第 2 章　JavaScript 语言基础

JavaScript 脚本语言作为一门功能强大、使用范围较广的程序语言，其语言基础包括数据类型、变量、运算符、函数以及核心语句等内容。本章主要介绍 JavaScript 脚本语言的基础知识，带领读者初步领会 JavaScript 脚本语言的精妙之处，并为后续章节的深入学习打下坚实的基础。

本章涉及到对象的相关知识，在本书后续章节将对其进行适当的分类和详细的论述，如读者理解有困难，可自行跳过，待学习了对象的基本概念和相关知识后再进行深入理解。

2.1 编程准备

在正式介绍 Javascript 脚本语言之前，先介绍使用 JavaScript 脚本进行编程需要首先了解的知识，包括编程术语、大小写敏感性、空白字符以及分号等内容，以及脚本编程过程中需遵守的一些约定，以编写合法的 JavaScript 脚本程序。

2.1.1 编程术语

首先我们来学习一下 Javascript 程序语言的基本术语，这些术语将贯穿 JavaScript 脚本编程的每个阶段，汇总如表 2.1 所示：

表 2.1　Javascript脚本编程基本术语

项目	简要说明	举例
Token(语言符号)	Javascrip脚本语言中最小的词汇单元，是一个字符序列	6，“I am a boy”，所有的标识符和关键字
Literal(常量)	拥有固定值的表达式	6，“I am a boy”，[1，2，3]
Identifer(标识符)	变量、函数、对象等的名称	num，alert，yourSex
Operator(运算符)	执行赋值、数学运算、比较等的符号	=，+，%，>
Expression(表达式)	标识符、运算符等组合起来的一个语句，返回该语句执行特定运算后的值	x+1，(num+1)/5

Statement(语句)	达到某个特定目的的强制性命令，脚本程序由多个语句构成	var num=5；
		function sum(x,y){
		result=x+y;
		return(result);
		}

Keyword(关键字)	作为脚本语言一部分的字符串，不能用作标识符使用	if，for，var，function
Reserved(保留字)	有可能作为脚本语言一部分的字符串，但并不严格限制其不能作为标识符	const，short，long

2.1.2 脚本执行顺序

JavaScript 脚本解释器将按照程序代码出现的顺序来解释程序语句，因此可以将函数定义和变量声明放在<head>和</head>之间，此时与函数体相关的操作不会被立即执行。

2.1.3 大小写敏感

JavaScript 脚本程序对大小写敏感，相同的字母，大小写不同，代表的意义也不同，如变量名 name、Name 和 NAME 代表三个不同的变量名。在 JavaScript 脚本程序中，变量名、函数名、运算符、关键字、对象属性等都是对大小写敏感的。同时，所有的关键字、内建函数以及对象属性等的大小写都是固定的，甚至混合大小写，因此在编写 JavaScript 脚本程序时，要确保输入正确，否则不能达到编写程序的目的。

2.1.4 空白字符

空白字符包括空格、制表符和换行符等，在编写脚本代码时占据一定的空间，但脚本被浏览器解释执行时无任何作用。脚本程序员经常使用空格作为空白字符，JavaScript 脚本解释器是忽略任何多余空格的。考察如下赋值语句：

```
s  =  s  +  5  ;
```

以及代码：

```
s=s+5;
```

上述代码的运行结果相同，浏览器解释执行第一个赋值语句时忽略了其中的空格。值得注意的是，浏览器解释执行脚本代码时，并非语句中所有的空格均被忽略掉。考察如下变量声明：

```
x=typeof y;
x=typeofy;
```

上面这两行代码代表的意义是不同的。第一行是将运算符 typeof 作用在变量 y 上，并将结果赋值给变量 x；而第二行是直接将变量 typeofy 的值赋给了 x，两行代码的意义完全不同。

在编写 JavaScript 脚本代码时经常使用一些多余的空格来增强脚本代码的可读性，并有助于专业的 JavaScript 脚本程序员（或者非专业人员）查看代码结构，同时有利于脚本代码的日后维护。

> 注意：在字符串中，空格不被忽略，而作为字符串的一部分显示出来，在编写 JavaScript 脚本代码时，经常需添加适当的空格使脚本代码层次明晰，方便相关人员查看和维护。

2.1.5 分号

在编写脚本语句时，用分号作为当前语句的结束符，例如：

```
var x=25;
var y=16;
var z=x+y;
```

当然，也可将多个语句写在同一行中，例如：

```
var x=25;var y=16;var z=x+y;
```

值得注意的是，为养成良好的编程习惯，尽量不要将多个语句写在一行中，避免降低脚本代码的可读性。

另外，语句分行后，作为语句结束符的分号可省略。例如可改写上述语句如下：

```
var x=25
var y=16
var z=x+y
```

代码运行结果相同，如将多个语句写在同一行中，则语句之间的分号不可省略。

2.1.6 块

在定义函数时，使用大括号“{}”将函数体封装起来，例如：

```
function muti(m,n)
{
   var result=m*n;
   return result;
}
```

在使用循环语句时，使用大括号“{}”将循环体封装起来，例如：

```
if(age<18)
{
    alert("对不起，您的年龄小于 18 岁，您无权浏览此网页");
}
```

从本质上讲，使用大括号“{}”将某段代码封装起来后，构成“块”的概念，JavaScript 脚本代码中的块，即为实现特定功能的多句（也可为空或一句）脚本代码构成的整体。

2.2 数值类型

2.2.1 整型和浮点数值

JavaScript 允许使用整数类型和浮点类型两种数值，其中整数类型包含正整数、0 和负整数；而浮点数则可以是包含小数点的实数，也可以是用科学计数法表示的实数，例如：

```
var age = 32;          //整数型
var num = 32.18;       //包含小数点的浮点型
var num = 3.7E-2;      //科学计数法表示的浮点型
```

2.2.2 八进制和十六进制

在整数类型的数值中，数制可使用十进制、八进制以及十六进制，例如：

```
var age = 32;          //十进制
var num = 010;         //八进制
var num = C33;         //十六进制
```

2.3 变量

几乎任何一种程序语言都会引入变量（variable），包括变量标识符、变量申明和变量作用域等内容。JavaScript 脚本语言中也将涉及到变量，其主要作用是存取数据以及提供存放信息的容器。在实际脚本开发过程中，变量为开发者与脚本程序交互的主要工具。下面分别介绍变量标识符、变量申明和变量作用域等内容。

2.3.1 变量标识符

与 C++、Java 等高级程序语言使用多个变量标识符不同，JavaScript 脚本语言使用关键字 var 作为其唯一的变量标识符，其用法为在关键字 var 后面加上变量名。例如：

```
var age;
var MyData;
```

2.3.2 变量申明

在 JavaScript 脚本语言中，声明变量的过程相当简单，例如通过下面的代码声明名为 age 的变量：

```
var age;
```

JavaScript 脚本语言允许开发者不首先声明变量就直接使用，而在变量赋值时自动申明该变量。一般来说，为培养良好的编程习惯，同时为了使程序结构更加清晰易懂，建议在使用变量前对变量进行申明。

变量赋值和变量声明可以同时进行，例如下面的代码声明名为 age 的变量，同时给该变量赋初值 25：

```
var age = 25;
```

当然，可在一句 JavaScript 脚本代码中同时声明两个以上的变量，例如：

```
var age , name;
```

同时初始化两个以上的变量也是允许的，例如：

```
var age = 35 , name = “tom”;
```

在编写 JavaScript 脚本代码时，养成良好的变量命名习惯相当重要。规范的变量命名，不仅有助于脚本代码的输入和阅读，也有助于脚本编程错误的排除。一般情况下，应尽量使用单词组合来描述变量的含义，并可在单词间添加下划线，或者第一个单词头字母小写而后续单词首字母大写。

> 注意：JavaScript 脚本语言中变量名的命名需遵循一定的规则，允许包含字母、数字、下划线和美元符号，而空格和标点符号都是不允许出现在变量名中，同时不允许出现中文变量名，且大小写敏感。

2.3.3 变量作用域

要讨论变量的作用域，首先要清楚全局变量和局部变量的联系和区别：

- 全局变量：可以在脚本中的任何位置被调用，全局变量的作用域是当前文档中整个脚本区域。
- 局部变量：只能在此变量声明语句所属的函数内部使用，局部变量的作用域仅为该函数体。

声明变量时，要根据编程的目的决定将变量声明为全局变量还是局部变量。一般而言，保存全局信息（如表格的原始大小、下拉框包含选项对应的字符串数组等）的变量需声明为全局变量，而保存临时信息（如待输出的格式字符串、数学运算中间变量等）的变量则声明为局部变量。

考察如下代码：

```
//源程序 2.1
<! DOCTYPE HTML PUBLIC    "-//W3C//DTD HTML 4.0//EN"
```

```
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var total=100;
function add(num)
{
   var total=2*num;
   alert("\n 局部变量 : \n\ntotal ="+total+"\n");
   return true;
}
-->
</script>
</head>
<body onload="add(total)">
<center>
<form>
   <input type=button value="局部和全局变量测试"
          onclick="javascript:alert('\n 全局变量 : \n\ntotal ='+total+'\n');">
</form>
</center>
</body>
</html>
```

