# 第1章 JavaScript 教程

JavaScript 是属于 HTML 和 Web 的编程语言。

编程令计算机完成您需要它们做的工作。

JavaScript 很容易学习。

本教程涵盖 JavaScript 基础和高级教程。

为何学习 JavaScript？

JavaScript 是 web 开发者必学的三种语言之一：

HTML 定义网页的内容

CSS 规定网页的布局

JavaScript 对网页行为进行编程

本教程提供关于 JavaScript，以及 JavaScript 如何与 HTML 和 CSS 协同工作的知识。

您知道吗？

JavaScript 和 Java 是完全不同的语言，不论是概念还是设计。

JavaScript 在 1995 年由 Brendan Eich 发明，并于 1997 年成为一部 ECMA 标准。

ECMA-262 是其官方名称。ECMAScript 6 （发布于 2015 年）是最新的 JavaScript 版本。

学习速度

在本教程中，您可以自由控制学习速度。

一切都取决于您。

如果您遇到困难，请休息一会儿，然后重新读一读材料。

请始终确保您理解了“亲自试一试”中的实例和练习题。

JavaScript 参考手册

W3School 提供并维护了完整的 JavaScript 参考手册，包括所有 HTML DOM 以及浏览器对象。

本参考手册包含了所有对象、属性及方法的示例，并根据最新 web 标准持续更新。

# 第2章 JavaScript 简介

JavaScript 能够改变 HTML 内容

getElementById() 是多个 JavaScript HTML 方法之一。

本例使用该方法来“查找” id="demo" 的 HTML 元素，并把元素内容（innerHTML）更改为 "Hello JavaScript"：

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 能做什么</h2>

<p id="demo">JavaScript 能够改变 HTML 内容。</p>

<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!"'>点击我！</button>

</body>

</html>

提示：JavaScript 同时接受双引号和单引号：

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 能做什么？</h1>

<p id="demo">JavaScript 可以更改 HTML 内容。</p>

<button type="button" onclick="document.getElementById('demo').innerHTML = 'Hello JavaScript!'">单击这里</button>

</body>

</html>

JavaScript 能够改变 HTML 属性

本例通过改变 <img> 标签的 src 属性（source）来改变一张 HTML 图像：

灯泡

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 能做什么？</h2>

<p>JavaScript 能够改变 HTML 属性值。</p>

<p>在本例中，JavaScript 改变了图像的 src 属性值。</p>

<button onclick="document.getElementById('myImage').src='/i/eg\_bulbon.gif'">开灯</button>

<img id="myImage" border="0" src="/i/eg\_bulboff.gif" style="text-align:center;">

<button onclick="document.getElementById('myImage').src='/i/eg\_bulboff.gif'">关灯</button>

</body>

</html>

JavaScript 能够改变 HTML 样式 (CSS)

改变 HTML 元素的样式，是改变 HTML 属性的一种变种：

实例

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 能够做什么</h2>

<p id="demo">JavaScript 能够改变 HTML 元素的样式。</p>

<button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">

点击我！

</button>

</body>

</html>

JavaScript 能够隐藏 HTML 元素

可通过改变 display 样式来隐藏 HTML 元素：

实例

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 能够做什么</h2>

<p id="demo">JavaScript 能够隐藏 HTML 元素。</p>

<button type="button" onclick="document.getElementById('demo').style.display='none'">

点击我！

</button>

</body>

</html>

JavaScript 能够显示 HTML 元素

可通过改变 display 样式来显示隐藏的 HTML 元素：

实例

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 能够做什么</h2>

<p>JavaScript 能够显示隐藏的 HTML 元素。</p>

<p id="demo" style="display:none">Hello JavaScript!</p>

<button type="button" onclick="document.getElementById('demo').style.display='block'">

点击我！

</button>

</body>

</html>

# 第3章 JavaScript 使用

<script> 标签

在 HTML 中，JavaScript 代码必须位于 <script> 与 </script> 标签之间。

实例

<!DOCTYPE html>

<html>

<body>

<h2>Body 中的 JavaScript</h2>

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

<script>

document.getElementById("demo").innerHTML = "我的第一段 JavaScript";

</script>

</body>

</html>

注释：旧的 JavaScript 例子也许会使用 type 属性：<script type="text/javascript">。

注释：type 属性不是必需的。JavaScript 是 HTML 中的默认脚本语言。

JavaScript 函数和事件

JavaScript 函数是一种 JavaScript 代码块，它可以在调用时被执行。

例如，当发生事件时调用函数，比如当用户点击按钮时。

提示：您将在稍后的章节学到更多有关函数和事件的知识。

<head> 或 <body> 中的 JavaScript

您能够在 HTML 文档中放置任意数量的脚本。

脚本可被放置与 HTML 页面的 <body> 或 <head> 部分中，或兼而有之。

<head> 中的 JavaScript

在本例中，JavaScript 函数被放置于 HTML 页面的 <head> 部分。

该函数会在按钮被点击时调用：

实例

<!DOCTYPE html>

<html>

<head>

<script>

function myFunction() {

document.getElementById("demo").innerHTML = "段落已被更改。";

}

</script>

</head>

<body>

<h2>Head 中的 JavaScript</h2>

<p id="demo">一个段落。</p>

<button type="button" onclick="myFunction()">试一试</button>

</body>

</html>

<body> 中的 JavaScript

在本例中，JavaScript 函数被放置于 HTML 页面的 <body> 部分。

该函数会在按钮被点击时调用：

实例

<!DOCTYPE html>

<html>

<body>

<h2>Body 中的 JavaScript</h2>

<p id="demo">一个段落。</p>

<button type="button" onclick="myFunction()">试一试</button>

<script>

function myFunction() {

document.getElementById("demo").innerHTML = "段落已被更改。";

}

</script>

</body>

</html>

提示：把脚本置于 <body> 元素的底部，可改善显示速度，因为脚本编译会拖慢显示。

外部脚本

脚本可放置与外部文件中：

外部文件：myScript.js

function myFunction() {

document.getElementById("demo").innerHTML = "段落被更改。";

