返回 20
```

尝试一下 »

两个方法都使用了对象本身作为第一个参数。 两者的区别在于第二个参数： apply传入的是一个参数数组，也就是将多个参数组合成为一个数组传入，而call则作为call的参数传入（从第二个参数开始）。

在 JavaScript 严格模式(strict mode)下, 在调用函数时第一个参数会成为 **this** 的值， 即使该参数不是一个对象。

在 JavaScript 非严格模式(non-strict mode)下, 如果第一个参数的值是 null 或 undefined, 它将使用全局对象替代。

Note 通过 call() 或 apply() 方法你可以设置 **this** 的值, 且作为已存在对象的新方法调用。

JavaScript 函数参数　　JavaScript 闭包

2 篇笔记

写笔记

#2

this 是 JavaScript 语言的一个关键字。
它代表函数运行时，自动生成的一个内部对象，只能在函数内部使用。比如：

```
function test() {
    this.x = 1;
}
```

随着函数使用场合的不同，this 的值会发生变化。但是有一个总的原则，那就是this指的是，调用函数的那个对象。

WooKong1年前 (2017-05-26)

#1

作为函数方法调用函数时，此函数执行了相当于 java 中静态函数的功能。

```
<script>
var myObject, myArray;
myObject={
```

```
    name: "hahaha ",
    hsk: "en"
};
function myFunction(a, b) {
    alert(this);
    return this.name +this.hsk;
}
myArray = [10, 2]
myObject = myFunction.apply(myObject, myArray);
document.getElementById("demo").innerHTML = myObject;
</script>
```

尝试一下 »

可以用的频率较高的函数作这样的设置，为对象执行相关操作。

ha oh1周前 (09-23)

反馈/建议

JavaScript 闭包

JavaScript 变量可以是局部变量或全局变量。

私有变量可以用到闭包。

全局变量

函数可以访问由函数内部定义的变量，如：

实例

```
function myFunction() {
var a = 4;
return a * a;
}
```

尝试一下 »

函数也可以访问函数外部定义的变量，如：

实例

```
var a = 4;
function myFunction() {
return a * a;
}
```

尝试一下 »

后面一个实例中， **a** 是一个 **全局** 变量。

在web页面中全局变量属于 window 对象。

全局变量可应用于页面上的所有脚本。

在第一个实例中， **a** 是一个 **局部** 变量。

局部变量只能用于定义它函数内部。对于其他的函数或脚本代码是不可用的。

全局和局部变量即便名称相同，它们也是两个不同的变量。修改其中一个，不会影响另一个的值。

Note 变量声明时如果不使用 **var** 关键字，那么它就是一个全局变量，即便它在函数内定义。

变量生命周期

全局变量的作用域是全局性的，即在整个JavaScript程序中，全局变量处处都在。

而在函数内部声明的变量，只在函数内部起作用。这些变量是局部变量，作用域是局部性的；函数的参数也是局部性的，只在函数内部起作用。

计数器困境

设想下如果你想统计一些数值，且该计数器在所有函数中都是可用的。

你可以使用全局变量，函数设置计数器递增：

实例

```
var counter = 0;
function add() {
return counter += 1;
}
add();
add();
add();
// 计数器现在为 3
```

尝试一下 »

计数器数值在执行 add() 函数时发生变化。

但问题来了，页面上的任何脚本都能改变计数器，即便没有调用 add() 函数。

如果我在函数内声明计数器，如果没有调用函数将无法修改计数器的值：

实例

```
function add() {
var counter = 0;
return counter += 1;
}
add();
add();
add();
// 本意是想输出 3, 但事与愿违，输出的都是 1 !
```

尝试一下 »

以上代码将无法正确输出，每次我调用 add() 函数，计数器都会设置为 1。

JavaScript 内嵌函数可以解决该问题。

JavaScript 内嵌函数

所有函数都能访问全局变量。

实际上，在 JavaScript 中，所有函数都能访问它们上一层的作用域。

JavaScript 支持嵌套函数。嵌套函数可以访问上一层的函数变量。

该实例中，内嵌函数 **plus()** 可以访问父函数的 **counter** 变量：

实例

```
function add() {
var counter = 0;
function plus() {counter += 1;}
plus();
return counter;
}
```

尝试一下 »

如果我们能在外部访问 **plus()** 函数，这样就能解决计数器的困境。

我们同样需要确保 **counter = 0** 只执行一次。

我们需要闭包。

JavaScript 闭包

还记得函数自我调用吗？该函数会做什么？

实例

```
var add = (function () {
var counter = 0;
return function () {return counter += 1;}
})();
add();
add();
add();
// 计数器为 3
```

尝试一下 »

实例解析

变量 **add** 指定了函数自我调用的返回字值。

自我调用函数只执行一次。设置计数器为 0。并返回函数表达式。

add变量可以作为一个函数使用。非常棒的部分是它可以访问函数上一层作用域的计数器。

这个叫作 JavaScript **闭包**。它使得函数拥有私有变量变成可能。

计数器受匿名函数的作用域保护，只能通过 add 方法修改。

Note 闭包是可访问上一层函数作用域里变量的函数，即便上一层函数已经关闭。

JavaScript 函数调用　　JavaScript 正则表达式

JavaScript HTML DOM

通过 HTML DOM，可访问 JavaScript HTML 文档的所有元素。

HTML DOM (文档对象模型)

当网页被加载时，浏览器会创建页面的文档对象模型（Document Object Model）。

HTML DOM 模型被构造为**对象**的树：

HTML DOM 树

```
DOM HTML tree
```

通过可编程的对象模型，JavaScript 获得了足够的能力来创建动态的 HTML。

- JavaScript 能够改变页面中的所有 HTML 元素
- JavaScript 能够改变页面中的所有 HTML 属性
- JavaScript 能够改变页面中的所有 CSS 样式
- JavaScript 能够对页面中的所有事件做出反应

查找 HTML 元素

通常，通过 JavaScript，您需要操作 HTML 元素。

为了做到这件事情，您必须首先找到该元素。有三种方法来做这件事：

- 通过 id 找到 HTML 元素
- 通过标签名找到 HTML 元素
- 通过类名找到 HTML 元素

通过 id 查找 HTML 元素

在 DOM 中查找 HTML 元素的最简单的方法，是通过使用元素的 id。

本例查找 id="intro" 元素：

实例

```
var x=document.getElementById("intro");
```

尝试一下 »

如果找到该元素，则该方法将以对象（在 x 中）的形式返回该元素。

如果未找到该元素，则 x 将包含 null。

通过标签名查找 HTML 元素

本例查找 id="main" 的元素，然后查找 id="main" 元素中的所有 <p> 元素：

实例

```
var x=document.getElementById("main");
var y=x.getElementsByTagName("p");
```

尝试一下 »

通过类名找到 HTML 元素

本例通过 getElementsByClassName 函数来查找 class="intro" 的元素：

实例

```
var x=document.getElementsByClassName("intro");
```

尝试一下 »

HTML DOM 教程

在本教程接下来的篇幅中，您将学到：

- 如何改变 HTML 元素的内容 (innerHTML)
- 如何改变 HTML 元素的样式 (CSS)
- 如何对 HTML DOM 事件做出反应
- 如何添加或删除 HTML 元素

JavaScript 表单验证　　JavaScript HTML DOM 改变 HTML 内容

点我分享笔记

反馈/建议

JavaScript HTML DOM　　JavaScript HTML DOM 改变 CSS

JavaScript HTML DOM - 改变 HTML

HTML DOM 允许 JavaScript 改变 HTML 元素的内容。

改变 HTML 输出流

JavaScript 能够创建动态的 HTML 内容：

今天的日期是： **Tue Oct 02 2018 11:49:21 GMT+0800 (China Standard Time)**

在 JavaScript 中，document.write() 可用于直接向 HTML 输出流写内容。

实例

```
<!DOCTYPE html>
<html>
<body>

<script>
document.write(Date());
</script>

</body>
</html>
```

尝试一下 »

绝对不要在文档(DOM)加载完成之后使用 document.write()。这会覆盖该文档。

改变 HTML 内容

修改 HTML 内容的最简单的方法是使用 innerHTML 属性。

如需改变 HTML 元素的内容，请使用这个语法：

```
document.getElementById(id).innerHTML=新的 HTML
```

本例改变了 <p>元素的内容：

实例

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML="新文本!";
</script>

</body>
</html>
```

尝试一下 »

本例改变了 <h1> 元素的内容：

实例

```
<!DOCTYPE html>
<html>
<body>

<h1 id="header">Old Header</h1>

<script>
var element=document.getElementById("header");
element.innerHTML="新标题";
</script>

</body>
</html>
```

尝试一下 »

实例讲解：

上面的 HTML 文档含有 id="header" 的 <h1> 元素

我们使用 HTML DOM 来获得 id="header" 的元素

JavaScript 更改此元素的内容 (innerHTML)

改变 HTML 属性

如需改变 HTML 元素的属性，请使用这个语法：

```
document.getElementById(id).attribute=新属性值
```

本例改变了 <img> 元素的 src 属性：

实例

```
<!DOCTYPE html>
<html>
<body>

<img id="image" src="smiley.gif">

<script>
document.getElementById("image").src="landscape.jpg";
</script>

</body>
</html>
```

尝试一下 »

实例讲解：

上面的 HTML 文档含有 id="image" 的 <img> 元素

我们使用 HTML DOM 来获得 id="image" 的元素

JavaScript 更改此元素的属性（把 "smiley.gif" 改为 "landscape.jpg"）

JavaScript HTML DOM　　JavaScript HTML DOM 改变 CSS

点我分享笔记

反馈/建议

JavaScript HTML DOM 改变 HTML 内容　　JavaScript HTML DOM 事件

JavaScript HTML DOM - 改变CSS

HTML DOM 允许 JavaScript 改变 HTML 元素的样式。

改变 HTML 样式

如需改变 HTML 元素的样式，请使用这个语法：

```
document.getElementById(id).style.property=新样式
```

下面的例子会改变 <p> 元素的样式：

实例

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
```

```
<body>
<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>
<script>
document.getElementById("p2").style.color="blue";
document.getElementById("p2").style.fontFamily="Arial";
document.getElementById("p2").style.fontSize="larger";
</script>
<p>以上段落通过脚本修改。</p>
</body>
</html>
```

尝试一下 »

使用事件

HTML DOM 允许我们通过触发事件来执行代码。

比如以下事件：

- 元素被点击。
- 页面加载完成。
- 输入框被修改。
- ……

在接下来的章节，你会学到更多关于事件的知识。

本例改变了 id="id1" 的 HTML 元素的样式，当用户点击按钮时：

实例

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id1">我的标题 1</h1>
<button type="button"
onclick="document.getElementById('id1').style.color='red'">
点我!</button>

</body>
</html>
```

尝试一下 »

更多实例

Visibility

如何使元素不可见。您希望元素显示或消失吗？

□ JavaScript HTML DOM 改变 HTML 内容　　JavaScript HTML DOM 事件 □

JavaScript HTML DOM 改变 CSS　　JavaScript HTML DOM 元素 (节点)

JavaScript HTML DOM 事件

HTML DOM 使 JavaScript 有能力对 HTML 事件做出反应。

实例

对事件做出反应

我们可以在事件发生时执行 JavaScript，比如当用户在 HTML 元素上点击时。

如需在用户点击某个元素时执行代码，请向一个 HTML 事件属性添加 JavaScript 代码：

```
onclick=JavaScript
```

HTML 事件的例子：

- 当用户点击鼠标时
- 当网页已加载时
- 当图像已加载时
- 当鼠标移动到元素上时
- 当输入字段被改变时
- 当提交 HTML 表单时
- 当用户触发按键时

在本例中，当用户在 <h1> 元素上点击时，会改变其内容：

实例

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="this.innerHTML='Ooops!'">点击文本!</h1>
</body>
</html>
```

尝试一下 »

本例从事件处理器调用一个函数：

实例

```
<!DOCTYPE html>
<html>
<head>
<script>
function changetext(id)
{
    id.innerHTML="Ooops!";
}
</script>
```

```
</head>
<body>
<h1 onclick="changetext(this)">点击文本!</h1>
</body>
</html>
```

尝试一下 »

HTML 事件属性

如需向 HTML 元素分配 事件，您可以使用事件属性。

实例

向 button 元素分配 onclick 事件：

```
<button onclick="displayDate()">点这里</button>
```

尝试一下 »

在上面的例子中，名为 displayDate 的函数将在按钮被点击时执行。

使用 HTML DOM 来分配事件

HTML DOM 允许您使用 JavaScript 来向 HTML 元素分配事件：

实例

向 button 元素分配 onclick 事件：