浏览器载入上述代码后，弹出警告框显示局部变量 total 的值，其结果如图 2.1 所示。

图 2.1 局部变量

在上述警告框中单击“确定”后，关闭该警告框。单击“局部和全局变量测试”按钮，弹出警告框显示全局变量 total 的值，如图 2.2 所示。

图 2.2 全局变量

代码载入后，add(num)函数响应 body 元素对象的 onload 事件处理程序，输出其函数体中定义的局部变量 total 的值（整数 200）。单击“局部和全局变量测试”按钮，触发其 onclick 事件处理程序运行与其关联的 JavaScript 代码输出全局变量 total 的值（整数 100）：

```
javascript:alert('\n 全局变量 : \n\ntotal ='+total+'\n');
```

由上述结果可以看出，全局变量可作用在当前文档的 JavaScript 脚本区域，而局部变量仅存在于其所属的函数体内。实际应用中，应根据全局变量和局部变量的作用范围恰当定义变量，并尽量避免全局变量与局部变量同名，否则容易出现不易发现的变量调用错误。同时注意应对代码中引入的任何变量均进行声明。

2.4 弱类型

JavaScript 脚本语言像其他程序语言一样，其变量都有数据类型，具体数据类型将在下一节中介绍。高级程序语言如 C++、Java 等为强类型语言，与此不同的是，JavaScript 脚本语言是弱类型语言，在变量声明时不需显式地指定其数据类型，变量的数据类型将根据变量的具体内容推导出来，且根据变量内容的改变而自动更改，而强类型语在变量声明时必须显式地指定其数据类型。

变量声明时不需显式指定其数据类型既是 JavaScript 脚本语言的优点也是缺点，优点是编写脚本代码时不需要指明数据类型，使变量声明过程简单明了；缺点就是有可能造成因微妙的拼写不当而引起致命的错误。

考察如下代码：

```
//源程序 2.2
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//弱类型测试函数
function Test()
{
   var msg="\n 弱类型语言测试 : \n\n";
   msg+=""600"*5 = "+('600'*6)+"\n";
   msg+=""600"-5 = "+('600'-5)+"\n";
   msg+=""600"/5 = "+('600'/5)+"\n";
   msg+=""600"+5 = "+('600'+5)+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="弱类型测试" onclick="Test()">
</form>
</center>
```

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

程序运行后，在原始页面单击“弱类型测试”按钮，弹出警告框如图 2.3 所示。

图 2.3　弱类型语言测试

由上图中前三个表达式运算结果可知，JavaScript 脚本在解释执行时自动将字符型数据转换为数值型数据，而最后一个结果由于加号“+”的特殊性导致运算结果不同，是将数值型数据转换为字符型数据。运算符“+”有两个作用：

- 作为数学运算的加和运算符
- 作为字符型数据的连接符

由于加号“+”作为后者使用时优先级较高，故实例中表达式“'600'+5”的结果为字符串“6005”，而不是整数 605。

2.5 基本数据类型

在实现预定功能的程序代码中，一般需定义变量来存储数据（作为初始值、中间值、最终值或函数参数等）。变量包含多种类型，JavaScript 脚本语言支持的基本数据类型包括 Number 型、String 型、Boolean 型、Undefined 型、Null 型和 Function 型，分别对应于不同的存储空间，汇总如表 2.2 所示：

表 2.2　六种基本数据类型

类型	举例	简要说明
Number	45 , -34 , 32.13 , 3.7E-2	数值型数据
String	"name" , 'Tom'	字符型数据，需加双引号或单引号
Boolean	true , flase	布尔型数据，不加引号，表示逻辑真或假
Undefined		
Null	null	表示空值
Function		表示函数

2.5.1 Number 型

Number 型数据即为数值型数据，包括整数型和浮点型，整数型数制可以使用十进制、八进制以及十六进制标识，而浮点型为包含小数点的实数，且可用科学计数法来表示。一般来说，Number 型数据为不在括号内的数字，例如：

```
var myDataA=8;
var myDataB=6.3;
```

上述代码分别定义值为整数 8 的 Number 型变量 myDataA 和值为浮点数 6.3 的 Number 型变量 myDataB。

2.5.2 String 型

String 型数据表示字符型数据。JavaScript 不区分单个字符和字符串，任何字符或字符串都可以用双引号或单引号引起来。例如下列语句中定义的 String 型变量 nameA 和 nameB 包含相同的内容：

```
var nameA = "Tom";
var nameB = 'Tom';
```

如果字符串本身含有双引号，则应使用单引号将字符串括起来；若字符串本身含有单引号，则应使用双引号将字符串引起来。一般来说，在编写脚本过程中，双引号或单引号的选择在整个 JavaScript 脚本代码中应尽量保持一致，以养成好的编程习惯。

2.5.3 Boolean 型

Boolean 型数据表示的是布尔型数据，取值为 ture 或 false，分别表示逻辑真和假，且任何时刻都只能使用两种状态中的一种，不能同时出现。例如下列语句分别定义 Boolean 变量 bChooseA 和 bChooseB，并分别赋予初值 true 和 false：

```
var bChooseA = true;
var bChooseB = false;
```

值得注意的是，Boolean 型变量赋值时，不能在 true 或 false 外面加引号，例如：

```
var happyA = true;
var happyB = “true”;
```

上述语句分别定义初始值为 true 的 Boolean 型变量 happyA 和初始值为字符串“true”的 String 型变量 happyB。

2.5.4 Undefined 型

Undefined 型即为未定义类型，用于不存在或者没有被赋初始值的变量或对象的属性，如下列语句定义变量 name 为 Undefined 型：

```
var name;
```

定义 Undefined 型变量后，可在后续的脚本代码中对其进行赋值操作，从而自动获得由其值决定的数据类型。

2.5.5 Null 型

Null 型数据表示空值，作用是表明数据空缺的值，一般在设定已存在的变量（或对象的属性）为空时较为常用。区分 Undefined 型和 Null 型数据比较麻烦，一般将 Undefined 型和 Null 型等同对待。

2.5.6 Function 型

Function 型表示函数，可以通过 new 操作符和构造函数 Function()来动态创建所需功能的函数，并为其添加函数体。例如：

```
var myFuntion = new Function()
```

```
{
   staments;
};
```

JavaScript 脚本语言除了支持上述六种基本数据类型外，也支持组合类型，如数组 Array 和对象 Object 等，下面介绍组合类型。

2.6 组合类型

JavaScript 脚本支持的组合类型比基本数据类型更为复杂，包括数组 Array 型和对象 Object 型。本节将简要介绍上述组合类型的基本概念及其用法，在本书后续章节将进行专门论述。

2.6.1 Array 型

Array 型即为数组，数组是包含基本和组合数据的序列。在 JavaScript 脚本语言中，每一种数据类型对应一种对象，数组本质上即为 Array 对象。考察如下定义：

```
var score = [56,34,23,76,45];
```

上述语句创建数组 score，中括号“[]”内的成员为数组元素。由于 JavaScript 是弱类型语言，因此不要求目标数组中各元素的数据类型均相同，例如：

```
var score = [56,34,”23”,76,”45”];
```

由于数组本质上为 Array 对象，则可用运算符 new 来创建新的数组，例如：

```
var score=new Array(56,34,”23”,76,”45”);
```

访问数组中特定元素可通过该元素的索引位置 index 来实现，如下列语句声明变量 m 返回数组 score 中第四个元素：

```
var m = score [3];
```

数组作为 Array 对象，具有最重要的属性 length，用来保存该数组的长度，考察如下的测试代码：

源程序 2.3

```
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 数组的 length 属性 : \n\n";
var myArray=new Array("Tom","Jerry","Lily","Hanks");
//响应按钮的 onclick 事件处理程序
function Test()
{
   GetInfo(myArray);
   msg+="\n 操作语句 :\nmyArray.length=3\n\n";
   myArray.length=3;
   GetInfo(myArray);
   alert(msg);
}
```

```
//输出数组内容
function GetInfo(tempArray)
{
   var myLength=tempArray.length;
   msg+="数组长度 : \n"+myLength+"\n";
   msg+="数组内容 : \n";
   for(var i=0;i<myLength;i++)
   {
      msg+="myArray[ "+i+" ] ="+tempArray[i]+"\n";
   }
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="数组测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“数组测试”按钮，弹出警告框如图 2.4 所示。

图 2.4　数组的 length 属性

值得注意的是，数组的 length 属性为可读可写属性，作为可写属性时，若新的属性值小于原始值时，将调整数组的长度为新的属性值，数组中其余元素将删除。

2.6.2 Object 型

对象为可包含基本和组合数据的组合类型，且对象的成员作为对象的属性，对象的成员函数作为对象的方法。在 JavaScript 脚本语言中，可通过在对象后面加句点“.”并加上对象属性（或方法）的名称来访问对象的属性（或方法），例如：

```
document.bgColor
document.write(“Welcome to JavaScript World!”);
```

考察如下的测试代码：

```
//源程序 2.4
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function Test()        //响应按钮的 onclick 事件处理程序
{
   var msg="\n 对象属性和方法的引用 : \n\n";
   msg+="引用语句 :\nvar myColor=document.bgColor\n";
   msg+="返回结果 :\n"+document.bgColor+"\n";
   msg+="引用语句 :\nwindow.close()\n";
   msg+="返回结果 :\nClose The Window!\n";
   alert(msg);
   window.close();
}
-->
</script>
</head>
<body bgColor="green">
<center>
<form>
   <input type=button value="数组测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“数组测试”按钮，弹出警告框如图 2.5 所示。