}

外部脚本很实用，如果相同的脚本被用于许多不同的网页。

JavaScript 文件的文件扩展名是 .js。

如需使用外部脚本，请在 <script> 标签的 src (source) 属性中设置脚本的名称：

实例

<!DOCTYPE html>

<html>

<body>

<h2>外部 JavaScript</h2>

<p id="demo">一个段落。</p>

<button type="button" onclick="myFunction()">试一试</button>

<p>（myFunction 存储在名为 "myScript.js" 的外部文件中。）</p>

<script src="/demo/myScript.js"></script>

</body>

</html>

您可以在 <head> 或 <body> 中放置外部脚本引用。

该脚本的表现与它被置于 <script> 标签中是一样的。

注释：外部脚本不能包含 <script> 标签。

外部 JavaScript 的优势

在外部文件中放置脚本有如下优势：

分离了 HTML 和代码

使 HTML 和 JavaScript 更易于阅读和维护

已缓存的 JavaScript 文件可加速页面加载

如需向一张页面添加多个脚本文件 - 请使用多个 script 标签：

实例

<script src="myScript1.js"></script>

<script src="myScript2.js"></script>

外部引用

可通过完整的 URL 或相对于当前网页的路径引用外部脚本：

本例使用完整的 URL 来链接至脚本：

实例

<!DOCTYPE html>

<html>

<body>

<h1>外部 JavaScript</h1>

<p id="demo">一个段落。</p>

<button type="button" onclick="myFunction()">试一试</button>

<p>（myFunction 存储在名为 "myScript.js" 的外部文件中。）</p>

<script src="https://www.w3school.com.cn/demo/myScript.js"></script>

</body>

</html>

本例使用了位于当前网站上指定文件夹中的脚本：

实例

<!DOCTYPE html>

<html>

<body>

<h2>外部 JavaScript</h2>

<p id="demo">一个段落。</p>

<button type="button" onclick="myFunction()">试一试</button>

<p>（myFunction 存储在名为 "myScript.js" 的外部文件中。）</p>

<script src="/demo/myScript.js"></script>

</body>

</html>

本例链接了与当前页面相同文件夹的脚本：

实例

<!DOCTYPE html>

<html>

<body>

<h2>外部 JavaScript</h2>

<p id="demo">一个段落。</p>

<button type="button" onclick="myFunction()">试一试</button>

<p>（myFunction 存储在名为 "myScript.js" 的外部文件中。）</p>

<script src="/demo/myScript.js"></script>

</body>

</html>

# 第4章 JavaScript 输出

JavaScript 不提供任何内建的打印或显示函数。

JavaScript 显示方案

JavaScript 能够以不同方式“显示”数据：

使用 window.alert() 写入警告框

使用 document.write() 写入 HTML 输出

使用 innerHTML 写入 HTML 元素

使用 console.log() 写入浏览器控制台

使用 innerHTML

如需访问 HTML 元素，JavaScript 可使用 document.getElementById(id) 方法。

id 属性定义 HTML 元素。innerHTML 属性定义 HTML 内容：

实例

<!DOCTYPE html>

<html>

<body>

<h2>我的第一张网页</h2>

<p>我的第一个段落。</p>

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

<script>

document.getElementById("demo").innerHTML = 5 + 6;

</script>

</body>

</html>

提示：更改 HTML 元素的 innerHTML 属性是在 HTML 中显示数据的常用方法。

使用 document.write()

出于测试目的，使用 document.write() 比较方便：

实例

<!DOCTYPE html>

<html>

<body>

<h2>我的第一张网页</h2>

<p>我的第一个段落。</p>

<script>

document.write(5 + 6);

</script>

</body>

</html>

注意：在 HTML 文档完全加载后使用 document.write() 将删除所有已有的 HTML ：

实例

<!DOCTYPE html>

<html>

<body>

<h2>我的第一张网页</h2>

<p>我的第一个段落。</p>

<button type="button" onclick="document.write(5 + 6)">试一试</button>

</body>

</html>

提示：document.write() 方法仅用于测试。

使用 window.alert()

您能够使用警告框来显示数据：

实例

<!DOCTYPE html>

<html>

<body>

<h2>我的第一张网页</h2>

<p>我的第一个段落。</p>

<script>

window.alert(5 + 6);

</script>

</body>

</html>

使用 console.log()

在浏览器中，您可使用 console.log() 方法来显示数据。

请通过 F12 来激活浏览器控制台，并在菜单中选择“控制台”。

实例

<!DOCTYPE html>

<html>

<body>

<h2>按 F12 启动</h2>

<p>在 debugger 中选择 "Console"。然后再次点击运行按钮。</p>

<script>

console.log(5 + 6);

</script>

</body>

</html>

# 第5章 JavaScript 语句

在 HTML 中，JavaScript 语句是由 web 浏览器“执行”的“指令”。

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 语句</h2>

<p><b>JavaScript 程序</b> 是一系列由计算机执行的 <b>语句</b>。</p>

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

<script>

var x, y, z; // 语句 1

x = 22; // 语句 2

y = 11; // 语句 3

z = x + y; // 语句 4

document.getElementById("demo").innerHTML =

"z 的值是" + z + "。";

</script>

</body>

</html>

JavaScript 程序

计算机程序是由计算机“执行”的一系列“指令”。

在编程语言中，这些编程指令被称为语句。

JavaScript 程序就是一系列的编程语句。

注释：在 HTML 中，JavaScript 程序由 web 浏览器执行。

JavaScript 语句

JavaScript 语句由以下构成：

值、运算符、表达式、关键词和注释。

这条语句告诉浏览器在 id="demo" 的 HTML 元素中输出 "Hello Kitty."：

实例

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 语句</h2>

<p>在 HTML 中，JavaScript 语句由浏览器执行。</p>

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

<script>

document.getElementById("demo").innerHTML = "Hello Kitty.";

</script>

</body>

</html>

大多数 JavaScript 程序都包含许多 JavaScript 语句。

这些语句会按照它们被编写的顺序逐一执行。

注释：JavaScript 程序（以及 JavaScript 语句）常被称为 JavaScript 代码。

分号 ;

分号分隔 JavaScript 语句。

请在每条可执行的语句之后添加分号：

a = 5;

b = 6;

c = a + b;

如果有分号分隔，允许在同一行写多条语句：

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

您可能在网上看到不带分号的例子。

提示：以分号结束语句不是必需的，但我们仍然强烈建议您这么做。

JavaScript 空白字符

JavaScript 会忽略多个空格。您可以向脚本添加空格，以增强可读性。

下面这两行是相等的：

var person = "Bill";

var person="Bill";

在运算符旁边（ = + - \* / ）添加空格是个好习惯：

var x = y + z;

JavaScript 行长度和折行

为了达到最佳的可读性，程序员们常常喜欢把代码行控制在 80 个字符以内。

如果 JavaScript 语句太长，对其进行折行的最佳位置是某个运算符：

实例

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 语句</h2>

<p>

折行的最佳位置是运算符或逗号之后。

</p>

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

<script>

document.getElementById("demo").innerHTML =

"Hello Kitty.";