```
<script>
document.getElementById("myBtn").onclick=function(){displayDate()};
</script>
```

尝试一下 »

在上面的例子中，名为 displayDate 的函数被分配给 id="myBtn" 的 HTML 元素。

按钮点击时Javascript函数将会被执行。

onload 和 onunload 事件

onload 和 onunload 事件会在用户进入或离开页面时被触发。

onload 事件可用于检测访问者的浏览器类型和浏览器版本，并基于这些信息来加载网页的正确版本。

onload 和 onunload 事件可用于处理 cookie。

实例

```
<body onload="checkCookies()">
```

尝试一下 »

onchange 事件

onchange 事件常结合对输入字段的验证来使用。

下面是一个如何使用 onchange 的例子。当用户改变输入字段的内容时，会调用 upperCase() 函数。

实例

```
<input type="text" id="fname" onchange="upperCase()">
```

尝试一下 »

onmouseover 和 onmouseout 事件

onmouseover 和 onmouseout 事件可用于在用户的鼠标移至 HTML 元素上方或移出元素时触发函数。

实例

一个简单的 onmouseover-onmouseout 实例：

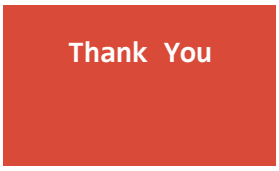
尝试一下 »

onmousedown、onmouseup 以及 onclick 事件

onmousedown, onmouseup 以及 onclick 构成了鼠标点击事件的所有部分。首先当点击鼠标按钮时，会触发 onmousedown 事件，当释放鼠标按钮时，会触发 onmouseup 事件，最后，当完成鼠标点击时，会触发 onclick 事件。

实例

一个简单的 onmousedown-onmouseup 实例：



更多实例

onmousedown 和onmouseup

当用户按下鼠标按钮时，更换一幅图像。

onload

当页面完成加载时，显示一个提示框。

onfocus

当输入字段获得焦点时，改变其背景色。

鼠标事件

当指针移动到元素上方时，改变其颜色；当指针移出文本后，会再次改变其颜色。

JavaScript HTML DOM 改变 CSS

JavaScript HTML DOM 元素 (节点)

点我分享笔记

反馈/建议

JavaScript HTML DOM EventListener

addEventListener() 方法

实例

在用户点击按钮时触发监听事件：

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

尝试一下 »

addEventListener() 方法用于向指定元素添加事件句柄。

addEventListener() 方法添加的事件句柄不会覆盖已存在的事件句柄。

你可以向一个元素添加多个事件句柄。

你可以向同个元素添加多个同类型的事件句柄，如：两个 "click" 事件。

你可以向任何 DOM 对象添加事件监听，不仅仅是 HTML 元素。如： window 对象。

addEventListener() 方法可以更简单的控制事件（冒泡与捕获）。

当你使用 addEventListener() 方法时, JavaScript 从 HTML 标记中分离开来，可读性更强， 在没有控制HTML标记时也可以添加事件监听。

你可以使用 removeEventListener() 方法来移除事件的监听。

语法

```
element.addEventListener(event, function, useCapture);
```

第一个参数是事件的类型 (如 "click" 或 "mousedown").

第二个参数是事件触发后调用的函数。

第三个参数是个布尔值用于描述事件是冒泡还是捕获。该参数是可选的。

Note 注意:不要使用 "on" 前缀。 例如，使用 "click" ,而不是使用 "onclick"。

向原元素添加事件句柄

实例

当用户点击元素时弹出 "Hello World!" :

```
element.addEventListener("click", function(){ alert("Hello World!"); });
```

尝试一下 »

你可以使用函数名，来引用外部函数:

实例

当用户点击元素时弹出 "Hello World!" :

```
element.addEventListener("click", myFunction);

function myFunction() {
    alert ("Hello World!");
}
```

尝试一下 »

向同一个元素中添加多个事件句柄

addEventListener() 方法允许向同一个元素添加多个事件，且不会覆盖已存在的事件：

实例

```
element.addEventListener("click", myFunction);
element.addEventListener("click", mySecondFunction);
```

尝试一下 »

你可以向同个元素添加不同类型的事件：

实例

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

尝试一下 »

向 Window 对象添加事件句柄

addEventListener() 方法允许你在 HTML DOM 对象添加事件监听， HTML DOM 对象如： HTML 元素, HTML 文档, window 对象。或者其他支出的事件对象如: xmlHttpRequest 对象。

实例

当用户重置窗口大小时添加事件监听：

```
window.addEventListener("resize", function(){
    document.getElementById("demo").innerHTML = sometext;
});
```

尝试一下 »

传递参数

当传递参数值时，使用"匿名函数"调用带参数的函数：

实例

```
element.addEventListener("click", function(){ myFunction(p1, p2); });
```

尝试一下 »

事件冒泡或事件捕获？

事件传递有两种方式：冒泡与捕获。

事件传递定义了元素事件触发的顺序。 如果你将 <p> 元素插入到 元素中，用户点击 <p> 元素, 哪个元素的 "click" 事件先被触发呢？

在 *冒泡* 中，内部元素的事件会先被触发，然后再触发外部元素，即： <p> 元素的点击事件先触发，然后会触发 元素的点击事件。

在 *捕获* 中，外部元素的事件会先被触发，然后才会触发内部元素的事件，即： 元素的点击事件先触发 ，然后再触发 <p> 元素的点击事件。

addEventListener() 方法可以指定 "useCapture" 参数来设置传递类型：

```
addEventListener(event, function, useCapture);
```

默认值为 false, 即冒泡传递，当值为 true 时, 事件使用捕获传递。

实例

```
document.getElementById("myDiv").addEventListener("click", myFunction, true);
```

尝试一下 »

removeEventListener() 方法

removeEventListener() 方法移除由 addEventListener() 方法添加的事件句柄:

实例

```
element.removeEventListener("mousemove", myFunction);
```

尝试一下 »

浏览器支持

表格中的数字表示支持该方法的第一个浏览器的版本号。

方法					
addEventListener()	1.0	9.0	1.0	1.0	7.0
removeEventListener()	1.0	9.0	1.0	1.0	7.0

注意： IE 8 及更早 IE 版本，Opera 7.0及其更早版本不支持 addEventListener() 和 removeEventListener() 方法。但是，对于这类浏览器版本可以使用 detachEvent() 方法来移除事件句柄:

```
element.attachEvent(event, function);
element.detachEvent(event, function);
```

实例

跨浏览器解决方法:

```
var x = document.getElementById("myBtn");
if (x.addEventListener) {                    // 所有主流浏览器，除了 IE 8 及更早版本
    x.addEventListener("click", myFunction);
} else if (x.attachEvent) {                  // IE 8 及更早版本
    x.attachEvent("onclick", myFunction);
}
```

尝试一下 »

HTML DOM 事件对象参考手册

所有 HTML DOM 事件，可以查看我们完整的 HTML DOM Event 对象参考手册。

JavaScript 保留关键字 JavaScript JSON

点我分享笔记

反馈/建议

JavaScript HTML DOM 元素 (节点)

本章节介绍如何向文档中添加和移除元素(节点)。

创建新的 HTML 元素 (节点) - appendChild()

要创建新的 HTML 元素 (节点)需要先创建一个元素，然后在已存在的元素中添加它。

实例

```
<div id="div1">
<p id="p1">这是一个段落。</p>
<p id="p2">这是另外一个段落。</p>
</div>
<script>
var para = document.createElement("p");
var node = document.createTextNode("这是一个新的段落。");
para.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(para);
</script>
```

尝试一下 »

实例解析

以下代码是用于创建 <p> 元素:

```
var para = document.createElement("p");
```

为 <p> 元素添加文本节点：

```
var node = document.createTextNode("这是一个新的段落。");
```

将文本节点添加到 <p> 元素中：

```
para.appendChild(node);
```

最后，在一个已存在的元素中添加 p 元素。

查找已存在的元素：

```
var element = document.getElementById("div1");
```

添加到已存在的元素中:

```
element.appendChild(para);
```

创建新的 HTML 元素 (节点) - insertBefore()

以上的实例我们使用了 `appendChild()` 方法，它用于添加新元素到尾部。

如果我们需要将新元素添加到开始位置，可以使用 `insertBefore()` 方法:

实例

```
<div id="div1">
<p id="p1">这是一个段落。</p>
<p id="p2">这是另外一个段落。</p>
</div>
<script>
```

```
var para = document.createElement("p");
var node = document.createTextNode("这是一个新的段落。");
para.appendChild(node);
var element = document.getElementById("div1");
var child = document.getElementById("p1");
element.insertBefore(para, child);
</script>
```

尝试一下 »

移除已存在的元素

要移除一个元素，你需要知道该元素的父元素。

实例

```
<div id="div1">
<p id="p1">这是一个段落。</p>
<p id="p2">这是另外一个段落。</p>
</div>
<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.removeChild(child);
</script>
```

尝试一下 »

注意：早期的 Internet Explorer 浏览器不支持 node.remove() 方法。

实例解析

HTML 文档中 元素包含两个子节点 (两个 <p> 元素):

```
<div id="div1">

<p id="p1">这是一个段落。</p>

<p id="p2">这是另外一个段落。</p>

</div>
```

查找 id="div1" 的元素:

```
var parent = document.getElementById("div1");
```

查找 id="p1" 的 <p> 元素:

```
var child = document.getElementById("p1");
```

从父元素中移除子节点：

```
parent.removeChild(child);
```

如果能够在不引用父元素的情况下删除某个元素，就太好了。
不过很遗憾。DOM 需要清楚您需要删除的元素，以及它的父元素。

以下代码是已知要查找的子元素，然后查找其父元素，再删除这个子元素（删除节点必须知道父节点）：

```
var child = document.getElementById("p1");

child.parentNode.removeChild(child);
```

替换 HTML 元素 - replaceChild()

我们可以使用 replaceChild() 方法来替换 HTML DOM 中的元素。

实例

```
<div id="div1">
<p id="p1">这是一个段落。</p>
<p id="p2">这是另外一个段落。</p>
</div>
<script>
var para = document.createElement("p");
var node = document.createTextNode("这是一个新的段落。");
para.appendChild(node);
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.replaceChild(para, child);
</script>
```

尝试一下 »

HTML DOM 教程

在我们的 JavaScript 教程的 HTML DOM 部分，您已经学到了：

- 如何改变 HTML 元素的内容 (innerHTML)
- 如何改变 HTML 元素的样式 (CSS)
- 如何对 HTML DOM 事件作出反应
- 如何添加或删除 HTML 元素

如果您希望学到更多有关使用 JavaScript 访问 HTML DOM 的知识，请访问我们完整的 HTML DOM 教程。

JavaScript HTML DOM 事件　　JavaScript 对象

JavaScript 验证 API　　JavaScript HTML DOM 节点列表

JavaScript HTML DOM 集合(Collection)

本章节介绍如何向文档中添加和移除元素(节点)。

HTMLCollection 对象

getElementsByTagName() 方法返回 HTMLCollection 对象。

HTMLCollection 对象类似 HTML 元素的一个数组。

以下代码获取文档所有的 <p> 元素：

实例

```
var x = document.getElementsByTagName("p");
```

集合中的元素可以通过索引(以 0 为起始位置)来访问。

访问第二个 <p> 元素可以是以下代码:

```
y = x[1];
```

尝试一下 »

HTMLCollection 对象 length 属性

HTMLCollection 对象的 length 属性定义了集合中元素的数量。

实例

```
var myCollection = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML = myCollection.length;
```

尝试一下 »

实例解析

获取 <p> 元素的集合：

```
var myCollection = document.getElementsByTagName("p");
```

显示集合元素个数：

```
document.getElementById("demo").innerHTML = myCollection.length;
```

集合 length 属性常用于遍历集合中的元素。

实例

修改所有 <p> 元素的背景颜色:

```
var myCollection = document.getElementsByTagName("p");
var i;
for (i = 0; i < myCollection.length; i++) {
myCollection[i].style.backgroundColor = "red";
}
```

尝试一下 »

注意

HTMLCollection 不是一个数组！

HTMLCollection 看起来可能是一个数组，但其实不是。

你可以像数组一样，使用索引来获取元素。

HTMLCollection 无法使用数组的方法： valueOf(), pop(), push(), 或 join() 。

JavaScript HTML DOM 节点列表

NodeList 对象是一个从文档中获取的节点列表 (集合) 。

NodeList 对象类似 HTMLCollection 对象。

一些旧版本浏览器中的方法（如：**getElementsByClassName()**）返回的是 NodeList 对象，而不是 HTMLCollection 对象。

所有浏览器的 **childNodes** 属性返回的是 NodeList 对象。

大部分浏览器的 **querySelectorAll()** 返回 NodeList 对象。

以下代码选取了文档中所有的 <p> 节点：

实例

