第 2 章

读懂代码的前提——JavaScript 编程基础

无论是传统编程语言，还是脚本语言，都具有数据类型、常量和变量、运算符、表达式、注释语句、流程控制语句等基本构成元素，了解这些基本元素是学会编程的第一步。本章将主要讲述 JavaScript 编程的基本知识。

2.1 JavaScript 的基本语法

JavaScript 可以直接用记事本编写，其中包括语句、相关的语句块以及注释。在一条语句内可以使用变量、表达式等。本节就来介绍相关编程语法的基础知识。

2.1.1 执行顺序

JavaScript 程序按照在 HTML 文件中出现的顺序逐行执行。如果需要在整个 HTML 文件中执行，最好将其放在 HTML 文件的<head></head>标签对中。某些代码，如函数体内的代码，不会被立即执行，只有当所在的函数被其他程序调用时，该代码才会被执行。

2.1.2 区分大小写

JavaScript 对字母大小写敏感，也就是说，在输入语言的关键字、函数、变量以及其他标识符时，一定要严格区分字母的大小写。例如 username 和 userName 是两个不同的变量。

HTML 不区分大小写。由于 JavaScript 与 HTML 紧密相关，这一点很容易混淆，许多 JavaScript 对象和属性都与其代表的 HTML 标签或属性同名，在 HTML 中，这些名称可以以任意的大小写方式输入，而不会引起混乱，但在 JavaScript 中，这些名称通常都是小写的。例如，在 HTML 中的单击事件处理器属性通常被声明为 onClick 或 Onclick，而在 JavaScript 中只能使用 onclick。

2.1.3 分号与空格

在 JavaScript 语句中，分号是可有可无的，这一点与 Java 语言不同，JavaScript 并不要求每行必须以分号作为语句的结束标志。如果语句的结束处没有分号，JavaScript 会自动地将该代码的结尾作为语句的结尾。

例如，下面两行代码书写方式都是正确的：

```
Alert("hello,JavaScript")
Alert("hello,JavaScript");
```

鉴于需要养成良好的编程习惯，最好在每行的最后加上一个分号，这样能保证每行代码都是正确的、易读的。

另外，JavaScript 会忽略多余的空格，用户可以向脚本中添加空格，来提高程序的可读性。例如，下面的两行代码是等效的：

```
var name="Hello";
var name = "Hello";
```

2.1.4 对代码行进行折行

当一段代码比较长时，用户可以在文本字符串中使用反斜杠对代码行进行折行。

例如，下面的代码会正确地运行：

```
document.write("Hello \
World!");
```

不过，用户不能像这样折行：

```
document.write \
("Hello World!");
```

2.1.5 注释

注释通常用来解释程序代码的功能(增加代码的可读性)或阻止代码的执行(调试程序时)，不参与程序的执行。在 JavaScript 中，注释分为单行注释和多行注释两种。

1. 单行注释

在 JavaScript 中，单行注释以双斜杠“//”开始，直到这一行结束。单行注释“//”可以放在行的开始或一行的末尾，无论放在哪里，只要是从“//”符号开始到本行结束为止的所有内容，就都不会执行。在一般情况下，如果“//”位于一行的开始，则用来解释下一行或一段代码的功能；如果“//”位于一行的末尾，则用来解释当前行代码的功能。如果用来阻止一行代码的执行，也常将“//”放在这一行的开始。

【例 2.1】(示例文件 ch02\2.1.html)

使用单行注释语句：

```
<!DOCTYPE html>
<html>
<head>
<title>date 对象</title>
<script type="text/javascript">
function disptime()
{
  //创建日期对象 now，并实现当前日期的输出
  var now = new Date();
  //输出当前日期
  document.write("<H2>今天日期:" + now.getFullYear() + "年"
    + (now.getMonth()+1) + "月"
    + now.getDate() + "日</H2>");   //在页面上显示当前年月日
}
</script>
</head>
<body onload="disptime()">
</body>
</html>
```

以上代码中，共使用了 3 个注释语句。第一个注释语句将“//”符号放在了行首，通常用来解释下面代码的功能与作用。第二个注释语句放在了代码的行首，阻止了该行代码的执

行。第三个注释语句放在了行的末尾，主要是对该行相关的代码进行解释说明。

在 IE 11.0 中的浏览效果如图 2-1 所示。可以看到代码中的注释不被执行。

2. 多行注释

单行注释语句只能注释一行代码，假设在调试程序时，希望有一段代码(若干行)不被浏览器执行或者对代码的功能说明一行书写不完，那么就可以使用多行注释语句。多行注释语句以“/*”开始，以“*/”结束，可以注释一段代码。

【例 2.2】(示例文件 ch02\2.2.html)

使用多行注释语句：

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1 id="myH1"></h1>
<p id="myP"></p>

<script type="text/javascript">
/*
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="Welcome to my Homepage";
document.getElementById("myP").innerHTML="This is my first paragraph.";
</script>

<p><b>注释：</b>注释块不会被执行。</p>
</body>
</html>
```

在 IE 11.0 中的浏览效果如图 2-2 所示。可以看到代码中的注释不被执行。

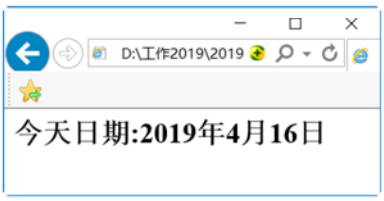

图 2-1　在 IE 11.0 中的浏览效果

图 2-2　使用多行注释语句

2.1.6　语句

JavaScript 程序是语句的集合，一条 JavaScript 语句相当于英语中的一个完整句子。JavaScript 语句将表达式组合起来，完成一定的任务。一条语句由一个或多个表达式、关键字或运算符组合而成，语句之间用分号(;)隔开，即分号是 JavaScript 语句的结束符号。

下面给出 JavaScript 语句的分隔示例，其中一行就是一条 JavaScript 语句。

```
Name = "张三";                    //将“张三”赋值给 Name
Var today = new Date();           //将今天的日期赋值给 today
```

【例 2.3】操作两个 HTML 元素(示例文件 ch02\2.3.html)：

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1>我的网站</h1>
<p id="demo">一个段落.</p>
<div id="myDIV">一个 div 块.</div>
<script type="text/javascript">
  document.getElementById("demo").innerHTML=
"Hello JavaScript";
  document.getElementById("myDIV").innerHTML=
"How are you?";
</script>
</body>
</html>
```

