第 6 章 String、Math、Array 等数据对象

JavaScript 脚本提供丰富的内置对象，包括同基本数据类型相关的对象（如 String、Boolean、 Number）、允许创建用户自定义和组合类型的对象（如 Object、Array）和其他能简化 JavaScript 操作的对象（如 Math、Date、RegExp、Function）。其中 RegExp 对象将在“正则表达式”章节进行详细的叙述，本章从实际应用出发，详细讨论其余的 JavaScript 脚本内置对象。

6.1 String 对象

String 对象是和原始字符串数据类型相对应的 JavaScript 脚本内置对象，属于 JavaScript 核心对象之一，主要提供诸多方法实现字符串检查、抽取子串、字符串连接、字符串分割等字符串相关操作。

语法如下：

```
var MyString=new String( );
var MyString=new String(string);
```

该方法使用关键字 new 返回一个使用可选参数“string”字符串初始化的 String 对象的实例 MyString，用于后续的字符串操作。

6.1.1 如何使用 String 对象方法操作字符串

使用 String 对象的方法来操作目标对象并不操作对象本身，而只是返回包含操作结果的字符串。例如要设置改变某个字符串的值，必须要定义该字符串等于将对象实施某种操作的结果。考察如下将字符串转换为大写的程序代码：

```
//源程序 6.1
<! 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">
<!--
function StringTest()
{
   var MyString=new String("Welcome to JavaScript world!");
   var msg="原始字符串 MyString:\n"
   msg+="MyString="+MyString+"\n\n";
   MyString.toUpperCase();
   msg+="运行语句:MyString.toUpperCase():\n";
   msg+="MyString="+MyString+"\n\n";
   MyString=MyString.toUpperCase();
   msg+="运行语句:MyString=MyString.toUpperCase():\n";
   msg+="MyString="+MyString+"\n\n";
```

```
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value=调试对象按钮  onclick="StringTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.1 所示。

图 6.1 如何使用 String 对象的方法实例

调用 String 对象的方法语句 MyString.toUpperCase()运行后，并没有改变字符串 MyString 的内容，要使用 String 对象的 toUpperCase( )方法改变字符串 MyString 的内容，必须将使用 toUpperCase( )方法操作字符串的结果返回给原字符串：

```
MyString=MyString.toUpperCase();
```

通过以上语句操作字符串后，字符串的内容才真正被改变。String 对象的其他方法也具有此种特性。

> 注意：String 对象的 toLowerCase( )方法与 toUpperCase( )方法的语法相同、作用类似，不同点在于前者将目标串中所有字符转换为小写状态并返回结果给新的字符串。在表单数据验证时，如果文本域不考虑大小写，可先将其全部字符转换为小写（当然也可大写）状态再进行相关验证操作。

6.1.2 获取目标字符串长度

字符串的长度 length 作为 String 对象的唯一属性，且为只读属性，它返回目标字符串（包含字符串里面的空格）所包含的字符数。改写源程序 6.1 的 StringTest( )函数：

```
function StringTest()
{
   var MyString=new String("Welcome to JavaScript world!");
   var strLength=MyString.length;
   var msg="获取目标字符串的长度:\n\n"
   msg+="访问方法: var strLength=MyString.length\n\n";
   msg+="原始字符串  内容 :"+MyString+"\n";
```

```
  msg+="原始字符串 长度 :"+strLength+"\n\n";
  MyString="This is the New string!";
  strLength=MyString.length;
  msg+="改变内容的字符串 内容 :"+MyString+"\n";
  msg+="改变内容的字符串 长度 :"+strLength+"\n";
  alert(msg);
}
```

程序运行结果如图 6.2 所示。

图 6.2 获取目标字符串的长度

其中脚本语句：

```
strLength=MyString.length;
```

将 MyString 的 length 属性保存在变量 strLength 中，并且其值随着字符串内容的变化自动更新。

6.1.3 连接两个字符串

String 对象的 concat( )方法能将作为参数传入的字符串加入到调用该方法的字符串的末尾并将结果返回给新的字符串，语法如下：

```
newString=targetString.concat(anotherString);
```

改写源程序 6.1 的 StringTest( )函数如下：

```
function StringTest()
{
  //定义两个 String 对象的实例
  var targetString=new String("Welcome to ");
  var strToBeAdded=new String("the world!");
  //调用 String 对象的方法连接字符串
  var finalString=targetString.concat(strToBeAdded);
  //输出返回的新字符串内容
  var msg="\n 连接字符串实例:\n\n";
  msg+="当前目标字符串 :  "+targetString+"\n";
  msg+="被连接的字符串 :  "+strToBeAdded+"\n";
  msg+="连接后的字符串 :  "+finalString+"\n";
  alert(msg);
}
```

程序运行结果如图 6.3 所示。

图 6.3 字符串连接实例

连接字符串的核心语句：

```
var finalString=targetString.concat(strToBeAdded);
```

该代码运行后，将字符串 strToBeAdded 添加到字符串 targetString 的后面，并将生成的新字符串赋值给 finalString，连接过程并不改变字符串 strToBeAdded 和 targetString 的值。

JavaScript 脚本中，也可通过如下的方法实现同样的功能：

```
var finalString="Welcome to "+"the world!";
var finalString="Welcome to ".concat("the world!");
var finalString="Welcome to ".concat("the ","world!");
```

String 对象的 concat( )方法可接受任意数目的参数字符串，并按顺序将它们连接起来添加到调用该方法的字符串后面，并将结果返回给新字符串。

6.1.4 验证邮箱地址合法性

在 Web 应用程序中，经常通过邮箱来进行网站与用户之间的信息交互，如网站通过注册用户的邮箱地址给该用户传递最新资讯。在注册该网站通行证的时候，一般都需提交用户的邮箱信息，此时，必须验证邮箱地址的有效性来保证信息交互的有效进行。

String 对象的 indexOf( )方法返回通过参数传入的字符串出现的开始位置，而邮箱地址必为类似于 username@website.com 的结构，在用户提交的标记为邮箱地址的字符串中，通过 indexOf("@")和 indexOf(".")方法返回值可以判断邮箱地址的有效性。考察如下代码：

```
//源程序 6.2
<! 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">
<!--
function EmailAddressTest()
{
  //获取用户输入的邮箱地址相关信息
  var EmailString=document.MyForm.MyEmail.value;
  var strLength=EmailString.length;
  var index1=EmailString.indexOf("@");
  var index2=EmailString.indexOf(".",index1);
  var msg="验证邮箱地址实例:\n\n";
```

```
  msg+="      邮箱地址 : "+EmailString+"\n";
  msg+="      验证信息 : ";
  //返回相关验证信息
  if(index1==-1||index2==-1||index2<=index1+1||index1==0||index2==strLength-1)
  {
     msg+="邮箱地址不合法!\n\n"
     msg+="不能同时满足如下条件:\n";
     msg+="      1、邮件地址中同时含有'@'和'.'字符；\n";
     msg+="      2、'@'后必须有'.'，且中间至少间隔一个字符；\n"
     msg+="      3、'@'不为第一个字符，'.'不为最后一个字符。\n"
  }
  else
  {
     msg+="邮箱地址合法!\n\n";
     msg+="能同时满足如下条件:\n";
     msg+="      1、邮件地址中同时含有'@'和'.'字符；\n";
     msg+="      2、'@'后必须有'.'，且中间至少间隔一个字符；\n"
     msg+="      3、'@'不为第一个字符，'.'不为最后一个字符。\n"
  }
  alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
   邮箱地址:<input type=text name=MyEmail id=MyEmail value="@">
   <input type=button value=验证邮箱地址  onclick="EmailAddressTest()">
</form>
</center>
</body>
</html>
```

运行上述代码，当文本框中输入 zangpu@gmail.com 等格式合法的邮箱地址时，弹出对话框提示输入的邮箱地址合法，如图 6.4 所示。

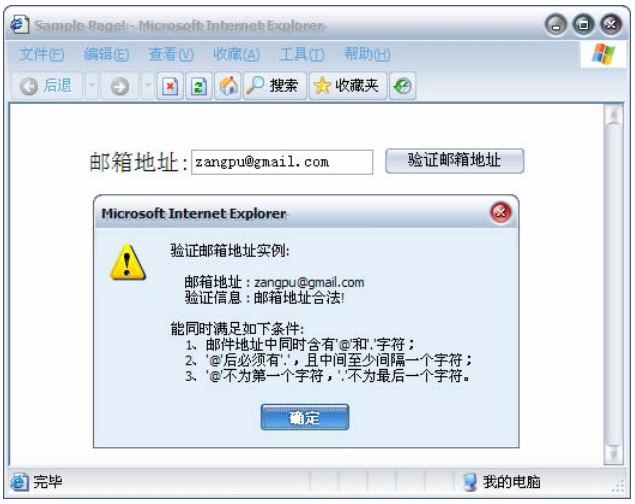

图 6.4 验证邮箱地址：输入 zangpu@gmail.com

当文本框中输入“zangpu@gmail”、“gmail.com”、“gmail.com@ zangpu”、“@gmail.com”、“zangpu@gmail.”、“zangpu”、“zangpu@.com”等格式不合法的邮箱地址时，弹出对话框提示示输入的邮箱地址不合法，如图 6.5 所示。

图 6.5 验证邮箱地址：输入@gmail.com

脚本代码中核心的语句：

```
var index1=EmailString.indexOf("@");
var index2=EmailString.indexOf(".",index1);
```

第一句获取目标字符串中'@'字符（也可为字符串）最先出现的位置并将结果返回 index1 变量，返回-1 表示未搜索到该字符；第二句从 index1 变量（即'@'字符后面开始）指定的位置开始搜索'.'字符最先出现的位置，并将结果返回 index2 变量，返回-1 表示未搜索到该字符，以确保在'@'字符后面存在'.'字符。判断语句：