</script>

</body>

</html>

JavaScript 代码块

JavaScript 语句可以用花括号（{...}）组合在代码块中。

代码块的作用是定义一同执行的语句。

您会在 JavaScript 中看到成块组合在一起的语句：

实例

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 语句</h2>

<p>JavaScript 代码块在 { 与 } 之间编写。</p>

<button type="button" onclick="myFunction()">点击我！</button>

<p id="demo1"></p>

<p id="demo2"></p>

<script>

function myFunction() {

document.getElementById("demo1").innerHTML = "Hello Kitty.";

document.getElementById("demo2").innerHTML = "How are you?";

}

</script>

</body>

</html>

注释：在本教程中我们为代码块用了 4 个空格的缩进。

提示：您将稍后在本教程中学到更多有关函数的内容。

JavaScript 关键词

JavaScript 语句常常通过某个关键词来标识需要执行的 JavaScript 动作。

下面的表格列出了一部分将在教程中学到的关键词：

|  |  |
| --- | --- |
| 关键词 | 描述 |
| break | 终止 switch 或循环。 |
| continue | 跳出循环并在顶端开始。 |
| debugger | 停止执行 JavaScript，并调用调试函数（如果可用）。 |
| do ... while | 执行语句块，并在条件为真时重复代码块。 |
| for | 标记需被执行的语句块，只要条件为真。 |
| function | 声明函数。 |
| if ... else | 标记需被执行的语句块，根据某个条件。 |
| return | 退出函数。 |
| switch | 标记需被执行的语句块，根据不同的情况。 |
| try ... catch | 对语句块实现错误处理。 |
| var | 声明变量。 |

注释：JavaScript 关键词指的是保留的单词。保留词无法用作变量名。

# 第6章 JavaScript 语法

JavaScript 语法是一套规则，它定义了 JavaScript 的语言结构。

var x, y; // 如何声明变量

x = 7; y = 8; // 如何赋值

z = x + y; // 如何计算值

JavaScript 值

JavaScript 语句定义两种类型的值：混合值和变量值。

混合值被称为字面量（literal）。变量值被称为变量。

JavaScript 字面量

书写混合值最重要的规则是：

写数值有无小数点均可：

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 数值</h2>

<p>写数值有无小数点均可。</p>

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

<script>

document.getElementById("demo").innerHTML = 15.90;

</script>

</body>

</html>

字符串是文本，由双引号或单引号包围：

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 字符串</h2>

<p>字符串可由双引号或单引号包围。</p>

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

<script>

document.getElementById("demo").innerHTML = 'Bill Gates';

</script>

</body>

</html>

JavaScript 变量

在编程语言中，变量用于存储数据值。

JavaScript 使用 var 关键词来声明变量。

= 号用于为变量赋值。

在本例中，x 被定义为变量。然后，x 被赋的值是 7：

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 变量</h2>

<p>在本例中，x 被定义为变量。然后 x 被赋值 7：</p>

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

<script>

var x;

x = 7;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

JavaScript 运算符

JavaScript 使用算数运算符（+ - \* /）来计算值：

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 运算符</h2>

<p>JavaScript 使用算数运算符来计算值（类似代数）。</p>

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

<script>

document.getElementById("demo").innerHTML = (7 + 8) \* 10;

</script>

</body>

</html>

JavaScript 使用赋值运算符（=）向变量赋值：

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 赋值</h2>

<p>在 JavaScript 中，= 运算符用于向变量赋值。</p>

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

<script>

var x, y;

x = 7;

y = 8;

document.getElementById("demo").innerHTML = x + y;

</script>

</body>

</html>

JavaScript 表达式

表达式是值、变量和运算符的组合，计算结果是值。

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 表达式</h2>

<p>表达式计算为值。</p>

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

<script>

document.getElementById("demo").innerHTML = 6 \* 10;

</script>

</body>

</html>

表达式也可包含变量值：

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 表达式</h2>

<p>表达式计算为值。</p>

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

<script>

var x;

x = 6;

document.getElementById("demo").innerHTML = x \* 10;

</script>

</body>

</html>

值可以是多种类型，比如数值和字符串。

例如，"Bill" + " " + "Gates"，计算为 "Bill Gates"：

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 表达式</h2>

<p>表达式计算为值。</p>

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

<script>

document.getElementById("demo").innerHTML = "Bill" + " " + "Gates";

</script>

</body>

</html>

JavaScript 关键词

JavaScript 关键词用于标识被执行的动作。

var 关键词告知浏览器创建新的变量：

<!DOCTYPE html>

<html>

<body>

<h2>var 关键词创建变量</h2>

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

<script>

var x, y;

x = 7 + 8;

y = x \* 10;

document.getElementById("demo").innerHTML = y;

</script>

</body>

</html>

JavaScript 注释

并非所有 JavaScript 语句都被“执行”。

双斜杠 // 或 /\* 与 \*/ 之间的代码被视为注释。

注释会被忽略，不会被执行：

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 注释<b>不会</b>被执行。</h2>

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

<script>

var x;

x = 7;

// x = 8; 我不会被执行

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

JavaScript 标识符

标识符是名称。

在 JavaScript 中，标识符用于命名变量（以及关键词、函数和标签）。

在大多数编程语言中，合法名称的规则大多相同。

在 JavaScript 中，首字符必须是字母、下划线（-）或美元符号（$）。

连串的字符可以是字母、数字、下划线或美元符号。

提示：数值不可以作为首字符。这样，JavaScript 就能轻松区分标识符和数值。

JavaScript 对大小写敏感

所有 JavaScript 标识符对大小写敏感。

变量 lastName 和 lastname，是两个不同的变量。

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 对大小写敏感</h2>

<p>请试着把 lastName 改为 lastname。</p>

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

<script>

var lastname, lastName;

lastName = "Gates";

lastname = "Jobs";

document.getElementById("demo").innerHTML = lastName;

</script>

</body>

</html>

JavaScript 不会把 VAR 或 Var 译作关键词 var。

JavaScript 与驼峰式大小写

历史上，程序员曾使用三种把多个单词连接为一个变量名的方法：

连字符：

first-name, last-name, master-card, inter-city.

注释：JavaScript 中不能使用连字符。它是为减法预留的。

下划线：

first\_name, last\_name, master\_card, inter\_city.

驼峰式大小写（Camel Case）：

FirstName, LastName, MasterCard, InterCity.