在 IE 11.0 中的浏览效果如图 2-3 所示。

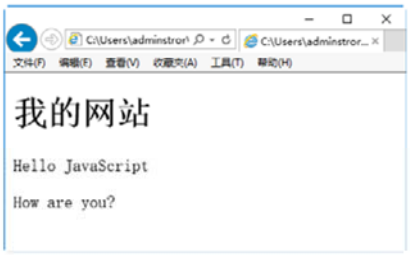

图 2-3　操作两个 HTML 元素

2.1.7　语句块

语句块是一些语句的组合，通常语句块都会用一对大括号包围起来。在调用语句块时，JavaScript 会按书写次序执行语句块中的语句。JavaScript 会把语句块中的语句看成是一个整体全部执行。

语句块通常用在函数中或流程控制语句中，如下所示的代码就包含一个语句块：

```
if (Fee < 2)
{
    Fee = 2;    //小于 2 元时，手续费为 2 元
}
```

语句块的作用是使语句序列一起执行。JavaScript 函数是将语句组合在块中的典型例子。

【例 2.4】运行可操作两个 HTML 元素的函数(示例文件 ch02\2.4.html)：

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1>我的网站</h1>
<p id="myPar">我是一个段落.</p>
<div id="myDiv">我是一个 div 块.</div>
<p>
<button type="button" onclick="myFunction()">点击这里</button>
</p>
<script type="text/javascript">
function myFunction()
{
```

```
  document.getElementById("myPar").innerHTML="Hello JavaScript";
  document.getElementById("myDiv").innerHTML="How are you?";
}
</script>
<p>当您点击上面的按钮时，两个元素会改变。</p>
</body>
</html>
```

在 IE 11.0 中浏览，效果如图 2-4 所示。单击其中的“点击这里”按钮，可以看到两个元素发生了变化，如图 2-5 所示。

图 2-4　初始效果

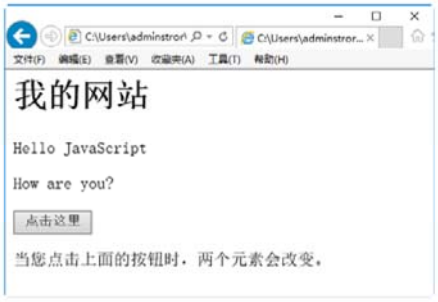

图 2-5　单击按钮后

2.2　JavaScript 的数据结构

每一种计算机编程语言都有自己的数据结构，JavaScript 脚本语言的数据结构包括标识符、常量、变量、关键字等。

2.2.1　标识符

用 JavaScript 编写程序时，很多地方都要求用户给定名称，例如，JavaScript 中的变量、函数等要素定义时都要求给定名称。可以将定义要素时使用的字符序列称为标识符。这些标识符必须遵循如下命名规则。

- 标识符只能由字母、数字或下划线和中文组成，而不能包含空格、标点符号、运算符等其他符号。
- 标识符的第一个字符必须是字母、下划线或者中文。
- 标识符不能与 JavaScript 中的关键字名称相同，即不能是 if、else 等。

例如，下面为合法的标识符：

```
UserName
Int2
_File_Open
Sex
```

又如，下面为不合法的标识符：

```
99BottlesofBeer
Namespace
```

```
It's-All-Over
```

2.2.2 关键字

关键字标识了 JavaScript 语句的开头或结尾。根据规定，关键字是保留的，不能用作变量名或函数名。JavaScript 中的关键字如表 2-1 所示。

表 2-1 JavaScript 中的关键字

break	case	catch	continue
default	delete	do	else
finally	for	function	if
in	instanceof	new	return
switch	this	throw	try
typeof	var	void	while
with			

JavaScript 关键字是不能作为变量名和函数名使用的。

2.2.3 保留字

保留字在某种意义上是为将来的关键字而保留的单词。因此保留字不能被用作变量名或函数名。

JavaScript 中的保留字如表 2-2 所示。

表 2-2 JavaScript 中的保留字

abstract	boolean	byte	char
class	const	debugger	double
enum	export	extends	final
float	goto	implements	import
int	interface	long	native
package	private	protected	public
short	static	super	synchronized
throws	transient	volatile	

如果将保留字用作变量名或函数名，那么除非将来的浏览器实现了该保留字，否则很可能收不到任何错误消息。当浏览器将其实现后，该单词将被看作关键字，如此将出现关键字错误。

2.2.4 常量

简单地说，常量是字面变量，是固化在程序代码中的信息，常量的值从定义开始就是固定的。常量主要用于为程序提供固定和精确的值，如数值、字符串、逻辑值真(true)、逻辑值假(false)等都是常量。

常量通常使用 const 来声明。语法格式如下。

```
const 常量名:数据类型 = 值;
```

2.2.5 变量

变量，顾名思义，在程序运行过程中，其值可以改变。变量是存储信息的单元，它对应于某个内存空间，变量用于存储特定数据类型的数据，用变量名代表其存储空间。程序能在变量中存储值和取出值，可以把变量比作超市的货架(内存)，货架上摆放着商品(变量)，可以把商品从货架上取出来(读取)，也可以把商品放入货架(赋值)。

1. 变量的命名

实际上，变量的名称是一个标识符。在 JavaScript 中，用标识符来命名变量和函数，变量的名称可以是任意长度。创建变量名称时，应遵循以下规则。

- 第一个字符必须是一个 ASCII 字符(大小写均可)或一个下划线(_)，但不能是文字或数字。
- 后续的字符必须是字母、数字或下划线。
- 变量名称不能是 JavaScript 的保留字。
- JavaScript 的变量名是严格区分大小写的。例如，变量名称 myCounter 与变量名称 MyCounter 是不同的。

下面给出一些合法的变量命名示例：

```
_pagecount
Part9
Number
```

下面给出一些错误的变量命名示例：

```
12balloon                //不能以数字开头
Summary&Went             //“与”符号不能用在变量名称中
```

2. 变量的声明与赋值

JavaScript 是一种弱类型的程序设计语言，变量可以不声明直接使用。所谓声明变量，就是为变量指定一个名称。声明变量后，就可以把它用作存储单元。

JavaScript 中使用关键字 var 来声明变量，在这个关键字之后的字符串将代表一个变量名。声明格式为：

```
var 标识符;
```

例如，声明变量 username，用来表示用户名，代码如下：

```
var username;
```

另外，一个关键字 var 也可以同时声明多个变量名，多个变量名之间必须用逗号“,”分隔。例如，同时声明变量 username、pwd、age，分别表示用户名、密码和年龄，代码如下：

```
var username,pwd,age;
```

要给变量赋值，可以使用 JavaScript 中的赋值运算符，即等于号(=)。
声明变量名时可以同时赋值，例如，声明变量 username 并赋值为“张三”，代码如下：