图 2.5　访问对象的属性和方法

单击“确定”按钮，将弹出如图 2.6 所示的警告框提示用户浏览器正试图关闭当前文档页面。

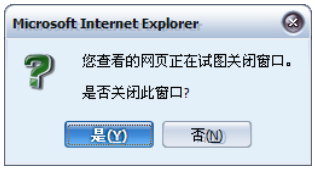

图 2.6 响应 window.close()方法

2.7 运算符

编写 JavaScript 脚本代码过程中，对目标数据进行运算操作需用到运算符。JavaScript 脚本语言支持的运算符包括：赋值运算符、基本数学运算符、位运算符、位移运算符、高级赋值语句、自加和自减、比较运算符、逻辑运算符、逗号运算符、空运算符、?...:…运算符、对象运算符以及 typedof 运算符等，下面分别予以介绍。

2.7.1 赋值运算符

JavaScript 脚本语言的赋值运算符包含“=”、“+=”、“—=”、“*=”、“/=”、“%=”、“&=”、“^=”等，汇总如表 2.3 所示：

表 2.3　赋值运算符

运算符	举例	简要说明
=	m=n	将运算符右边变量的值赋给左边变量
+=	m+=n	将运算符两侧变量的值相加并将结果赋给左边变量
-=	m-=n	将运算符两侧变量的值相减并将结果赋给左边变量
*=	m*=n	将运算符两侧变量的值相乘并将结果赋给左边变量
/=	m/=n	将运算符两侧变量的值相除并将整除的结果赋给左边变量
%=	m%=n	将运算符两侧变量的值相除并将余数赋给左边变量
&=	m&=n	将运算符两侧变量的值进行按位与操作并将结果赋值给左边变量
^=	m^=n	将运算符两侧变量的值进行按位或操作并将结果赋值给左边变量
<<=	m<<=n	将运算符左边变量的值左移由右边变量的值指定的位数，并将操作的结果赋予左边变量
>>=	m>>=n	将运算符左边变量的值右移由右边变量的值指定的位数，并将操作的结果赋予左边变量
>>>=	m>>>=n	将运算符左边变量的值逻辑右移由右边变量的值指定的位数，并将操作的结果赋给左边变量

赋值运算符是编写 JavaScript 脚本代码时最为常用的操作，读者应熟练掌握各个运算符的功能，避免混淆其具体作用。

考察如下的测试程序：

```
//源程序 2.5
<! DOCTYPE HTML PUBLIC    "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var lValue=48;                  //设定初始值
var rValue=3;
function Test()                 //响应按钮的 onclick 事件处理程序
{
   var msg="\n 赋值运算符操作 :   \n\n";
   msg+="原始数值 :\nlValue="+lValue+"rValue="+rValue+"\n\n";
   msg+="操作语句及返回结果 :\n\n";
   lValue=rValue;
   msg+="语句 : lValue=rValue          结果 : lValue="+lValue+"rValue="+rValue+"\n";
```

```
    lValue+=rValue;
    msg+="语句 : lValue+=rValue        结果 : lValue="+lValue+"rValue="+rValue+"\n";
    lValue-=rValue;
    msg+="语句 : lValue-=rValue        结果 : lValue="+lValue+"rValue="+rValue+"\n";
    lValue*=rValue;
    msg+="语句 : lValue*=rValue        结果 : lValue="+lValue+"rValue="+rValue+"\n";
    lValue/=rValue;
    msg+="语句 : lValue/=rValue        结果 : lValue="+lValue+"rValue="+rValue+"\n";
    lValue%=rValue;
    msg+="语句 : lValue%=rValue        结果 : lValue="+lValue+"rValue="+rValue+"\n";
    lValue=13;
    lValue&=rValue;
    msg+="语句 : lValue&=rValue        结果 : lValue="+lValue+" rValue="+rValue+"\n";
    lValue^=rValue;
    msg+="语句 : lValue^=rValue        结果 : lValue="+lValue+"rValue="+rValue+"\n";
    lValue<<=rValue;
    msg+="语句 : lValue<<=rValue       结果 : lValue="+lValue+"rValue="+rValue+"\n";
    lValue>>=rValue;
    msg+="语句 : lValue>>=rValue       结果 : lValue="+lValue+" rValue="+rValue+"\n";
    lValue>>>=rValue;
    msg+="语句 : lValue>>>=rValue      结果 : lValue="+lValue+"rValue="+rValue+"\n";
    alert(msg);
}
-->
</script>
</head>
<body>
<hr>
<form>
    <input type=button value="运算符测试" onclick="Test()">
</form>
</body>
</html>
```

程序运行后，在原始页面单击“运算符测试”按钮，弹出警告框如图 2.7 所示。

图 2.6　赋值运算符

由上述结果可知，JavaScript 脚本语言的运算符在参与数值运算时，其右侧的变量将保持不变。从本质上讲，运算符右侧的变量作为运算的参数而存在，脚本解释器执行指定的操作后，将运算结果作为返回值赋予运算符左侧的变量。

2.7.2 基本数学运算符

JavaScript 脚本语言中基本的数学运算包括加、减、乘、除以及取余等，其对应的数学运算符分别为“+”、“-”、“*”、“/”和“%”等，如表 2.4 所示：

表 2.4　　基本数学运算符

基本数学运算符	举例	简要说明
+	m=5+5	将两个数据相加，并将结果返回操作符左侧的变量
-	m=9-4	将两个数据相减，并将结果返回操作符左侧的变量
*	m=3*4	将两个数据相乘，并将结果返回操作符左侧的变量
/	m=20/5	将两个数据相除，并将结果返回操作符左侧的变量
%	m=14%3	求两个数据相除的余数，并将结果返回操作符左侧的变量

考察如下测试代码：

```
//源程序 2.6
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var lValue=25;                  //设定初始值
var rValue=4;
function Test()                 //响应按钮的 onclick 事件处理程序
{
   var tempData=0;
   var msg="\n 基本数学运算符 : \n\n";
   msg+="原始数值 :\n\ntempData="+tempData+"lValue="+lValue+" rValue="+rValue+"\n\n";
   msg+="操作语句及返回结果 :\n\n";
   tempData=lValue+rValue;
   msg+="语句 : tempData=lValue+rValue      结果 : tempData="+tempData+"\n";
   tempData=lValue-rValue;
   msg+="语句 : tempData=lValue-rValue       结果 : tempData="+tempData+"\n";
   tempData=lValue*rValue;
   msg+="语句 : tempData=lValue*rValue       结果 : tempData="+tempData+"\n";
   tempData=lValue/rValue;
   msg+="语句 : tempData=lValue/rValue       结果 : tempData="+tempData+"\n";
   tempData=lValue%rValue;
   msg+="语句 : tempData=lValue%rValue      结果 : tempData="+tempData+"\n";
   alert(msg);
}
-->
</script>
</head>
<body bgColor="green">
<center>
<form>
   <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
```

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

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出警告框如图 2.8 所示。

图 2.8　基本数学运算符

2.7.3 位运算符

JavaScript 脚本语言支持的基本位运算符包括：“&”、“|”、“^”和“~”等。脚本代码执行位运算时先将操作数转换为二进制数，操作完成后将返回值转换为十进制。位运算符的作用如表 2.5 所示:

表 2.5 位运算符

位运算符	举例	简要说明
&	9&4	按位与，若两数据对应位都是1，则该位为1，否则为0
^	9^4	按位异或，若两数据对应位相反，则该位为1，否则为0
|	9|4	按位或，若两数据对应位都是0，则该位为0，否则为1
~	~4	按位非，若数据对应位为0，则该位为1，否则为0

位运算符在进行数据处理、逻辑判断等方面使用较为广泛，恰当应用位运算符，可节省大量脚本代码。

考察如下测试代码：

```
//源程序 2.7
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var lValue=6;                    //二进制值 0000 0110b
var rValue=36;                   //二进制值 0010 0100b
function Test()                  //响应按钮的 onclick 事件处理程序
{
   var tempData=0;
   var msg="\n 基本位运算符 ：  \n\n";
   msg+="原始数值 :\n\ntempData="+tempData+" lValue="+lValue+" rValue="+rValue+"\n\n";
   msg+="操作语句及返回结果 :\n\n";
   tempData=lValue&rValue;
   msg+="语句 : tempData=lValue&rValue     结果 : tempData="+tempData+"\n";
```

```
    tempData=lValue^rValue;
    msg+="语句 : tempData=lValue^rValue      结果 : tempData="+tempData+"\n";
    tempData=lValue|rValue;
    msg+="语句 : tempData=lValue|rValue      结果 : tempData="+tempData+" \n";
    tempData=~lValue;
    msg+="语句 : tempData=~lValue            结果 : tempData="+tempData+"\n";
    alert(msg);
}
-->
</script>
</head>
<body bgColor="green">
<center>
<form>
    <input type=button value="位运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出警告框如图 2.9 所示。

图 2.9　位运算符

原始操作数分别为二进制 0000 0110b 和 0010 0100b，执行位与、位异或、位或和位非操作后，其结果分别为二进制 0000 0100b、0010 0010b、0010 0110b 和 1000 0111b，对应的十进制结果分别为 4、34、38 和-7。

2.7.4 位移运算符

位移运算符用于将目标数据往指定方向移动指定的位数。JavaScript 脚本语言支持“<<”、“>>”和“>>>”等位移运算符，其具体作用如见表 2.6：

表 2.6　　位移运算符

运算符	举例	简要说明
>>	9>>2	算术右移，将左侧数据的二进制值向左移动由右侧数值表示的位数，右边空位补0
<<	9<<2	算术左移，将左侧数据的二进制值向右移动由右侧数值表示的位数，忽略被移出的位
>>>	9>>>2	逻辑右移，将左边数据表示的二进制值向右移动由右边数值表示的位数，忽略被移出的位，左侧空位补0

位移运算符在逻辑控制、数值处理等方面应用较为广泛，恰当应用位移运算符，可节省大量脚本代码。

考察如下测试代码：