JavaScript 程序员倾向于使用以小写字母开头的驼峰大小写：

firstName, lastName, masterCard, interCity

JavaScript 字符集

JavaScript 使用 Unicode 字符集。

Unicode 覆盖世界上几乎所有的字符、标点和符号。

如需了解详情，请访问我们完整的 Unicode 参考手册。

# 第7章 JavaScript 注释

JavaScript 注释用于解释 JavaScript 代码，增强其可读性。

JavaScript 注释也可以用于在测试替代代码时阻止执行。

单行注释

单行注释以 // 开头。

任何位于 // 与行末之间的文本都会被 JavaScript 忽略（不会执行）。

本例在每条代码行之前使用单行注释：

<!DOCTYPE html>

<html>

<body>

<h1 id="myH"></h1>

<p id="myP"></p>

<script>

// 改变标题：

document.getElementById("myH").innerHTML = "JavaScript 注释";

// 改变段落：

document.getElementById("myP").innerHTML = "我的第一个段落。";

</script>

</body>

</html>

多行注释

多行注释以 /\* 开头，以 \*/ 结尾。

任何位于 /\* 和 \*/ 之间的文本都会被 JavaScript 忽略。

本例使用多行注释（注释块）来解释代码：

实例

<!DOCTYPE html>

<html>

<body>

<h1 id="myH"></h1>

<p id="myP"></p>

<script>

/\*

下面的代码会改变

id = "myH" 的标题

以及 id = "myp" 的段落

\*/

document.getElementById("myH").innerHTML = "JavaScript 注释";

document.getElementById("myP").innerHTML = "我的第一个段落。";

</script>

</body>

</html>

注释：使用单行注释最为常见。

提示：注释块常用于官方声明。

使用注释来阻止执行

使用注释来防止代码执行很适合代码测试。

在代码行之前添加 // 会把可执行的代码行更改为注释。

本例使用 // 来阻止代码行的执行：

实例

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 注释</h2>

<h1 id="myH"></h1>

<p id="myP"></p>

<script>

//document.getElementById("myH").innerHTML = "我的首页";

document.getElementById("myP").innerHTML = "我的第一个段落。";

</script>

<p>// 开头的代码行不会被执行。</p>

</body>

</html>

本例使用注释块来阻止多行代码的执行：

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 注释</h2>

<h1 id="myH"></h1>

<p id="myP"></p>

<script>

/\*

document.getElementById("myH").innerHTML = "欢迎访问我的首页";

document.getElementById("myP").innerHTML = "这是我的第一个段落。";

\*/

document.getElementById("myP").innerHTML = "注释块不会被执行。";

</script>

</body>

</html>

# 第8章 JavaScript 变量

JavaScript 变量

JavaScript 变量是存储数据值的容器。

在本例中，x、y 和 z 是变量：

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 变量</h1>

<p>在本例中，x、y 以及 z 都是变量。</p>

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

<script>

var x = 7;

var y = 8;

var z = x + y;

document.getElementById("demo").innerHTML =

"z 的值是：" + z;

</script>

</body>

</html>

从上例中，您可获得：

x 存储值 7

y 存储值 8

z 存储值 15

类似代数

在本例中，price1、price2 以及 total 是变量：

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 变量</h1>

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

<script>

var price1 = 7;

var price2 = 8;

var price3 = 12;

var total = price1 + price2 + price3;

document.getElementById("demo").innerHTML = "总计：" + total;

</script>

</body>

</html>

在编程中，类似代数，我们使用变量（比如 price1）来存放值。

在编程中，类似代数，我们在表达式中使用变量（total = price1 + price2）。

从上例中，您可以算出 total 的值是 27。

提示：JavaScript 变量是存储数据值的容器。

JavaScript 标识符

所有 JavaScript 变量必须以唯一的名称的标识。

这些唯一的名称称为标识符。

标识符可以是短名称（比如 x 和 y），或者更具描述性的名称（age、sum、totalVolume）。

构造变量名称（唯一标识符）的通用规则是：

名称可包含字母、数字、下划线和美元符号

名称必须以字母开头

名称也可以 $ 和 \_ 开头（但是在本教程中我们不会这么做）

名称对大小写敏感（y 和 Y 是不同的变量）

保留字（比如 JavaScript 的关键词）无法用作变量名称

提示：JavaScript 标识符对大小写敏感。

赋值运算符

在 JavaScript 中，等号（=）是赋值运算符，而不是“等于”运算符。

这一点与代数不同。下面的代码在代数中是不合理的：

x = x + 5

然而在 JavaScript 中，它非常合理：把 x + 5 的值赋给 x。

（计算 x + 5 的值并把结果放入 x 中。x 的值递增 5。）

注释：JavaScript 中的“等于”运算符是 ==。

JavaScript 数据类型

JavaScript 变量可存放数值，比如 100，以及文本值，比如 "Bill Gates"。

在编程中，文本值被称为字符串。

JavaScript 可处理多种数据类型，但是现在，我们只关注数值和字符串值。

字符串被包围在双引号或单引号中。数值不用引号。

如果把数值放在引号中，会被视作文本字符串。

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 变量</h1>

<p>字符串用引号包围。</p>

<p>数值不用引号包围。</p>

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

<script>

var pi = 3.14;

var person = "Bill Gates";

var answer = 'How are you!';

document.getElementById("demo").innerHTML =

pi + "<br>" + person + "<br>" + answer;

</script>

</body>

</html>

声明（创建） JavaScript 变量

在 JavaScript 中创建变量被称为“声明”变量。

您可以通过 var 关键词来声明 JavaScript 变量：

var carName;

声明之后，变量是没有值的。（技术上，它的值是 undefined。）

如需赋值给变量，请使用等号：

carName = "porsche";

您可以在声明变量时向它赋值：

var carName = "porsche";

在上面的例子中，我们创建了名为 carName 的变量，并向其赋值 "porsche"。

然后，我们在 id="demo" 的 HTML 段落中“输出”该值：

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 变量</h1>

<p>创建一个变量，向其赋值，然后显示它：</p>

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

<script>

var carName = "porsche";

document.getElementById("demo").innerHTML = carName;

</script>

</body>

</html>

提示：在脚本的开头声明所有变量是个好习惯！

一条语句，多个变量

您可以在一条语句中声明许多变量。

以 var 作为语句的开头，并以逗号分隔变量：

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 变量</h1>

<p>您可以在一条语句中声明许多变量。</p>

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

<script>

var person = "Bill Gates", carName = "porsche", price = 150000;

document.getElementById("demo").innerHTML = carName;