```
if(index1==-1||index2==-1||index2<=index1+1||index1==0||index2==strLength-1)
```

该句是根据邮箱地址规范设定的条件，如果其中一项不满足，则邮箱地址不合法，反之则合法。

String 对象的 indexOf( )方法有个类似的方法，即 lastIndexOf( )方法，该方法与 indexOf( )方法不同点在于其搜索的顺序是由右向左（由后至前），与 indexOf( )方法正好相反。

注意：一般而言邮件地址的格式如下：somebody@domain_name＋后缀 ，domain_name 为域名标识符，即邮件必须要交付到的邮件目的地的域名；somebody 为该域名对应的服务器上存在的邮箱用户的 id；后缀一般则代表了该域名的性质或地区的代码。例如：.com、.edu.cn、.gov、.org、.tw 等。

6.1.5 返回指定位置的字符串

String 对象提供几种方法用于获取指定位置的字符串，且均在 JavaScript 1.0+、JScript 1.0+中获得支持。如表 6.1 所示：

表 6.1 获取指定位置的字符串

方法	语法	说明
slice( )	slice(num1,num2); slice(num)	以参数num1和num2作为开始和结束索引位置，返回目标字符串中num1和num2之间的子串。当num2为负时，从字符串结束位置向前num2个字符即为结束索引位置；当参数num2大于字符串的长度时，字符串结束索引位置为字符串末尾。若只有参数num，返回从num

		索引位置至字符串结束位置的子串。
substr( )	substr(num1,num2); substr(num)	返回字符串在指定初始位置num1、长度为num2个字符的子串。参数num1为负时，返回字符串起始位置开始、长度为num2个字符的子串；当参数num2大于字符串的长度时，字符串结束位置为字符串的末尾。使用单一参数num时，返回从该参数指定的位置到字符串结尾的字符串。
substring( )	substring(num1,num2); substring(num);	返回字符串在指定的索引位置num1和num2之间的字符。如果num2为负，返回从字符串起始位置开始的num1个字符；如果参数num1为负，将被视为0；如果参数num2大于字符串长度，将被视为string.length。使用单一参数num时返回从该参数指定的位置到字符串结尾的子串。

利用 String 对象的这三个方法，可方便地生成指定的子串，考虑下面的代码：

```
//源程序 6.3
<! 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">
<!--
function StringTest()
{
   var MyString=new String("Congratulations!");
   var msg="返回指定位置字符串实例：                \n\n"
   msg+="原始字符串信息：\n";
   msg+="      内容："+MyString+"\n      长度："+MyString.length+"\n\n";
   msg+="slice()方法:\n";
   msg+="      MyString.slice(2,9) : "+MyString.slice(2,9)+"\n";
   msg+="      MyString.slice(2,-2) : "+MyString.slice(2,-2)+"\n";
   msg+="      MyString.slice(2,19) : "+MyString.slice(2,19)+"\n";
   msg+="      MyString.slice(2) : "+MyString.slice(2)+"\n\n";
   msg+="substr()方法:\n";
   msg+="      MyString.substr(2,9) : "+MyString.substr(2,9)+"\n";
   msg+="      MyString.substr(-2,9) : "+MyString.substr(-2,9)+"\n";
   msg+="      MyString.substr(2,19) : "+MyString.substr(2,19)+"\n";
   msg+="      MyString.substr(2) : "+MyString.substr(2)+"\n\n";
   msg+="substring()方法:\n";
   msg+="      MyString.substring(2,9) : "+MyString.substring(2,9)+"\n";
   msg+="      MyString.substring(2,-2) : "+MyString.substring(2,-3)+"\n";
   msg+="      MyString.substring(-2,9) : "+MyString.substring(-2,9)+"\n";
   msg+="      MyString.substring(2,19) : "+MyString.substring(2,19)+"\n";
   msg+="      MyString.substring(2) : "+MyString.substring(2)+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value=调试对象按钮  onclick="StringTest()">
</form>
</center>
```

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

程序运行结果如图 6.6 所示。

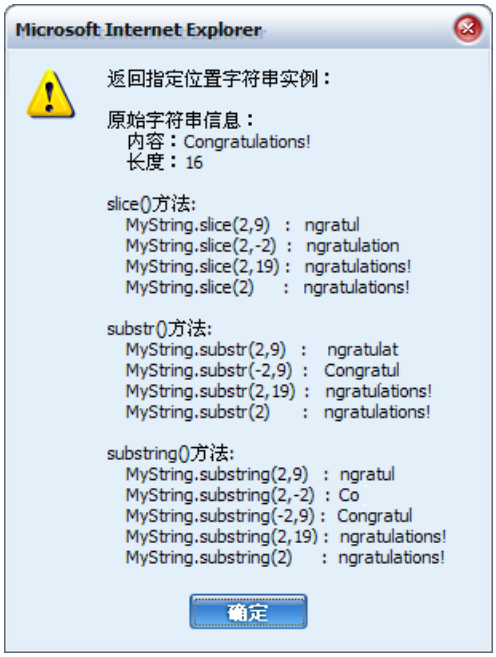

图 6.6 返回指定位置的字符串实例

在 Web 应用程序开发过程中，可以充分结合这三种方法的优缺点，实现更为复杂的字符串选取功能。

String 对象还提供 charAt(num)方法返回字符串中由参数 num 指定位置处的字符，如果 num 不是字符串中的有效索引位置则返回-1；提供 charCodeAt(num)方法返回字符串中由 num 指定位置处字符的 ISO_Latin_1 值，如果 num 不是字符串中的有效索引位置则返回-1。

6.1.6 在 URL 中定位字符串

在 HTML 页面引用、跳转中，经常需要在提交的 URL 中提取感兴趣的内容，如提交表单的用户名等。综合运用以上几个 String 对象关于字符串选取的方法，可在目标字符串中定位到指定的字符串，并能实现指定的字符串多次出现情况下的有效定位。

考察如下代码：

```
//源程序 6.4
<! 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 URLDetect()
{
  //获取文本框内容
```

```
  var MyURL=document.MyForm.MyURL.value;
  var MyStr=document.MyForm.MyStr.value;
  //获取字符串长度
  var URLlength=MyURL.length;
  var Strlength=MyStr.length;
  var msg="";
  //判断文本框是否为空,若空,返回错误信息
  if(URLlength==0||Strlength==0)
  {
     msg+="对不起,你的输入有误，文本框不能为空，但可以为空格!";
     alert(msg);
     return;
  }
  //若不空,执行定位操作
  else
  {
     msg+="在 URL 中定位目标字符串实例：\n\n";
     msg+="原始地址 : " + MyURL + "\n";
     msg+="目标子串 : " + MyStr + "\n\n";
     msg+="定位结果 : \n";
     var index=MyURL.indexOf(MyStr);
     var i=0;
     if(index==-1)
     {
        msg+="目标字符串中没有找到指定的字符串!";
     }
     else
     {
        //搜索到一个位置后,设定标记,然后继续搜索
        while(index!=-1)
        {
           i+=1;
           msg+="位置" +i+ " : " +index+ "\n";
           index=MyURL.indexOf(MyStr, index+1);
        }
     }
  }
  //输出定位信息
  alert(msg);
  return;
}
-->
</script>
</head>
<body>
<form name=MyForm>
  原始地址:
  <input type=text name=MyURL size=60><br>
  搜索子串:
  <input type=text name=MyStr size=60><br><br>
  <center>
     <input type=button value=定位指定字符串  onclick="URLDetect()">
  </center>
```

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

运行上述代码，若目标字符串中存在指定的字符串，返回如图 6.7 所示警告框。

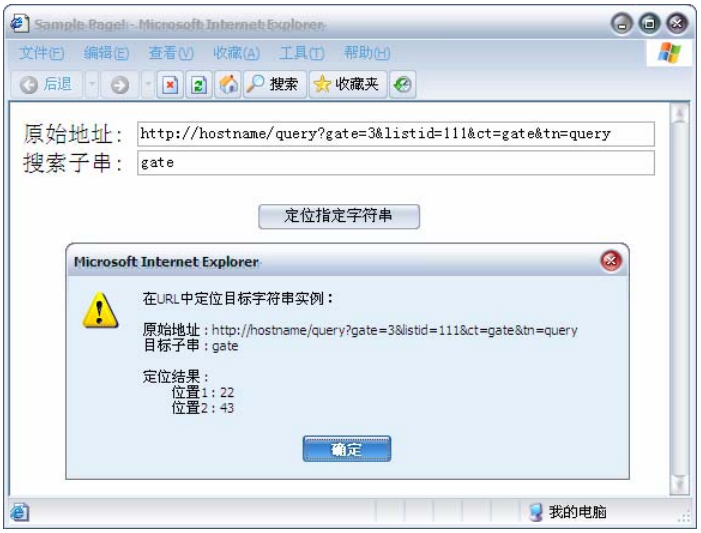

图 6.7 在 URL 中定位字符串：存在指定字符串

若目标字符串中不存在指定的字符串，则返回如图 6.8 所示警告框。

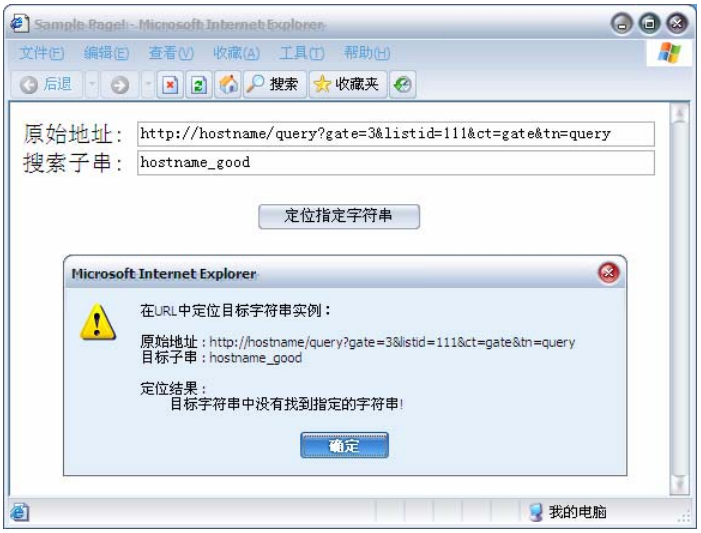

图 6.8 在 URL 中定位字符串：不存在指定字符串

本实例扩展了 String 对象的 indexOf( )方法，将其应用于指定字符串在目标 URL 字符串中存在若干次的情况，同时综合使用了 String 对象的 length 属性。

6.1.7 分隔字符串

String 对象提供 split( )方法来进行字符串的分割操作，split( )方法根据通过参数传入的规则表达式或分隔符来分隔调用此方法的字符串。split( )方法的语法如下：