```
var myNodeList = document.querySelectorAll("p");
```

NodeList 中的元素可以通过索引(以 0 为起始位置)来访问。

访问第二个 <p> 元素可以是以下代码:

```
y = myNodeList[1];
```

尝试一下 »

NodeList 对象 length 属性

NodeList 对象 length 属性定义了节点列表中元素的数量。

实例

```
var myNodelist = document.querySelectorAll("p");
document.getElementById("demo").innerHTML = myNodelist.length;
```

尝试一下 »

实例解析

获取 <p> 元素的集合：

```
var myNodelist = document.querySelectorAll("p");
```

显示节点列表的元素个数：

```
document.getElementById("demo").innerHTML = myNodelist.length;
```

length 属性常用于遍历节点列表。

实例

修改节点列表中所有 <p> 元素的背景颜色:

```
var myNodelist = document.querySelectorAll("p");
var i;
for (i = 0; i < myNodelist.length; i++) {
myNodelist[i].style.backgroundColor = "red";
}
```

尝试一下 »

HTMLCollection 与 NodeList 的区别

HTMLCollection 是 HTML 元素的集合。

NodeList 是一个文档节点的集合。

NodeList 与 HTMLCollection 有很多类似的地方。

NodeList 与 HTMLCollection 都与数组对象有点类似，可以使用索引 (0, 1, 2, 3, 4, ...) 来获取元素。

NodeList 与 HTMLCollection 都有 length 属性。

HTMLCollection 元素可以通过 name，id 或索引来获取。

NodeList 只能通过索引来获取。

只有 NodeList 对象有包含属性节点和文本节点。

> ***节点列表不是一个数组！***
>
> *节点列表看起来可能是一个数组，但其实不是。*
>
> *你可以像数组一样，使用索引来获取元素。*
>
> *节点列表无法使用数组的方法： valueOf(), pop(), push(), 或 join() 。*

JavaScript HTML DOM 集合(Collection)

1 篇笔记
#1

写笔记

```
pcoll=document.querySelectorAll("p")



plist=document.getElementsByTagName("p")
```

以上 **pcoll** 返回的就是固定的值。
而获取 **plist** 后, 若 html 页面有变化且刚好添加或移除了 p 标签, 此时plist也会跟着变。

athey3个月前 (07-03)

反馈/建议

JavaScript HTML DOM 元素 (节点)　　JavaScript Number 对象

JavaScript 对象

JavaScript 中的所有事物都是对象：字符串、数值、数组、函数...

此外，JavaScript 允许自定义对象。

所有事物都是对象

JavaScript 提供多个内建对象，比如 String、Date、Array 等等。 对象只是带有属性和方法的特殊数据类型。

布尔型可以是一个对象。

数字型可以是一个对象。

字符串也可以是一个对象

日期是一个对象

数学和正则表达式也是对象

数组是一个对象

甚至函数也可以是对象

JavaScript 对象

对象只是一种特殊的数据。对象拥有**属性**和**方法**。

访问对象的属性

属性是与对象相关的值。

访问对象属性的语法是：

```
objectName.propertyName
```

这个例子使用了 String 对象的 length 属性来获得字符串的长度：

```
var message="Hello World!";
var x=message.length;
```

在以上代码执行后，x 的值将是：

```
12
```

访问对象的方法

方法是能够在对象上执行的动作。

您可以通过以下语法来调用方法：

```
objectName.methodName()
```

这个例子使用了 String 对象的 toUpperCase() 方法来将文本转换为大写：

```
var message="Hello world!";
var x=message.toUpperCase();
```

在以上代码执行后，x 的值将是：

```
HELLO WORLD!
```

创建 JavaScript 对象

通过 JavaScript，您能够定义并创建自己的对象。

创建新对象有两种不同的方法：

定义并创建对象的实例

使用函数来定义对象，然后创建新的对象实例

创建直接的实例

这个例子创建了对象的一个新实例，并向其添加了四个属性：

实例

```
person=new Object();
person.firstname="John";
person.lastname="Doe";
person.age=50;
person.eyecolor="blue";
```

尝试一下 »

替代语法（使用对象 literals）：

实例

```
person={firstname:"John",lastname:"Doe",age:50,eyecolor:"blue"};
```

尝试一下 »

使用对象构造器

本例使用函数来构造对象：

实例

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;
}
```

尝试一下 »

在JavaScript中，this通常指向的是我们正在执行的函数本身，或者是指向该函数所属的对象（运行时）

创建 JavaScript 对象实例

一旦您有了对象构造器，就可以创建新的对象实例，就像这样：

```
var myFather=new person("John","Doe",50,"blue");
var myMother=new person("Sally","Rally",48,"green");
```

把属性添加到 JavaScript 对象

您可以通过为对象赋值，向已有对象添加新属性：

假设 personObj 已存在 - 您可以为其添加这些新属性：firstname、lastname、age 以及 eyecolor：

```
person.firstname="John";
person.lastname="Doe";
person.age=30;
person.eyecolor="blue";

x=person.firstname;
```

在以上代码执行后，x 的值将是：

```
John
```

把方法添加到 JavaScript 对象

方法只不过是附加在对象上的函数。

在构造器函数内部定义对象的方法：

```
function person(firstname,lastname,age,eyecolor)

{

    this.firstname=firstname;

    this.lastname=lastname;

    this.age=age;
```

```
    this.eyecolor=eyecolor;



    this.changeName=changeName;

    function changeName(name)

    {

        this.lastname=name;

    }

}
```

changeName() 函数 name 的值赋给 person 的 lastname 属性。

现在您可以试一下：

```
myMother.changeName("Doe");
```

尝试一下 »

JavaScript 类

JavaScript 是面向对象的语言，但 JavaScript 不使用类。

在 JavaScript 中，不会创建类，也不会通过类来创建对象（就像在其他面向对象的语言中那样）。

JavaScript 基于 prototype，而不是基于类的。

JavaScript for...in 循环

JavaScript for...in 语句循环遍历对象的属性。

语法

```
for (variable in object)

{

    执行的代码.....

}
```

注意： for...in 循环中的代码块将针对每个属性执行一次。

实例

循环遍历对象的属性：

实例

```
var person={fname:"John",lname:"Doe",age:25};
for (x in person)
{
txt=txt + person[x];
}
```

尝试一下 »

点我分享笔记

反馈/建议

JavaScript Number 对象

JavaScript 只有一种数字类型。

可以使用也可以不使用小数点来书写数字。

JavaScript 数字

JavaScript 数字可以使用也可以不使用小数点来书写：

实例

```
var pi=3.14;    // 使用小数点
var x=34;       // 不使用小数点
```

极大或极小的数字可通过科学（指数）计数法来写：

实例

```
var y=123e5;    // 12300000
var z=123e-5;   // 0.00123
```

所有 JavaScript 数字均为 64 位

JavaScript 不是类型语言。与许多其他编程语言不同，JavaScript 不定义不同类型的数字，比如整数、短、长、浮点等等。

在JavaScript中，数字不分为整数类型和浮点型类型，所有的数字都是由 浮点型类型。JavaScript采用IEEE754标准定义的64位浮点格式表示数字，它能表示最大值为±1.7976931348623157 x 10308，最小值为±5 x 10 -324

值 (aka Fraction/Mantissa)	指数	Sign
52 bits (0 - 51)	11 bits (50 - 62)	1 bit (63)

精度

整数（不使用小数点或指数计数法）最多为 15 位。

实例

```
var x = 999999999999999; // x 为 999999999999999
var y = 9999999999999999; // y 为 10000000000000000
```

尝试一下 »

小数的最大位数是 17，但是浮点运算并不总是 100% 准确：

实例

```
var x = 0.2+0.1; // 输出结果为 0.30000000000000004
```

尝试一下 »

八进制和十六进制

如果前缀为 0，则 JavaScript 会把数值常量解释为八进制数，如果前缀为 0 和 "x"，则解释为十六进制数。

实例

```
var y = 0377;
var z = 0xFF;
```

尝试一下 »

绝不要在数字前面写零，除非您需要进行八进制转换。

默认情况下，JavaScript 数字为十进制显示。

但是你可以使用 toString() 方法 输出16进制、8进制、2进制。

实例

```
var myNumber=128;
myNumber.toString(16);   // 返回 80
myNumber.toString(8);    // 返回 200
myNumber.toString(2);    // 返回 10000000
```

尝试一下 »

无穷大（Infinity）

当数字运算结果超过了JavaScript所能表示的数字上限（溢出），结果为一个特殊的无穷大（infinity）值，在JavaScript中以Infinity表示。同样地，当负数的值超过了JavaScript所能表示的负数范围，结果为负无穷大，在JavaScript中以-Infinity表示。无穷大值的行为特性和我们所期望的是一致的：基于它们的加、减、乘和除运算结果还是无穷大（当然还保留它们的正负号）。

实例

```
myNumber=2;
while (myNumber!=Infinity)
{
    myNumber=myNumber*myNumber; // 重复计算直到 myNumber 等于 Infinity
}
```

尝试一下 »

除以0也产生了无限:

实例

```
var x = 2/0;
var y = -2/0;
```

尝试一下 »

NaN - 非数字值

NaN 属性是代表非数字值的特殊值。该属性用于指示某个值不是数字。可以把 Number 对象设置为该值，来指示其不是数字值。

你可以使用 isNaN() 全局函数来判断一个值是否是 NaN 值。

实例

```
var x = 1000 / "Apple";
isNaN(x); // 返回 true
var y = 100 / "1000";
isNaN(y); // 返回 false
```

尝试一下 »

除以0是无穷大，无穷大是一个数字:

实例

```
var x = 1000 / 0;
isNaN(x); // 返回 false
```

尝试一下 »

数字可以是数字或者对象

数字可以私有数据进行初始化，就像 x = 123;

JavaScript 数字对象初始化数据， var y = new Number(123);

实例

```
var x = 123;
var y = new Number(123);
typeof(x) // 返回 Number
typeof(y) // 返回 Object
```

尝试一下 »

实例

```
var x = 123;
var y = new Number(123);
(x === y) // 为 false，因为 x 是一个数字，y 是一个对象
```

尝试一下 »

数字属性

- MAX_VALUE
- MIN_VALUE
- NEGATIVE_INFINITY
- POSITIVE_INFINITY
- NaN
- prototype
- constructor

数字方法

- toExponential()
- toFixed()
- toPrecision()

toString()

valueOf()

JavaScript 对象 JavaScript 字符串（String）对象

JavaScript 字符串（String） 对象

String 对象用于处理已有的字符块。

JavaScript 字符串

一个字符串用于存储一系列字符就像 "John Doe".

一个字符串可以使用单引号或双引号：

实例

```
var carname="Volvo XC60";
var carname='Volvo XC60';
```

你使用位置（索引）可以访问字符串中任何的字符：

实例

```
var character=carname[7];
```

字符串的索引从零开始, 所以字符串第一字符为 [0],第二个字符为 [1], 等等。

你可以在字符串中使用引号，如下实例：

实例

```
var answer="It's alright";
var answer="He is called 'Johnny'";
var answer='He is called "Johnny"';
```

或者你可以在字符串中使用转义字符(\)使用引号：

实例

```
var answer='It\'s alright';
var answer="He is called \"Johnny\"";
```

尝试一下 »

字符串（String）

字符串（String）使用长度属性**length**来计算字符串的长度：

实例

```
var txt="Hello World!";
document.write(txt.length);

var txt="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
document.write(txt.length);
```

尝试一下 »

在字符串中查找字符串

字符串使用 indexOf() 来定位字符串中某一个指定的字符首次出现的位置：

实例

```
var str="Hello world, welcome to the universe.";
var n=str.indexOf("welcome");
```

尝试一下 »

如果没找到对应的字符函数返回-1

lastIndexOf() 方法在字符串末尾开始查找字符串出现的位置。

内容匹配

match()函数用来查找字符串中特定的字符，并且如果找到的话，则返回这个字符。

实例

```
var str="Hello world!";
document.write(str.match("world") + "<br>");
document.write(str.match("World") + "<br>");
document.write(str.match("world!"));
```

尝试一下 »

替换内容

replace() 方法在字符串中用某些字符替换另一些字符。

实例

```
str="Please visit Microsoft!"
var n=str.replace("Microsoft","Runoob");
```

尝试一下 »

字符串大小写转换

字符串大小写转换使用函数 **toUpperCase()** / **toLowerCase()**:

实例

```
var txt="Hello World!";       // String
var txt1=txt.toUpperCase();   // txt1 文本会转换为大写
var txt2=txt.toLowerCase();   // txt2 文本会转换为小写
```

尝试一下 »

字符串转为数组

字符串使用**split()**函数转为数组:

实例

```
txt="a,b,c,d,e"   // String
txt.split(",");   // 使用逗号分隔
txt.split(" ");   // 使用空格分隔
txt.split("|");   // 使用竖线分隔
```

尝试一下 »

特殊字符

Javascript 中可以使用反斜线（\）插入特殊符号，如：撇号,引号等其他特殊符号。

查看如下 JavaScript 代码:

```
var txt="We are the so-called "Vikings" from the north.";
document.write(txt);
```

在JavaScript中，字符串的开始和停止使用单引号或双引号。这意味着，上面的字符串将被切成： We are the so-called

解决以上的问题可以使用反斜线来转义引号：

```
var txt="We are the so-called \"Vikings\" from the north.";
document.write(txt);
```

JavaScript将输出正确的文本字符串：We are the so-called "Vikings" from the north.