</script>

</body>

</html>

声明可横跨多行：

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 变量</h1>

<p>您可以在一条语句中声明多个变量。</p>

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

<script>

var person = "Bill Gates",

carName = "porsche",

price = 150000;

document.getElementById("demo").innerHTML = carName;

</script>

</body>

</html>

Value = undefined

在计算机程序中，被声明的变量经常是不带值的。值可以是需被计算的内容，或是之后被提供的数据，比如数据输入。

不带有值的变量，它的值将是 undefined。

变量 carName 在这条语句执行后的值是 undefined：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 变量</h1>

<p>如果变量声明时没有赋值，那么其值是 undefined。</p>

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

<script>

var carName;

document.getElementById("demo").innerHTML = carName;

</script>

</body>

</html>

重复声明 JavaScript 变量

如果再次声明某个 JavaScript 变量，将不会丢它的值。

在这两条语句执行后，变量 carName 的值仍然是 "porsche"：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 变量</h1>

<p>如果您再次声明某个 JavaScript 变量，那么它的值不会丢失。</p>

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

<script>

var carName = "porsche";

var carName;

document.getElementById("demo").innerHTML = carName;

</script>

</body>

</html>

JavaScript 算术

与代数类似，您能够通过 JavaScript 变量进行算术运算，使用 = 和 + 之类的运算符：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 变量</h1>

<p>相加 5 + 2 + 3 的结果：</p>

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

<script>

var x = 3 + 5 + 8;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

字符串也可以使用加号，但是字符串将被级联：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 变量</h1>

<p>相加 "Bill" + " " + "Gates" 的结果是：</p>

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

<script>

var x = "Bill" + " " + "Gates";

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

还可以试试这个：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 变量</h1>

<p>相加 "8" + 3 + 5 的结果是：</p>

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

<script>

x = "8" + 3 + 5;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

提示：如果把要给数值放入引号中，其余数值会被视作字符串并被级联。

现在试试这个：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 变量</h1>

<p>相加 3 + 5 + "8" 的结果是：</p>

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

<script>

var x = 3 + 5 + "8"

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

# 第9章 JavaScript Let

ECMAScript 2015

ES2015 引入了两个重要的 JavaScript 新关键词：let 和 const。

这两个关键字在 JavaScript 中提供了块作用域（Block Scope）变量（和常量）。

在 ES2015 之前，JavaScript 只有两种类型的作用域：全局作用域和函数作用域。

全局作用域

全局（在函数之外）声明的变量拥有全局作用域。

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 全局作用域</h1>

<p>可以从任何脚本或函数中访问全局变量。</p>

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

<script>

var carName = "porsche";

myFunction();

function myFunction() {

document.getElementById("demo").innerHTML =

"I can display " + carName;

}

</script>

</body>

</html>

全局变量可以在 JavaScript 程序中的任何位置访问。

函数作用域

局部（函数内）声明的变量拥有函数作用域。

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 局部作用域</h1>

<p>myFunction() 之外的 carName 未定义。</p>

<p id="demo1"></p>

<p id="demo2"></p>

<script>

myFunction();

function myFunction() {

var carName = "porsche";

document.getElementById("demo1").innerHTML =

typeof carName + " " + carName;

}

document.getElementById("demo2").innerHTML =

typeof carName;

</script>

</body>

</html>

局部变量只能在它们被声明的函数内访问。

JavaScript 块作用域

通过 var 关键词声明的变量没有块作用域。

在块 {} 内声明的变量可以从块之外进行访问。

实例

{

var x = 10;

}

// 此处可以使用 x

在 ES2015 之前，JavaScript 是没有块作用域的。

可以使用 let 关键词声明拥有块作用域的变量。

在块 {} 内声明的变量无法从块外访问：

实例

{

let x = 10;

}

// 此处不可以使用 x

重新声明变量

使用 var 关键字重新声明变量会带来问题。

在块中重新声明变量也将重新声明块外的变量：

实例

<!DOCTYPE html>

<html>

<body>

<h1>使用 var 声明变量</h1>

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

<script>

var x = 10;

// Here x is 10

{

var x = 2;

// Here x is 2

}

// Here x is 2

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

使用 let 关键字重新声明变量可以解决这个问题。

在块中重新声明变量不会重新声明块外的变量：

实例

<!DOCTYPE html>

<html>

<body>

<h1>使用 let 声明变量</h1>

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

<script>

var x = 10;

// Here x is 10

{

let x = 2;

// Here x is 2

}

// Here x is 10

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

循环作用域

在循环中使用 var：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript let</h1>

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

<script>

var i = 5;

for (var i = 0; i < 10; i++) {

// 一些语句

}

document.getElementById("demo").innerHTML = i;

</script>

</body>

</html>

在循环中使用 let：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript let</h1>

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

<script>

let i = 5;

for (let i = 0; i < 10; i++) {

// 一些语句

}

document.getElementById("demo").innerHTML = i;

</script>

</body>

</html>

在第一个例子中，在循环中使用的变量使用 var 重新声明了循环之外的变量。

在第二个例子中，在循环中使用的变量使用 let 并没有重新声明循环外的变量。

如果在循环中用 let 声明了变量 i，那么只有在循环内，变量 i 才是可见的。

函数作用域

在函数内声明变量时，使用 var 和 let 很相似。

它们都有函数作用域：

function myFunction() {

var carName = "porsche"; // 函数作用域

}

function myFunction() {

let carName = "porsche"; // 函数作用域

}

全局作用域

如果在块外声明声明，那么 var 和 let 也很相似。

它们都拥有全局作用域：

var x = 10; // 全局作用域

let y = 6; // 全局作用域

HTML 中的全局变量

使用 JavaScript 的情况下，全局作用域是 JavaScript 环境。

在 HTML 中，全局作用域是 window 对象。

通过 var 关键词定义的全局变量属于 window 对象：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 全局变量</h1>

<p>在 HTML 中，通过 <b>var</b> 定义的全局变量会成为 window 变量。</p>

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

<script>

var carName = "Audi";

// 此处的代码能够使用 window.carName

document.getElementById("demo").innerHTML = "我可以显示 " + window.carName;

</script>

</body>

</html>

通过 let 关键词定义的全局变量不属于 window 对象：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 全局变量</h1>

<p>在 HTML 中，由 <b>let</b> 定义的全局变量，不会成为 window 变量。</p>

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

<script>

let carName = "Audi";

// 此处的代码能够使用 window.carName

document.getElementById("demo").innerHTML = "我不能显示 " + window.carName;