```
var username = "张三";
```

声明变量之后，对变量赋值，或者对未声明的变量直接赋值。例如，声明变量 age，然后再为它赋值，以及直接对变量 count 赋值：

```
var age;          //声明变量
age = 18;         //对已声明的变量赋值
count = 4;        //对未声明的变量直接赋值
```

JavaScript 中的变量如果未初始化(赋值)，默认值为 undefind。

3. 变量的作用范围

所谓变量的作用范围，是指可以访问该变量的代码区域。JavaScript 中，变量按其作用范围分为全局变量和局部变量。

- 全局变量：可以在整个 HTML 文档范围内使用的变量，这种变量通常都是在函数体外定义的变量。
- 局部变量：只能在局部范围内使用的变量，这种变量通常都是在函数体内定义的变量，所以只能在函数体中有效。

省略关键字 var 声明的变量，无论是在函数体内还是函数体外，都是全局变量。

【例 2.5】创建名为 carname 的变量，并向其赋值 Volvo，然后把它放入 id="demo"的 HTML 段落中(示例文件 ch02\2.5.html)。代码如下：

```
<!DOCTYPE html>
<html><head></head>
<body>
  <p>点击这里来创建变量，并显示结果。</p>
  <button onclick="myFunction()">点击这里</button>
  <p id="demo"></p>
<script type="text/javascript">
function myFunction()
{
  var carname="Volvo";
  document.getElementById("demo").innerHTML=carname;
}
```

```
</script>
</body>
</html>
```

在 IE 11.0 中浏览的效果如图 2-6 所示。单击其中的“点击这里”按钮，可以看到元素发生了变化，如图 2-7 所示。

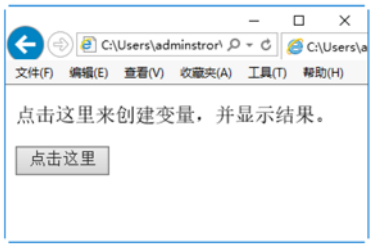

图 2-6　初始效果

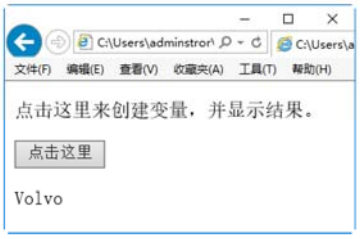

图 2-7　单击按钮后

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。

2.3　看透代码中的数据类型

每一种计算机语言除了有自己的数据结构外，还具有自己所支持的数据类型。在 JavaScript 脚本语言中，采用的是弱类型方式，即一个变量不必首先做声明，可以在使用或赋值时再确定其数据类型，当然也可以先声明该变量的类型。

2.3.1　typeof 运算符

typeof 运算符有一个参数，即要检查的变量或值。例如：

```
var sTemp = "test string";
alert(typeof sTemp);        //输出“string”
alert(typeof 86);           //输出“number”
```

对变量或值调用 typeof 运算符将返回下列值之一。

- undefined：如果变量是 Undefined 类型的，则返回 undefined。
- boolean：如果变量是 Boolean 类型的，则返回 boolean。
- number：如果变量是 Number 类型的，则返回 number。
- string：如果变量是 String 类型的，则返回 string。
- object：如果变量是一种引用类型或 Null 类型的，则返回 object。

【例 2.6】typeof 运算符的使用(示例文件 ch02\2.6.html)：

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
```

```
<script type="text/javascript">
  typeof(1);
  typeof(NaN);
  typeof(Number.MIN_VALUE);
  typeof(Infinity);
  typeof("123");
  typeof(true);
  typeof(window);
  typeof(document);
  typeof(null);
  typeof(eval);
  typeof(Date);
  typeof(sss);
  typeof(undefined);
  document.write("typeof(1): "+typeof(1)+"<br>");
  document.write("typeof(NaN): "+typeof(NaN)+"<br>");
  document.write("typeof(Number.MIN_VALUE): "
    + typeof(Number.MIN_VALUE)+"<br>")
  document.write("typeof(Infinity): "+typeof(Infinity)+"<br>")
  document.write("typeof(\"123\"): "+typeof("123")+"<br>")
  document.write("typeof(true): "+typeof(true)+"<br>")
  document.write("typeof(window): "+typeof(window)+"<br>")
  document.write("typeof(document): "+typeof(document)+"<br>")
  document.write("typeof(null): "+typeof(null)+"<br>")
  document.write("typeof(eval): "+typeof(eval)+"<br>")
  document.write("typeof(Date): "+typeof(Date)+"<br>")
  document.write("typeof(sss): "+typeof(sss)+"<br>")
  document.write("typeof(undefined): "+typeof(undefined)+"<br>")
</script>
</body>
</html>
```

在 IE 11.0 中浏览，效果如图 2-8 所示。

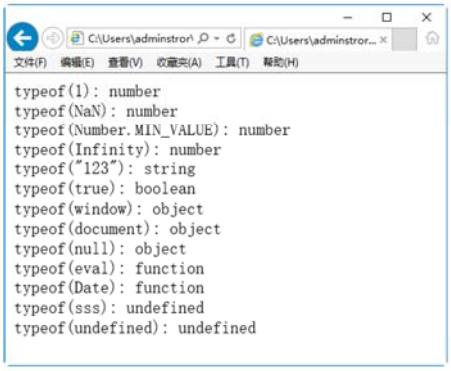

图 2-8 程序运行结果

2.3.2 未定义类型

undefined 是未定义类型的变量，表示变量还没有赋值，如 var a;，或者赋予一个不存在的属性值，例如 var a = String.notProperty;。

此外，JavaScript 中有一种特殊类型的常量 NaN，表示“非数字”，当在程序中由于某种原因发生计算错误后，将产生一个没有意义的值，此时 JavaScript 返回的就是 NaN。

【例 2.7】使用 undefined(示例文件 ch02\2.7.html)：

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<script type="text/javascript">
```

```
  var person;
  document.write(person + "<br />");
</script>
</body>
</html>
```

在 IE 11.0 中的浏览效果如图 2-9 所示。

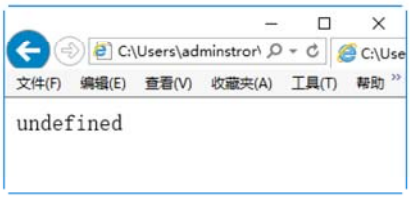

图 2-9　使用 undefined 的变量

2.3.3　空值类型

JavaScript 中的关键字 null 是一个特殊的值，表示空值，用于定义空的或不存在的引用。不过，null 不等同于空的字符串或 0。由此可见，null 与 undefined 的区别是：null 表示一个变量被赋予了一个空值，而 undefined 则表示该变量还未被赋值。

【例 2.8】使用 null(示例文件 ch02\2.8.html)：

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var person;
  document.write(person + "<br />");
  var car = null;
  document.write(car + "<br />");
</script>
</body>
</html>
```