```
//源程序 2.8
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var targetValue=189;          //目标数据二进制值 1011 1101b
var iPos=2;                    //目标数据移动的位数
function Test()                //响应按钮的 onclick 事件处理程序
{
   var tempData=0;
   var msg="\n 位移运算符 :   \n\n";
   msg+="原始数值:\n\ntempData="+tempData+"targetValue="+targetValue+"iPos="+iPos+"\n\n";
   msg+="操作语句及返回结果 :\n\n";
   tempData=targetValue>>iPos;
   msg+="语句 : tempData=targetValue>>iPos       结果 : tempData="+tempData+"\n";
   tempData=targetValue<<iPos;
   msg+="语句 : tempData=targetValue<<iPos       结果 : tempData="+tempData+"\n";
   tempData=targetValue>>>iPos;
   msg+="语句 : tempData=targetValue>>>iPos      结果 : tempData="+tempData+"\n";
   alert(msg);
}
-->
</script>
</head>
<body bgColor="green">
<center>
<form>
   <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出警告框如图 2.10 所示。

图 2.10　位移运算符

目标数据的二进制值 1011 1101b，执行算术右移两位、算术左移两位和逻辑右移两位后，

其结果分别为二进制 0010 1111b、10 1110 1100b 和 0010 1111b，分别对应于十进制值 47、756 和 47。

2.7.5 自加和自减

自加运算符为“++”和自减运算符为“--”分别将操作数加 1 或减 1。值得注意的是，自加和自减运算符放置在操作数的前面和后面含义不同。运算符写在变量名前面，则返回值为自加或自减前的值；而写在后面，则返回值为自加或自减后的值。

考察如下测试代码：

```
//源程序 2.9
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var targetValue=9;                  //原始数据
function Test()                      //响应按钮的 onclick 事件处理程序
{
   var tempData=0;
   var msg="\自加和自减运算符 :   \n\n";
   msg+="原始数值 :\n\ntempData="+tempData+"targetValue="+targetValue+"\n\n";
   msg+="操作语句及返回结果 :\n\n";
   tempData=targetValue++;
   msg+="语句 : tempData=targetValue++       结果 : tempData="
          +tempData+"    targetValue="+targetValue+"\n";
   tempData=++targetValue;
   msg+="语句 : tempData=++targetValue       结果 : tempData="
          +tempData+"    targetValue="+targetValue+"\n";
   tempData=targetValue--;
   msg+="语句 : tempData=targetValue--       结果 : tempData="
          +tempData+"    targetValue="+targetValue+"\n";
   tempData=--targetValue;
   msg+="语句 : tempData=--targetValue       结果 : tempData="
          +tempData+"    targetValue="+targetValue+"\n";
   alert(msg);
}
-->
</script>
</head>
<body bgColor="green">
<center>
<form>
   <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出警告框如图 2.11 所示。

图 2.11　自加和自减运算符

由程序运行结果可以看出：

- 若自加（或自减）运算符放置在操作数之后，执行该自加（或自减）操作时，先将操作数的值赋值给运算符前面的变量，然后操作数自加（或自减）；
- 若自加（或自减）运算符放置在操作数之前，执行该自加（或自减）操作时，操作数先进行自加（或自减），然后将操作数的值赋值给运算符前面的变量。

2.7.6 比较运算符

JavaScript 脚本语言中用于比较两个数据的运算符称为比较运算符，包括“= =”、“!=”、“>”、“<”、“<=”、“>=”等，其具体作用见表 2.7。

表 2.7　　比较运算符

运算符	举例	作用
==	num==8	相等，若两数据相等，则返回布尔值true，否则返回false
!=	num!=8	不相等，若两数据不相等，则返回布尔值true，否则返回false
>	num>8	大于，若左边数据大于右边数据，则返回布尔值true，否则返回false
<	num<8	小于，若左边数据小于右边数据，则返回布尔值true，否则返回false
>=	num>=8	大于或等于，若左边数据大于或等于右边数据，则返回布尔值true，否则返回false
<=	num<=8	小于或等于，若左边数据小于或等于右边数据，则返回布尔值true，否则返回false

比较运算符主要用于数值判断及流程控制等方面，考察如下的测试代码。

```
//源程序 2.10
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//响应按钮的 onclick 事件处理程序
function Test()
{
   var myAge=prompt("请输入您的年龄(数值) : ",25);
   var msg="\n 年龄测试 : \n\n";
   msg+="年龄 : "+myAge+" 岁\n";
   if(myAge<18)
      msg+="结果 : 您处于青少年时期! \n";
   if(myAge>=18&&myAge<30)
```

```
    msg+="结果 ： 您处于青年时期! \n";
  if(myAge>=30&&myAge<55)
    msg+="结果 ： 您处于中年时期! \n";
  if(myAge>=55)
    msg+="结果 ： 您处于老年时期! \n";
  alert(msg);
}
-->
</script>
</head>
<body bgColor="green">
<center>
<form>
  <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出提示框提示用户输入相关信息，如图 2.12 所示。

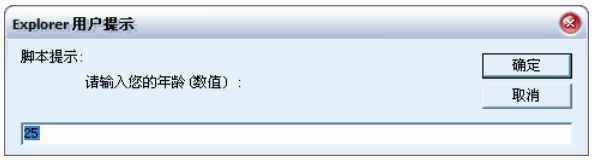

图 2.12　提示用户输入相关信息

在上述提示框输入相关信息（如年龄 35）后，单击“确定”按钮，弹出警告框如图 2.13 所示。

图 2.13 根据用户输入进行特定操作

可以看出，脚本代码采集用户输入的数值，然后通过比较运算符进行判定，再做出相对应的操作，实现了程序流程的有效控制。

> 注意：比较运算符 “= =”与赋值运算符“=”截然不同，前者用于比较运算符前后的两个数据，主要用语数值比较和流程控制；后者用于将运算符后面的变量的值赋予运算符前面的变量，主要用于变量赋值。

2.7.7 逻辑运算符

JavaScript 脚本语言的逻辑运算符包括 “&&”、“||”和“!”等，用于两个逻辑型数据

之间的操作，返回值的数据类型为布尔型。逻辑运算符的功能如表 2.8 所示：

表 2.8　　逻辑运算符

运算符	举例	作用
&&	num<5&&num>2	逻辑与，如果符号两边的操作数为真，则返回true，否则返回false
||	num<5||num>2	逻辑或，如果符号两边的操作数为假，则返回false，否则返回true
!	!num<5	逻辑非，如果符号右边的操作数为真，则返回false，否则返回true

逻辑运算符一般与比较运算符捆绑使用，用以引入多个控制的条件，以控制 JavaScript 脚本代码的流向。

2.7.8 逗号运算符

编写 JavaScript 脚本代码时，可使用逗号“,”将多个语句连在一起，浏览器载入该代码时，将其作为一个完整的语句来调用，但语句的返回值是最右边的语句。

考察如下的测试代码：

```
//源程序 2.11
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//响应按钮的 onclick 事件处理程序
function Test()
{
   var dataA,dataB,dataC,dataD;
   dataA=(dataB=1,dataC=2,dataD=3);
   var msg="\n 逗号运算符测试 : \n\n";
   msg+="赋值语句 : \ndataA=(dataB=1,dataC=2,dataD=3);\n";
   msg+="赋值结果 : \n";
   msg+="dataA = "+dataA+"\n";
   msg+="dataB = "+dataB+"\n";
   msg+="dataC = "+dataC+"\n";
   msg+="dataD = "+dataD+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出警告框如图 2.14 所示。

图 2.14　逗号“,”运算符

由运行结果可知，使用长语句赋值时，返回值为赋值语句最右边变量的值，为养成良好的编程习惯，建议不是用该方法。逗号“,”一般用于在函数定义和调用时分隔多个参数，例如：

```
function sum(a,b,c){
   statements
}
```

2.7.9 空运算符

空运算符对应的关键字为“void”，其作用是定义一个表达式，但该表达式并不返回任何值。修改源程序 2.11 中变量赋值语句为：

```
dataA=(dataB=1,dataC=2,dataD=3);
```

保存代码后，使用浏览器载入，在原始页面单击“运算符测试”按钮，弹出警告框如图 2.15 所示。

图 2.15　空运算符

由程序运行结果可知，使用空运算符 void 后，变量 dataA 定义为 undefined 型，并不返回任何值。

2.7.10 ?...:运算符

在 JavaScript 脚本语言中，“?...:”运算符用于创建条件分支。在动作较为简单的情况下，较之 if…else 语句更加简便，其语法结构如下：