```
String.split(separator,num);
```

```
String.split(separator);
String.split(regexpression,num);
```

如果传入的是一个规则表达式 regexpression，则该表达式由定义如何匹配的 pattern 和 flags 组成；如果传入的是分隔符 separator，则分隔符是一个字符串或字符，使用它将调用此方法的字符串分隔开，num 表示返回的子串数目，无此参数则默认为返回所有子串。考察如下的代码：

```
//源程序 6.5
<! 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">
<!--
//显示分隔之后形成的字符串数组信息
function getMsg(arrayName,MyStr)
{
   var tempMsg="分隔模式 : "+MyStr+"            \n";
   if(arrayName==null)
   {
      tempMsg+="没有搜索到匹配模式!";
   }
   else
   {
      for(var i=0;i<arrayName.length;i++)
      {
         tempMsg+='['+i+'] : '+arrayName[i]+"\n";
      }
   }
   tempMsg+="\n";
   return tempMsg;
}
//实施分隔操作
function MySplit()
{
   var MyString=new String("Mr. R. Allen Wyke");
   var MyRegExp=/\s/g;
   var MySeparator=" ";
   var msg="分隔字符串实例:\n\n";
   msg+="原始字符串 : "+MyString+"\n";
   msg+="分隔符 : "+MySeparator+"(空格)\n";
   msg+="正则表达式 : "+MyRegExp+"\n\n";
   msg+=getMsg(MyString.split(MySeparator),"MyString.split(MySeparator)");
   msg+=getMsg(MyString.split(MySeparator,2),"MyString.split(MySeparator,2)");
   msg+=getMsg(MyString.split(MyRegExp),"MyString.split(MyRegExp)");
   msg+=getMsg(MyString.split(MyRegExp,3),"MyString.split(MyRegExp,3)");
   alert(msg);
}
-->
</script>
</head>
```

```
<body>
<center>
<form name=MyForm>
   <input type=button value=分隔指定字符串  onclick="MySplit()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.9 所示。

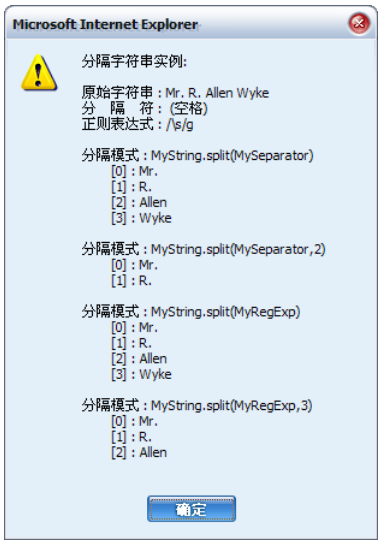

图 6.9 分隔字符串实例

核心语句：

```
MyString.split(MySeparator);
MyString.split(MySeparator,2);
MyString.split(MyRegExp);
MyString.split(MyRegExp,3);
```

代码运行后，按照 String 对象的 split( )方法，将分隔而成的子串形成的字符串数组的相关信息通过函数 getMsg(arrayName,MyStr)返回 msg 变量输出。

正则表达式 RegExp 对象的相关知识在“正则表达式”章节将详细讲述。

6.1.8 将字符串标记为 HTML 语句

客户端 JavaScript 脚本主要用于处理 HTML 文档中各元素对象，同时提供大量的方法将字符串转化为 HTML 语句，这些方法返回使用了 HTML 标记对的字符串。如下面的代码：

```
var MyString="Welcome to JavaScript world!".big();
document.write(MyString);
```

脚本运行后，相当于下面的 HTML 语句：

```
<big> Welcome to JavaScript world!</big>
```

在 HTML 文档中，标记之间可以相互嵌套，JavaScript 脚本也可通过链式方法调用实现该效果，如 HTML 语句：

```
<a url="parent.html"><big> <strike>Welcome to JavaScript world! </strike></big></a>
```

使用 JavaScript 脚本实现的代码如下：

```
var MyStr="Welcome to JavaScript world!".strike().big().link('parent.html');
```

其中调用的先后顺序对应于 HTML 语句由内向外的顺序。考察如下的代码：

```
//源程序 6.6
<! 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>
</head>
<body>
<script language="JavaScript" type="text/javascript">
<!--
var MyString=new String("How Are You?");
document.write("                    原始字符串："+MyString+"<br><hr>");
document.write("                  anchor()方法："+MyString.anchor("New")+"<br>");
document.write("                     big()方法："+MyString.big()+"<br>");
document.write("                   small()方法："+MyString.small()+"<br>");
document.write("                    bold()方法："+MyString.bold()+"<br>");
document.write("        fontcolor('blue')方法："+MyString.fontcolor('blue')+"<br>");
document.write("      fontcolor('ff0000')方法："+MyString.fontcolor('ff0000')+"<br>");
document.write("              fontsize(5)方法："+MyString.fontsize(5)+"<br>");
document.write("             fontsize('-2')方法："+MyString.fontsize('-2')+"<br>");
document.write("                  italics()方法："+MyString.italics()+"<br>");
document.write("       link('parent.html')方法："+MyString.link('parent.html')+"<br>");
document.write("                    strike()方法："+MyString.strike()+"<br>");
document.write("                     sub()方法："+MyString.sub()+"<br>");
document.write("                     sup()方法："+MyString.sup()+"<br>");
-->
</script>
</body>
</html>
```

程序运行的结果如图 6.10 所示。

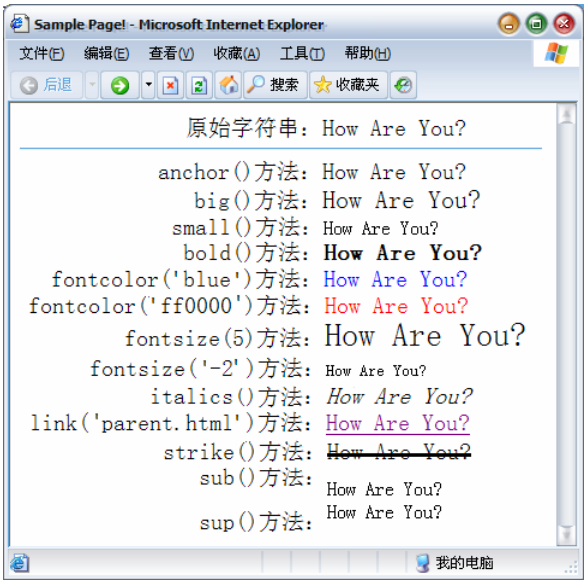

图 6.10 将字符串标记为 HTML 语句实例

在 JavaScript 脚本生成 HTML 语句诸多方法中，有些方法如 fontcolor()方法等会生成包含有 HTML 4 标准不赞成使用或已被淘汰标记如<blink>等的字符串，在 XHTML 中样式表（CSS）将逐步取代用 JavaScript 脚本生成 HTML 语句的方法。

6.1.9 常见属性和方法汇总

JavaScript 脚本的核心对象 String 提供大量的属性和方法来操作字符串。表 6.2 列出了其常用的属性、方法以及脚本版本支持情况。

表 6.2 String对象常用的属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	length	返回目标字符串的长度。详情请见6.1.2应用实例	①②
	prototype	用于给String对象增加属性和方法。	③⑥
方法	anchor(name)	创建<a>标签，并用参数nane设置其NAME属性	①②
	big( ) 、blink( )、 bold( )、fixed( )、italics()、small()、strike()、sub()、sup()	对应于方法，分别创建HTML语句中<big>、<blink>、<bold>、<tt>、<i>、<small>、<strike>、<sub>、<sup>等标签。详情请见6.1.8应用实例	①②
	link(URL)	创建<a>标签，并用参数URL指定其HREF属性。	①②
	charAt(num)	用于返回参数num指定索引位置的字符。如果参数num不是字符串中的有效索引位置则返回-1。	①②
	charCodeAt(num)	与charAt( )方法相同，但其返回ISO_Latin_1值。	①②
	concat(string2)	把参数string2传入的字符串连接到当前字符串的末尾并返回新的字符串。详见6.1.3应用实例	④⑥
	fontcolor(hexnum) fontcolor(color)	创建<font>标签并设置其color属性。详情请见6.1.8应用实例	①②
	fontsize(num) fontsize(string2)	创建<font>标签并设置size属性为数字num或由字符串string2表示的相对于<basefont>标签的增减值i。详情请见6.1.8应用实例	①②
	fromCharCode(num1,…numN) fromCharCode(keyevent.which)	返回对应于通过参数num1至numN传入的ISO_Latin_1值位置处的字符，或者传入一个键盘事件并由其which属性指定哪个键被按下。	④⑥
	indexOf(string,num) indexOf(string)	返回通过字符串传入的字符串传入的字符串string出现的位置，详情请见6.1.6应用实例。	①②
	LastIndexOf()	参数与indexOf相同，功能相似，索引方向相反。	①②
	match(regexpression)	查找目标字符串中通过参数传入的规则表达式regexpression所指定的字符串。	④⑥
	replace(regExpression,strin2)	查找目标字符串中通过参数传入的规则表达式指定的字符串，若找到匹配字符串，返回由参数字符串string2替换匹配字符串后的新字符串。	④⑥
	search(regexpression)	查找目标字符串中通过参数传入的规则表达式指定的字符串，找到配对时返回字符串的索引位置否则返回-1。	④⑥
	slice(num1,num2) slice(num)	返回目标字符串指定位置的字符串。详情请见6.1.5应用实例	①②
	split(separetor,um) split(separetor) split(regexpression,num)	根据参数传入的规则表达式regexpression或分隔符separetor来分隔目标字符串，并返回字符串数组。详情请见6.1.7应用实例	③②
	substr(num1,num2) substr(num)	返回目标字符串中指定位置的字符串，详情请见6.1.5应用实例	①②
	substring(num1,num2) substring(num)	返回目标字符串中指定位置的字符串，详情请见6.1.5应用实例	①②
	toLowerCases()	将字符串的全部字符转化为小写。	①②
	toUpperCase()	将字符串的全部字符转化为大写。	①②
	valueOf()	返回String对象的原始值。	④⑥