在 IE 11.0 中的浏览效果如图 2-10 所示。

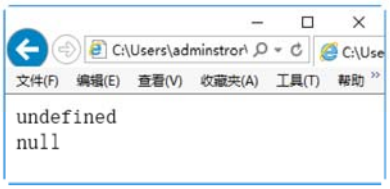

图 2-10　使用被赋予了一个空值的变量

2.3.4　布尔类型

布尔类型表示一个逻辑数值，用于表示两种可能的情况。逻辑真，用 true 表示；逻辑假，用 false 来表示。通常，我们使用 1 表示真，0 表示假。

【例 2.9】使用 Boolean 类型(示例文件 ch02\2.9.html)：

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var b1 = Boolean("");          //返回 false，空字符串
  var b2 = Boolean("s");         //返回 true，非空字符串
  var b3 = Boolean(0);           //返回 false，数字 0
  var b4 = Boolean(1);           //返回 true，非 0 数字
  var b5 = Boolean(-1);              //返回 true，非 0 数字
  var b6 = Boolean(null);            //返回 false
  var b7 = Boolean(undefined);       //返回 false
  var b8 = Boolean(new Object());    //返回 true，对象
```

```
  document.write(b1 + "<br>")
  document.write(b2 + "<br>")
  document.write(b3 + "<br>")
  document.write(b4 + "<br>")
  document.write(b5 + "<br>")
  document.write(b6 + "<br>")
  document.write(b7 + "<br>")
  document.write(b8 + "<br>")
</script>
</body>
</html>
```

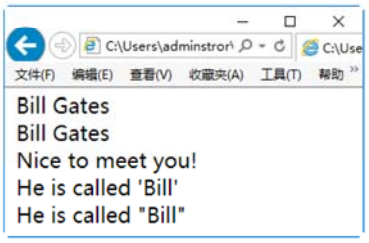

图 2-11 程序运行结果

在 IE 11.0 中的浏览效果如图 2-11 所示。

2.3.5 数值类型

JavaScript 的数值类型可以分为 4 类，即整数、浮点数、内部常量和特殊值。整数可以为正数、0 或者负数；浮点数可以包含小数点，也可以包含一个“e”(大小写均可，在科学记数法中表示“10 的幂”)，或者同时包含这两项。整数可以以 10(十进制)、8(八进制)和 16(十六进制)作为基数来表示。

【例 2.10】输出数值(示例文件 ch02\2.10.html)：

```
<!DOCTYPE html>
<html><head></head>
<body>
<script type="text/javascript">
  var x1 = 36.00;
  var x2 = 36;
  var y = 123e5;
  var z = 123e-5;
  document.write(x1 + "<br />")
  document.write(x2 + "<br />")
  document.write(y + "<br />")
  document.write(z + "<br />")
</script>
</body>
</html>
```

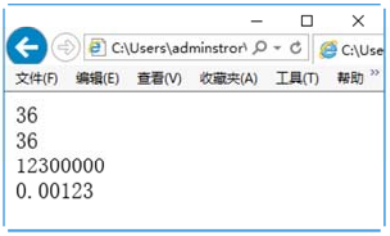

图 2-12 输出数值

在 IE 11.0 中的浏览效果如图 2-12 所示。

2.3.6 字符串类型

字符串是用一对单引号(')或双引号("")和引号中的内容构成的。字符串也是 JavaScript 中的一种对象，有专门的属性。引号中间的部分可以是任意多的字符，如果没有，则是一个空字符串。

如果要在字符串中使用双引号，则应该将其包含在使用单引号的字符串中，使用单引号时则反之。

【例 2.11】输出字符串(示例文件 ch02\2.11.html)：

```
<!DOCTYPE html>
<html><head></head>
<body>
<script type="text/javascript">
  var string1 = "Bill Gates";
  var string2 = 'Bill Gates';
  var string3 = "Nice to meet you!";
  var string4 = "He is called 'Bill'";
  var string5 = 'He is called "Bill"';
  document.write(string1 + "<br>")
  document.write(string2 + "<br>")
  document.write(string3 + "<br>")
  document.write(string4 + "<br>")
  document.write(string5 + "<br>")
</script>
</body>
</html>
```

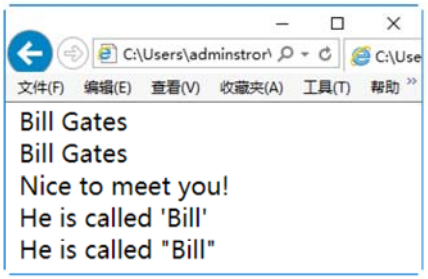

图 2-13　输出字符串

在 IE 11.0 中的浏览效果如图 2-13 所示。

2.3.7　对象类型

前面介绍的 5 种数据类型是 JavaScript 的原始数据类型，而 Object 是对象类型。该数据类型中包括 Object、Function、String、Number、Boolean、Array、Regexp、Date、Global、Math、Error，以及宿主环境提供的 Object 类型。

【例 2.12】Object 数据类型的使用(示例文件 ch02\2.12.html)：

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<script type="text/javascript">
  person = new Object();
  person.firstname = "Bill";
  person.lastname = "Gates";
  person.age = 56;
  person.eyecolor = "blue";
  document.write(person.firstname + " is "
    + person.age + " years old.");
</script>
</body>
</html>
```

图 2-14　使用 Object 数据类型

在 IE 11.0 中的浏览效果如图 2-14 所示。

2.4　数据间的计算法则——运算符

在 JavaScript 程序中，要完成各种各样的运算，是离不开运算符的。运算符用于将一个或几个值进行运算，而得出所需要的结果值。在 JavaScript 中，按运算符类型，可以分为算术运

算符、比较运算符、位运算符、赋值运算符、逻辑运算符和条件运算符等。

2.4.1 算术运算符

算术运算符是最简单、最常用的运算符，所以有时也称为简单运算符，可以使用算术运算符进行通用的数学计算。

JavaScript 语言中提供的算术运算符有+、-、*、/、%、++、--七种。分别表示加、减、乘、除、求余数、自增和自减。其中+、-、*、/、%五种为二元运算符，表示对运算符左右两边的操作数做算术运算，其运算规则与数学中的运算规则相同，即先乘除后加减。而++、--两种运算符都是一元运算符，其结合性为自右向左，在默认情况下表示对运算符右边的变量的值增 1 或减 1，而且它们的优先级比其他算术运算符的优先级高。