```
(condition)?statementA:statementB;
```

载入上述语句后，首先判断条件 condition，若结果为真则执行语句 statementA，否则执行语句 statementB。值得注意的是，由于 JavaScript 脚本解释器将分号“;”作为语句的结束符，statementA 和 statementB 语句均必须为单个脚本代码，若使用多个语句会报错，例如下

列代码浏览器解释执行时得不到正确的结果：

```
(condition)?statementA:statementB;ststementC;
```

考察如下简单的分支语句：

```
var age= prompt("请输入您的年龄(数值) : ",25);
var contentA="\n 系统提示 ： \n 对不起，您未满 18 岁，不能浏览该网站！ \n";
var contentB="\n 系统提示 ： \n 点击"确定"按钮，注册网上商城开始欢乐之旅！ "
if(age<18)
{
   alert(contentA);
}
else{
   alert(contentB);
}
```

程序运行后，单击原始页面中“测试”按钮，弹出提示框提示用户输入年龄，并根据输入值决定后续操作。例如在提示框中输入整数 17，然后单击“确定”按钮，则弹出警告框如图 2.16 所示：

图 2.16 输入值为 12

若在提示框中输入整数 24，然后单击“确定”按钮，则弹出警告框如图 2.17 所示：

图 2.17 输入值为 24

上述语句中的条件分支语句完全可由“?...:”运算符简单表述：

```
(age<18)?alert(contentA):alert(contentB);
```

可以看出，使用“?...:”运算符进行简单的条件分支，语法简单明了，但若要实现较为复杂的条件分支，推荐使用 if…else 语句或者 switch 语句。

2.7.11 对象运算符

JavaScript 脚本语言主要支持四种对象运算符，包括点号运算符、new 运算符、delete 运算符以及（）运算符等。

对象包含属性和方法，点号运算符用来访问对象的属性和方法。其用法是将对象名称与对象的属性（或方法）用点号隔开，例如：

```
var myColor=document.bgColor;
window.alert(msg);
```

语句一使用变量 myColor 返回 Document 对象的 bgColor 属性，语句二调用 Window 对象的 alert()方法输出提示信息。当然，也可使用双引号“[]”来访问对象的属性，改写上述语句：

```
var myColor=document[" bgColor "];
```

new 运算符用来创建新的对象，例如创建一个新的数组对象，可以写成：

```
var exam = new Array (43,76,34 89,90);
```

new 运算符可以创建程序员自定义的对象，以可以创建 JavaScript 内建对象的实例。下列函数创建 Date 对象，并调用 Window 对象的 alert()方法输出当前时间信息：

```
function createDate()
{
   var myDate=new Date();
   var msg="\n 当前时间 : \n\n";
   msg+="          "+myDate+"          \n";
   alert(msg)
}
```

上述函数被调用后，弹出警告框显示当前时间信息，如图 2.18 所示。

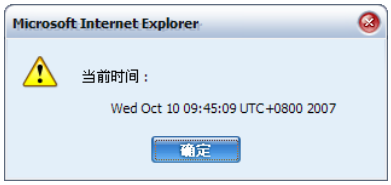

图 2.18　new 运算符

delete 运算符主要用于删除数组的特定元素，也可用来删除对象的属性、方法等。考察如下的测试代码：

```
//源程序 2.12
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\ndelete 运算符测试 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
   var myClassmate=new Array("JHX","LJY","QZY","HZF");
   getInfo(myClassmate);
   delete myClassmate[2];
   msg+="\n 执行语句 : \n delete myClassmate[2];\n\n";
   getInfo(myClassmate);
   alert(msg);
}
//获取当前数组信息
function getInfo(iArray)
{
   var myLength=iArray.length;
```

```
   msg+="数组长度 : \n "+myLength+"\n";
   msg+="数组元素 : \n";
   for(i=0;i<myLength;i++)
   {
      msg+="iArray[ "+i+" ]="+iArray[i]+"\n";
   }
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出警告框如图 2.19 所示。

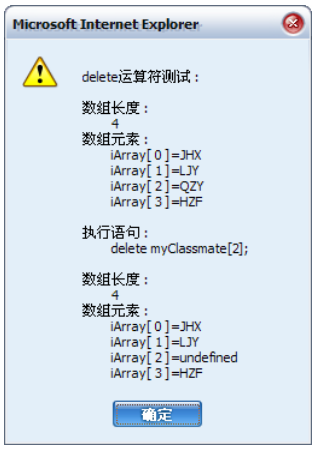

图 2.19　delete 运算符

由上图可知，执行“delete myClassmate[2];”语句后，数组元素 myClassmate[2]被定义为 undefined 类型。

“()”运算符用来调用对象的方法，例如：

```
window.alert(msg);
```

上述四种对象运算符，在后续章节将进行更为深入的介绍。

2.7.12 typeof 运算符

typeof 运算符用于表明操作数的数据类型，返回数值类型为一个字符串。在 JavaScript 脚本语言中，其使用格式如下：

```
var myString=typeof(data);
```

考察如下实例：

```
//源程序 2.13
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\ntypeof 运算符测试 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
   var myData;
   msg+="语句 : var myData;     类型 : "+typeof(myData)+" \n";
   myData=5;
   msg+="语句 : myData=5;       类型 : "+typeof(myData)+"\n";
   myData="5";
   msg+="语句 : myData="5";     类型 : "+typeof(myData)+"\n";
   myData=true;
   msg+="语句 : myData=true;    类型 : "+typeof(myData)+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，出现如图 2.20 所示页面：

图 2.20　typeof 运算符

可以看出，使用关键字 var 定义变量时，若不指定其初始值，则变量的数据类型默认为 undefined。同时，若在程序执行过程中，变量被赋予其他隐性包含特定数据类型的数值时，其数据类型也随之发生更改。

2.7.13 运算符优先级

JavaScript 脚本编程中，运算表达式中可能含有多个运算符，同其他程序语言一样，这些运算符也是有处理的先后顺序的，运算符的优先级如表 2.9 所示。

表 2.9　运算符优先级

运算符优先级	运算符	简要说明
1	( )	
	[ ]	
2	!	逻辑非
	~	按位非
	—	取负
	+ +	自加
	——	自减
	typeof	表明数据类型
3	*	乘
	/	除
	%	取余
4	+	
	—	
5	<<	按位移
	>	
	>>	
6	<	比较运算符
	>	
	<=	
	>=	
7	= =	
	!=	
8	&	按位与
9	^	按位异或
10	|	按位或
11	&&	逻辑与
12	||	逻辑或
13	?	条件表达式
14	=	赋值运算符
	+=	
	—=	
	*=	
	/=	
	%=	
	<<=	
	>=	
	>>=	
	&=	
	^=	
	|=	
15	,	参数分隔

进行表达式求值时，先执行优先级脚高的运算符，再执行优先级较低的运算符；若优先级相同则按照从左至右的顺序执行。构造特定运算功能的表达式时，应根据上述表格中列举的运算符优先级合理安排。

2.8 核心语句

前面小节讲述了 JavaScript 脚本语言数据结构方面的基础知识，包括基本数据类型、运算符、运算符优先级等，本节将重点介绍 JavaScript 脚本的核心语句。

在 JavaScript 脚本语言中，语句的基本格式为：

<statement>;

分号为语句结束标志符，为养成良好的编程习惯，在编程中应使用分号。值得注意的是，JavaScript 脚本支持符号匹配，如双引号、单引号等。若分号嵌套在上述匹配符号内，脚本解释器搜索匹配的符号：

- 若存在匹配符，则将其中的分号作为普通符号而不是作为语句结束符对待。例如：