注意：在以上版本支持情况中，①指 JavaScript 1.0+； ②指 JScript 1.0+； ③指 JavaScript 1.1+； ④指 JavaScript 1.2+； ⑤指 JavaScript 1.3 +；⑥指 JScript 3.0+。此项设定下同。

在 JavaScript 脚本程序编写过程中，String 对象是最为常见的处理目标，用于存储较短的数据。JavaScript 语言提供了丰富的属性和方法支持，方便 Web 应用程序开发者灵活地操纵 String 对象的实例。

6.2 Math 对象

Math 对象是 JavaScript 核心对象之一，拥有一系列的属性和方法，能够进行比基本算术运算更为复杂的运算。但 Math 对象所有的属性和方法都是静态的，并不能生成对象的实例，但能直接访问它的属性和方法。例如可直接访问 Math 对象的 PI 属性和 abs(num)方法：

```
var MyPI=Math.PI;
var MyAbs=Math.abs(-5);
```

需要注意的是，JavaScript 脚本中浮点运算精确度不高，常导致计算结果产生微小误差从而导致最终结果的致命错误。例如：

```
alert(Math.sin(Math.PI));
```

代码运行结果如图 6.11 所示。

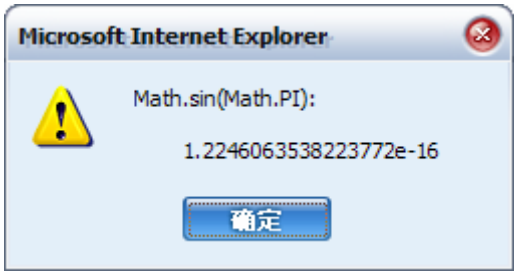

图 6.11 调用 Math.sin(Math.PI)方法的返回结果

可见，JavaScript 脚本中 Math.sin(Math.PI)返回的结果与理论上的 0 非常接近，但微小的误差足以导致精确计算的失败。

6.2.1 基本数学运算

Math 对象提供丰富的方法用于数学运算，特别是三角函数方面的方法。由于三角函数的参数使用弧度制，要在参数上乘以 $\pi/180$。考察如下的代码：

```
//源程序 6.7
<! 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">
<!--
function MyTest()
```

```
{
   var msg="Math 对象基本数学运算实例:                    \n\n";
   msg+="绝对值  Math.abs(-1)="+Math.abs(-1)+"\n";
   msg+="平方根  Math.sqrt(9)="+Math.sqrt(9)+"\n";
   msg+="反余弦  Math.acos(-1)="+Math.acos(-1)+"\n";
   msg+="反正弦  Math.asin(-1)="+Math.asin(-1)+"\n";
   msg+="反正切  Math.atan(-1)="+Math.atan(-1)+"\n";
   msg+="反正切(方位角) Math.atan2(1,2)="+Math.atan2(1,2)+"\n";
   msg+="正弦值  Math.sin(45*Math.PI/180)="+Math.sin(45*Math.PI/180)+"\n";
   msg+="余弦值  Math.cos(45*Math.PI/180)="+Math.cos(45*Math.PI/180)+"\n";
   msg+="正切值  Math.tan(45*Math.PI/180)="+Math.tan(45*Math.PI/180)+"\n";
   msg+="大于一个数的最小整数  Math.ceil(1.2)="+Math.ceil(1.2)+"\n";
   msg+="小于一个数的最小整数  Math.floor(1.2)="+Math.floor(1.2)+"\n";
   msg+="欧拉常数的次幂  Math.exp(2)="+Math.exp(2)+"\n";
   msg+="数的自然对数  Math.log(3)="+Math.log(3)+"\n";
   msg+="两数中的最大值  Math.max(1,2)="+Math.max(1,2)+"\n";
   msg+="两数中的最小值  Math.min(1,2)="+Math.min(1,2)+"\n";
   msg+="数的次方  Math.pow(3,4)="+Math.pow(3,4)+"\n";
   msg+="最接近的整数  Math.round(2.1)="+Math.round(2.1)+"\n";
   msg+="最接近的整数  Math.round(2.6)="+Math.round(2.6)+"\n";
   msg+="最接近的整数  Math.round(2.1)="+Math.round(2.1)+"\n";
   msg+="将数值转换为字符串  Math.toString(123456)="+Math.toString(123456)+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
   <input type=button value=数学运算  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.12 所示。

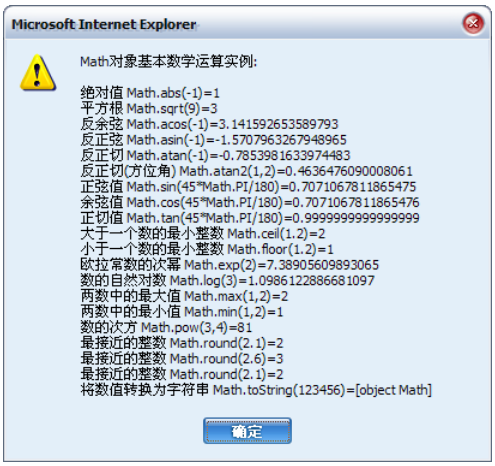

图 6.12 Math 对象的方法中基本数学运算

可见，Math 对象提供很多的数学方法用于基本运算，这些基本能满足 Web 应用程序的要求，但在实际应用中要充分考虑程序的精度要求，并在满足精度的情况下对获得的数据进行适当的截尾。

6.2.2 任意范围随机数发生器

在 JavaScript 脚本中，可使用 Math 对象的 random()方法生成 0 到 1 之间的随机数，考察下面任意范围的随机数发生器代码：

```
//源程序 6.8
<! 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">
<!--
function MyTest()
{
   var m=document.MyForm.MyM.value;
   var n=document.MyForm.MyN.value;
   var msg="m 到 n 之间的随机数产生实例:\n\n";
   msg+="随机数范围设定:\n";
   msg+="下限:"+m+"\n";
   msg+="上限:"+n+"\n\n"
   if(m==n)
   {
       msg+="错误提示信息:\n"
       msg+="上限与下限相等,请返回重新输入!";
   }
   else
   {
      msg+="随机数产生结果:\n"
      for(var i=0;i<10;i++)
      {
         //产生 0-1 之间随机数，并通过系数变换到 m-n 之间
         msg+="第  "+(i+1)+"  个: "+(Math.random()*(n-m)+m)+"\n";
      }
   }
   alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
   随机数产生范围下限 : <input type=text name=MyM size=30 value=1><br>
   随机数产生范围上限 : <input type=text name=MyN size=30 value=10><br><br>
   <input type=button value=数学运算  onclick="MyTest()">
```

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

程序运行结果如图 6.13 所示。

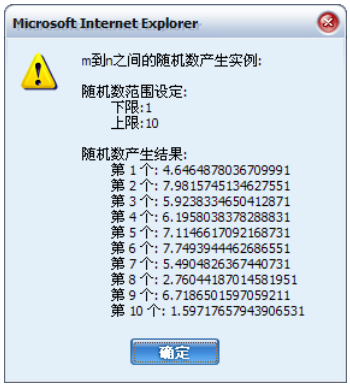

图 6.13 任意范围的随机数发生器

程序中关键代码：

```
Math.random()*(n-m)+m;
```

首先产生 0 和 1 之间随机数，然后通过系数变换，将其限定在 m 和 n(n>m)之间的随机数，并可通过更改文本框内容的形式，产生任意范围的随机数。

6.2.3 访问其基本属性

Math 对象拥有很多基本属性，如圆周率 Math.PI、Math.SQRT2、Math.log10E 等，表示数学运算中经常使用的常量。考察下面的代码：

```
//源程序 6.9
<! 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">
<!--
function MyTest()
{
   var msg="";
   msg+="Math 对象基本属性: \n\n";
   msg+="欧拉常数 Math.E = "+Math.E+"\n\n";
   msg+="圆周率 Math.PI = "+Math.PI+"\n\n";
   msg+="10 的自然对数 Math.LN10 = "+Math.LN10+"\n\n";
   msg+="2 的自然对数 Math.LN2 = "+Math.LN2+"\n\n";
   msg+="E 为底 10 的对数 Math.LOG10E = "+Math.LOG10E+"\n\n";
   msg+="E 为底 2 的对数 Math.LOG2E = "+Math.LOG2E+"\n\n";
   msg+="0.5 的平方根 Math.SQRT1_2 = "+Math.SQRT1_2+"\n\n";
   msg+="2 的平方根 Math.SQRT2 = "+Math.SQRT2+"\n";
```

```
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form name=MyForm>
   <input type=button value=数学运算  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.14 所示。