算术运算符的说明和示例如表 2-3 所示。

表 2-3 算术运算符

运 算 符	说 明	示 例
+	加法运算符，用于实现对两个数字进行求和	x+100、100+1000、+100
-	减法运算符或负值运算符	100-60、-100
*	乘法运算符	100*6
/	除法运算符	100/50
%	求模运算符，也就是算术中的求余	100%30
++	将变量值加 1 后再将结果赋值给该变量	x++：在参与其他运算之前先将自己加 1 后，再用新的值参与其他运算 ++x：先用原值运算后，再将自己加 1
--	将变量值减 1 后再将结果赋值给该变量	x--、--x，与++的用法相同

【例 2.13】通过 JavaScript 在页面中定义变量，再通过运算符计算变量的运行结果(示例文件 ch02\2.13.html)：

```
<!DOCTYPE html>
<html>
<head>
<title>运用 JavaScript 运算符</title>
</head>
<body>
<script type="text/javascript">
  var num1=120,num2 = 25;                              //定义两个变量
  document.write("120+25="+(num1+num2)+"<br>");        //计算两个变量的和
  document.write("120-25="+(num1-num2)+"<br>");        //计算两个变量的差
  document.write("120*25="+(num1*num2)+"<br>");        //计算两个变量的积
  document.write("120/25="+(num1/num2)+"<br>");        //计算两个变量的商
  document.write("(120++)="+(num1++)+"<br>");          //自增运算
  document.write("++120="+(++num1)+"<br>");
```

```
</script>
</body>
</html>
```

在 IE 11.0 中的浏览效果如图 2-15 所示。

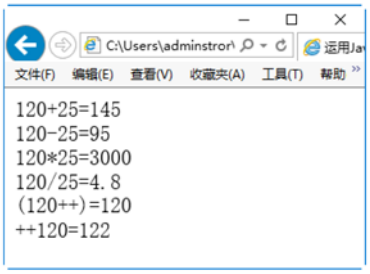

图 2-15 使用算术运算符

2.4.2 比较运算符

比较运算符用于对运算符的两个表达式进行比较，然后根据比较结果返回布尔类型的值 true 或 false。

在表 2-4 中列出了 JavaScript 支持的比较运算符。

表 2-4 比较运算符

运 算 符	说 明	示 例
==	判断左右两边表达式的值是否相等，当左边表达式的值等于右边表达式的值时返回 true，否则返回 false	Number == 100 Number1 == Number2
!=	判断左边表达式的值是否不等于右边表达式的值，当左边表达式的值不等于右边表达式的值时返回 true，否则返回 false	Number != 100 Number1 != Number2
>	判断左边表达式的值是否大于右边表达式的值，当左边表达式的值大于右边表达式的值时返回 true，否则返回 false	Number > 100 Number1 > Number2
>=	判断左边表达式的值是否大于等于右边表达式的值，当左边表达式的值大于等于右边表达式的值时返回 true，否则返回 false	Number >= 100 Number1 >= Number2
<	判断左边表达式的值是否小于右边表达式的值，当左边表达式的值小于右边表达式的值时返回 true，否则返回 false	Number < 100 Number1 < Number2
<=	判断左边表达式的值是否小于等于右边表达式的值，当左边表达式的值小于等于右边表达式的值时返回 true，否则返回 false	Number <= 100 Number <= Number2

【例 2.14】使用比较运算符比较两个数值的大小(示例文件 ch02\2.14.html)：

```
<!DOCTYPE html>
<html>
<head>
<title>比较运算符的使用</title>
</head>
<body>
<script type="text/javascript">
  var age = 25;                                          //定义变量
  document.write("age 变量的值为: "+age+"<br>");          //输出变量值
  document.write("age>=20: "+(age>=20)+"<br>");           //实现变量值比较
```

```
  document.write("age<20: "+(age<20)+"<br>");
  document.write("age!=20: "+(age!=20)+"<br>");
  document.write("age>20: "+(age>20)+"<br>");
</script>
</body>
</html>
```

在 IE 11.0 中的浏览效果如图 2-16 所示。

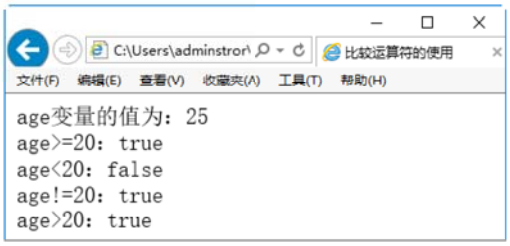

图 2-16 程序运行结果

2.4.3 位运算符

任何信息在计算机中都是以二进制的形式保存的。位运算符就是对数据按二进制位进行运算的运算符。JavaScript 语言中的位运算符有：&(与)、|(或)、^(异或)、~(取补)、<<(左移)、>>(右移)几种，如表 2-5 所示。其中，取补运算符为一元运算符，而其他的位运算符都是二元运算符。这些运算都不会产生溢出。位运算符的操作数为整型或者是可以转换为整型的任何其他类型。

表 2-5 位运算符

运 算 符	描 述
&	与运算。操作数中的两个位都为 1，结果为 1，两个位中有一个为 0，结果为 0
|	或运算。操作数中的两个位都为 0，结果为 0，否则，结果为 1
^	异或运算。两个操作位相同时，结果为 0，不相同时，结果为 1
~	取补运算。操作数的各个位取反，即 1 变为 0，0 变为 1
<<	左移位。操作数按位左移，高位被丢弃，低位顺序补 0
>>	右移位。操作数按位右移，低位被丢弃，其他各位顺序依次右移

【例 2.15】输出十进制数 18 对应的二进制数(示例文件 ch02\2.15.html)：

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1>输出十进制 18 的二进制数</h1>
<script type="text/javascript">
    var iNum = 18;
    alert(iNum.toString(2));
</script>
```