</script>

</body>

</html>

重新声明

允许在程序的任何位置使用 var 重新声明 JavaScript 变量：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript let</h1>

<p>在程序的任何位置都允许使用 <b>var</b> 重新声明JavaScript变量：</p>

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

<script>

var x = 2;

// 此时 x 是 2

var x = 3;

// 此时 x 是 3

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

在相同的作用域，或在相同的块中，通过 let 重新声明一个 var 变量是不允许的：

实例

var x = 10; // 允许

let x = 6; // 不允许

{

var x = 10; // 允许

let x = 6; // 不允许

}

在相同的作用域，或在相同的块中，通过 let 重新声明一个 let 变量是不允许的：

实例

let x = 10; // 允许

let x = 6; // 不允许

{

let x = 10; // 允许

let x = 6; // 不允许

}

在相同的作用域，或在相同的块中，通过 var 重新声明一个 let 变量是不允许的：

实例

let x = 10; // 允许

var x = 6; // 不允许

{

let x = 10; // 允许

var x = 6; // 不允许

}

在不同的作用域或块中，通过 let 重新声明变量是允许的：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript let</h1>

<p>使用 <b>let</b>，在另一个作用域或另一个块中重新声明变量是允许的：</p>

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

<script>

let x = 2; // 允许

{

let x = 3; // 允许

}

{

let x = 4; // 允许

}

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

提升

通过 var 声明的变量会提升到顶端。如果您不了解什么是提升（Hoisting），请学习我们的提升这一章。

您可以在声明变量之前就使用它：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 提升</h1>

<p>使用 <b>var</b>，您可以在声明之前使用变量：</p>

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

<script>

carName = "Audi";

document.getElementById("demo").innerHTML = carName;

var carName;

</script>

</body>

</html>

通过 let 定义的变量不会被提升到顶端。

在声明 let 变量之前就使用它会导致 ReferenceError。

变量从块的开头一直处于“暂时死区”，直到声明为止：

实例

// 在此处，您不可以使用 carName

let carName;

# 第10章 JavaScript Const

ECMAScript 2015

ES2015 引入了两个重要的 JavaScript 新关键词：let 和 const。

通过 const 定义的变量与 let 变量类似，但不能重新赋值：

实例

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript const</h2>

<p>您无法更改原始值。</p>

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

<script>

try {

const PI = 3.141592653589793;

PI = 3.14;

}

catch (err) {

document.getElementById("demo").innerHTML = err;

}

</script>

</body>

</html>

块作用域

在块作用域内使用 const 声明的变量与 let 变量相似。

在本例中，x 在块中声明，不同于在块之外声明的 x：

实例

<!DOCTYPE html>

<html>

<body>

<h2>使用 const 声明变量</h2>

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

<script>

var x = 10;

// 此处 x 是 10

{

const x = 2;

// 此处 x 是 2

}

// 此处 x 是 10

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

在声明时赋值

JavaScript const 变量必须在声明时赋值：

不正确

const PI;

PI = 3.14159265359;

正确

const PI = 3.14159265359;

不是真正的常数

关键字 const 有一定的误导性。

它没有定义常量值。它定义了对值的常量引用。

因此，我们不能更改常量原始值，但我们可以更改常量对象的属性。

原始值

如果我们将一个原始值赋给常量，我们就不能改变原始值：

实例

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript const</h2>

<p>您无法更改原始值。</p>

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

<script>

try {

const PI = 3.141592653589793;

PI = 3.14;

}

catch (err) {

document.getElementById("demo").innerHTML = err;

}

</script>

</body>

</html>

常量对象可以更改

您可以更改常量对象的属性：

实例

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript const</h2>

<p>声明常量对象不会使对象属性不可更改：</p>

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

<script>

// 创建对象：

const car = {type:"Porsche", model:"911", color:"white"};

// 更改属性：

car.color = "red";

// 添加属性：

car.owner = "Steve";

// 显示属性：

document.getElementById("demo").innerHTML = "车主是：" + car.owner;

</script>

</body>

</html>

但是您无法重新为常量对象赋值：

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript const</h2>

<p>但是你不能重新赋值一个常量对象：</p>

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

<script>

try {

const car = {type:"Porsche", model:"911", color:"white"};

car = {type:"BMW", model:"M5", color:"red"};

}

catch (err) {

document.getElementById("demo").innerHTML = err;

}

</script>

</body>

</html>

常量数组可以更改

您可以更改常量数组的元素：

实例

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript const</h2>

<p>声明常量数组不会使元素不可更改：</p>

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

<script>

// 创建数组：

const cars = ["Audi", "BMW", "Porsche"];

// 更改元素：

cars[0] = "Toyota";

// 添加元素：

cars.push("Honda");

// 显示数组：

document.getElementById("demo").innerHTML = cars;

</script>

</body>

</html>

但是您无法重新为常量数组赋值：

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript const</h2>

<p>您无法重新赋值常量数组：</p>

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

<script>

try {

const cars = ["Saab", "Volvo", "BMW"];

cars = ["Toyota", "Volvo", "Audi"];

}

catch (err) {

document.getElementById("demo").innerHTML = err;

}

</script>

</body>

</html>

重新声明

在程序中的任何位置都允许重新声明 JavaScript var 变量：

实例

var x = 2; // 允许

var x = 3; // 允许

x = 4; // 允许

在同一作用域或块中，不允许将已有的 var 或 let 变量重新声明或重新赋值给 const：

实例

var x = 2; // 允许

const x = 2; // 不允许

{

let x = 2; // 允许

const x = 2; // 不允许

}

在同一作用域或块中，为已有的 const 变量重新声明声明或赋值是不允许的：

实例

const x = 2; // 允许

const x = 3; // 不允许

x = 3; // 不允许

var x = 3; // 不允许

let x = 3; // 不允许

{

const x = 2; // 允许

const x = 3; // 不允许

x = 3; // 不允许

var x = 3; // 不允许

let x = 3; // 不允许

}

在另外的作用域或块中重新声明 const 是允许的：

实例

const x = 2; // 允许

{

const x = 3; // 允许

}

{

const x = 4; // 允许

}

提升

通过 var 定义的变量会被提升到顶端。如果您不了解什么是提升（Hoisting），请学习提升这一章。

您可以在声明 var 变量之前就使用它：

实例

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript 提升</h2>

<p>使用 <b>var</b>，您可以在声明变量之前使用它：</p>

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

<script>

carName = "Audi";

document.getElementById("demo").innerHTML = carName;

var carName;

