**Function 对象**

创建新的函数。

**语法 1**

**function** *functionName***([***argname1* **[, ...[,** *argnameN***]]])**  
**{**   *body*  
**}**

**语法 2**

*functionName* **= new Function( [***argname1*, **[...** *argnameN***,]]** *body* **);**

**参数**

*functionName*

必选项。最新创建函数的名称

argname1...argnameN

可选项。函数接收的参数列表。

body

可选项。包含调用该函数时被执行的 JScript 代码块的字符串。

**说明**

函数 JScript 中的基本数据类型。语法 1 创建在必要时由 JScript 转换为 **Function** 对象的函数值。JScript 在调用函数时将用语法 2 创建的函数转换为 **Fnction** 对象。

语法 1 是 JScript 中创建函数的基本方法。语法 2 是用于显式创建函数对象的另一方法。

例如，要创建将传递给它的两个参数相加的函数，可以用两种方法中的任一种完成：

**例子 1**

**function** add(x, y)

{

   return(x + y); // 执行加法并返回结果。

}

**例子 2**

var add = new **Function(**"x", "y", "return(x+y)"**)**;

在两种情况下，都可以用如下代码行调用该函数：

add(2, 3);

**注意**   在调用函数时，请确保包含了括号和必需的参数。调用函数时不用括号导致返回函数的文本而不是函数执行的结果。

# 属性

## arguments 属性

为当前执行的 **function** 对象返回一个**arguments** 对象。

*function*.**arguments**

function 参数是当前执行函数的名称，可以省略。

#### 说明

通过 **arguments** 属性，函数可以处理可变数量的参数。 **arguments** 对象的 **length** 属性包含了传递给函数的参数的数目。对于**arguments** 对象所包含的单个参数，其访问方法与数组中所包含的参数的访问方法相同。

#### 示例

下面的例子说明了 **arguments** 属性的用法：

function ArgTest(){

   var i, s, numargs = arguments.length;

   s = numargs;

   if (numargs < 2)

      s += " argument was passed to ArgTest. It was ";

   else

      s += " arguments were passed to ArgTest. They were " ;

   for (i = 0; i < numargs; i++)

      {

         s += **arguments[**i**]** + " ";

      }

   return(s);

}

## caller 属性

返回一个对函数的引用，该函数调用了当前函数。

*functionName*.**caller**

functionName 对象是所执行函数的名称。

#### 说明

对于函数来说，**caller** 属性只有在函数执行时才有定义。 如果函数是由 JScript 程序的顶层调用的，那么 **caller** 包含的就是 **null** 。

如果在字符串上下文中使用 **caller** 属性，那么结果和 *functionName*.**toString** 一样，也就是说，显示的是函数的反编译文本。

#### 例子

下面的例子说明了 **caller** 属性的用法：

function CallLevel(){

   if (**CallLevel.caller** == null)

      return("CallLevel was called from the top level.");

   else

      return("CallLevel was called by another function.");

}

## constructor 属性

表示创建对象的函数。

*object*.**constructor**

必需的 *object*是对象或函数的名称。

#### 说明

**constructor** 属性是所有具有 prototype 的对象的成员。它们包括除 **Global** 和 **Math** 对象以外的所有 [JScript 固有对象](mk:@MSITStore:C:\Users\Administrator.win7-PC\Desktop\Javascipt语言手册.chm::/html/glossary.htm#defintrinsicobjectjs)。**constructor** 属性保存了对构造特定对象实例的函数的引用。

### 例子：

x = new String("Hi");

if (x.**constructor** == String)

   // 进行处理（条件为真）。

或

function MyFunc {

   // 函数体。

}

y = new MyFunc;

if (y.**constructor** == MyFunc)

   // 进行处理（条件为真）。

## prototype 属性

返回对象类型原型的引用。

*objectName*.**prototype**

*objectName* 参数是对象的名称。

#### 说明

用 **prototype** 属性提供对象的类的一组基本功能。 对象的新实例“继承”赋予该对象原型的操作。

例如，要为 **Array** 对象添加返回数组中最大元素值的方法。 要完成这一点，声明该函数，将它加入 **Array.prototype**， 并使用它。

function array\_max( ){

   var i, max = this[0];

   for (i = 1; i < this.length; i++)

   {

   if (max < this[i])

   max = this[i];

   }

   return max;

}

Array.**prototype**.max = array\_max;

var x = new Array(1, 2, 3, 4, 5, 6);

var y = x.max( );