下表列出其他特殊字符，可以使用反斜线转义特殊字符：

代码	输出
\'	单引号
\"	双引号
\\	斜杆
\n	换行
\r	回车
\t	tab
\b	空格
\f	换页

字符串属性和方法

属性:

- length
- prototype
- constructor

方法:

- charAt()
- charCodeAt()
- concat()
- fromCharCode()
- indexOf()

lastIndexOf()

match()

replace()

search()

slice()

split()

substr()

substring()

toLowerCase()

toUpperCase()

valueOf()

更多方法与属性介绍可以参考：JavaScript String 对象

JavaScript Number 对象 JavaScript Date（日期）对象

3 篇笔记

写笔记

反馈/建议

JavaScript 字符串（String）对象 JavaScript Array（数组）对象

JavaScript Date（日期） 对象

日期对象用于处理日期和时间。

在线实例

如何使用 Date() 方法获得当日的日期。

getFullYear()

使用 getFullYear() 获取年份。

getTime()

getTime() 返回从 1970 年 1 月 1 日至今的毫秒数。

setFullYear()

如何使用 setFullYear() 设置具体的日期。

toUTCString()

如何使用 toUTCString() 将当日的日期（根据 UTC）转换为字符串。

getDay()

如何使用 getDay() 和数组来显示星期，而不仅仅是数字。

Display a clock

如何在网页上显示一个钟表。

完整的 Date 对象参考手册

我们提供 JavaScript Date 对象参考手册，其中包括所有可用于日期对象的属性和方法。JavaScript Date 对象参考手册。

该手册包含了对每个属性和方法的详细描述以及相关实例。

创建日期

Date 对象用于处理日期和时间。

可以通过 new 关键词来定义 Date 对象。以下代码定义了名为 myDate 的 Date 对象：

有四种方式初始化日期:

```
new Date() // 当前日期和时间
new Date(milliseconds) //返回从 1970 年 1 月 1 日至今的毫秒数
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

上面的参数大多数都是可选的，在不指定的情况下，默认参数是0。

实例化一个日期的一些例子：

```
var today = new Date()
var d1 = new Date("October 13, 1975 11:13:00")
var d2 = new Date(79,5,24)
var d3 = new Date(79,5,24,11,33,0)
```

设置日期

通过使用针对日期对象的方法，我们可以很容易地对日期进行操作。

在下面的例子中，我们为日期对象设置了一个特定的日期 (2010 年 1 月 14 日)：

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
```

在下面的例子中，我们将日期对象设置为 5 天后的日期：

```
var myDate=new Date();
myDate.setDate(myDate.getDate()+5);
```

注意: 如果增加天数会改变月份或者年份，那么日期对象会自动完成这种转换。

两个日期比较

日期对象也可用于比较两个日期。

下面的代码将当前日期与 2100 年 1 月 14 日做了比较：

```
var x=new Date();

x.setFullYear(2100,0,14);

var today = new Date();



if (x>today)

{

    alert("今天是2100年1月14日之前");

}

else

{
```

```
    alert("今天是2100年1月14日之后");

}
```

JavaScript 字符串（String）对象 JavaScript Array（数组）对象

JavaScript Date（日期）对象 JavaScript Boolean（布尔）对象

JavaScript Array（数组） 对象

数组对象的作用是：使用单独的变量名来存储一系列的值。

在线实例

创建数组, 为其赋值：

实例

```
var mycars = new Array();
mycars[0] = "Saab";
mycars[1] = "Volvo";
mycars[2] = "BMW";
```

尝试一下 »

页面底部你可以找到更多的实例。

什么是数组?

数组对象是使用单独的变量名来存储一系列的值。

如果你有一组数据（例如：车名字），存在单独变量如下所示：

```
var car1="Saab";
var car2="Volvo";
var car3="BMW";
```

然而，如果你想从中找出某一辆车？并且不是3辆，而是300辆呢？这将不是一件容易的事！

最好的方法就是用数组。

数组可以用一个变量名存储所有的值，并且可以用变量名访问任何一个值。

数组中的每个元素都有自己的的ID，以便它可以很容易地被访问到。

创建一个数组

创建一个数组，有三种方法。

下面的代码定义了一个名为 myCars的数组对象：

1: 常规方式:

```
var myCars=new Array();
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";
```

2: 简洁方式:

```
var myCars=new Array("Saab","Volvo","BMW");
```

3: 字面:

```
var myCars=["Saab","Volvo","BMW"];
```

访问数组

通过指定数组名以及索引号码，你可以访问某个特定的元素。

以下实例可以访问myCars数组的第一个值：

```
var name=myCars[0];
```

以下实例修改了数组 myCars 的第一个元素:

```
myCars[0]="Opel";
```

[0] 是数组的第一个元素。[1] 是数组的第二个元素。

在一个数组中你可以有不同的对象

所有的JavaScript变量都是对象。数组元素是对象。函数是对象。

因此，你可以在数组中有不同的变量类型。

你可以在一个数组中包含对象元素、函数、数组：

```
myArray[0]=Date.now;
myArray[1]=myFunction;
myArray[2]=myCars;
```

数组方法和属性

使用数组对象预定义属性和方法：

```
var x=myCars.length              // myCars 中元素的数量
var y=myCars.indexOf("Volvo")    // "Volvo" 值的索引值
```

完整的数组对象参考手册

你可以参考本站关于数组的所有属性和方法的完整参考手册。

参考手册包含了所有属性和方法的描述（和更多的例子）。

完整数组对象参考手册

创建新方法

原型是JavaScript全局构造函数。它可以构建新Javascript对象的属性和方法。

实例：创建一个新的方法。

```
Array.prototype.myUcase=function(){
    for (i=0;i<this.length;i++){
        this[i]=this[i].toUpperCase();
    }
}
```

尝试一下 »

上面的例子创建了新的数组方法用于将数组小写字符转为大写字符。

更多实例

合并两个数组 - concat()

合并三个数组 - concat()

用数组的元素组成字符串 - join()

删除数组的最后一个元素 - pop()

数组的末尾添加新的元素 - push()

将一个数组中的元素的顺序反转排序 - reverse()

删除数组的第一个元素 - shift()

从一个数组中选择元素 - slice()

数组排序（按字母顺序升序）- sort()

数字排序（按数字顺序升序）- sort()

数字排序（按数字顺序降序）- sort()

在数组的第2位置添加一个元素 - splice()

转换数组到字符串 -toString()

在数组的开头添加新元素 - unshift()

JavaScript Date（日期）对象 JavaScript Boolean（布尔）对象

5 篇笔记

写笔记

反馈/建议

JavaScript Array（数组）对象 JavaScript Math（算数）对象

JavaScript Boolean（布尔） 对象

Boolean（布尔）对象用于将非布尔值转换为布尔值（true 或者 false）。

在线实例

检查布尔值

检查布尔对象是 true 还是 false。

完整的 Boolean（布尔） 对象参考手册

我们提供 JavaScript Boolean 对象参考手册，其中包括所有可用于布尔对象的属性和方法。

该手册包含了对每个属性和方法的详细描述以及相关实例。

创建 Boolean 对象

Boolean 对象代表两个值:"true" 或者 "false"

下面的代码定义了一个名为 myBoolean 的布尔对象：

```
var myBoolean=new Boolean();
```

如果布尔对象无初始值或者其值为:

0

-0

null

""

false

undefined

NaN

那么对象的值为 false。否则，其值为 true（即使当变量值为字符串 "false" 时）！

□ JavaScript Array（数组）对象 JavaScript Math（算数）对象 □

□ JavaScript Boolean（布尔）对象 JavaScript RegExp 对象 □

JavaScript Math（算数） 对象

Math（算数）对象的作用是：执行常见的算数任务。

在线实例

round()

如何使用 round()。

random()

如何使用 random() 来返回 0 到 1 之间的随机数。

max()

如何使用 max() 来返回两个给定的数中的较大的数。（在 ECMASCript v3 之前，该方法只有两个参数。）

min()

如何使用 min() 来返回两个给定的数中的较小的数。（在 ECMASCript v3 之前，该方法只有两个参数。）

完整的 Math 对象参考手册

我们提供 JavaScript Math 对象的参考手册，其中包括所有可用于算术对象的属性和方法。

该手册包含了对每个属性和方法的详细描述以及相关实例。

Math 对象

Math（算数）对象的作用是：执行普通的算数任务。

Math 对象提供多种算数值类型和函数。无需在使用这个对象之前对它进行定义。

使用Math的属性/方法的语法：

```
var x=Math.PI;
var y=Math.sqrt(16);
```

注意： Math对象无需在使用这个对象之前对它进行定义。

算数值

JavaScript 提供 8 种可被 Math 对象访问的算数值：

你可以参考如下Javascript常量使用方法：

```
Math.E
Math.PI
Math.SQRT2
Math.SQRT1_2
Math.LN2
Math.LN10
Math.LOG2E
Math.LOG10E
```

算数方法

除了可被 Math 对象访问的算数值以外，还有几个函数（方法）可以使用。

下面的例子使用了 Math 对象的 round 方法对一个数进行四舍五入。

```
document.write(Math.round(4.7));
```

上面的代码输出为：

```
5
```

下面的例子使用了 Math 对象的 random() 方法来返回一个介于 0 和 1 之间的随机数：

```
document.write(Math.random());
```

上面的代码输出为：

```
0.10329115577042103
```

下面的例子使用了 Math 对象的 floor() 方法和 random() 来返回一个介于 0 和 11 之间的随机数：

```
document.write(Math.floor(Math.random()*11));
```

上面的代码输出为：

```
8
```

□ JavaScript Boolean（布尔）对象　　JavaScript RegExp 对象 □

对于伪随机数，JS 有很多种玩法来生成我们所需要的伪随机数。
根据上下限生成随机数：

```
var rand = (min,max) => Math.round(Math.random()*(max-min))+min;

//Max为最大值，Min为最小值
```

根据概率随机生成 bool 值：

```
function randBool(percent=0.5){

  //percent为概率，默认0.5（50%）。

  if(Math.random()<percent)//如果随机数小于概率值，返回true，否则返回false。

    return true;
```

```
  else
    return false;
}
```

随机生成指定字符：

```
function randChar(length,characters="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"){
  //length为所需长度，characters为所包含的所有字符，默认为字母+数字。
  characters=characters.split("");//分割字符。
  result="";//返回的结果。
  while(result.length<length) result+=characters[Math.round(Math.random()*characters.length)];
  return result;
}
```

随机生成字符的另一种玩法：

```
function randCharAnother(length,rangeMin=0x80,rangeMax=0x7FF){
  //length长度，rangeMin为最小Unicode码，rangeMax为最大Unicode码。
  result="";
  while(result.length<length) result+=String.fromCharCode(Math.round(Math.random()*(rangeMax-rangeMin))-rangeMin);
  return result;
}
```

随机从数组中取出一个东东：

```
Array.prototype.pick = function(){
  //不能为 ()=>{/*函数*/}，否则this会指向Window。
  return this.length?Math.round(Math.random()*(this.length-1)):undefined;//如果长度为0，返回undefined。
}
```

学神之女4个月前 (06-05)

反馈/建议

JavaScript RegExp 对象

RegExp：是正则表达式（regular expression）的简写。

完整 RegExp 对象参考手册

请查看我们的 JavaScript RegExp 对象的参考手册，其中提供了可以与字符串对象一同使用的所有的属性和方法。

这个手册包含的关于每个属性和方法的用法的详细描述和实例。

什么是 RegExp?