```
var msg="语句 : var myData;";
```

- 若不存在匹配符，则提示脚本出现语法错误。例如：

```
var msg="语句 : var myData;
```

基本语句构成代码段，下面介绍 JavaScript 脚本代码的基本处理流程

2.8.1 基本处理流程

基本处理流程就是对数据结构的处理流程，在 JavaScript 里，基本的处理流程包含三种结构，即顺序结构、选择结构和循环结构。

顺序结构即按照语句出现的先后顺序依次执行，为 JavaScript 脚本程序中最基本的结构，如图 2.21 所示。

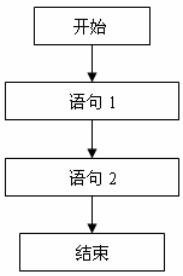

图 2.21 顺序结构

选择结构即按照给定的逻辑条件来决定执行顺序，可以分为单向选择、双向选择和多向选择。但无论是单向还是多向选择，程序在执行过程中都只能执行其中一条分支。单向选择和双向选择结构如图 2.22 所示。

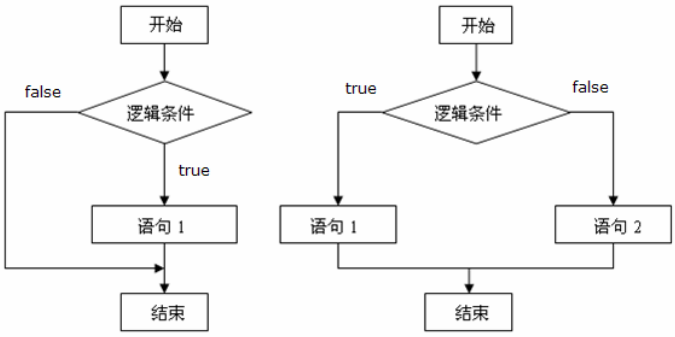

图 2.22 单向和双向选择结构

循环结构即根据代码的逻辑条件来判断是否重复执行某一段程序。若逻辑条件为 true，则重复执行，即进入循环，否则结束循环。循环结构可分为条件循环和计数循环，如图 2.23 所示。

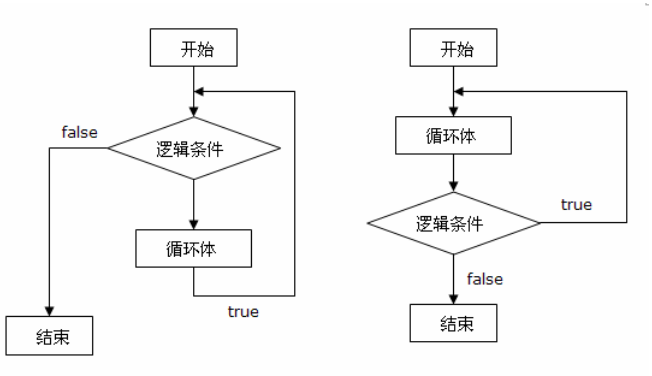

图 2.23 循环结构

一般而言，在 JavaScript 脚本语言中，程序总体是按照顺序结构执行的，而在顺序结构中可以包含选择结构和循环结构。

2.8.2 if 条件假设语句

if 条件假设语句是比较简单的一种选择结构语句，若给定的逻辑条件表达式为真，则执行一组给定的语句。其基本结构如下：

```
if(conditions)
{
 statements;
}
```

逻辑条件表达式 conditions 必须放在小括号里，且仅当该表达式为真时，执行大括号内包含的语句，否则将跳过该条件语句而执行其下的语句。大括号内的语句可为一个或多个，当仅有一个语句时，大括号可以省略。但一般而言，为养成良好的编程习惯，同时增强程序代码的结构化和可读性，建议使用大括号将指定执行的语句括起来。

if 后面可增加 else 进行扩展，即组成 if…else 语句，其基本结构如下：

```
if(conditions)
{
   statement1;
}
else
{
   statement2;
}
```

当逻辑条件表达式 conditions 运算结果为真时，执行 statement1 语句（或语句块），否则执行 statement2 语句（或语句块）。

if(或 if…else)结构可以嵌套使用来表示所示条件的一种层次结构关系。值得注意的是，嵌套时应重点考虑各逻辑条件表达式所表示的范围。

2.8.3 switch 流程控制语句

在 if 条件假设语句中，逻辑条件只能有一个，如果有多个条件，可以使用嵌套的 if 语句来解决，但此种方法会增加程序的复杂度，并降低程序的可读性。若使用 switch 流程控制语句就可完美地解决此问题，其基本结构如下：

```
switch (a)
{
   case a1:
     statement 1;
     [break;]
   case a2:
     statement 2;
     [break];
   ……
   default:
     [statement n;]
}
```

其中 a 是数值型或字符型数据，将 a 的值与 a1、a2、……比较，若 a 与其中某个值相等时，执行相应数据后面的语句，且当遇到关键字 break 时，程序跳出 statement n 语句，并重新进行比较；若找不到与 a 相等的值，则执行关键字 default 下面的语句。

考察如下的测试代码：

```
//源程序 2.14
<! DOCTYPE HTML PUBLIC    "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\nswitch 流程控制语句 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
   var year=window. prompt("请输入您的军龄(整数值,0 表示未参军) : ",25);
   var army;
   switch(year)
   {
      case 0:
         army="平民";
         break;
      case 1:
         army="列兵";
         break;
      case 2:
         army="上等兵";
         break;
      case 3:
      case 4:
      case 5:
         army="一级士官";
         break;
      case 6:
      case 7:
      case 8:
         army="二级士官";
         break;
```

```
      default:
         if (year>8)
            army="中高级士官";
   }
   msg+="军龄 : "+year+"年\n";
   msg+="结论 : "+army+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<form>
   <input type=button value="测试" onclick="Test()">
</form>
</body>
</html>
```

程序运行后，在原始页面中单击“测试”按钮，将弹出提示框提示用户输入相关信息，例如输入 12，单击“确定”按钮提交，弹出警告框如图 2.24 所示。

图 2.24　switch 流程控制语句

2.8.4 for 循环语句

for 循环语句是循环结构语句，按照指定的循环次数，循环执行循环体内语句（或语句块），其基本结构如下：

```
for(initial condition; test condition; alter condition)
{
    statements;
}
```

循环控制代码（即小括号内代码）内各参数的含义如下：

- initial condition 表示循环变量初始值；
- test condition 为控制循环结束与否的条件表达式，程序每执行完一次循环体内语句（或语句块），均要计算该表达式是否为真，若结果为真，则继续运行下一次循环体内语句（或语句块）；若结果为假，则跳出循环体。
- alter condition 指循环变量更新的方式，程序每执行完一次循环体内语句（或语句块），均需要更新循环变量。

上述循环控制参数之间使用分号“;”间隔开来，考察如下的测试函数：

```
function Test()
{
   var iArray=new Array("JHX","QZY","LJY","HZF");
   var iLength=iArray.length;
   var msg="\nfor 循环语句测试 :\n\n";
```

```
   msg+="数组长度 : \n "+iLength+"\n";
   msg+="数组元素 : \n";
   for(var i=0;i<iLength;i++)
   {
      msg+="iArray["+i+"] ="+iArray[i]+" \n";
   }
   alert(msg);
}
```

上述函数被调用后，弹出警告框如图 2.25 所示。

图 2.25　for 循环语句举例

2.8.5 while 和 do-while 循环语句

while 语句与 if 语句相似，均有条件地控制语句（或语句块）的执行，其语言结构基本相同:

```
while(conditions)
{
 statements;
}
```

while 语句与 if 语句的不同之处在于：在 if 条件假设语句中，若逻辑条件表达式为真，则运行 statements 语句（或语句块），且仅运行一次；while 循环语句则是在逻辑条件表达式为真的情况下，反复执行循环体内包含的语句（或语句块）。

> 注意：while 语句的循环变量的赋值语句在循环体前，循环变量更新则放在循环体内；for 循环语句的循环变量赋值和更新语句都在 for 后面的小括号中，在编程中应注意二者的区别。

改写 Test()函数代码如下，程序运行结果不变：

```
function Test()
{
   var iCount=0;
   var iArray=new Array("JHX","QZY","LJY","HZF");
   var iLength=iArray.length;
   var msg="\nfor 循环语句测试 :\n\n";
   msg+="数组长度 : \n "+iLength+"\n";
   msg+="数组元素 : \n";
   while(iCount<iLength)
   {
      msg+="iArray["+iCount+"] ="+iArray[iCount]+" \n";
      iCount+=1;
   }
   alert(msg);
```

```
}
```

在某些情况下，while 循环大括号内的 statements 语句（或语句块）可能一次也不被执行，因为对逻辑条件表达式的运算在执行 statements 语句（或语句块）之前。若逻辑条件表达式运算结果为假，则程序直接跳过循环而一次也不执行 statements 语句（或语句块）。

若希望至少执行一次 statements 语句（或语句块），可改用 do…while 语句，其基本语法结构如下：

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

改写 Test()函数代码如下，程序运行结果不变：

```
function Test()
{
   var iCount=0;
   var iArray=new Array("JHX","QZY","LJY","HZF");
   var iLength=iArray.length;
   var msg="\nfor 循环语句测试 :\n\n";
   msg+="数组长度 : \n "+iLength+"\n";
   msg+="数组元素 : \n";
   do{
      msg+="iArray["+iCount+"] ="+iArray[iCount]+" \n";
      iCount+=1;
   }while(iCount<iLength);
   alert(msg);
}
```

for、while、do…while 三种循环语句具有基本相同的功能，在实际编程过程中，应根据实际需要和本着使程序简单易懂的原则来选择到底使用哪种循环语句。

2.8.6 使用 break 和 continue 进行循环控制

在循环语句中，某些情况下需要跳出循环或者跳过循环体内剩余语句，而直接执行下一次循环，此时需要通过 break 和 continue 语句来实现。break 语句的作用是立即跳出循环，continue 语句的作用是停止正在进行的循环，而直接进入下一次循环。

考察如下测试代码：

```
//源程序 2.15
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 使用 break 和 continue 控制循环 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
   var n=-1;
   var iArray=new Array("YSQ","JHX","QZY","LJY","HZF","XGM","LJY","LHZ");
   var iLength=iArray.length;
```

```
   msg+="数组长度 : \n "+iLength+"\n";
   msg+="数组元素 : \n";
   while(n<iLength)
   {
      n+=1;
      if(n==3)
         continue;
      if(n==6)
         break;
      msg+="iArray["+n+"] = "+iArray[n]+"\n";
   }
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“测试”按钮，弹出警告框如图 2.26 所示。

图 2.26 break 和 continue 语句

从上图的结果可以看出：

- 当 n=3 时，跳出当前循环而直接进行下一个循环，故 iArray[3]不进行显示；
- 当 n=6 时，直接跳出 while 循环，不再执行余下的循环，故 iArray[5]之后的数组元素不进行显示。