```
</body>
</html>
```

在 IE 11.0 中的浏览效果如图 2-17 所示。18 的二进制数只用了前 5 位，它们是这个数字的有效位。把数字转换成二进制字符串，就能看到有效位。这段代码只输出“10010”，而不是 18 的 32 位表示。这是因为其他的数位并不重要，仅使用前 5 位即可确定这个十进制数值。

图 2-17　输出十进制数 18 的二进制数

2.4.4　逻辑运算符

逻辑运算符通常用于执行布尔运算，它们常与比较运算符一起使用，来表示复杂比较运算，这些运算涉及的变量通常不止一个，而且常用于 if、while 和 for 语句中。

表 2-6 列出了 JavaScript 支持的逻辑运算符。

表 2-6　逻辑运算符

运 算 符	说　明	示　例
&&	逻辑与。若两边表达式的值都为 true，则返回 true；任意一个值为 false，则返回 false	100>60 &&100<200　返回 true 100>50&&10>100　返回 false
||	逻辑或。只有表达式的值都为 false 时，才返回 false	100>60||10>100　返回 true 100>600||50>60　返回 false
!	逻辑非。若表达式的值为 true，则返回 false，否则返回 true	!(100>60)　返回 false !(100>600)　返回 true

【例 2.16】逻辑运算符的使用(示例文件 ch02\2.16.html)：

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1>逻辑运算符的使用</h1>
<script type="text/javascript">
   var a=true,b=false;
   document.write(!a);
   document.write("<br />");
   document.write(!b);
   document.write("<br />");
   a=true,b=true;
```

```
    document.write(a&&b);
    document.write("<br />");
    document.write(a||b);
    document.write("<br />");
    a=true,b=false;
    document.write(a&&b);
    document.write("<br />");
    document.write(a||b);
    document.write("<br />");
    a=false,b=false;
    document.write(a&&b);
    document.write("<br />");
    document.write(a||b);
    document.write("<br />");
    a=false,b=true;
    document.write(a&&b);
    document.write("<br />");
    document.write(a||b);
</script>
</body>
</html>
```

图 2-18　逻辑运算符的使用

在 IE 11.0 中的浏览效果如图 2-18 所示。

从运行结果可以看出逻辑运算的规律。具体如下。

- true 的!为 false，false 的!为 true。
- a&&b：a、b 全为 true 时表达式为 true，否则表达式为 false。
- a||b：a、b 全为 false 时表达式为 false，否则表达式为 true。

2.4.5　条件运算符

除了上面介绍的常用运算符外，JavaScript 还支持条件表达式运算符“？：”，这个运算符是个三元运算符，它有三个部分：一个计算值的条件和两个根据条件返回的真假值。

格式如下所示：

```
条件？表示式 1 ： 表达式 2
```

在使用条件运算符时，如果条件为真，则表达式使用表达式 1 的值，否则使用表达式 2 的值。示例如下：

```
(x>y)？100*3 ： 11
```

这里，如果 x 的值大于 y 值，则表达式的值为 300；否则 x 的值小于或等于 y 值，即表达式的值为 11。

【例 2.17】使用条件运算符(示例文件 ch02\2.17.html)：

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1>条件运算符的使用</h1>
<script type="text/javascript">
```

```
    var a = 3;
    var b = 5;
    var c = b - a;
    document.write(c+"<br>");
    if(a>b)
    { document.write("a 大于 b<br>");}
    else
    { document.write("a 小于 b<br>");}
    document.write(a>b? "2" : "3");
</script>
</body>
</html>
```

上面的代码创建了两个变量 a 和 b，变量 c 的值是 b 和 a 的差。下面使用 if 语句判断 a 和 b 的大小，并输出结果。最后使用了一个三元运算符，如果 a>b，则输出 2，否则输出 3。
表示在网页中换行，“+”是一个连接字符串。

在 IE 11.0 中的浏览效果如图 2-19 所示，即网页中输出了 JavaScript 语句的执行结果。

图 2-19　条件运算符的使用

2.4.6　赋值运算符

赋值就是把一个数据赋值给一个变量。例如，myName="张三";的作用是执行一次赋值操作，把常量"张三"赋值给变量 myName。赋值运算符为二元运算符，要求运算符两侧的操作数类型必须一致。

JavaScript 中提供了简单赋值运算符和复合赋值运算符两种，如表 2-7 所示。

表 2-7　赋值运算符

运 算 符	说　明	示　例
=	将右边表达式的值赋值给左边的变量	Username="Bill"
+=	将运算符左边的变量加上右边表达式的值赋值给左边的变量	a+=b　//相当于 a=a+b
-=	将运算符左边的变量减去右边表达式的值赋值给左边的变量	a-=b　//相当于 a=a-b
*=	将运算符左边的变量乘以右边表达式的值赋值给左边的变量	a*=b　//相当于 a=a*b
/=	将运算符左边的变量除以右边表达式的值赋值给左边的变量	a/=b　//相当于 a=a/b
%=	将运算符左边的变量用右边表达式的值求模，并将结果赋给左边的变量	a%=b　//相当于 a=a%b
&=	将运算符左边的变量与右边表达式的变量进行逻辑与运算，将结果赋给左边的变量	a&=b　//相当于 a=a&b

续表

运 算 符	说　明	示　例
|=	将运算符左边的变量与右边表达式的变量进行逻辑或运算，将结果赋给左边的变量	a|=b　//相当于 a=a||b
^=	将运算符左边的变量与右边表达式的变量进行逻辑异或运算，将结果赋给左边的变量	a^=b　//相当于 a=a^b

在书写复合赋值运算符时，两个符号之间一定不能有空格，否则将会出错。

【例 2.18】赋值运算符的使用(示例文件 ch02\2.18.html)：

```
<!DOCTYPE html>
<html><head></head>
<body>
  <h3>赋值运算符的使用规则</h3>
  <p><strong>如果把数字与字符串相加，结果将成为字符串。</strong></p>
    <script type="text/javascript">
       x=5+5;
       document.write(x);
       document.write("<br />");
       x="5"+"5";
       document.write(x);
       document.write("<br />");
       x=5+"5";
       document.write(x);
       document.write("<br />");
       x="5"+5;
       document.write(x);
       document.write("<br />");
    </script>
</body>
</html>
```

在 IE 11.0 中的浏览效果如图 2-20 所示。

图 2-20　赋值运算符的使用

2.4.7　运算符的优先级

运算符的种类非常多，通常不同的运算符又构成了不同的表达式，甚至一个表达式中又包含有多种运算符，因此它们的运算方法有一定的规律性。JavaScript 语言规定了各类运算符的运算级别及结合性等，如表 2-8 所示。

表 2-8　运算符的优先级

优先级(1 最高)	说　明	运 算 符	结 合 性
1	括号	()	从左到右
2	自加/自减运算符	++　--	从右到左

续表

优先级(1 最高)	说　明	运 算 符	结 合 性
3	乘法运算符、除法运算符、取模运算符	* / %	从左到右
4	加法运算符、减法运算符	+ -	从左到右
5	小于、小于等于、大于、大于等于	< <= > >=	从左到右
6	等于、不等于	== !=	从左到右
7	逻辑与	&&	从左到右
8	逻辑或	||	从左到右
9	赋值运算符和快捷运算符	= += *= /= %= -=	从右到左

建议在写表达式的时候，如果无法确定运算符的有效顺序，则尽量采用括号来保证运算的顺序，这样也使得程序一目了然，而且自己在编程时能够思路清晰。

【例 2.19】演示使用运算符的优先级(示例文件 ch02\2.19.html)：