正则表达式描述了字符的模式对象。

当您检索某个文本时，可以使用一种模式来描述要检索的内容。RegExp 就是这种模式。

简单的模式可以是一个单独的字符。

更复杂的模式包括了更多的字符，并可用于解析、格式检查、替换等等。

您可以规定字符串中的检索位置，以及要检索的字符类型，等等。

语法

```
var patt=new RegExp(pattern,modifiers);

或更简单的方法

var patt=/pattern/modifiers;
```

模式描述了一个表达式模型。

修饰符(modifiers)描述了检索是否是全局，区分大小写等。

注意： *当使用构造函数创造正则对象时，需要常规的字符转义规则（在前面加反斜杠 \）。比如，以下是等价的：*

```
var re = new RegExp("\\w+");

var re = /\w+/;
```

RegExp 修饰符

修饰符用于执行不区分大小写和全文的搜索。

i - 修饰符是用来执行不区分大小写的匹配。

g - 修饰符是用于执行全文的搜索（而不是在找到第一个就停止查找,而是找到所有的匹配）。

实例 1

在字符串中不区分大小写找"runoob"

```
var str = "Visit RUnoob";
var patt1 = /runoob/i;
```

以下**标记**的文本是获得的匹配的表达式：

```
Visit RUnoob
```

尝试一下 »

实例 2

全文查找 "is"

```
var str="Is this all there is?";
var patt1=/is/g;
```

以下 **标记** 的文本是获得的匹配的表达式：

```
Is this all there is?
```

尝试一下 »

实例 3

全文查找和不区分大小写搜索 "is"

```
var str="Is this all there is?";
var patt1=/is/gi;
```

以下 **标记** 的文本是获得的匹配的表达式：

```
Is this all there is?
```

尝试一下 »

test()

test()方法搜索字符串指定的值，根据结果并返回真或假。

下面的示例是从字符串中搜索字符 "e"：

实例

```
var patt1=new RegExp("e");
document.write(patt1.test("The best things in life are free"));
```

由于该字符串中存在字母 "e"，以上代码的输出将是：

```
true
```

尝试一下 »

当使用构造函数创造正则对象时，需要常规的字符转义规则（在前面加反斜杠 \）

实例

```
var re = new RegExp("\\w+");
```

尝试一下 »

exec()

exec() 方法检索字符串中的指定值。返回值是被找到的值。如果没有发现匹配，则返回 null。

下面的示例是从字符串中搜索字符 "e"：

实例 1

```
var patt1=new RegExp("e");
document.write(patt1.exec("The best things in life are free"));
```

由于该字符串中存在字母 "e"，以上代码的输出将是：

```
e
```

尝试一下 »

JavaScript Window - 浏览器对象模型

浏览器对象模型 (BOM) 使 JavaScript 有能力与浏览器"对话"。

浏览器对象模型 (BOM)

浏览器对象模型（Browser Object Model (BOM)）尚无正式标准。

由于现代浏览器已经（几乎）实现了 JavaScript 交互性方面的相同方法和属性，因此常被认为是 BOM 的方法和属性。

Window 对象

所有浏览器都支持 window 对象。它表示浏览器窗口。

所有 JavaScript 全局对象、函数以及变量均自动成为 window 对象的成员。

全局变量是 window 对象的属性。

全局函数是 window 对象的方法。

甚至 HTML DOM 的 document 也是 window 对象的属性之一：

```
window.document.getElementById("header");
```

与此相同：

```
document.getElementById("header");
```

Window 尺寸

有三种方法能够确定浏览器窗口的尺寸。

对于Internet Explorer、Chrome、Firefox、Opera 以及 Safari：

window.innerHeight - 浏览器窗口的内部高度(包括滚动条)

window.innerWidth - 浏览器窗口的内部宽度(包括滚动条)

对于 Internet Explorer 8、7、6、5:

document.documentElement.clientHeight

document.documentElement.clientWidth

或者

document.body.clientHeight

document.body.clientWidth

实用的 JavaScript 方案（涵盖所有浏览器）：

实例

```
var w=window.innerWidth
|| document.documentElement.clientWidth
|| document.body.clientWidth;
```

```
var h=window.innerHeight
|| document.documentElement.clientHeight
|| document.body.clientHeight;
```

尝试一下 »

该例显示浏览器窗口的高度和宽度。

其他 Window 方法

一些其他方法：

- window.open() - 打开新窗口
- window.close() - 关闭当前窗口
- window.moveTo() - 移动当前窗口
- window.resizeTo() - 调整当前窗口的尺寸

JavaScript RegExp 对象 JavaScript Window Screen

点我分享笔记

反馈/建议

JavaScript Window Screen

window.screen 对象包含有关用户屏幕的信息。

Window Screen

window.screen对象在编写时可以不使用 window 这个前缀。

一些属性：

- screen.availWidth - 可用的屏幕宽度
- screen.availHeight - 可用的屏幕高度

Window Screen 可用宽度

screen.availWidth 属性返回访问者屏幕的宽度，以像素计，减去界面特性，比如窗口任务栏。

实例

返回您的屏幕的可用宽度：

```
<script>
document.write("可用宽度: " + screen.availWidth);
</script>
```

以上代码输出为：

```
可用宽度: 1366
```

尝试一下 »

Window Screen 可用高度

screen.availHeight 属性返回访问者屏幕的高度，以像素计，减去界面特性，比如窗口任务栏。

实例

返回您的屏幕的可用高度：

```
<script>

document.write("可用高度: " + screen.availHeight);

</script>
```

以上代码将输出：

```
可用高度: 728
```

尝试一下 »

所有 screen 属性实例

JavaScript Window　　JavaScript Window Location

点我分享笔记

反馈/建议

JavaScript Window Screen　　JavaScript Window History

JavaScript Window Location

window.location 对象用于获得当前页面的地址 (URL)，并把浏览器重定向到新的页面。

Window Location

window.location 对象在编写时可不使用 window 这个前缀。 一些例子：

一些实例:

location.hostname 返回 web 主机的域名

location.pathname 返回当前页面的路径和文件名

location.port 返回 web 主机的端口 （80 或 443）

location.protocol 返回所使用的 web 协议（http:// 或 https://）

Window Location Href

location.href 属性返回当前页面的 URL。

实例

返回（当前页面的）整个 URL：

```
<script>

document.write(location.href);

</script>
```

以上代码输出为：

```
http://localhost:8000/JavaScript.html
```

Window Location Pathname

location.pathname 属性返回 URL 的路径名。

实例

返回当前 URL 的路径名：

```
<script>

document.write(location.pathname);

</script>
```

以上代码输出为：

```
/JavaScript.html
```

Window Location Assign

location.assign() 方法加载新的文档。

实例

加载一个新的文档：

```
<html>
<head>
<script>
function newDoc()
  {
  window.location.assign("http://www.w3cschool.cc")
  }
</script>
</head>
<body>

<input type="button" value="Load new document" onclick="newDoc()">

</body>
</html>
```

尝试一下 »

JavaScript Window Screen

JavaScript Window History

点我分享笔记

JavaScript Window History

window.history 对象包含浏览器的历史。

Window History

window.history对象在编写时可不使用 window 这个前缀。

为了保护用户隐私，对 JavaScript 访问该对象的方法做出了限制。

一些方法：

history.back() - 与在浏览器点击后退按钮相同

history.forward() - 与在浏览器中点击向前按钮相同

Window History Back

history.back() 方法加载历史列表中的前一个 URL。

这与在浏览器中点击后退按钮是相同的：

实例

在页面上创建后退按钮：

```
<html>
<head>
<script>
function goBack()
  {
  window.history.back()
  }
</script>
</head>
<body>

<input type="button" value="Back" onclick="goBack()">

</body>
</html>
```

以上代码输出为：

返回上一页

Window History Forward

history forward() 方法加载历史列表中的下一个 URL。

这与在浏览器中点击前进按钮是相同的：

实例

在页面上创建一个向前的按钮：

```
<html>
<head>
<script>
function goForward()
  {
  window.history.forward()
  }
</script>
</head>
<body>

<input type="button" value="Forward" onclick="goForward()">

</body>
</html>
```

以上代码输出为：

返回下一页

JavaScript Window Location　　JavaScript Window Navigator

JavaScript Window History　　JavaScript 弹窗

JavaScript Window Navigator

window.navigator 对象包含有关访问者浏览器的信息。

Window Navigator

window.navigator 对象在编写时可不使用 window 这个前缀。

实例

```
<divid="example"></div>
<script>
txt = "<p>浏览器代号: " + navigator.appCodeName + "</p>";
txt+= "<p>浏览器名称: " + navigator.appName + "</p>";
txt+= "<p>浏览器版本: " + navigator.appVersion + "</p>";
txt+= "<p>启用Cookies: " + navigator.cookieEnabled + "</p>";
txt+= "<p>硬件平台: " + navigator.platform + "</p>";
txt+= "<p>用户代理: " + navigator.userAgent + "</p>";
txt+= "<p>用户代理语言: " + navigator.systemLanguage + "</p>";
document.getElementById("example").innerHTML=txt;
</script>
```

尝试一下 »

警告!!!

来自 navigator 对象的信息具有误导性，不应该被用于检测浏览器版本，这是因为：

navigator 数据可被浏览器使用者更改

一些浏览器对测试站点会识别错误

浏览器无法报告晚于浏览器发布的新操作系统

浏览器检测

由于 navigator 可误导浏览器检测，使用对象检测可用来嗅探不同的浏览器。

由于不同的浏览器支持不同的对象，您可以使用对象来检测浏览器。例如，由于只有 Opera 支持属性 "window.opera"，您可以据此识别出 Opera。

例子：if (window.opera) {...some action...}

JavaScript Window History　　JavaScript 弹窗

JavaScript Window Navigator　　JavaScript 计时事件

JavaScript 弹窗

可以在 JavaScript 中创建三种消息框：警告框、确认框、提示框。

警告框

警告框经常用于确保用户可以得到某些信息。

当警告框出现后，用户需要点击确定按钮才能继续进行操作。

语法

```
window.alert("sometext");
```

window.alert() 方法可以不带上window对象，直接使用**alert()**方法。

实例

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction()
{
    alert("你好，我是一个警告框！");
}
</script>
</head>
<body>
```

```
<input type="button" onclick="myFunction()" value="显示警告框">

</body>
</html>
```

尝试一下 »

确认框

确认框通常用于验证是否接受用户操作。

当确认卡弹出时，用户可以点击 "确认" 或者 "取消" 来确定用户操作。

当你点击 "确认", 确认框返回 true， 如果点击 "取消", 确认框返回 false。

语法

```
window.confirm("sometext");
```

window.confirm() 方法可以不带上window对象，直接使用**confirm()**方法。

实例

```
var r=confirm("按下按钮");
if (r==true)
{
    x="你按下了\"确定\"按钮!";
}
else
{
    x="你按下了\"取消\"按钮!";
}
```

尝试一下 »

提示框

提示框经常用于提示用户在进入页面前输入某个值。

当提示框出现后，用户需要输入某个值，然后点击确认或取消按钮才能继续操纵。

如果用户点击确认，那么返回值为输入的值。如果用户点击取消，那么返回值为 null。

语法

```
window.prompt("sometext","defaultvalue");
```

window.prompt() 方法可以不带上window对象，直接使用**prompt()**方法。

实例

```
var person=prompt("请输入你的名字","Harry Potter");
if (person!=null && person!="")
{
    x="你好 " + person + "! 今天感觉如何?";
    document.getElementById("demo").innerHTML=x;
}
```

尝试一下 »

换行

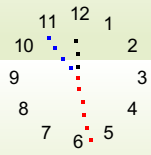
弹窗使用 反斜杠 + "n"(\n) 来设置换行。

实例

```
alert("Hello\nHow are you?");
```

尝试一下 »

JavaScript 计时事件



JavaScript 一个设定的时间间隔之后来执行代码

我们称之为计时事件

JavaScript 计时事件

通过使用 JavaScript，我们有能力做到在一个设定的时间间隔之后来执行代码，而不是在函数被调用后立即执行。我们称之为计时事件。

在 JavaScritp 中使用计时事件是很容易的，两个关键方法是:

setInterval() - 间隔指定的毫秒数不停地执行指定的代码。

setTimeout() - 在指定的毫秒数后执行指定代码。

注意: setInterval() 和 setTimeout() 是 HTML DOM Window对象的两个方法。

setInterval() 方法

setInterval() 间隔指定的毫秒数不停地执行指定的代码

语法

```
window.setInterval("javascript function",milliseconds);
```

window.setInterval() 方法可以不使用window前缀，直接使用函数**setInterval()**。

setInterval() 第一个参数是函数（function）。

第二个参数间隔的毫秒数

注意: 1000 毫秒是一秒。

实例

每三秒弹出 "hello"：

```
setInterval(function(){alert("Hello")},3000);
```

尝试一下 »

实例展示了如何使用 setInterval() 方法，但是每三秒弹出一次对用户体验并不好。

以下实例将显示当前时间。 setInterval() 方法设置每秒钟执行一次代码，就是手表一样。

实例

显示当前时间