在编写 JavaScript 脚本过程中，应根据实际需要来选择使用哪一种循环语句，并确保在使用了循环控制语句 continue 和 break 后，循环不出现任何差错。

2.8.7 with 对象操作语句

在编写 JavaScript 脚本过程中，经常需引用同一对象的多个属性或方法，正常的对象属性或方法的引用途径能达到既定的目的，但代码显得尤为复杂。JavaScript 脚本语言提供 with 操作语句来简化对象属性和方法的引用过程，其语法结构如下：

```
with (objct)
```

```
{
    statements;
}
```

例如下列连续引用 document 对象的 write()方法的语句：

```
document.write("Welcome to China");
document.write("Welcome to Beijing");
document.write("Welcome to Shanghai");
```

可以使用 with 语句简化为：

```
with(document)
{
    write("Welcome to China");
    write("Welcome to Beijing");
    write("Welcome to Shanghai");
}
```

在脚本代码中适当使用 with 语句可使脚本代码简明易懂，避免不必要的重复输入。若脚本代码中涉及到多个对象，不推荐使用 with 语句，避免造成属性或方法引用的混乱。

2.8.8 使用 for...in 进行对象循环

使用 for…in 循环语句可以对指定对象的属性和方法进行遍历，其语法结构如下：

```
for (变量名  in  对象名)
{
    statements;
}
```

考察如下测试代码：

源程序 2.16

```
<! DOCTYPE HTML PUBLIC    "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\nfor...in 对象循环语句遍历对象 :                    \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
   var i=0;
   msg+="Window 对象支持的属性和方法 : \n";
   for(num in window)
   {
      msg+=num+"              ";
      i+=1;
      if((i%5)==0)
         msg+="\n";
   }
   alert(msg);
}
-->
</script>
```

```
</head>
<body>
<form>
   <input type=button value="测试" onclick="Test()">
</form>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.27 所示。

图 2.27　for…in 对象循环语句

2.8.9 含标签的语句

经常在循环标志前加上标签文本来引用该循环，其使用方法是标识符后面加冒号“:”。在使用 break 和 continue 语句配合使用控制循环语句时，可使用 break 或 continue 加上标识符的形式使循环跳转到指定的位置。

考察如下的测试代码：

```
//源程序 2.17
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 使用标签控制循环语句 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
   msg+="循环流程 : \n";
   outer:
   for(m=1;m<4;m++)
   {
      msg+="外循环   第"+m+"次\n";
      for(n=1;n<4;n++)
      {
         if(n==2)
            break outer;
         msg+="内循环   第"+n+"次\n";
      }
```

```
    }
    alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
    <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.28 所示。

图 2.28　使用标签控制循环

若不加标签 outer，而直接使用 break 语句跳出循环，其运行结果如图 2.29 所示。

图 2.29　不使用标签 outer

比较上述两个实例可知：

- 若增加标签 outer，则执行到 break outer 语句时跳出整个 while 循环；
- 若直接使用 break 语句仅跳出 break 所在的 for 循环。

由此可见，标签对循环控制非常有效。若配合 break 和 continue 语句使用，可精确控制循环的走向，在实际编写脚本代码过程中应根据需要选择添加标签与否。

2.9 函数

JavaScript 脚本语言允许开发者通过编写函数的方式组合一些可重复使用的脚本代码块，增加了脚本代码的结构化和模块化。函数是通过参数接口进行数据传递，以实现特定的

功能。本小节将重点介绍函数的基本概念、组成、全局函数与局部函数、作为对象的函数以及递归函数等知识，让读者从头开始，学习如何编写执行效率高、代码利用率高，且易于查看和维护的函数。

2.9.1 函数的基本组成

函数由函数定义和函数调用两部分组成，应首先定义函数，然后再进行调用，以养成良好的编程习惯。

函数的定义应使用关键字 function，其语法规则如下：

```
function funcName ([parameters])
{
   statements;
   [return 表达式;]
}
```

函数的各部分含义如下：

- funcName 为函数名，函数名可由开发者自行定义，与变量的命名规则基本相同；
- parameters 为函数的参数，在调用目标函数时，需将实际数据传递给参数列表以完成函数特定的功能。参数列表中可定义一个或多个参数，各参数之间加逗号“,”分隔开来，当然，参数列表也可为空；
- statements 是函数体，规定了函数的功能，本质上相当于一个脚本程序；
- return 指定函数的返回值，为可选参数。

自定义函数一般放置在 HTML 文档的<head>和</head>标记对之间。除了自定义函数外，JavaScript 脚本语言提供大量的内建函数，无需开发者定义即可直接调用，例如 window 对象的 alert()方法即为 JavaScript 脚本语言支持的内建函数。

函数定义过程结束后，可在文档中任意位置调用该函数。引用目标函数时，只需在函数名后加上小括号即可。若目标函数需引入参数，则需在小括号内添加传递参数。如果函数有返回值，可将最终结果赋值给一个自定义的变量并用关键字 return 返回。

考察如下测试代码：

```
//源程序 2.18
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 函数调用实例 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
   var i=10;
   var j=15;
   var temp=sum(i,j);
   msg+="函数参数 : \n";
   msg+="参数 1:     i="+i+"\n";
   msg+="参数 2:     j="+j+"\n";
   msg+="调用语句 : \n";
```

```
    msg+="var temp=sum(i,j); \n";
    msg+="返回结果 : \n";
    msg+=""+i+"+"+j+" = "+temp+"\n";
    alert(msg);
}
//计算两个数的加和
function sum(data1,data2)
{
    var tempData=data1+data2;
    return tempData;
}
-->
</script>
</head>
<body>
<center>
<form>
    <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.30 所示。

图 2.30　函数调用

上述代码中，定义了实现两数加和的函数 sum(data1,data2)及响应“测试”按钮 onclick 事件处理程序的 Test()函数，并在后者内部调用了 window 对象的内建函数 alert()，实现了函数的相互引用。

如果函数中引用的外部函数较多或函数的功能很复杂，势必导致函数代码过长而降低脚本代码可读性，违反了开发者使用函数实现特定功能的初衷。因此，在编写函数时，应尽量保持函数功能的单一性，使脚本代码结构清晰、简单易懂。

2.9.2 全局函数与局部函数

JavaScript 脚本语言提供了很多全局（内建）函数，在脚本编程过程中可直接调用，在此介绍四种简单的全局函数：parseInt()、parseFloat()、escape()和 unescape()。

parseInt()函数的作用是将字符串转换为整数，parseFloat()函数的作用是将字符串转换为浮点数；escape()函数的作用是将一些特殊字符转换成 ASCII 码，而 unescape()函数的作用是将 ASCII 码转换成字符。

考察如下测试代码：

```
//源程序 2.19
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 全局函数调用实例 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
   var string1="30121";
   var string2="34.12";
   var string3="Money*#100";
   var temp1,temp2,temp3,temp4;
   msg+="原始变量 : \n";
   msg+="string1 = "+string1+"类型 : "+typeof(string1)+"\n";
   msg+="string2 = "+string2+"类型 : "+typeof(string2)+"\n";
   msg+="string3 = "+string3+"类型 : "+typeof(string3)+"\n";
   msg+="执行语句与结果:\n";
   temp1=parseInt(string1);
   temp2=parseInt(string2);
   msg+="语句 : parseInt(string1)   结果 : string1="+temp1+"类型 : "+typeof(temp1)+"\n";
   msg+="语句 : parseInt(string2)   结果 : string1="+temp2+"类型 : "+typeof(temp2)+"\n";
   temp1=parseFloat(string1);
   temp2=parseFloat(string2);
   msg+="语句 : parseFloat(string1)   结果 : string1="+temp1+"类型 : "+typeof(temp1)+"\n";
   msg+="语句 : parseFloat(string2)   结果 : string1="+temp2+"类型 : "+typeof(temp2)+"\n";
   temp3=escape(string3);
   msg+="语句 : temp3=escape(string3)      结果 : temp3="+temp3+"类型 : "+typeof(temp3)+"\n";
   temp4=unescape(temp3);
   msg+="语句 : temp4=unescape(temp3)   结果 : temp4="+temp4+"类型 : "+typeof(temp4)+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.31 所示。