该代码执行后，*y* 保存数组 *x* 中的最大值，或说 6。

所有 [JScript 固有对象](mk:@MSITStore:C:\Users\Administrator.win7-PC\Desktop\Javascipt语言手册.chm::/html/glossary.htm#defintrinsicobjectjs)都有只读的 **prototype** 属性。可以象该例中那样为原型添加功能，但该对象不能被赋予不同的原型。然而，[用户定义的对象](mk:@MSITStore:C:\\Users\\Administrator.win7-PC\\Desktop\\Javascipt语言手册.chm::/html/glossary.htm" \l "defuserobject)可以被赋给新的原型。

本语言参考中每个内部对象的方法和属性列表指出哪些是对象原型的部分，哪些不是

# function 语句

声明一个新的函数。

**function** *functionname***(**[*arg1* [, *arg2* [,...[, *argN*]]]]**)**  
{  
   *statements*  
}

#### 参数

functionname

必选项。函数名。

arg1...argN

可选项。函数理解的参数列表、各个参数之间用逗号分开。

statements

可选项。一个或多个 JScript 语句。

#### 说明

使用 **function** 语句来声明一个稍后要使用的函数。在脚本的其他地方调用该函数前，*statements* 中包含的代码不被执行。

#### 示例

下面示例说明了 **function** 语句的用法。

**function** myfunction**(**arg1, arg2**)**{

   var r;

   r = arg1 \* arg2;

   return(r);

}

**注意**在调用函数时，确保总是包括了圆括号以及任何需要的参数。不带圆括号调用函数将导致返回函数的文本，而不是函数的结果。

# 方法

## toString 方法

返回对象的字符串表示。

objectname.**toString(**[radix]**)**

**参数**

objectname

必选项。要得到字符串表示的对象。

radix

可选项。指定将数字值转换为字符串时的进制。

**说明**

**toString** 方法是所有内建的 JScript 对象的成员。它的操作依赖于对象的类型：

|  |  |
| --- | --- |
| **对象** | **操作** |
| Array | 将 **Array** 的元素转换为字符串。结果字符串由逗号分隔，且连接起来。 |
| Boolean | 如果 Boolean 值是 **true**，则返回 “true”。否则，返回 “false”。 |
| Date | 返回日期的文字表示法。 |
| Error | 返回一个包含相关错误消息的字符串。 |
| Function | 返回如下格式的字符串，其中 *functionname* 是被调用 **toString** 方法函数的名称：  function functionname( ) { [native code] } |
| Number | 返回数字的文字表示。 |
| String | 返回 **String** 对象的值。 |
| 默认 | 返回 “[object objectname]”，其中 objectname 是对象类型的名称。 |

**示例**

下面的例子演示了使用带有 radix 参数的 toString 方法。上面所示函数的返回值是一个 Radix 转换表。

function CreateRadixTable (){

var s, s1, s2, s3, x; // 声明变量。

s = "Hex Dec Bin \n"; // 创建表头。

for (x = 0; x < 16; x++) // 根据所示值的

{ // 数字建立

switch(x) // 表尺寸。

{ // 设置栏目间空间。

case 0 :

s1 = " ";

s2 = " ";

s3 = " ";

break;

case 1 :

s1 = " ";

s2 = " ";

s3 = " ";

break;

case 2 :

s3 = " ";

break;

case 3 :

s3 = " ";

break;

case 4 :

s3 = " ";

break;

case 5 :

s3 = " ";

break;

case 6 :

s3 = " ";

break;

case 7 :

s3 = " ";

break;

case 8 :

s3 = "" ;

break;

case 9 :

s3 = "";

break;

default:

s1 = " ";

s2 = "";

s3 = " ";

} // 转换为十六进制、十进制、二进制。

s += " " + **x.toString(**16**)** + s1 + **x.toString(**10**)**

s += s2 + s3 + **x.toString(**2**)**+ "\n";

}

return(s); // 返回整个 radix 表。

}

## valueOf 方法

返回指定对象的原始值。

*object*.**valueOf( )**

必选项 *object* 参数是任意固有 JScript 对象。

**说明**

每个 JScript 固有对象的 **valueOf** 方法定义不同。

|  |  |
| --- | --- |
| **对象** | **返回值** |
| Array | 数组的元素被转换为字符串，这些字符串由逗号分隔，连接在一起。其操作与 **Array.toString** 和 **Array.join** 方法相同。 |
| Boolean | Boolean 值。 |
| Date | 存储的时间是从 1970 年 1 月 1 日午夜开始计的毫秒数 [UTC](mk:@MSITStore:C:\Users\Administrator.win7-PC\Desktop\Javascipt语言手册.chm::/html/glossary.htm#defutctime)。 |
| Function | 函数本身。 |
| Number | 数字值。 |
| Object | 对象本身。这是默认情况。 |
| String | 字符串值。 |

**Math** 和 **Error** 对象没有 **valueOf** 方法。

# JScript 的变量

任何编程语言中，用一块数据量化一个概念。

How old am I?