```
var myVar=setInterval(function(){myTimer()},1000);
function myTimer()
{
var d=new Date();
var t=d.toLocaleTimeString();
document.getElementById("demo").innerHTML=t;
}
```

尝试一下 »

如何停止执行?

clearInterval() 方法用于停止 setInterval() 方法执行的函数代码。

语法

```
window.clearInterval(intervalVariable)
```

window.clearInterval() 方法可以不使用window前缀，直接使用函数**clearInterval()**。

要使用 clearInterval() 方法, 在创建计时方法时你必须使用全局变量：

```
myVar=setInterval("javascript function",milliseconds);
```

然后你可以使用clearInterval() 方法来停止执行。

实例

以下例子,我们添加了 "停止" 按钮：

```
<p id="demo"></p>
<button onclick="myStopFunction()">停止</button>
<script>
var myVar=setInterval(function(){myTimer()},1000);
function myTimer(){
var d=new Date();
var t=d.toLocaleTimeString();
document.getElementById("demo").innerHTML=t;
}
function myStopFunction(){
clearInterval(myVar);
}
</script>
```

尝试一下 »

---

setTimeout() 方法

语法

```
myVar= window.setTimeout("javascript function", milliseconds);
```

setTimeout() 方法会返回某个值。在上面的语句中，值被储存在名为 myVar 的变量中。假如你希望取消这个 setTimeout()，你可以使用这个变量名来指定它。

setTimeout() 的第一个参数是含有 JavaScript 语句的字符串。这个语句可能诸如 "alert('5 seconds!')"，或者对函数的调用，诸如 alertMsg。

第二个参数指示从当前起多少毫秒后执行第一个参数。

提示：1000 毫秒等于一秒。

实例

等待3秒，然后弹出 "Hello":

```
setTimeout(function(){alert("Hello")},3000);
```

尝试一下 »

如何停止执行?

clearTimeout() 方法用于停止执行setTimeout()方法的函数代码。

语法

```
window.clearTimeout(timeoutVariable)
```

window.clearTimeout() 方法可以不使用window 前缀。

要使用clearTimeout() 方法, 你必须在创建超时方法中（setTimeout）使用全局变量:

```
myVar=setTimeout("javascript function",milliseconds);
```

如果函数还未被执行，你可以使用 clearTimeout() 方法来停止执行函数代码。

实例

以下是同一个实例, 但是添加了 "Stop the alert" 按钮:

```
var myVar;
function myFunction()
{
myVar=setTimeout(function(){alert("Hello")},3000);
}
function myStopFunction()
{
clearTimeout(myVar);
}
```

尝试一下 »

更多实例

另一个简单的计时

JavaScript 弹窗 JavaScript Cookie

JavaScript 计时事件 JavaScript 库

JavaScript Cookie

Cookie 用于存储 web 页面的用户信息。

什么是 Cookie?

Cookie 是一些数据, 存储于你电脑上的文本文件中。

当 web 服务器向浏览器发送 web 页面时，在连接关闭后，服务端不会记录用户的信息。

Cookie 的作用就是用于解决 "如何记录客户端的用户信息":

当用户访问 web 页面时，他的名字可以记录在 cookie 中。

在用户下一次访问该页面时，可以在 cookie 中读取用户访问记录。

Cookie 以名/值对形式存储，如下所示:

```
username=John Doe
```

当浏览器从服务器上请求 web 页面时， 属于该页面的 cookie 会被添加到该请求中。服务端通过这种方式来获取用户的信息。

使用 JavaScript 创建Cookie

JavaScript 可以使用 **document.cookie** 属性来创建 、读取、及删除 cookie。

JavaScript 中，创建 cookie 如下所示：

```
document.cookie="username=John Doe";
```

您还可以为 cookie 添加一个过期时间（以 UTC 或 GMT 时间）。默认情况下，cookie 在浏览器关闭时删除：

```
document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 GMT";
```

您可以使用 path 参数告诉浏览器 cookie 的路径。默认情况下，cookie 属于当前页面。

```
document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 GMT; path=/";
```

使用 JavaScript 读取 Cookie

在 JavaScript 中, 可以使用以下代码来读取 cookie：

```
var x = document.cookie;
```

Note document.cookie 将以字符串的方式返回所有的 cookie，类型格式： cookie1=value; cookie2=value; cookie3=value;

使用 JavaScript 修改 Cookie

在 JavaScript 中，修改 cookie 类似于创建 cookie，如下所示：

```
document.cookie="username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 GMT; path=/";
```

旧的 cookie 将被覆盖。

使用 JavaScript 删除 Cookie

删除 cookie 非常简单。您只需要设置 expires 参数为以前的时间即可，如下所示，设置为 Thu, 01 Jan 1970 00:00:00 GMT:

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 GMT";
```

注意，当您删除时不必指定 cookie 的值。

Cookie 字符串

document.cookie 属性看起来像一个普通的文本字符串，其实它不是。

即使您在 document.cookie 中写入一个完整的 cookie 字符串, 当您重新读取该 cookie 信息时，cookie 信息是以名/值对的形式展示的。

如果您设置了新的 cookie，旧的 cookie 不会被覆盖。 新 cookie 将添加到 document.cookie 中，所以如果您重新读取document.cookie，您将获得如下所示的数据：

cookie1=value; cookie2=value;

显示所有 Cookie　创建 Cookie 1　创建 Cookie 2　删除 Cookie 1　删除 Cookie 2

如果您需要查找一个指定 cookie 值，您必须创建一个JavaScript 函数在 cookie 字符串中查找 cookie 值。

JavaScript Cookie 实例

在以下实例中，我们将创建 cookie 来存储访问者名称。

首先，访问者访问 web 页面, 他将被要求填写自己的名字。该名字会存储在 cookie 中。

访问者下一次访问页面时，他会看到一个欢迎的消息。

在这个实例中我们会创建 3 个 JavaScript 函数:

1. 设置 cookie 值的函数
2. 获取 cookie 值的函数
3. 检测 cookie 值的函数

设置 cookie 值的函数

首先，我们创建一个函数用于存储访问者的名字：

```
function setCookie(cname,cvalue,exdays)
{
  var d = new Date();
  d.setTime(d.getTime()+(exdays*24*60*60*1000));
  var expires = "expires="+d.toGMTString();
  document.cookie = cname + "=" + cvalue + "; " + expires;
}
```

函数解析：

以上的函数参数中，cookie 的名称为 cname，cookie 的值为 cvalue，并设置了 cookie 的过期时间 expires。

该函数设置了 cookie 名、cookie 值、cookie过期时间。

获取 cookie 值的函数

然后，我们创建一个函数用户返回指定 cookie 的值：

```
function getCookie(cname)
{
  var name = cname + "=";
  var ca = document.cookie.split(';');
  for(var i=0; i<ca.length; i++)
  {
    var c = ca[i].trim();
    if (c.indexOf(name)==0) return c.substring(name.length,c.length);
  }
  return "";
}
```

函数解析：

cookie 名的参数为 cname。

创建一个文本变量用于检索指定 cookie :cname + "="。

使用分号来分割 document.cookie 字符串，并将分割后的字符串数组赋值给 ca (ca = document.cookie.split(';'))。

循环 ca 数组 (i=0;i<ca.length;i++)，然后读取数组中的每个值，并去除前后空格 (c=ca[i].trim())。

如果找到 cookie(c.indexOf(name) == 0)，返回 cookie 的值 (c.substring(name.length,c.length)。

如果没有找到 cookie, 返回 ""。

检测 cookie 值的函数

最后，我们可以创建一个检测 cookie 是否创建的函数。

如果设置了 cookie，将显示一个问候信息。

如果没有设置 cookie，将会显示一个弹窗用于询问访问者的名字，并调用 setCookie 函数将访问者的名字存储 365 天：

```
function checkCookie()
{
  var username=getCookie("username");
  if (username!="")
  {
    alert("Welcome again " + username);
  }
  else
  {
    username = prompt("Please enter your name:","");
    if (username!="" && username!=null)
    {
      setCookie("username",username,365);
    }
  }
}
```

完整实例

实例

```
function setCookie(cname,cvalue,exdays){
var d = new Date();
d.setTime(d.getTime()+(exdays*24*60*60*1000));
var expires = "expires="+d.toGMTString();
document.cookie = cname+"="+cvalue+"; "+expires;
}
function getCookie(cname){
var name = cname + "=";
var ca = document.cookie.split(';');
for(var i=0; i<ca.length; i++) {
var c = ca[i].trim();
if (c.indexOf(name)==0) { return c.substring(name.length,c.length); }
}
return "";
}
```

```
function checkCookie(){
var user=getCookie("username");
if (user!=""){
alert("欢迎 " + user + " 再次访问");
}
else {
user = prompt("请输入你的名字:","");
if (user!="" && user!=null){
setCookie("username",user,30);
}
}
}
```

尝试一下 »

以下实例在页面载入时执行 checkCookie() 函数。

JavaScript 计时事件 JavaScript 库

点我分享笔记

反馈/建议

JavaScript Cookie JavaScript 测试 jQuery

JavaScript 库

JavaScript 库 - jQuery、Prototype、MooTools。

JavaScript 框架（库）

JavaScript 高级程序设计（特别是对浏览器差异的复杂处理），通常很困难也很耗时。

为了应对这些调整，许多的 **JavaScript (helper)** 库应运而生。

这些 JavaScript 库常被称为 **JavaScript 框架**。

在本教程中，我们将了解到一些广受欢迎的 JavaScript 框架：

- jQuery
- Prototype
- MooTools

所有这些框架都提供针对常见 JavaScript 任务的函数，包括动画、DOM 操作以及 Ajax 处理。

在本教程中，您将学习到如何开始使用它们，来使得 JavaScript 编程更容易、更安全且更有乐趣。

jQuery

jQuery 是目前最受欢迎的 JavaScript 框架。

它使用 CSS 选择器来访问和操作网页上的 HTML 元素（DOM 对象）。

jQuery 同时提供 companion UI（用户界面）和插件。

许多大公司在网站上使用 jQuery：

- Google
- Microsoft
- IBM
- Netflix

如需更深入地学习 jQuery，请访问我们的 jQuery 教程。

Prototype

Prototype 是一种库，提供用于执行常见 web 任务的简单 API。

API 是应用程序编程接口（Application Programming Interface）的缩写。它是包含属性和方法的库，用于操作 HTML DOM。

Prototype 通过提供类和继承，实现了对 JavaScript 的增强。

MooTools

MooTools 也是一个框架，提供了可使常见的 JavaScript 编程更为简单的 API。

MooTools 也含有一些轻量级的效果和动画函数。

其他框架

下面是其他一些在上面未涉及的框架：

YUI - Yahoo! User Interface Framework，涵盖大量函数的大型库，从简单的 JavaScript 功能到完整的 internet widget。

Ext JS - 可定制的 widget，用于构建富因特网应用程序（rich Internet applications）。

Dojo - 用于 DOM 操作、事件、widget 等的工具包。

script.aculo.us - 开源的 JavaScript 框架，针对可视效果和界面行为。

UIZE - Widget、AJAX、DOM、模板等等。

CDN -内容分发网络

您总是希望网页可以尽可能地快。您希望页面的容量尽可能地小，同时您希望浏览器尽可能多地进行缓存。

如果许多不同的网站使用相同的 JavaScript 框架，那么把框架库存放在一个通用的位置供每个网页分享就变得很有意义了。

CDN (Content Delivery Network) 解决了这个问题。CDN 是包含可分享代码库的服务器网络。

Google 为一系列 JavaScript 库提供了免费的 CDN，包括：

- jQuery
- Prototype
- MooTools
- Dojo
- Yahoo! YUI

但是由于 Google 在中国经常被GFW（防火长城，英文名称Great Firewall of China，简写为Great Firewall，缩写GFW）屏蔽，造成访问不稳定，所以建议使用百度静态资源公共库。

如需在您的网页中使用 JavaScript 框架库，只需在 <script> 标签中引用该库即可：

引用 jQuery

```
<script src="http://apps.bdimg.com/libs/jquery/2.1.1/jquery.min.js">
</script>
```

使用框架

在您决定为网页使用 JavaScript 框架之前，首先对框架进行测试是明智的。

JavaScript 框架很容易进行测试。您无需在计算机上安装它们，同时也没有安装程序。

通常您只需从网页中引用一个库文件。

在本教程的下一章，我们会为您完整地讲解 jQuery 的测试过程。

JavaScript - 测试 jQuery

测试 JavaScript 框架库 - jQuery

引用 jQuery

如需测试 JavaScript 库，您需要在网页中引用它。

为了引用某个库，请使用 <script> 标签，其 src 属性设置为库的 URL：

引用 jQuery

```
<!DOCTYPE html>
<html>
<head>
<script src="http://apps.bdimg.com/libs/jquery/2.1.1/jquery.min.js">
</script>
</head>
<body>
</body>
</html>
```

jQuery 描述

主要的 jQuery 函数是 $() 函数（jQuery 函数）。如果您向该函数传递 DOM 对象，它会返回 jQuery 对象，带有向其添加的 jQuery 功能。

jQuery 允许您通过 CSS 选择器来选取元素。

在 JavaScript 中，您可以分配一个函数以处理窗口加载事件：

JavaScript 方式：

```
function myFunction()
{
    var obj=document.getElementById("h01");
    obj.innerHTML="Hello jQuery";
}
onload=myFunction;
```