</script>

</body>

</html>

通过 const 定义的变量不会被提升到顶端。

const 变量不能在声明之前使用：

实例

carName = "Volvo"; // 您不可以在此处使用 carName

const carName = "Volvo";

# 第11章 JavaScript 运算符

JavaScript 运算符

实例

向变量赋值，并把它们相加：

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 运算符</h1>

<p>x = 7，y = 8，计算 z = x + y，然后显示 z：</p>

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

<script>

var x = 7;

var y = 8;

var z = x + y;

document.getElementById("demo").innerHTML = z;

</script>

</body>

</html>

赋值运算符（=）把值赋给变量。

赋值

<!DOCTYPE html>

<html>

<body>

<h1>= 运算符</h1>

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

<script>

var x = 15;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

加法运算符（+）对数字相加：

加法

<!DOCTYPE html>

<html>

<body>

<h1>+ 运算符</h1>

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

<script>

var x = 7;

var y = 8;

var z = x + y;

document.getElementById("demo").innerHTML = z;

</script>

</body>

</html>

乘法运算符（\*）对数字相乘：

乘法

<!DOCTYPE html>

<html>

<body>

<h1>\* 运算符</h1>

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

<script>

var x = 7;

var y = 8;

var z = x \* y;

document.getElementById("demo").innerHTML = z;

</script>

</body>

</html>

JavaScript 算数运算符

算数运算符用于对数字执行算数运算：

|  |  |
| --- | --- |
| 运算符 | 描述 |
| + | 加法 |
| - | 减法 |
| \* | 乘法 |
| / | 除法 |
| % | 取模（余数） |
| ++ | 递加 |
| -- | 递减 |

注释：JS 算数这一章对算数运算符进行了完整描述。

JavaScript 赋值运算符

赋值运算符向 JavaScript 变量赋值。

|  |  |  |
| --- | --- | --- |
| 运算符 | 例子 | 等同于 |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |

加法赋值运算符（+=）向变量添加一个值。

赋值

<!DOCTYPE html>

<html>

<body>

<h1>+= 运算符</h1>

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

<script>

var x = 7;

x += 8;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

注释：JS 赋值这一章中完整描述了赋值运算符。

JavaScript 字符串运算符

+ 运算符也可用于对字符串进行相加（concatenate，级联）。

实例

txt1 = "Bill";

txt2 = "Gates";

txt3 = txt1 + " " + txt2;

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 运算符</h1>

<p>+ 运算符串联（相加）字符串。</p>

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

<script>

var txt1 = "Bill";

var txt2 = "Gates";

document.getElementById("demo").innerHTML = txt1 + " " + txt2;

</script>

</body>

</html>

+= 赋值运算符也可用于相加（级联）字符串：

实例

txt1 = "Hello ";

txt1 += "Kitty!";

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 运算符</h1>

<p>赋值运算符 += 串联字符串。</p>

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

<script>

txt1 = "Hello ";

txt1 += "Kitty！";

document.getElementById("demo").innerHTML = txt1;

</script>

</body>

</html>

提示：在用于字符串时，+ 运算符被称为级联运算符。

字符串和数字的相加

相加两个数字，将返回和，但对一个数字和一个字符串相加将返回一个字符串：

实例

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript 运算符</h1>

<p>对数字和字符串相加，会返回字符串。</p>

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

<script>

var x = 7 + 8;

var y = "7" + 8;

var z = "Hello" + 7;

document.getElementById("demo").innerHTML =

x + "<br>" + y + "<br>" + z;

</script>

</body>

</html>

提示：如果您对数字和字符串相加，结果将是字符串！

JavaScript 比较运算符

|  |  |
| --- | --- |
| 运算符 | 描述 |
| == | 等于 |
| === | 等值等型 |
| != | 不相等 |
| !== | 不等值或不等型 |
| > | 大于 |
| < | 小于 |
| >= | 大于或等于 |
| <= | 小于或等于 |
| ? | 三元运算符 |

注释：JS 比较这一章中完整描述了比较运算符。

JavaScript 逻辑运算符

|  |  |
| --- | --- |
| 运算符 | 描述 |
| && | 逻辑与 |
| || | 逻辑或 |
| ! | 逻辑非 |

JavaScript 类型运算符

|  |  |
| --- | --- |
| 运算符 | 描述 |
| typeof | 返回变量的类型。 |
| instanceof | 返回 true，如果对象是对象类型的实例。 |

JavaScript 位运算符

位运算符处理 32 位数。

该运算中的任何数值运算数都会被转换为 32 位的数。结果会被转换回 JavaScript 数。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 运算符 | 描述 | 例子 | 等同于 | 结果 | 十进制 |
| & | 与 | 5 & 1 | 0101 & 0001 | 0001 | 1 |
| | | 或 | 5 | 1 | 0101 | 0001 | 0101 | 5 |
| ~ | 非 | ~ 5 | ~0101 | 1010 | 10 |
| ^ | 异或 | 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | 零填充左位移 | 5 << 1 | 0101 << 1 | 1010 | 10 |
| >> | 有符号右位移 | 5 >> 1 | 0101 >> 1 | 0010 | 2 |
| >>> | 零填充右位移 | 5 >>> 1 | 0101 >>> 1 | 0010 | 2 |

上例使用 4 位无符号的例子。但是 JavaScript 使用 32 位有符号数。

因此，在 JavaScript 中，~ 5 不会返回 10，而是返回 -6。

~00000000000000000000000000000101 将返回 11111111111111111111111111111010。

注释：我们将在 JS 位运算这一章为您详解位运算符。

# 第12章 JavaScript 算数

处理数值的典型情景是算数。

JavaScript 算数运算符

算术运算符对数值（文字或变量）执行算术运算。

|  |  |
| --- | --- |
| 运算符 | 描述 |
| + | 加法 |
| - | 减法 |
| \* | 乘法 |
| \*\* | 幂（ES2016） |
| / | 除法 |
| % | 系数 |
| ++ | 递增 |
| -- | 递减 |