图 6.14 Math 对象的基本属性

6.2.4 使用 with 声明简化表达式

在 Math 对象进行数学计算时，常涉及到它的很多种属性或方法，如果每次都使用 Math.abs()、Math.PI 方式调用的话，代码复杂度高。可以使用 with 申明来简化调用的代码。考察如下 JavaScript 脚本：

```
//源程序 6.10
<! 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">
<!--
function MyTest()
{
   var msg="使用 with 声明简化表达式实例:\n\n";
   //使用 with 声明简化其表达式
   with(Math)
   {
       var a=document.MyForm.Mya.value;
       var b=document.MyForm.Myb.value;
```

```
        var AngleC=document.MyForm.MyC.value;
        var AngleA=atan(a/b);
        var AngleB=atan(b/a);
        var powerC=pow(a,2)+pow(b,2)+2*a*b*cos(AngleC*PI/180);
        msg+="运用余弦定理计算:\n";
        msg+="角 A="+AngleA+"\n";
        msg+="角 B="+AngleB+"\n";
        msg+="边 c="+sqrt(powerC)+"\n";
    }
    alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
    边 a<input type=text name=Mya value=3><br>
    边 b<input type=text name=Myb value=4><br>
    角 C<input type=text name=MyC value=90><br>
    <input type=button value=简化数学运算  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.15 所示。

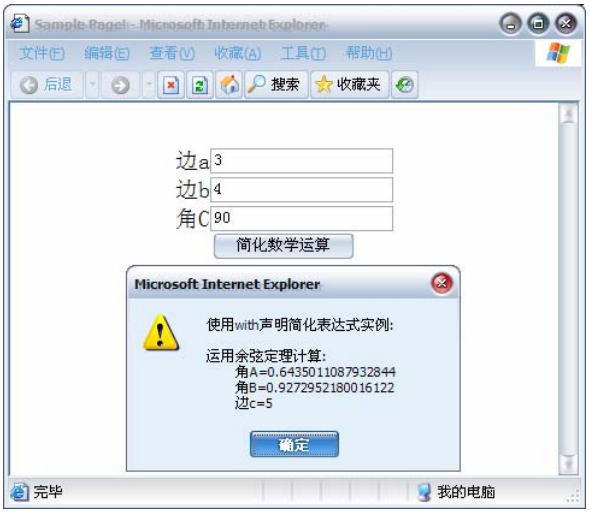

图 6.15 使用 with 声明简化表达式

在 JavaScript 脚本代码中声明 with 后，在 with 作用范围内的 Math 对象的属性和方法都可以直接使用，而不需要使用 Math 对象引用的方式调用。

6.2.5 常见属性汇总

Math 对象的常见属性汇总情况见表 6.3 所示：

表 6.3 Math对象的常见属性列表

属性	说明	脚本版本支持	浏览器版本支持
Math.E	返回欧拉常数e的值	①②	①⑤⑦
Math.LN2	返回2的自然对数	①②	①⑤⑦
Math.LN10	返回10的自然对数	①②	①⑤⑦
Math.LOG2E	返回e为底2的对数	①②	①⑤⑦
Math.LOG10E	返回e为底10的对数	①②	①⑤⑦
Math.PI	返回圆周率PI的值	①②	①⑤⑦
Math.SQRT1_2	返回0.5的平方根	①②	①⑤⑦
Math.SQRT2	返回2的平方根	①②	①⑤⑦

注意：在浏览器版本支持中，①指 Navgator 2+； ②指 Navgator 3+； ③指 Navgator 4+； ④指 Navgator 4.06+； ⑤指 Internet Explorer 3 +；⑥指 Internet Explorer 4 +；⑦指 Opera 3+。此项设定下同。

6.2.6 常见方法汇总

Math 对象的常见方法汇总情况见表 6.4 所示：

表 6.4 Math对象的常见方法列表

方法	说明	脚本版本支持	浏览器支持
Math.abs(num)	返回num的绝对值	①②	①⑤⑦
Math.acos(num)	返回num的反余弦	①②	①⑤⑦
Math.asin(num)	返回num的反正弦	①②	①⑤⑦
Math.atan(num)	返回num的反正切	①②①	①⑤⑦
Math.atan2(num1, num2)	返回一个除法表示式的反正切值	①⑥	①⑥
Math.ceil(num)	返回大于等于一个数的最小整数	①②①	①⑤⑦
Math.cos(num)	返回num的余弦值	①②	①⑤⑦
Math.exp(num)	返回底为欧拉常数e的num次方	①②	①⑤⑦
Math.floor(num)	返回小于等于一个数的最大整数	①②	①⑤⑦
Math.log(num)	返回以欧拉常数e为底num的自然对数	①②	①⑤⑦
Math.max(num1, num2)	返回num1和num2中较大的一个数	①②	①⑤⑦
Math.min(num1, num2)	返回num1和num2中较大的一个数	①②	①⑤⑦
Math.pow(num1, num2)	返回num1的num2次方	①②	①⑤⑦
Math.random()	返回0至1间的随机数	③②	①⑤⑦
Math.round(num)	返回最接近num的整数	①②	①⑤⑦
Math.sin(num)	返回num的正弦值	①②	①⑤⑦
Math.sqrt(num)	返回num的平方根	①②	①⑤⑦
Math.tan(num)	返回num的正切值	①②	①⑤⑦
Math.toSource(object)	返回Math对象object的拷贝	⑤	④
Math.toString()	返回表示Math对象的字符串	①⑥	①⑥

Math 对象提供大量的属性和方法实现 JavaScript 脚本中的数学运算，但由于其为静态对象，不能创建对象的实例，更不能动态添加属性和方法，导致其使用范围较窄。下面介绍功能完善，且扩展方便的 Array 对象。

6.3 Array 对象

数组是包含基本和组合数据类型的有序序列，在 JavaScript 脚本语言中实际指 Array 对象。数组可用构造函数 Array( )产生，主要有三种构造方法：

```
Var MyArray=new Array();
var MyArray =new Array(4);
var MyArray =new Array(arg1,arg2,...,argN);
```

第一句声明一个空数组并将其存放在以 MyArray 命名的空间里，可用数组对象的方法动态添加数组元素；第二句声明长度为 4 的空数组，JavaScript 脚本中支持最大的数组长度为 4294967295；第三句声明一个长度为 N 的数组，并用参数 arg1、arg2、...、argN 直接初始化数组元素，该方法在实际应用中最为广泛。

在 JavaScript 脚本版本更新过程中，渐渐支持使用数组字面值来声明数组的方法。与上面构造方法相对应，出现了如下的数组构造形式：

```
var MyArray=[];
var MyArray =[,,,];
var MyArray =[arg1,arg2,...,argN];
```

该构造方法在 JavaScript 1.2+版本中首先获得支持。其中第二种方式中表明数组长度为 4，并且数组元素未被指定，浏览器解释时，把其看成拥有 4 个未指定初始值的元素的数组。将其扩展如下：

```
var MyArray =[234,,24,,56,,,,3,4];
```

该数组构造方式构造一个长度为 10、某些位置指定初始值、其他位置未指定初始值的数组 MyArray，MyArray 又被称为稀疏数组，可通过给指定位置赋值的方法来修改该数组。

Array 对象提供较多的属性和方法来访问、操作目标 Array 对象实例，如增加、修改数组元素等。

6.3.1 创建数组并访问其特定位置元素

JavaScript 脚本中，使用 new 操作符来创建新数组，并可通过数组元素的下标实现对任意元素的访问。考察如下代码：

```
//源程序 6.11
<! 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">
<!--
function MyTest()
{
   var MyArray=new Array("TOM","Allen","Lily","Jack");
   var strLength=MyArray.length;
   var msg="创建数组并通过下标访问实例:\n\n";
   msg+="创建目标数组语法:\n"
   msg+="new Array('TOM','Allen','Lily','Jack');\n\n";
   msg+="目标数组信息:\n";
   msg+="MyArray.length = "+ strLength +"\n";
```

```
    for(var i=0;i<strLength;i++)
    {
       msg+="MyArray["+i+"] = "+MyArray[i]+"\n";
    }
    msg+="MyArray[5] = "+MyArray[5]+"\n";
    alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
    <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.16 所示。

图 6.16 创建数组并通过下标访问其元素实例

数组元素下标从 0 开始顺序递增，可通过数组元素的下标实现对它的访问：

```
var data=MyArray[i];
```

但访问数组中未被定义的元素时将返回未定义的值，如下列代码：

```
var data=MyArray[5];
```

运行后，data 返回 undefined。同样，使用稀疏数组时，访问未被定义的元素也将返回未定义的值 undefined。

6.3.2 数组中元素的顺序问题

Array 对象提供相关相关方法实现数组中元素的顺序操作，如颠倒元素顺序、按 Web 应用程序开发者制定的规则进行排列等，主要有 Array 对象的 reverse()和 sort()方法。

reverse()方法将按照数组的索引号的顺序将数组中元素完全颠倒，语法如下：

```
arrayName.reverse();
```

sort()方法较之 reverse()方法复杂，它基于某种顺序重新排列数组的元素，语法如下：

```
arrayName.sort();
arrayName.sort(function);
```

第一种调用方式不指定排列顺序，JavaScript 脚本将数组元素转化为字符串，然后按照

字母顺序进行排序。

第二种调用方式由参数 function 指定排序算法，该算法需遵循如下的规则：

- 算法必须接受两个可以比较的参数 a 和 b，即 function(a,b)；
- 算法必须返回一个值以表示两个参数之间的关系；
- 若参数 a 在参数 b 之前出现，函数返回小于零的值；
- 若参数 a 在参数 b 之后出现，函数返回大于零的值；
- 若参数 a 等于 b，则返回零。