等价的 jQuery 是不同的：

jQuery 方式：

```
function myFunction()
{
```

```
    $("#h01").html("Hello jQuery");
}
$(document).ready(myFunction);
```

上面代码的最后一行，HTML DOM 文档对象被传递到 jQuery ：$(document)。

当您向 jQuery 传递 DOM 对象时，jQuery 会返回以 HTML DOM 对象包装的 jQuery 对象。

jQuery 函数会返回新的 jQuery 对象，其中的 ready() 是一个方法。

由于在 JavaScript 中函数就是变量，因此可以把 myFunction 作为变量传递给 jQuery 的 ready 方法。

jQuery 返回 jQuery 对象，与已传递的 DOM 对象不同。
jQuery 对象拥有的属性和方法，与 DOM 对象的不同。
您不能在 jQuery 对象上使用 HTML DOM 的属性和方法。

测试 jQuery

请试一下下面这个例子：

实例

```
<!DOCTYPE html>
<html>
<head>
<script src="http://apps.bdimg.com/libs/jquery/2.1.1/jquery.min.js">
</script>
<script>
function myFunction()
{
    $("#h01").html("Hello jQuery")
}
$(document).ready(myFunction);
</script>
</head>
<body>
<h1 id="h01"></h1>
</body>
</html>
```

尝试一下 »

请再试一下这个例子：

实例

```
<!DOCTYPE html>
<html>
<head>
<script src="http://apps.bdimg.com/libs/jquery/2.1.1/jquery.min.js">
</script>
<script>
function myFunction()
{
    $("#h01").attr("style","color:red").html("Hello jQuery")
}
$(document).ready(myFunction);
</script>
</head>
<body>
<h1 id="h01"></h1>
</body>
</html>
```

尝试一下 »

正如您在上面的例子中看到的，jQuery 允许链接（链式语法）。

链接（Chaining）是一种在同一对象上执行多个任务的便捷方法。

需要学习更多内容吗？菜鸟教程为您提供了非常棒的 jQuery 教程。

JavaScript - 测试 Prototype

测试 JavaScript 框架库 - Prototype

引用 Prototype

如需测试 JavaScript 库，您需要在网页中引用它。

为了引用某个库，请使用 <script> 标签，其 src 属性设置为库的 URL：

引用 Prototype

```
<!DOCTYPE html>
<html>
<head>
<script
src="http://apps.bdimg.com/libs/prototype/1.7.1.0/prototype.js">
</script>
</head>
<body>
</body>
</html>
```

Prototype 描述

Prototype 提供的函数可使 HTML DOM 编程更容易。

与 jQuery 类似，Prototype 也有自己的 $() 函数。

$() 函数接受 HTML DOM 元素的 id 值（或 DOM 元素），并会向 DOM 对象添加新的功能。

与 jQuery 不同，Prototype 没有用以取代 window.onload() 的 ready() 方法。相反，Prototype 会向浏览器及 HTML DOM 添加扩展。

在 JavaScript 中，您可以分配一个函数以处理窗口加载事件：

JavaScript 方式：

```
function myFunction()
{
var obj=document.getElementById("h01");
obj.innerHTML="Hello Prototype";
}
onload=myFunction;
```

等价的 Prototype 是不同的：

Prototype 方式：

```
function myFunction()
{
$("h01").insert("Hello Prototype!");
}
Event.observe(window,"load",myFunction);
```

Event.observe() 接受三个参数：

您希望处理的 HTML DOM 或 BOM（浏览器对象模型）对象

您希望处理的事件

您希望调用的函数

测试 Prototype

请试一下下面这个例子：

Example

```
<!DOCTYPE html>
<html>
<script
src="http://apps.bdimg.com/libs/prototype/1.7.1.0/prototype.js">
</script>
<script>
function myFunction()
{
$("h01").insert("Hello Prototype!");
}
Event.observe(window,"load",myFunction);
</script>
</head>
<body>
<h1 id="h01"></h1>
</body>
</html>
```

尝试一下 »

请再试一下这个例子：

Example

```
<!DOCTYPE html>
<html>
<script
src="http://apps.bdimg.com/libs/prototype/1.7.1.0/prototype.js">
</script>
<script>
function myFunction()
{
$("h01").writeAttribute("style","color:red").insert("Hello Prototype!");
}
Event.observe(window,"load",myFunction);
</script>
</head>
<body>
<h1 id="h01"></h1>
</body>
</html>
```

测试一下 »

正如您在上面的例子中看到的，与 jQuery 相同，Prototype 允许链式语法。

链接（Chaining）是一种在同一对象上执行多个任务的便捷方法。

基础 JavaScript 实例

用JavaScript输出文本

用JavaScript改变HTML元素

一个外部JavaScript

实例解析

JavaScript 语句、注释和代码块

JavaScript 语句

JavaScript 代码块

JavaScript 单行注释

JavaScript 多行注释

使用单行注释来防止执行

使用多行注释来防止执行

实例解析

JavaScript 变量

声明一个变量，为它赋值，然后显示出来

实例解析

JavaScript 条件语句 If ... Else

If 语句

If...else 语句

随机链接

Switch 语句

实例解析

JavaScript 消息框

Alert(警告)框

带有换行的警告框

确认框

提示框

实例解析

JavaScript 函数

函数

带有参数的函数

带有参数的函数 2

返回值的函数

带有参数并返回值的函数

实例解析

JavaScript 循环

For 循环

循环输出 HTML 标题

While 循环

Do while 循环

break 语句

continue 语句

使用 For...In 声明来遍历数组内的元素

实例解析

JavaScript 事件

onclick事件

onmouseover 事件

实例解析

JavaScript 错误处理

try...catch 语句

带有确认框的 try...catch 语句

onerror 事件

实例解析

高级 JavaScript 实例

创建一个欢迎 cookie

简单的计时

另一个简单的计时

在一个无穷循环中的计时事件

带有停止按钮的无穷循环中的计时事件

使用计时事件制作的钟表

创建对象的实例

创建用于对象的模板

JavaScript 应用实例

javascript 幻灯片代码(含自动播放)

CSS 日历样式

JavaScript 弹窗

JavaScript 图片弹窗

JavaScript Lightbox

javascript 搜索框自动提示

JavaScript 表格数据搜索

JavaScript 实现列表按字母排序

JavaScript 实现表格单列按字母排序

JavaScript 动画应用实例

JavaScript 进度条实例

JavaScript 百分比进度条

JavaScript/CSS 实现提示弹窗

JavaScript/CSS 实现待办事项列表(To Do List)

HTML CSS, JavaScript 计算器

HTML、CSS、JavaScript 实现下拉菜单效果

JS/CSS 各种操作信息提示效果

JS/CSS 在屏幕底部弹出消息(snackbar)

JS/CSS 登录表单

JS/CSS 注册表单

JavaScript 计算器(倒计时)

JS/CSS 菜单按钮切换(打开/关闭)

JS/CSS 手风琴动画效果

JS/CSS 带图标手风琴动画效果

JS/CSS 选项卡

JS/CSS 选项卡 – 淡入效果

JS/CSS 选项卡 – 设置默认选项

JS/CSS 选项卡 – 设置关闭按钮

JS/CSS 选项卡 – 垂直方向

JS/CSS 选项卡 – 幻灯片效果

JS/CSS 响应式顶部导航样式实例

JS/CSS 侧边栏动画实例

JS/CSS 侧边栏动画实例 - 页面主体内容向右移动

JS/CSS 侧边栏动画实例 - 页面主体内容黑色透明背景

JS/CSS 全屏幕侧边栏

JS/CSS 侧边栏 - 无动画效果

JS/CSS 右侧侧边栏

JS/CSS 全屏幕导航 – 从上到下动画

JS/CSS 点击式下拉菜单

JS/CSS 点击式下拉菜单 - 右对齐

JS/CSS 点击式导航栏下拉菜单

JS/CSS 下拉菜单可进行搜索/过滤操作

JS 联想、自动补齐功能

JavaScript 按下回车(Enter)键触发按钮点击事件

JavaScript 创建一个菜单搜索

JavaScript 测试 Prototype JavaScript 对象实例

点我分享笔记

反馈/建议

JavaScript 实例 JavaScript HTML DOM 实例

JavaScript 对象 实例

使用内置的JavaScript对象实例。

String（字符串）对象

返回字符串的长度

为字符串添加样式

返回字符串中指定文本首次出现的位置 - indexOf()方法

查找字符串中特定的字符，若找到，则返回该字符 - match() 方法

替换字符串中的字符 - replace()

更多的字符串对象的例子，在我们的JavaScript String 对象参考手册。

Date（日期）对象

使用 Date() 方法来返回今天的日期和时间

使用 getTime() 计算从1970年到今天有多少毫秒

使用 setFullYear() 设置具体的日期

使用 toUTCString() 把当日的日期（根据 UTC）转换为字符串

使用 getDay() 来显示星期，而不仅仅是数字

显示一个钟表

更多的Date（日期）对象的例子，在我们的JavaScript Date 对象参考手册。

Array（数组）对象

创建数组

合并两个数组 - concat()

合并三个数组 - concat()

用数组的元素组成字符串 - join()

删除数组的最后一个元素 - pop()

数组的末尾添加新的元素 - push()

反转一个数组中的元素的顺序 - reverse()

删除数组的第一个元素 - shift()

从一个数组中的选择元素 - slice()

数组排序（按字母顺序升序）- sort()

数字排序（按数字顺序升序）- sort()

数字排序（按数字顺序降序）- sort()

在数组的第2位置添加一个元素 - splice()

转换数组到字符串 -toString()

在数组的开头添加新元素 - unshift()