```
<!DOCTYPE html>
<html>
<head>
<title>运算符的优先级</title>
</head>
<body>
<script language="javascript">
   var a = 1+2*3;            //按自动优先级计算
   var b = (1+2)*3;          //使用()改变运算优先级
   alert("a="+a+"\nb="+b);   //分行输出结果
</script>
</body>
</html>
```

在 IE 11.0 中的浏览效果如图 2-21 所示。

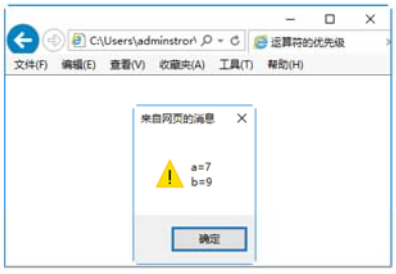

图 2-21　演示使用运算符的优先级

2.5　JavaScript 的表达式

表达式是一个语句的集合，像一个组一样，计算结果是单一值，然后该结果被 JavaScript 归入下列数据类型之一：布尔、数值、字符串、对象等。

一个表达式本身可以是一个数值或者变量，或者它可以包含许多连接在一起的变量关键字以及运算符。

例如，表达式 x/y，若分别使自由变量 x 和 y 的值为 10 和 5，其输出为数值 2；但在 y 值为 0 时则没有定义。一个表达式的赋值和运算符的定义以及数值的定义域是有关联的。

2.5.1　赋值表达式

在 JavaScript 中，赋值表达式的一般语法形式为“变量 赋值运算符 表达式”，在计算过

程中是按照自右而左结合的。其中有简单的赋值表达式，如 i=1；也有定义变量时，给变量赋初始值的赋值表达式，如 var str = "Happy World！"；还有使用比较复杂的赋值运算符连接的赋值表达式，如 k+=18。

【例 2.20】赋值表达式的用法(示例文件 ch02\2.20.html)：

```
<!DOCTYPE html>
<html>
<head>
<title>赋值表达式</title>
</head>
<body>
 <script language="javascript">
    var x = 15;
    document.write("<p>目前变量 x 的值为：x="+ x);
    x+=x-=x*x;
    document.write("<p>执行语句“x+=x-=x*x”后，变量 x 的值为：x=" + x);
    var y = 15;
    document.write("<p>目前变量 y 的值为：y="+ y);
    y+=(y-=y*y);
    document.write("<p>执行语句“y+=(y-=y*y)”后，变量 y 的值为：y=" + y);
</script>
</body>
</html>
```

在上述代码中，表达式 x+=x−=x*x 的运算流程如下：先计算 x=x−(x*x)，得到 x=−210，再计算 x=x+(x−=x*x)，得到 x=−195。同理，表达式 y+=(y−=y*y)的结果为 y=−195，如图 2-22 所示。

图 2-22 运行结果

由于运算符的优先级规定较多并且容易混淆，为提高程序的可读性，在使用多操作符的运算时，应该尽量使用括号“()”来保证程序的正常运行。

2.5.2 算术表达式

算术表达式就是用算术运算符连接的 JavaScript 语句元素。如 i+j+k、20−x、a*b、j/k、sum%2 等即为合法的算术运算符的表达式。

算术运算符的两边必须都是数值，若在“+”运算中存在字符或字符串，则该表达式将是字符串表达式，因为 JavaScript 会自动将数值型数据转换成字符串型数据。例如，下面的表达式将被看作是字符串表达式：

```
"好好学习" + i + "天天向上" + j
```

2.5.3 布尔表达式

布尔表达式一般用来判断某个条件或者表达式是否成立，其结果只能为 true 或 false。

【例 2.21】布尔表达式的用法(示例文件 ch02\2.21.html)：

```
<!DOCTYPE html>
<html><head>
<title>布尔表达式</title>
</head><body>
<script language="javascript" type="text/javaScript">
function checkYear()
{
   var txtYearObj = document.all.txtYear; //文本框对象
   var txtYear = txtYearObj.value;
   if((txtYear==null) || (txtYear.length<1) || (txtYear<0))  //文本框值为空
   {
       window.alert("请在文本框中输入正确的年份！");
       txtYearObj.focus();
       return;
   }
   if(isNaN(txtYear))  //用户输入不是数值
   {
       window.alert("年份必须为整型数字！");
       txtYearObj.focus();
       return;
   }
   if(isLeapYear(txtYear))
       window.alert(txtYear + "年是闰年！");
   else
       window.alert(txtYear + "年不是闰年！");
}
function isLeapYear(yearVal)  //判断是否为闰年
{
   if((yearVal%100==0) && (yearVal%400==0))
       return true;
   if(yearVal%4 == 0) return true;
   return false;
}
</script>
<form action="#" name="frmYear">
请输入当前年份：
   <input type="text" name="txtYear">
   <p>请单击按钮以判断是否为闰年：
   <input type="button" value="按钮" onclick="checkYear()">
</form>
</body>
</html>
```

在代码中多次使用布尔表达式进行数值的判断。运行该段代码，在显示的文本框中输入“2019”，单击“按钮”按钮后，系统先判断文本框是否为空，再判断文本框输入的数值是否

合法，最后判断其是否为闰年，并弹出相应的提示框，如图 2-23 所示。

同理，如果输入值为“2020”，具体的显示效果则如图 2-24 所示。

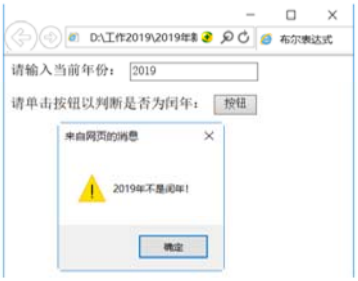

图 2-23　输入“2019”的运行结果

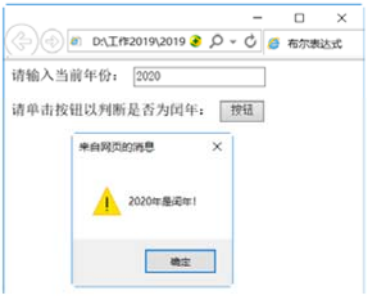

图 2-24　输入“2020”的运行结果

2.5.4　字符串表达式

字符串表达式是操作字符串的 JavaScript 语句。JavaScript 的字符串表达式只能使用“+”与“+=”两个字符串运算符。如果在同一个表达式中既有数值又有字符串，同时还没有将字符串转换成数值的方法，则返回值一定是字符串型。

【例 2.22】字符串表达式的用法(示例文件 ch02\2.22.html)：