考察如下的代码：

```
//源程序 6.12
<! 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">
<!--
//输出数组信息
function getMsg(arrayName)
{
   var arrayLength=arrayName.length;
   var tempMsg="";
   for(var i=0;i<arrayLength;i++)
   {
      tempMsg+="            MyArray["+i+"]="+arrayName[i]+"\n";
   }
   return tempMsg;
}
//根据排序算法规则构造排序算法 MyFunction(arg1,arg2)
function MyFunction(arg1,arg2)
{
   var dataReturn;
   if(arg1.length<arg2.length)
      dataReturn=-1;
   else if(arg1.length>arg2.length)
      dataReturn=1;
   else   dataReturn=0;
   return dataReturn;
}
function MyTest()
{
   var MyArray=new Array("First","Second","Third","Forth");
   var msg="数组元素的顺序操作实例:\n\n";
   msg+="原始数组:\n"+getMsg(MyArray)+"\n";
   msg+="前后颠倒:\n"+getMsg(MyArray.reverse())+"\n";
   msg+="字母顺序:\n"+getMsg(MyArray.sort())+"\n";
   msg+="排序算法:\n"+getMsg(MyArray.sort(MyFunction))+"\n";
   alert(msg);
}
-->
</script>
</head>
```

```
<body>
<br>
<center>
<form name=MyForm>
   <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.17 所示。

图 6.17 数组元素排序实例

在上述程序中，构造了排序算法 MyFunction(arg1,arg2)，并根据其返回值决定数组中各元素之间的相对顺序。

6.3.3 模拟堆栈和队列操作的方法

为实现数组的动态操作，从 JavaScript 1.2+和 JScript 5.5+开始，Array 对象提供了诸如 pop()、push()、unshift()、shift()等方法来动态添加和删除数组元素。先来了解两个抽象的数据类型：

- 堆栈（LIFO）：用于以“后进先出”的顺序存储数据的结构。在读取堆栈的时候，最后存入的数据最先被读取出来；
- 队列（FIFO）：用于以“先进先出”的顺序储存数据的结构。在读取队列的时候，最先存入的数据最先被读取出来。

其中 pop()方法模拟堆栈的“压栈”动作，将数组中最后一个元素删除，并将该元素作为操作的结果返回，同时更新数组的 length 属性；push()方法模拟堆栈的“出栈”动作，将以参数传入的元素按参数顺序添加到数组的尾部，并将插入的元素作为操作的结果返回，同时更新数组的 length 属性。

unshift()、shift()方法与 pop()、push()方法相对，都是删除和添加数组元素，仅仅是删除和添加目标的位置不同，前者与后者相反方向，即从数组的第一个元素开始操作，后面的元素将分别向前和向后移动，数组的 length 属性自动更新。

考察如下的代码：

```
//源程序 6.13
<! 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">
<!--
//输出数组信息
function getMsg(arrayName)
{
   var arrayLength=arrayName.length;
   var tempMsg="              长度 : "+arrayLength+"\n";
   for(var i=0;i<arrayLength;i++)
   {
      tempMsg+="              MyArray["+i+"]="+arrayName[i];
   }
   return tempMsg;
}
function MyTest()
{
   var MyArray=new Array("First","Second","Third","Forth");
   var msg="添加和删除数组元素实例:\n\n";
   msg+="原始数组:\n"+getMsg(MyArray)+"\n\n";
   var arrayPop=MyArray.pop();
   msg+="使用 pop()方法:\n"+getMsg(MyArray)+"\n";
   var arrayPush=MyArray.push("Forth");
   msg+="使用 push("Forth")方法:\n"+getMsg(MyArray)+"\n";
   var arrayShift=MyArray.shift();
   msg+="使用 shift()方法:\n"+getMsg(MyArray)+"\n";
   var arrayUnshift=MyArray.unshift("First");
   msg+="使用 unshift("First")方法:\n"+getMsg(MyArray)+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
   <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.18 所示。

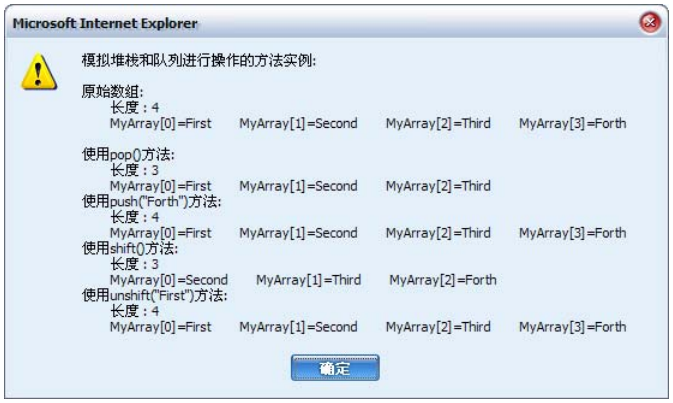

图 6.18 模拟堆栈和队列进行操作的方法实例

Array 对象提供的模拟堆栈和队列进行数组元素添加、删除的方法，使用非常简单。它致命的缺陷就是只能操作数组的头部或末端的数组元素，不能进行任意位置数组元素的添加和删除操作，下面介绍可以在数组中任意位置执行该操作的 splice()方法。

> 注意：Array 对象的 pop()、push()、unshift()、shift()等添加和删除数组元素的方法，虽模拟了堆栈和队列的基本动作，但并不能因此而将数组看成堆栈或队列。

6.3.4 使用 splice()方法添加和删除数组元素

Array 对象的 splice()方法提供一种在数组任意位置添加、删除数组元素的方法。语法如下：

```
MyArray.splice(start,delete,arg3,…,argN);
```

参数说明如下：

- 当参数 delete 为 0 时，不执行任何删除操作；
- 当参数 delete 非 0 时，在调用此方法的数组中删除下标从 start 到 shart+delete 的数组元素，其后的数组元素的下标均减小 delete；
- 如果在参数 delete 之后还有参数，在执行删除操作之后，这些参数将作为新元素添加到数组中由 start 指定的开始位置，原数组该位置之后的元素往后顺移。

考察如下的代码：

```
//源程序 6.14
<! 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">
<!--
//返回数组信息
function getMsg(arrayName)
{
   var arrayLength=arrayName.length;
   var tempMsg="      长度 : "+arrayLength+"\n";
   for(var i=0;i<arrayLength;i++)
   {
```

```
    tempMsg+="        MyArray["+i+"]="+arrayName[i];
  }
  tempMsg+="          ";
  return tempMsg;
}
//执行相关操作
function MyTest()
{
  var MyArray=new Array("First","Second","Third","Forth");
  var msg="添加和删除数组元素实例:\n\n";
  msg+="原始数组:\n"+getMsg(MyArray)+"\n\n";
  MyArray.splice(1,0);
  msg+="使用 splice(1,0)方法:\n"+getMsg(MyArray)+"\n";
  MyArray.splice(1,1);
  msg+="使用 splice(1,1)方法:\n"+getMsg(MyArray)+"\n";
  MyArray.splice(1,1,"New1","New2");
  msg+="使用 splice(1,1,"New1","New2")方法:\n"+getMsg(MyArray)+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
  <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.19 所示。

图 6.19 使用 splice()方法增加和删除数组元素实例

其中核心语句：

```
MyArray.splice(1,0);
MyArray.splice(1,1);
MyArray.splice(1,1,"New1","New2");
```

第一句中参数 delete 为 0，不执行任何操作，MyArray 数组保持不变：

```
MyArray=["First","Second","Third","Forth"];
```

第二句参数 delete 不为 0（=1），执行删除下标为 start（=1）到 start+delete（=2）之间的数组元素，即 MyString[1]=Second，其后的数组元素往前挪动 delete（=1）位，此时 MyArray 数组变为：

```
MyArray=["First","Third","Forth"];
```

第三句在继续执行一次第二句的删除操作（删除 MyString[1]=Third）基础上，将以参数传入的“New1”和“New2”元素作为数组元素插入到 start（=1）指定的位置，原位置上的数组元素顺移，相当于执行两个步骤：

```
MyArray=["First","Forth"];
MyArray=["First","New1","New2","Forth"];
```

注意：Array 对象的 splice()方法在 Navigator 4 中存在一个缺陷，当删除数组中指定的元素时，返回的不是删除了指定元素的数组而是该指定的元素，同时，如果数组中没有元素被删除，返回的不是空数组而是 null。

6.3.5 修改 length 属性更改数组

Array 对象的 length 属性保存目标数组的长度：

```
var strLength=MyArray.length;
```

Array 对象的 length 属性检索的是数组末尾的下一个可及（未被填充）的位置的索引值，即使前面有些索引没被使用，length 属性也返回最后一个元素后面第一个可及位置的索引值。考察下面代码：

```
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
   var MyArray=new Array();
   MyArray[10]="Welcome!";
   var arrayLength=MyArray.length;
   var msg="数组的 length 属性实例:\n\n";
   msg+=" MyArray.length = "+arrayLength +"\n";
   alert(msg);
}
-->
</script>
```

程序运行结果如图 6.20 所示。

图 6.20　Array 对象的 length 属性实例

同时，当脚本动态添加、删除数组元素时，数组的 length 属性会自动更新。在循环访问数组元素的过程中，应十分注意控制循环的变量的变化情况。

length 属性可读可写，在 JavaScript 脚本中可通过修改数组的 length 属性来更改数组的

内容，如通过减小数组的 length 属性，改变数组所含的元素，即凡是下标在新 length-1 后的数组元素将被删除。考察如下代码：

```
//源程序 6.15
<! 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">
<!--
//返回数组信息
function getMsg(arrayName)
{
   var arrayLength=arrayName.length;
   var tempMsg="       长度 : "+arrayLength+"\n";
   for(var i=0;i<arrayLength;i++)
   {
      tempMsg+="        MyArray["+i+"]="+arrayName[i];
   }
   tempMsg+="                    ";
   return tempMsg;
}
//执行相关操作
function MyTest()
{
   var MyArray=new Array("First","Second","Third","Forth");
   var msg="修改 length 属性更改数组:\n\n";
   msg+="原始数组:\n"+getMsg(MyArray)+"\n\n";
   MyArray.length=3;
   msg+="使用 MyArray.length=3 语句 : \n"+getMsg(MyArray)+"\n";
   MyArray.length=4;
   msg+="使用 MyArray.length=4 语句 : \n"+getMsg(MyArray)+"\n";
   MyArray[3]="Fifth";
   msg+="使用 MyArray[3]="Fifth"语句直接赋值 : \n"+getMsg(MyArray)+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
   <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.21 所示。