在 Jscript 中，变量是给概念的名称；它代表了给出瞬间的值。当使用该变量时，实际是用的它所代表的数据。给出示例：

NumberOfDaysLeft = EndDate – TodaysDate;

机械的理解是使用变量来存储、得到并操作脚本中出现的所有的不同值。创建有意义的变量名称；便于别人理解脚本。

## 变量声明

变量在脚本中的第一次出现是在声明中。变量在第一次用到时就设置于内存中，便于后来在脚本中引用。使用变量之前先进行声明。可以使用 **var** 关键字来进行变量声明。

var count; // 单个声明。

var count, amount, level; // 用单个 var 关键字声明的多个声明。

var count = 0, amount = 100; // 一条语句中的变量声明和初始化。

如果在 **var** 语句中没有初始化变量，变量自动取 JScript 值 **undefined**。尽管并不安全，但声明语句中忽略 **var** 关键字是合法的 JScript 语法。这时，JScript 解释器给予变量全局范围的可见度。当在过程级中声明一个变量时，它不能用于全局范围；这种情况下，变量声明必须用 **var** 关键字。

## 变量命名

变量名称是一个标识符。Jscript 中，用标识符来：

* 命名变量，
* 命名函数，
* 给出循环的标签。

JScript 是一种区分大小写的语言。因此变量名称 *myCounter* 和变量名称 *mYCounter* 是不一样的。变量的名称可以是任意长度。创建合法的变量名称应遵循如下规则：

* 第一个字符必须是一个 ASCII 字母（大小写均可），或一个下划线(\_)。注意第一个字符不能是数字。
* 后续的字符必须是字母、数字或下划线。
* 变量名称一定不能是 [保留字](mk:@MSITStore:C:\Users\Administrator.win7-PC\Desktop\Javascipt语言手册.chm::/html/jsreserved.htm)。

下面给出合法变量名称的一些示例：

\_pagecount

Part9

Number\_Items

下面给出无效变量名称的一些示例：

99Balloons // 不能以数字开头。

Smith&Wesson // “与”符号（&）字符用于变量名称是无效的。

当要声明一个变量并进行初始化，但又不想指定任何特殊值，可以赋值为 JScript 值 null。下面给出示例。

var bestAge = null;

var muchTooOld = 3 \* bestAge; // muchTooOld 的值为 0。

如果声明了一个变量但没有对其赋值，该变量存在，其值为Jscript 值 undefined。下面给出示例。

var currentCount;

var finalCount = 1 \* currentCount; // finalCount 的值为 NaN，因为 currentCount 为 undefined。

注意在 JScript 中 **null** 和 **undefined** 的主要区别是 **null** 的操作象数字 0，而 **undefined** 的操作象特殊值**NaN** （不是一个数字）。对 **null** 值和 **undefined** 值作比较总是相等的。

可以不用 **var** 关键字声明变量，并赋值。这就是隐式声明。

noStringAtAll = ""; // 隐式声明变量 noStringAtAll。

不能使用未经过声明的变量。

var volume = length \* width; // 错误 — length 和 width 并不存在。

## 强制转换

表达式中操作项的数据类型相同时 JScript 解释器才能对其求值。如果表达式不经过强制转换就试图对两个不同的数据类型（如一个为数字，另一个为字符串）执行运算，将产生错误结果。但在 Jscript 中情况就不同了。

JScript 是一种自由类型的语言。它的变量没有预定类型（相对于强类型语言，如 C++）。相反，JScript 变量的类型相应于他们包含的值的类型。这种操作的好处是能将值作为另一类型处理。

在 Jscript 中，可以对不同类型的值执行运算，不必担心 JScript 解释器产生异常。相反，JScript 解释器自动将数据类型之一改变（强制转换）为另一种数据类型，然后执行运算。例如：

|  |  |
| --- | --- |
| **运算** | **结果** |
| 数值与字符串相加 | 将数值强制转换为字符串。 |
| 布尔值与字符串相加 | 将布尔值强制转换为字符串。 |
| 数值与布尔值相加 | 将布尔值强制转换为数值。 |

考虑下面的示例。

var x = 2000; // 一个数字。

var y = "Hello"; // 一个字符串。

x = x + y; // 将数字强制转换为字符串。

document.write(x); // 输出 2000Hello。

要想显式地将字符串转换为整数，使用 parseInt 方法。要想显式地将字符串转换为数字，使用 parseFloat 方法。请注意，比较大小时字符串自动转换为相等的数字，但加法（连接）运算时保留为字符串。