```
<!DOCTYPE html>
<html><head>
<title>字符串表达式</title>
</head><body>
<script language="javascript">
  var x=10;
  document.write("<p>目前变量 x 的值为：x=" + x);
  x=1+4+8;
  document.write("<p>执行语句“x=1+4+8”后，变量 x 的值为：x=" + x);
  document.write("<p>此时，变量 x 的数据类型为：" + (typeof x));
  x=1+4+'8';
  document.write("<p>执行语句“x=1+4+'8'”后，变量 x 的值为：x=" + x);
  document.write("<p>此时，变量 x 的数据类型为：" + (typeof x));
</script>
</body>
</html>
```

运行上述代码，对于一般表达式“1+4+8”，将三者相加的和为 13；而在表达式“1+4+'8'”中，表达式按照从左至右的运算顺序，先计算数值 1、4 的和，结果为 5；再把和转换成字符串型，与最后的字符串连接；得到的结果是字符串“58”，如图 2-25 所示。

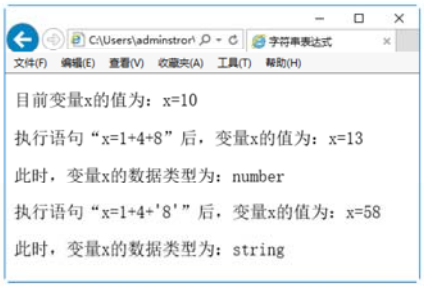

图 2-25　使用字符串表达式

2.5.5　类型转换

相对于强类型语言，JavaScript 的变量没有预定类

型，其类型对应于包含值的类型。当对不同类型的值进行运算时，JavaScript 解释器将自动把数据类型执意改变(强制转换)为另一种数据类型，再执行相应的运算。除自动类型转换外，为避免自动转换或不转换产生的不良后果，有时需要手动进行显式的类型转换，此时可利用 JavaScript 中提供的类型转换工具，如 parseInt()方法和 parseFloat()方法等。

【例 2.23】将字符串型转换为逻辑型数据(示例文件 ch02\2.23.html)：

```
<!DOCTYPE html>
<html><head>
<title>类型转换</title>
</head><body>
<script language="javascript">
var x = "happy";  //x 值为非空字符串
if (x)
{
    alert("字符串型变量 x 转换为逻辑型后，结果为
true");
}
else
{
    alert("字符串型变量 x 转换为逻辑型后，结果为false");
}
</script>
</body>
</html>
```

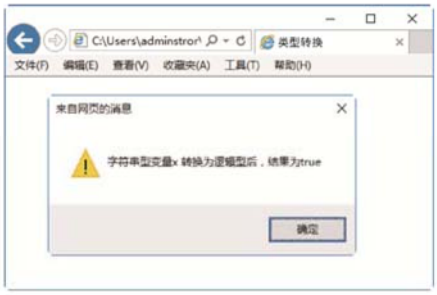

图 2-26　程序运行结果

代码的运行结果如图 2-26 所示。对于非空字符串变量 x，按照数据类型转换规则，自动转换为逻辑型后的结果为 true。

2.6　实战演练——局部变量和全局变量的优先级

在函数内部，局部变量的优先级高于同名的全局变量。也就是说，如果存在与全局变量名称相同的局部变量，或者在函数内部声明了与全局变量同名的参数，则该全局变量将不再起作用。

【例 2.24】变量的优先级(示例文件 ch02\2.24.html)：

```
<!DOCTYPE html>
<html>
<head><title>变量的优先级</title></head>
<body>
<script language="javascript">
  var scope = "全局变量"; //声明一个全局变量
  function checkscope()
  {
    var scope = "局部变量"; //声明一个同名的局部变量
    document.write(scope); //使用的是局部变量，而不是全局变量
```

```
  }
  checkscope(); //调用函数，输出结果
</script>
</body>
</html>
```

图 2-27 使用变量的优先级

在 IE 11.0 中的浏览效果如图 2-27 所示。将输出“局部变量”。

虽然在全局作用域中可以不使用 var 声明变量，但声明局部变量时，一定要使用 var 语句。

JavaScript 没有块级作用域，函数中的所有变量无论是在哪里声明的，在整个函数中都有意义。

【例 2.25】JavaScript 无块级作用域(示例文件 ch02\2.25.html)：

```
<!DOCTYPE html>
<html><head>
<title>变量的优先级</title>
</head><body>
<script languagc="javascript">
   var scope = "全局变量"; //声明一个全局变量
   function checkscope()
   {
      alert(scope); //调用局部变量，将显示“undefined”而不是“局部变量”
      var scope = "局部变量"; //声明一个同名的局部变量
      alert(scope); //使用的是局部变量，将显示“局部变量”
   }
    checkscope(); //调用函数，输出结果
</script>
</body>
</html>
```

程序运行结果如图 2-28 所示。单击“确定”按钮，弹出的结果如图 2-29 所示。

图 2-28 程序运行结果

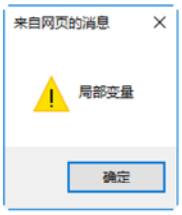

图 2-29 弹出的结果

本例中，用户可能认为因为声明局部变量的 var 语句还没有执行而调用全局变量 scope，但由于“无块级作用域”的限制，局部变量在整个函数体内是有定义的。这就意味着在整个函数体中都隐藏了同名的全局变量，因此，输出的并不是“全局变量”。虽然局部变量在整个函数体中都是有定义的，但在执行 var 语句之前不会被初始化。

2.7 疑 难 解 惑

疑问 1：声明变量时需要遵循哪几种规则？

可以使用一个关键字 var 同时声明多个变量，如语句“var x,y;”就同时声明了 x 和 y 两个变量。可以在声明变量的同时对其赋值(称为初始化)，例如“var president="henan"; var x=5, y=12;”声明了 3 个变量 president、x 和 y，并分别对其进行了初始化。如果出现重复声明的变量，且该变量已有一个初始值，则此时的声明相当于对变量的重新赋值。如果只是声明了变量，并未对其赋值，其值默认为 undefined。var 语句可以用作 for 循环和 for/in 循环的一部分，这样可使得循环变量的声明成为循环语法自身的一部分，使用起来较为方便。

疑问 2：比较运算符“ ==” 与赋值运算符“ =” 的不同之处是什么？

在各种运算符中，比较运算符“==”与赋值运算符“=”完全不同，运算符“=”是用于给操作数赋值的；而运算符“==”则是用于比较两个操作数的值是否相等的。如果在需要比较两个表达式的值是否相等的情况时错误地使用了赋值运算符“=”，则会将右操作数的值赋给左操作数。