算数运算符

典型的算术运算会操作两个数值。

这两个数可以是字面量：

实例

<!DOCTYPE html>

<html>

<body>

<p>典型的算术运算会操作两个数（或表达式）并返回一个新的数。</p>

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

<script>

var x = 7 + 8;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

或变量：

实例

<!DOCTYPE html>

<html>

<body>

<p>典型的算术运算会操作两个数（或表达式）并返回一个新的数。</p>

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

<script>

var a = 7;

var b = 8;

var x = a + b;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

或表达式：

实例

<!DOCTYPE html>

<html>

<body>

<p>典型的算术运算会操作两个数（或表达式）并返回一个新的数。</p>

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

<script>

var a = 2;

var x = (7 + 8) \* a;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

运算符和操作数

在算术运算中，数被称为操作数。

（两个操作数之间执行的）运算由运算符定义

加法

加法运算符（+）加数：

<!DOCTYPE html>

<html>

<body>

<h1>+ 运算符</h1>

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

<script>

var x = 7;

var y = 8;

var z = x + y;

document.getElementById("demo").innerHTML = z;

</script>

</body>

</html>

减法

减法运算符（-）减数。

var x = 7;

var y = 8;

var z = x - y;

亲自试一试

乘法

乘法运算符（\*）乘数。

var x = 7;

var y = 8;

var z = x \* y;

亲自试一试

除法

除法运算符（/）除数。

var x = 7;

var y = 2;

var z = x / y;

亲自试一试

系数

系数运算符（%）返回除法的余数。

var x = 7;

var y = 2;

var z = x % y;

亲自试一试

注释：在算术中，两个整数的除法产生商和余数。

注释：在数学中，模运算的结果是算术除法的余数。

递增

递增运算符（++）对数值进行递增。

var x = 7;

x++;

var z = x;

亲自试一试

递减

递减运算符（--）对数值进行递减。

var x = 7;

x--;

var z = x;

亲自试一试

幂

取幂运算符（\*\*）将第一个操作数提升到第二个操作数的幂。

实例

var x = 5;

var z = x \*\* 2; // 结果是 25

亲自试一试

x \*\* y 产生的结果与 Math.pow(x,y) 相同:

实例

var x = 5;

var z = Math.pow(x,2); // 结果是 25

亲自试一试

运算符优先级

运算符优先级（Operator precedence）描述了在算术表达式中所执行操作的顺序。

实例

var x = 200 + 50 \* 2;

亲自试一试

上例的结果是 250 \* 2 还是 200 + 100 呢？

是加法还是乘法优先呢？

在传统的学校数学中，乘法是优先的。

乘法（\*）和除法（%）比加法（+）和减法（-）拥有更高的优先级。

同时，（就像在学校的数学中那样）能够通过使用括号来改变优先级：

实例

var x = (200 + 50) \* 2;

亲自试一试

当使用括号时，括号中的运算符会首先被计算。

当多个运算拥有相同的优先级时（比如加法和减法），对它们的计算是从左向右的：

实例

var x = 200 + 50 - 2;

JavaScript 运算符优先级值

|  |  |  |  |
| --- | --- | --- | --- |
| 值 | 运算符 | 描述 | 实例 |
| 20 | ( ) | 表达式分组 | (3 + 4) |
|  |  |  |  |
| 19 | . | 成员 | person.name |
| 19 | [] | 成员 | person["name"] |
| 19 | () | 函数调用 | myFunction() |
| 19 | new | 创建 | new Date() |
|  |  |  |  |
| 17 | ++ | 后缀递增 | i++ |
| 17 | -- | 后缀递减 | i-- |
|  |  |  |  |
| 16 | ++ | 前缀递增 | ++i |
| 16 | -- | 前缀递减 | --i |
| 16 | ! | 逻辑否 | !(x==y) |
| 16 | typeof | 类型 | typeof x |
|  |  |  |  |
| 15 | \*\* | 求幂 (ES7) | 10 \*\* 2 |
|  |  |  |  |
| 14 | \* | 乘 | 10 \* 5 |
| 14 | / | 除 | 10 / 5 |
| 14 | % | 模数除法 | 10 % 5 |
|  |  |  |  |
| 13 | + | 加 | 10 + 5 |
| 13 | - | 减 | 10 - 5 |
|  |  |  |  |
| 12 | << | 左位移 | x << 2 |
| 12 | >> | 右位移 | x >> 2 |
| 12 | >>> | 右位移（无符号） | x >>> 2 |
|  |  |  |  |
| 11 | < | 小于 | x < y |
| 11 | <= | 小于或等于 | x <= y |
| 11 | > | 大于 | x > y |
| 11 | >= | 大于或等于 | x >= y |
| 11 | in | 对象中的属性 | "PI" in Math |
| 11 | instanceof | 对象的实例 | instanceof Array |
|  |  |  |  |
| 10 | == | 相等 | x == y |
| 10 | === | 严格相等 | x === y |
| 10 | != | 不相等 | x != y |
| 10 | !== | 严格不相等 | x !== y |
|  |  |  |  |
| 9 | & | 按位与 | x & y |
| 8 | ^ | 按位 XOR | x ^ y |
| 7 | | | 按位或 | x | y |
| 6 | && | 逻辑与 | x && y |
| 5 | || | 逻辑否 | x || y |
| 4 | ? : | 条件 | ? "Yes" : "No" |
|  |  |  |  |
| 3 | = | 赋值 | x = y |
| 3 | += | 赋值 | x += y |
| 3 | -= | 赋值 | x -= y |
| 3 | \*= | 赋值 | x \*= y |
| 3 | %= | 赋值 | x %= y |
| 3 | <<= | 赋值 | x <<= y |
| 3 | >>= | 赋值 | x >>= y |
| 3 | >>>= | 赋值 | x >>>= y |
| 3 | &= | 赋值 | x &= y |
| 3 | ^= | 赋值 | x ^= y |
| 3 | |= | 赋值 | x |= y |
|  |  |  |  |
| 2 | yield | 暂停函数 | yield x |
| 1 | , | 逗号 | 7 , 8 |

注意：淡红色指示实验性或建议性的技术（ECMASScript 2016 或 ES7）

提示：括号中的表达式会在值在表达式的其余部分中被使用之前进行完全计算。

# 第13章 JavaScript 赋值

JavaScript 赋值运算符

赋值运算符向 JavaScript 变量赋值。

|  |  |  |
| --- | --- | --- |
| 运算符 | 例子 | 等同于 |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |
| <<= | x <<= y | x = x << y |
| >>= | x >>= y | x = x >> y |
| >>>= | x >>>= y | x = x >>> y |
| &= | x &= y | x = x & y |
| ^= | x ^= y | x = x ^ y |
| |= | x |= y | x = x | y |
| \*\*= | x \*\*= y | x = x \*\* y |
|  |  |  |
|  |  |  |
|  |  |  |

提示：\*\*= 运算符属于 ECMAScript 2016 proposal (ES7) 的实验部分。它的跨浏览器表现并不稳定。请勿使用。

第14章