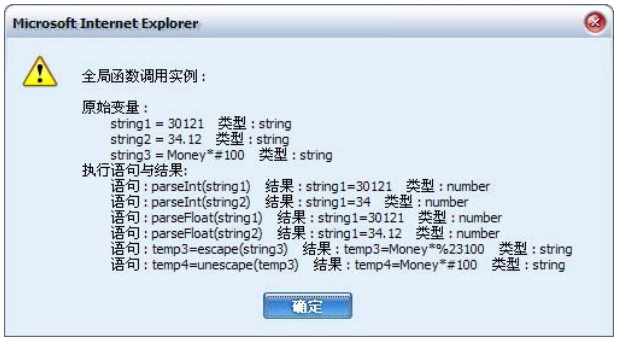

图 2.31 全局函数

由程序运行结果可知上述全局函数的具体作用，当然 JavaScript 脚本语言还支持很多其他的全局函数，在编程中适当使用它们可大大提高编程效率。

与全局函数相对应的函数是局部函数，即定义在某特定函数内部，并仅能在其内使用的函数。

考察如下测试代码：

```
//源程序 2.20
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 局部函数调用实例 ：\n\n";
//响应按钮的 onclick 事件处理程序
function multi(m,n)
{
   var result;
   function inner(m)
   {
      if (m%2!=0)
         return 0;
      else
         return 1;
   }
   result=inner(m)*n;
   msg+="输入参数 ：\n";
   msg+="m = "+m+"\n             n = "+n+"\n";
   msg+="乘积结果 ：\n";
   msg+="result = "+result+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="测试" onclick="multi(4,3)">
```

```
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.32 所示。

图 2.32 局部函数

函数 muti()内部定义了局部函数 inner()，判断变量 m 是否为偶数，如果是偶数则返回 1，否则返回 0。根据调用语句 muti(4,3)，m=4 为偶数，故局部函数 inner()返回值为 1，函数 muti()的返回值为 3。

> 注意：通过上述方式定义的函数为局部函数，函数的作用域为自所属的框架函数，任何处于框架函数外部对局部函数的引用均为不合法。

2.9.3 作为对象的函数

JavaScript 脚本语言中所有的数据类型、数组等均可作为对象对待，函数也不例外。可以使用 new 操作符和 Function 对象的构造函数 Function()来生成指定规则的函数，其基本语法如下：

```
var funcName = new Function (arguments,statements;);
```

值得注意的是，上述的构造函数 Function()首字母必须为大写，同时函数的参数列表与操作代码之间使用逗号隔开。考察如下测试代码：

```
//源程序 2.21
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 使用构造函数 Function 构造函数 : \n\n";
//使用 new 操作符和 Function()构造函数生成函数 newFunc()
var newFunc=new Function("result","alert(msg+'     '+result)");
//响应按钮的 onclick 事件处理程序
function Test()
{
   msg+="生成语句: \n";
   msg+="var newFunc=new Function("result","alert(msg+result)"); \n"
   msg+="调用语句: \n";
   msg+="newFunc("Welcome to JavaScript World!");\n";
```

```
   msg+="返回结果: \n";
   newFunc("Welcome to JavaScript World!");
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.33 所示。

图 2.33　作为对象的函数

通过 new 操作符和 Function()构造函数定义函数对象时，并没有给函数赋予名称，而是定义函数后直接将其赋值给变量 newFunc，并通过 newFunc 进行访问，与通常的函数定义不同。

> 注意：在定义函数对象时，参数列表可以为空，也可有一个或多个参数，使用变量引用该函数时，应将函数执行所需要的参数传递给函数体。

作为对象的函数最重要的性质即为它可以创建静态变量，给函数增加实例属性，使得函数在被调用之间也能发挥作用。考察如下测试代码：

```
//源程序 2.22
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 作为对象的函数创建静态变量 : \n\n";
function sum(x,y)
{
   sum.result=sum.result+x+y;
   return(sum.result);
}
sum.result=0;
//响应按钮的 onclick 事件处理程序
```

```
function Test()
{
   var tempData;
   msg+="调用语句及返回结果: \n";
   tempData=sum(2,3);
   msg+="语句 : tempData=sum(2,3);";
   msg+="结果 : tempData = "+tempData+"       sum.result = "+sum.result+"\n";
   tempData=sum(4,5);
   msg+="语句 : tempData=sum(4,5);";
   msg+="结果 : tempData = "+tempData+"       sum.result = "+sum.result+"\n";
   tempData=sum(6,7);
   msg+="语句 : tempData=sum(6,7);";
   msg+="结果 : tempData = "+tempData+"       sum.result = "+sum.result+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.34 所示。

图 2.34　创建静态变量

由上述结果可以看出，作为对象的函数使用静态变量后，可以用来保存其运行的环境参数如中间值等数据。

2.9.4 函数递归调用

函数的递归调用即函数在定义时调用自身，考察如下实例代码：

```
//源程序 2.35
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
```

```
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 函数的递归调用 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
   var result;
   msg+="调用语句 : \n";
   msg+="result = sum(6);\n";
   msg+="调用步骤 : \n";
   result=sum(6);
   msg+="计算结果 : \n";
   msg+="result = "+result+"\n";
   alert(msg);
}
//计算当前步骤加和值
function sum(m)
{
   if(m==0)
      return 0;
   else
   {
      msg+="语句 : result = " +m+ "+sum(" +(m-1)+"); \n";
      result=m+sum(m-1);
   }
   return result;
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.35 所示。

图 2.35　函数递归调用

函数递归调用能使代码显得紧凑、简练，但也存在执行效率并低、容易出错、资源耗费

较多等问题，推荐在递归调用次数较少的情况下使用该方法，其余情况尽量使用其余方法来代替。

2.9.5 语言注释语句

在 JavaScript 脚本代码中，可加如一些提示性的语句，以便提示代码的用途及跟踪程序执行的流程，并增加程序的可读性，同时有利于代码的后期维护。上述提示性语句称作语言注释语句，JavaScript 脚本解释器并不执行语言解释语句。

一般使用双反斜杠“//”作为每行解释语句的开头，例如：

```
// 程序解释语句
```

值得注意的是，必须在每行注释语句前均加上双反斜杠“//”。例如：

```
// 程序解释语句 1
// 程序解释语句 2
```

下面的语句为错误的注释语句：

```
// 程序解释语句 1
   程序解释语句 2
```

上述语句在中第二行不被脚本解释器作为解释语句，而作为普通的代码对待，即由脚本解释器解释执行。

对于多行的解释语句，可以在解释语句开头加上“/*”，末尾加上*/，不须在每行开头加上双反斜杠“//”。例如：

```
/* 程序解释语句 1
    程序解释语句 2
    程序解释语句 3 */
```

JavaScript 脚本语言中，还允许使用 HTML 风格的语言注释，即用“<!--”来代替双反斜杠“//”。与 HTML 文档注释不同的是，HTML 允许“<!--”跨越多行进行注释，而 JavaScript 脚本里必须在每行注释前加上“<!--”。另外，JavaScript 脚本里不需要以“-->”来结束注释语句，而 HTML 中则必须用“-->”来结束注释语句。

为养成良好的编程习惯，最好不用 HTML 风格的语言注释语句，而使用双反斜杠“//”来加入单行注释语句，用“/*”和“*/”加入多行注释语句。

2.9.6 函数应用注意事项

最后介绍一下在使用函数过程中应特别予以注意的几个问题，以帮助读者更好、更准确确地使用函数，并养成良好的编程习惯。具体表现在如下几点：

- 定义函数的位置：如果函数代码较为复杂，函数之间相互调用较多，应将所有函数的定义部分放在 HTML 文档的<head>和</head>标记对之间，既可保证所有的函数在调用之前均已定义，又可使开发者后期的维护工作更为简便；
- 函数的命名：函数的命名原则与变量的命名原则相同，但尽量不要将函数和变量取同一个名字。如因实际情况需要将函数和变量定义相近的名字，也应给函数加上可以清楚辨认的字符（如前缀 func 等）以示区别；
- 函数返回值：在函数定义代码结束时，应使用 return 语句返回，即使函数不需要返回任何值；
- 变量的作用域：区分函数中使用的变量是全局变量还是局部变量，避免调用过程中出现难以检查的错误；

- 函数注释：在编写脚本代码时，应在适当的地方给代码的特定行添加注释语句，例如将函数的参数数量、数据类型、返回值、功能等注释清楚，既方便开发者对程序的后期维护，也方便其他人阅读和使用该函数，便于模块化编程；
- 函数参数传递：由于 JavaScript 是弱类型语言，使用变量时并不检查其数据类型，导致一个潜在的威胁，即开发者调用函数时，传递给函数的参数数量或数据类型不满足要求而导致错误的出现。在函数调用时，应仔细检查传递给目标函数的参数变量的数量和数据类型。

其中第五点尤为值得特别关注，因由其导致的错误非常难于检测。考察如下两数乘法的测试代码：

```
function muti(x,y)
{
   if(muti.arguments.length==2)
      return (x*y);
}
```

以上代码检查了输入参数的数量，但未检查操作数的数据类型，例如输入字符串“num1”和“num2”，并调用 multi(num1,num2)时，浏览器报错。修改函数 multi()如下：

```
function muti(x,y)
{
   if(muti.arguments.length==2)
   {
      if( (typeof(x)!="number" || (typeof(y)!="number"))
          return errorNum;
      else
          return (x*y);
      }
return;
}
```

这个函数既检查了参数的数量，又检查了参数的数据类型，避免了我们在调用时传递了错误的参数数量或数据类型。

以上简要讨论了在使用函数时应注意的问题，应该说编写一个好的函数，一个好的脚本程序是不容易的，需要读者朋友在以后的编程实践中去摸索，去总结，养成良好的编程习惯。

2.10 本章小结

本章介绍了 JavaScript 脚本语言的基本语法知识，包括数据类型、变量、运算符、核心语句以及函数等相关内容。其中，数据类型和运算符较为简单，通过本章的学习相信读者可以完全掌握；变量、核心语句和函数等知识，本章只作简要的介绍，在后续章节将加大介绍的力度。

本章还涉及到小部分与对象相关的知识，在后续章节中将进行深入的讲解。JavaScript 脚本是基于事件的程序开发语言，下一章将重点介绍“JavaScript 事件处理”的相关知识。