图 6.21 修改 length 属性更改数组实例

在使用 MyArray.length=3 语句后，数组长度变为 3，直接删除数组元素 MyArray[3]；在使用 MyArray.length=4 语句后，数组长度变变为 4，在数组末端添加元素 MyArray[3]，且为未定义类型；在使用 MyArray[3]="Fifth"语句直接给 MyArray[3]赋值“Fifth”后，数组：

MyArray=["First","Second","Third","Fifth"];

MyArray.length=4;

更改 Array 对象的 length 属性后，任何包含数据的索引只要大于 length-1，将立即被设定为未定义类型。

6.3.6 调用 Array 对象的方法生成字符串

在 Web 应用程序开发过程中，常常需要将数组元素按某种形式转化为字符串，如需将存放用户名的数组中各个元素转换为字符串并赋值给各用户等。先考察如下代码：

```
//源程序 6.16
<! 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">
<!--
//返回数组信息
function getMsg(arrayName)
{
   var arrayLength=arrayName.length;
   var tempMsg="";
   for(var i=0;i<arrayLength;i++)
   {
      tempMsg+="              MyArray["+i+"]="+arrayName[i]+"\n";
   }
   return tempMsg;
}
//执行相关操作
function MyTest()
{
   var MyArray=new Array("First","Second","Third","Forth");
   var msg="调用 Array 对象的方法生成字符串实例: \n\n";
```

```
    msg+="原始数组 : \n"+getMsg(MyArray)+"\n";
    var tempStr="";
    tempStr=MyArray.join();
    msg+="1、调用 MyArray.join()方法返回字符串 : \n"+tempStr+"\n\n";
    tempStr=MyArray.join("-");
    msg+="2、调用 MyArray.join("-")方法返回字符串 : \n"+tempStr+"\n\n";
    msg+="3、调用 MyArray.toString()方法返回字符串 : \n"+MyArray+"\n";
    alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
    <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.22 所示。

图 6.22 调用 Array 对象的方法生成字符串实例

Array 对象的 join()方法由两种调用方式：

```
MyArray.join();
MyArray.join(string);
```

join()方法将数组中所有元素转化为字符串，并将这些串由逗号隔开合并成一个字符串作为方法的结果返回。如果调用时给定参数 string，就将 string 作为在结果字符串中分开由各个数组元素形成的字符串的分隔符。

toString()方法返回一个包含数组中所有元素，且元素之间以逗号隔开的字符串。该方法在将数组作为字符串使用时强制使用，且不需显性申明此方法的调用。

在 Navigator 3+浏览器获得支持的还有一种操作数组并返回字符串的方法，即 Array 对象的 toSource()方法：

```
MyArray.toSource();
```

该方法返回一个字符串表示该 Array 对象的源定义，其包含数组中所有元素，元素之间用逗号隔开，整个字符串用方括号“[]”括起表示它是一个数组。

注意：Array 对象的 toSource()方法在 IE 等浏览器环境中没有获得很好的支持，但属于 ECMAScript 2.0+

内置的标准方法。

6.3.7 连接两个数组

Array 对象提供 concat()方法将以参数传入的数组连接到目标数组的后面，并将结果返回新数组，从而实现数组的连接。concat()方法的语法如下：

```
var   myNewArray=MyArray.concat(arg1,arg2,…,argN);
```

该方法将按照参数的顺序将它们添加到目标数组 MyArray 的后面，并将最终的结果返回新数组 myNewArray。考察如下代码：

```
//源程序 6.17
<! 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">
<!--
//输出数组信息
function getMsg(arrayName)
{
   var arrayLength=arrayName.length;
   var tempMsg="              长度 : "+arrayLength+"\n";
   for(var i=0;i<arrayLength;i++)
   {
      tempMsg+="             ["+i+"]="+arrayName[i];
      if((i+1)%3==0)
         tempMsg+="\n";
   }
   return tempMsg;
}
//执行相关操作
function MyTest()
{
   var MyArray=new Array("First","Second","Third");
   var ArrayToAdd1=new Array("Forth","Fifth");
   var ArrayToAdd2=new Array("Sixth");
   var msg="使用 concat()方法连接数组实例:\n\n";
   msg+="目标数组:\n"+getMsg(MyArray)+"\n";
   msg+="参数数组 ArrayToAdd1:\n"+getMsg(ArrayToAdd1)+"\n";
   msg+="参数数组 ArrayToAdd2:\n"+getMsg(ArrayToAdd2)+"\n\n";
   var myNewArray=MyArray.concat(ArrayToAdd1,ArrayToAdd2);
   msg+="使用 concat()方法产生新数组:\n"+getMsg(myNewArray)+"\n";
   msg+="目标数组 MyArray:\n"+getMsg(MyArray)+"\n";
   msg+="参数数组 ArrayToAdd1:\n"+getMsg(ArrayToAdd1)+"\n";
   msg+="参数数组 ArrayToAdd2:\n"+getMsg(ArrayToAdd2)+"\n";
   alert(msg);
}
-->
</script>
</head>
```

```
<body>
<br>
<center>
<form name=MyForm>
   <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.23 所示。

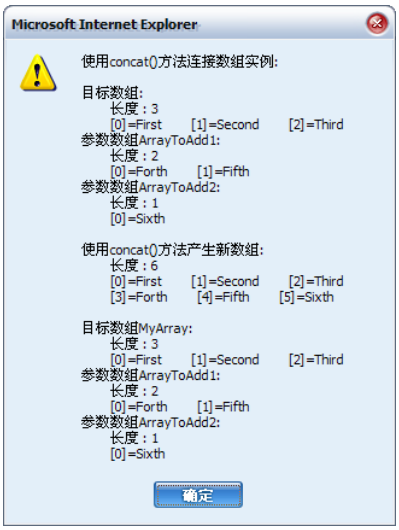

图 6.23 使用 concat()方法连接数组实例

由使用 concat()方法后目标数组和参数数组的内容不变可知，concat()方法并不修改数组本身，而是将操作结果返回给新数组。

6.3.8 常见属性和方法汇总

JavaScript 脚本的核心对象 Array 提供大量的属性和方法来操作数组。表 6.5 列出了其常用的属性、方法以及脚本版本支持情况。

表 6.5 Array对象常用的属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	Array.length	返回数组的长度，为可读可写属性	③⑥
	Array.prototype	用来给Array对象添加属性和方法	③⑥
方法	Array.concat(arg1,arg2,…argN)	将参数中的元素添加到目标数组后面并将结果返回到新数组	④⑥
	Array.join() Array.join(string)	将数组中所有元素转化为字符串，并把这些字符串连接成一个字符串，若有参数string，则表示使用string作为分开各个数组元素的分隔符	③⑥
	Array.pop()	删除数组末尾的元素并将数组length属性值减1	④
	Array.push(arg1,arg2,…,argN)	把参数中的元素按顺序添加到数组的末尾	④
	Array.reverse()	按照数组的索引号将数组元素的顺序完全颠倒	③⑥
	Array.shift()	删除数组的第一个元素并将该元素作为操作的结果返回。删除后所有剩下的元素将下移1位	④
	Array.slice(start)	返回包含参数start和stop之间的数组元素的新数	④⑥

	Array.slice(start,stop)	组，若无stop参数，则默认stop为数组的末尾	
	Array.sort() Array.sort(function)	基于一种顺序重新排列数组的元素。若有参数，则它表示一定的排序算法。详情见6.3.2实例	③⑥
	Array.splice(start,delete,arg3,…,argN)	按参数start和delete的具体值添加、删除数组元素。详情请见6.3.4应用实例	④
	Array.toSource()	返回一个表示Array对象源定义的字符串	④⑥
	Array.toString()	返回一个包含数组中所有元素的字符串，并用逗号隔开各个数组元素	③⑥

JavaScript 核心对象 Array 为我们提供了访问和操作数组的途径，使 JavaScript 脚本程序开发人员很方便、快捷操作数组这种储存数据序列的复合类型。

6.4 Date 对象

在 Web 应用中，经常碰到需要处理时间和日期的情况。JavaScript 脚本内置了核心对象 Date，该对象可以表示从毫秒到年的所有时间和日期，并提供了一系列操作时间和日期的方法。在深入了解 Array 对象前，首先了解两个有关时间标准的概念：

- GMT：格林尼治标准时间的英文缩写，指位于伦敦郊区的皇家格林尼治天文台的标准时间，该地点的经线被定义为本初子午线。理论上来说，格林尼治标准时间的正午是指当太阳横穿格林尼治子午线时的时间。
- UTC：世界协调时间的英文缩写，是由国际无线电咨询委员会规定和推荐,并由国际时间局(BIH)负责保持的以秒为基础的时间标度，相当于本初子午线上的平均太阳时。由于地球自转速度不规则，目前采用由原子钟授时的 UTC 作为时间标准。

在大多数情况下，可以假定 GMT 时间和 UTC 时间一致，电脑的时钟严格按照 GMT 时间运行。

JavaScript 脚本中采用 UNIX 系统存储时间的人工方式，即以毫秒数存储内部日期。同时，脚本在读取当前日期和时间时，依赖于客户端电脑的时钟，如果客户端电脑时钟有误，将造成一定的问题。

> 注意：为方便表述，将 GMT 时间 1970 年 1 月 1 日 0 点定义为 GMT 标准零点，下同。

6.4.1 生成日期对象的实例

Date 对象的构造函数通过可选的参数，可生成表示过去、现在和将来的 Date 对象。其构造方式有四种，分别如下：

```
var MyDate=new Date();
var MyDate=new Date(milliseconds);
var MyDate=new Date(string);
var MyDate=new Date(year,month,day,hours,minutes,seconds,milliseconds);
```