更多的Array（数组）对象的例子，在我们的JavaScript Array 对象的参考手册。

Boolean（布尔）对象

检查逻辑值

更多的Boolean（布尔）对象对象的例子，在我们的JavaScript Boolean 对象的参考手册。

Math（算数）对象

使用 round() 对数字进行舍入

使用 random() 来返回 0 到 1 之间的随机数

使用 max() 来返回两个给定的数中的较大的数

使用 min() 来返回两个给定的数中的较小的数

摄氏度与华氏转换

一般

通过对象元素使用for...in语句

更多Math 对象实例在我们的**JavaScript Math 对象的参考手册。**

JavaScript 实例 JavaScript HTML DOM 实例

点我分享笔记

反馈/建议

JavaScript Browser 对象 实例

使用JavaScript来访问和控制浏览器对象实例。

Window 对象

弹出一个警告框

弹出一个带折行的警告框

弹出一个确认框，并提醒访客点击的内容

弹出一个提示框

点击一个按钮时，打开一个新窗口

打开一个新窗口，并控制其外观

打开多个新窗口

确保新的窗口没有获得焦点

确保新的窗口获得焦点

关闭新窗口

检查新的窗口是否已关闭

返回新窗口的名字

传输一些文本到源（父）窗口

相对于当前位置移动新窗口

移动新窗口到指定位置

打印当前页面

用像素指定窗口大小

指定窗口大小

由指定的像素数滚动内容

滚动到指定内容处

一个简单的时钟

用setTimeout() 和 clearTimeout()设置和停止定时器

用setInterval() 和 clearInterval()设置和停止定时器

更多的Window 对象的例子，在我们的JavaScript 参考手册。

Navigator 对象

访问者的浏览器的详细

更多的Navigator 对象的例子，在我们的JavaScript 参考手册。

Screen 对象

访问者的屏幕的详细

更多的Screen 对象的例子，在我们的JavaScript 参考手册。

History 对象

返回一个url的历史清单

创建一个后退按钮

创建一个前进按钮

从url的历史清单转到指定的url

更多的History 对象对象的例子，在我们的JavaScript 参考手册。

Location 对象

返回主机名和当前url的端口号

返回当前页面的整个URL

返回当前url的路径名

返回当前URL的协议部分

加载个新文档

重新载入当前文档

替代当前文档

跳出框架

更多Location 对象实例在我们的**JavaScript 参考手册。**

JavaScript HTML DOM 实例 JavaScript 总结

点我分享笔记

反馈/建议

JavaScript 对象实例 JavaScript 浏览器对象实例

JavaScript 对象 实例

使用内置JavaScript的对象实例。

Document 对象

使用 document.write() 输出文本

使用 document.write() 输出 HTML

返回文档中锚的数目

返回文档中第一个锚的 innerHTML

返回文档中表单的数目

返回文档中第一个表单的名字

返回文档中的图像数

返回文档中第一个图像的ID

返回文档中的链接数

返回文档中的第一个链接的ID

返回文档中的所有cookies的名称/值对

返回加载的文档的服务器域名

返回文档的最后一次修改时间

返回加载的当前文档的URL

返回文档的标题

返回文档的完整的URL

打开输出流，向流中输入文本

write() 和 writeln()的不同

用指定的ID弹出一个元素的innerHTML

用指定的Name弹出元素的数量

用指定的tagname弹出元素的数量

更多的Document 对象的例子，在我们的JavaScript 参考手册。

Anchor 对象

返回和设置链接的charset属性

返回和设置链接的href属性

返回和设置链接的hreflang属性

返回一个锚的名字

返回当前的文件和链接的文档之间的关系

改变链接的target属性

返回一个链接的type属性的值

更多的Anchor 对象的例子，在我们的JavaScript 参考手册。

Area 对象

返回图像映射某个区域的替代文字

返回图像映射某个区域的坐标

返回一个区域的href属性的锚部分

返回的主机名：图像映射的某个区域的端口

返回图像映射的某个区域的hostname

返回图像映射的某个区域的port

返回图像映射的某个区域的href

返回图像映射的某个区域的pathname

返回图像映射的某个区域的protocol

返回一个区域的href属性的querystring部分

返回图像映射的某个区域的shape

返回图像映射的某个区域的target的值

更多的Area 对象的例子，在我们的JavaScript 参考手册。

Base 对象

返回页面上所有相对URL的基URL

返回页面上所有相对链接的基链接

更多的Base 对象对象的例子，在我们的JavaScript 参考手册。

Button 对象

当点击完button不可用

返回一个button的name

返回一个button的type

返回一个button的value

返回一个button所属表的ID

更多Button 对象实例在我们的**JavaScript 参考手册**。

Form 对象

返回一个表单中所有元素的value

返回一个表单acceptCharset属性的值

返回一个表单action属性的值

返回表单中的enctype属性的值

返回一个表单中元素的数量

返回发送表单数据的方法

返回一个表单的name

返回一个表单target属性的值

重置表单

提交表单

更多Button 对象实例在我们的**JavaScript 参考手册**。

Frame/IFrame 对象

对iframe排版

改变一个包含在iframe中的文档的背景颜色

返回一个iframe中的frameborder属性的值

删除iframe的frameborder

改变iframe的高度和宽度

返回一个iframe中的longdesc属性的值

返回一个iframe中的marginheight属性的值

返回一个iframe中的marginwidth属性的值

返回一个iframe中的name属性的值

返回和设置一个iframe中的scrolling属性的值

改变一个iframe的src

更多Frame/IFrame 对象实例在我们的**JavaScript 参考手册**。

Image 对象

对image排版

返回image的替代文本

给image加上border

改变image的高度和宽度

设置image的hspace和vspace属性

返回image的longdesc属性的值

创建一个链接指向一个低分辨率的image

返回image的name

改变image的src

返回一个客户端图像映射的usemap的值

更多Image 对象实例在我们的**JavaScript 参考手册**。

Event 对象

被按下的键盘键的keycode?

鼠标的坐标?

鼠标相对于屏幕的坐标?

shift键被按下了吗?

哪个事件发生了?

Option 和 Select 对象

禁用和启用下拉列表

获得有下拉列表的表单的ID

获得下拉列表的选项数量

将下拉列表变成多行列表

在下拉列表中选择多个选项

弹出下拉列表中被选中的选项

弹出下拉列表中被选中的选项的索引

改变下拉列表中被选中的选项的文本

删除下拉列表中的选项

Table, TableHeader, TableRow, TableData 对象

改变表格边框的宽度

改变表格的cellpadding和cellspacing

指定表格的frame

为表格指定规则

一个行的innerHTML

JavaScript 对象实例 JavaScript 浏览器对象实例

点我分享笔记

反馈/建议

JavaScript 浏览器对象实例 JavaScript 保留关键字

现在您已经学习了 JavaScript，接下来该学习什么呢？

JavaScript 总结

本教程中我们向您讲授了如何向 html 页面添加 JavaScript，使得网站的动态性和交互性更强。

你已经学习了如何创建对事件的响应，验证表单，以及如何根据不同的情况运行不同的脚本。

你也学到了如何创建和使用对象，以及如何使用 JavaScript 的内置对象。

如需更多关于 JavaScript 的信息和知识，请参阅我们的 JavaScript 实例 和 JavaScript 参考手册。

现在你已经学习了 JavaScript，接下来该学习什么呢？

下一步应该学习 HTML DOM 和 DHTML。

如果你希望学习关于服务器端脚本的知识，那么下一步应该学习 ASP,PHP, .Net。

HTML DOM

HTML DOM 定义了访问和操作 HTML 文档的标准方法。 HTML DOM 独立于平台和语言，可被任何编程语言使用，比如 Java、JavaScript 和 VBscript。 假如希望了解更多关于 DOM 的知识，请访问我们的 HTML DOM 教程。

jQuery

jQuery 是一个 JavaScript 库。

jQuery 极大地简化了 JavaScript 编程。

jQuery 很容易学习。

假如希望了解更多关于 jQuery 的知识，请访问我们的 jQuery 教程。

AJAX

AJAX = 异步 JavaScript 和 XML。

AJAX 不是一种新的编程语言，而是一种使用现有标准的新方法。

通过与服务器进行数据交换，AJAX 可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

有很多使用 AJAX 的应用程序案例：新浪微博、Google 地图、开心网等等。

假如您希望学习更多关于 AJAX 的知识，请访问我们的 AJAX 教程。

ASP / PHP / .NET

和 HTML 文档中的脚本运行于客户端（浏览器）不同，ASP/PHP 文件中的脚本在服务器上运行。

使用 ASP，你可以动态地编辑、改变或者添加网站内容，对由 HTML 表单提交而来的数据进行响应，访问数据或者数据库并向浏览器返回结果，或者定制对不同的用户来说更有帮助的网页。

由于 ASP/PHP 文件返回的是纯粹的 HTML，因此可显示在任何浏览器中。

如果希望学习更多关于 ASP 的知识，请访问我们的 ASP 教程。

如果希望学习更多关于 PHP 的知识，请访问我们的 PHP 教程。

如果希望学习更多关于 .NET 的知识，请访问我们的.NET 实例/教程

□ JavaScript 浏览器对象实例　　JavaScript 保留关键字 □

□点我分享笔记

反馈/建议

□

□ HTML DOM Textarea 对象　　HTML DOM Style margin 属性 □

JavaScript 和 HTML DOM 参考手册

所有内置的JavaScript对象

所有浏览器对象

所有HTML DOM对象

JavaScript 对象参考手册

参考手册描述了每个对象的属性和方法，并提供了在线实例。

- Array 对象
- Boolean 对象
- Date 对象
- Math 对象
- Number 对象
- String 对象
- RegExp 对象
- 全局属性和函数

Browser 对象参考手册

参考手册描述了每个对象的属性和方法，并提供了在线实例。

- Window 对象
- Navigator 对象
- Screen 对象
- History 对象
- Location 对象

HTML DOM 参考手册

参考手册描述了 HTML DOM 的属性和方法，并提供在线实例。

- HTML Document
- HTML Element
- HTML Attributes
- HTML Events

HTML DOM 元素对象参考手册

参考手册描述了每个对象的属性和方法，并提供了在线实例。

- Anchor 对象
- Area 对象
- Base 对象
- Body 对象
- Button 对象
- Form 对象
- Frame/IFrame 对象
- Frameset 对象
- Image 对象
- Input Button 对象
- Input Checkbox 对象
- Input File 对象
- Input Hidden 对象
- Input Password 对象
- Input Radio 对象
- Input Reset 对象
- Input Submit 对象
- Input Text 对象
- Link 对象
- Meta 对象
- Object 对象
- Option 对象
- Select 对象
- Style 对象

HTML DOM Document 对象

HTML DOM 节点

在 HTML DOM (Document Object Model) 中，每一个元素都是 **节点**：

文档是一个文档节点。

所有的HTML元素都是元素节点。

所有 HTML 属性都是属性节点。

文本插入到 HTML 元素是文本节点。are text nodes。

注释是注释节点。

Document 对象

当浏览器载入 HTML 文档, 它就会成为 **Document 对象**。

Document 对象是 HTML 文档的根节点。

Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问。

提示： Document 对象是 Window 对象的一部分，可通过 window.document 属性对其进行访问。

浏览器支持

所有主要浏览器都支持 Document 对象。

Document 对象属性和方法

HTML文档中可以使用以下属性和方法:

属性 / 方法	描述
document.activeElement	返回当前获取焦点元素

document.addEventListener()	向文档添加句柄
document.adoptNode(node)	从另外一个文档返回 adapded 节点到当前文档。
document.anchors	返回对文档中所有 Anchor 对象的引用。
document.applets	返回对文档中所有 Applet 对象的引用。 注意：HTML5 已不支持 <applet> 元素。
document.baseURI	返回文档的绝对基础 URI
document.body	返回文档的body元素
document.close()	关闭用 document.open() 方法打开的输出流，并显示选定的数据。
document.cookie	设置或返回与当前文档有关的所有 cookie。
document.createAttribute()	创建一个属性节点
document.createComment()	createComment() 方法可创建注释节点。
document.createDocumentFragment()	创建空的 DocumentFragment 对象，并返回此对象。
document.createElement()	创建元素节点。
document.createTextNode()	创建文本节点。
document.doctype	返回与文档相关的文档类型声明 (DTD)。
document.documentElement	返回文档的根节点
document.documentMode	返回用于通过浏览器渲染文档的模式
document.documentURI	设置或返回文档的位置
document.domain	返回当前文档的域名。
document.domConfig	已废弃。返回 normalizeDocument() 被调用时所使用的配置。
document.embeds	返回文档中所有嵌入的内容（embed）集合
document.forms	返回对文档中所有 Form 对象引用。
document.getElementsByClassName()	返回文档中所有指定类名的元素集合，作为 NodeList 对象。
document.getElementById()	返回对拥有指定 id 的第一个对象的引用。
document.getElementsByName()	返回带有指定名称的对象集合。
document.getElementsByTagName()	返回带有指定标签名的对象集合。
document.images	返回对文档中所有 Image 对象引用。
document.implementation	返回处理该文档的 DOMImplementation 对象。
document.importNode()	把一个节点从另一个文档复制到该文档以便应用。
document.inputEncoding	返回用于文档的编码方式（在解析时）。
document.lastModified	返回文档被最后修改的日期和时间。
document.links	返回对文档中所有 Area 和 Link 对象引用。

document.normalize()	删除空文本节点，并连接相邻节点
document.normalizeDocument()	删除空文本节点，并连接相邻节点的
document.open()	打开一个流，以收集来自任何 document.write() 或 document.writeln() 方法的输出。
document.querySelector()	返回文档中匹配指定的CSS选择器的第一元素
document.querySelectorAll()	document.querySelectorAll() 是 HTML5中引入的新方法，返回文档中匹配的CSS选择器的所有元素节点列表
document.readyState	返回文档状态 (载入中……)
document.referrer	返回载入当前文档的文档的 URL。
document.removeEventListener()	移除文档中的事件句柄(由 addEventListener() 方法添加)
document.renameNode()	重命名元素或者属性节点。
document.scripts	返回页面中所有脚本的集合。
document.strictErrorChecking	设置或返回是否强制进行错误检查。
document.title	返回当前文档的标题。
document.URL	返回文档完整的URL
document.write()	向文档写 HTML 表达式 或 JavaScript 代码。
document.writeln()	等同于 write() 方法，不同的是在每个表达式之后写一个换行符。

警告 !!!

在 W3C DOM核心，文档对象 继承节点对象的所有属性和方法。

很多属性和方法在文档中是没有意义的。

HTML 文档对象可以避免使用这些节点对象和属性：

属性 / 方法	避免的原因
document.attributes	文档没有该属性
document.hasAttributes()	文档没有该属性
document.nextSibling	文档没有下一节点
document.nodeName	这个通常是 #document
document.nodeType	这个通常是 9(DOCUMENT_NODE)
document.nodeValue	文档没有一个节点值
document.ownerDocument	文档没有主文档
document.ownerElement	文档没有自己的节点
document.parentNode	文档没有父节点
document.previousSibling	文档没有兄弟节点
document.textContent	文档没有文本节点

点我分享笔记

反馈/建议