第一句生成一个空的 Date 对象实例 MyDate，可在后续操作中通过 Date 对象提供的诸多方法来设定其时间，如果不设定则代表客户端当前日期；在第二句的构造函数中传入唯一参数 milliseconds，表示构造与 GMT 标准零点相距 milliseconds 毫秒的 Date 对象实例 MyDate；第三句构造一个用参数 string 指定的 Date 对象实例 MyDate，其中 string 为表示期望日期的字符串，符合特定的格式；第四句通过具体的日期属性，如 year、month 等构造指定的 Date 对象实例 MyDate。

考察如下的代码：

```
//源程序 6.18
```

```
<! 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">
<!--
function MyTest()
{
   var msg="生成日期对象实例 ：  \n\n";
   var MyDate4=new Date(2007,8,1,19,8,20,100);
   var MyDate1=new Date();
   MyDate1=MyDate4;
   var MyDate2=new Date(MyDate1.getTime());
   var MyDate3=new Date(MyDate1.toString());
   msg+="MyDate1   :   "+MyDate1.toString()+"\n";
   msg+="MyDate2   :   "+MyDate2.toString()+"\n";
   msg+="MyDate3   :   "+MyDate3.toString()+"\n";
   msg+="MyDate4   :   "+MyDate3.toString()+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
   <input type=button value=日期测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.24 所示。

图 6.24 生成 Date 对象实例

该程序分为几步：

（1）通过第四种构造方式构造代表 2007 年 9 月 1 日 17 点 8 分 20 秒 100 毫秒的 Date 对象实例 MyDate4；

（2）通过第一种构造方式构造空的 Date 对象实例 MyDate1，并通过对象拷贝的方法，将 MyDate4 复制到 MyDate1；

（3）通过 Date 对象的 getTime()方法返回 MyDate4 表示的时间与 GMT 标准零点之间的毫秒数，然后将此毫秒数作为参数通过第二种构造方式构造 Date 对象实例 MyDate2；

（4）通过 Date 对象的 toString()方法返回表示 Date 对象实例 MyDate4 代表的时间的字符串，然后将此字符串作为参数通过第三种构造方式构造 Date 对象实例 MyDate3。

通过程序结果可知，上述四种构造 Date 对象实例的方式都能构造同样的日期对象。但上述的四种构造方法都是以当地时间创建新日期，JavaScript 提供了 UTC()方法按世界时间创建日期，语法如下：

```
date.UTC(year,month,day,hours,minutes,seconds,millseconds);
```

UTC()方法的参数意义与上述的第四种方法完全相同，只不过构建的对象基于 UTC 世界时间而已，如下的代码显示两者的不同之处：

```
var MyDate=new Date();
msg+="本地时间："+MyDate.toString()+"\n";
msg+="世界标准时间："+MyDate.toUTCString();
document.write(msg);
```

上述代码的运行后，返回：

本地时间：Fri Aug 3 21:21:43 UTC+0800 2007

世界标准时间：Fri, 3 Aug 2007 13:21:43 UTC

可以看出两者之间的差异，本地时间（东 8 区：北京时间）与 UTC 世界标准时间之间相差 8 小时，且输出字符串格式不同。

> 注意：欧美时间制中，星期及月份数都从 0 开始计数。如星期中第 0 天为 Sunday，第 7 天为 Saturday；月份中的第 0 月为 January，第 11 月为 December。但月的天数从 1 开始计数。

6.4.2 如何提取日期各字段

Date 对象以目标日期与 GMT 标准零点之间的毫秒数来储存该日期，给脚本程序员操作 Date 对象带来一定的难度。为解决这个难题，JavaScript 提供大量的方法而不是通过直接设置或读取属性的方式来设置和提取日期各字段，这些方法将毫秒数转化为对用户友好的格式。下面的程序以中文方式在文本框中动态显示系统时间：

```
//源程序 6.19
<! 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>
</head>
<body onLoad="StartClock()">
<br><br>
<script language="JavaScript">
<!--
var timerID = null;
var timerRunning = false;
//获取日期各字段并赋值
function MyTimer()
{
   //生成 Date 对象实例，同时通过其方法提取日期的各字段
   var nowTime = new Date();
   var iyear = nowTime.getYear();
   var imonth = nowTime.getMonth();
   var iweek = nowTime.getDay();
```

```
   var idate = nowTime.getDate();
   var ihours= nowTime.getHours();
   var iminutes = nowTime.getMinutes();
   var iseconds = nowTime.getSeconds();
   //通过函数返回月份和日期的英文形式
   var Month = MyMonth(imonth);
   var Week = MyWeek(iweek);
   //设定输出格式
   var myDate = ((idate<10) ? "0" : "") + idate;
   var AM_PM = (ihours>= 12) ? "P.M." : "A.M.";
   var tempHours = (ihours>= 12) ? (ihours-12) : ihours;
   var Hours = ((tempHours < 10) ? "0" : "") + tempHours;
   var Minutes = ((iminutes < 10) ? ":0" : ":") + iminutes;
   var Seconds = ((iseconds < 10) ? ":0" : ":") + iseconds;
   //设定输出字符串
   var iTime = (Week+ "    " +Month+ "    " +myDate+ "," +iyear+ "    " +Hours+Minutes+Seconds+ " "
+AM_PM);
   document.MyForm.MyText.value=iTime;
   //设定定时器及时更新文本框内容
   timerID=setTimeout("MyTimer()",1000);
   timerRunning = true;
}
//转换月份，0 对应一月份，依此类推
function MyMonth(month)
{
   var strMonth;
   if(month==0) strMonth = "January";
   if(month==1) strMonth = "February";
   if(month==2) strMonth = "March";
   if(month==3) strMonth = "April";
   if(month==4) strMonth = "May";
   if(month==5) strMonth = "June";
   if(month==6) strMonth = "July";
   if(month==7) strMonth = "August";
   if(month==8) strMonth = "September";
   if(month==9) strMonth = "October";
   if(month==10) strMonth = "November";
   if(month==11) strMonth = "December";
   return strMonth;
}
//转换星期，0 对应星期日，依此类推
function MyWeek(week)
{
   var strWeek;
   if(week==0) strWeek = "Sunday";
   if(week==1) strWeek = "Monday";
   if(week==2) strWeek = "Tuesday";
   if(week==3) strWeek = "Wednesday";
   if(week==4) strWeek = "Thursday";
   if(week==5) strWeek = "Friday";
   if(week==6) strWeek = "Saturday";
   return strWeek;
}
```

```
//文档载入的同时启动定时器
function StartClock()
{
   if(timerRunning)
      clearTimeout(timerID);
   timerRunning = false;
   MyTimer();
}
-->
</script>
<center>
<form name="MyForm">
   <input type=text name="MyText" size=40>
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.25 所示，并根据客户端时钟及时更新文本框内容。

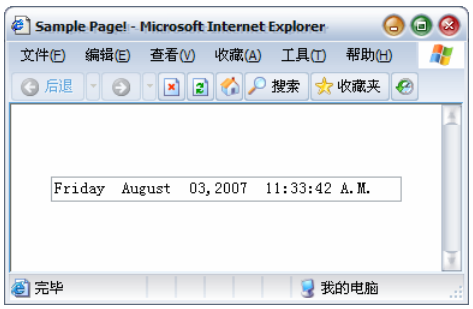

图 6.25 提取日期各字段实例

上述代码主要包括如下内容：

（1）MyTimer()函数：该函数首先构造空的 Date 对象实例 nowTime，用于保存当前系统的日期信息，然后通过 Date 对象的各种提取日期中信息的方法，获得如年、月、日、时、分、秒等信息，并更改输出格式；启动定时器 timerID 以及时更新 Date 对象实例 nowTime；

（2）MyWeek(week)函数：该函数将以数值参数传入的星期转化为英文表示的星期，并把结果以字符串的形式返回；

（3）MyMonth(month)函数：该函数将以数值参数传入的月份转化为英文表示的月份，并把结果以字符串的形式返回；

（4）StartClock()函数：该函数用于在文档载入时响应其 onload()事件，并设置初始状态，并将主动权交给 MyTimer()函数。

提取日期各字段的关键代码如下：

```
var nowTime = new Date();
var iyear = nowTime.getYear();
var imonth = nowTime.getMonth();
var iweek = nowTime.getDay();
var idate = nowTime.getDate();
var ihours= nowTime.getHours();
var iminutes = nowTime.getMinutes();
var iseconds = nowTime.getSeconds();
```

上述代码依次为构造用于保存当前日期的空对象和获取年、月、星期、日、小时、分、

秒等日期字段，并分别用变量保存个各字段信息，用于后续处理。

上述日期都是客户端日期，Date 对象也提供了基于 UTC 世界标准时间提取目标日期中各字段的诸多方法，如 getUTCDay()、getUTCSeconds()等。这些方法的使用过程与实例中的相同，只不过操作的基础不是客户端日期，而是 UTC 世界标准时间。

理解了如何从现有 Date 对象实例中提取日期各字段的问题后，下面了解设置日期中的各字段的方法。

6.4.3 如何设置日期各字段

在实际应用中，通常需要在原有的日期基础上得到新的日期，如电影中的“二十年后”等。Date 对象提供一系列的操作日期的方法。考察如下代码：

```
//源程序 6.20
<! 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">
<!--
function MyTest()
{
   var MyDate=new Date(2007,8,1,19,8,20,100);
   var msg="设置日期各字段实例 : \n\n";
   msg+="原始日期   :   "+MyDate.toString()+"\n\n";
   var temp="";
   temp=MyDate.setFullYear(1997);
   msg+="setFullYear(1997)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setYear(98);
   msg+="setYear(98)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setYear(1999);
   msg+="setYear(1999)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setMonth(3);
   msg+="setMonth(3)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setDate(21);
   msg+="setDate(21)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setHours(11);
   msg+="setHours(11)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setMinutes(45);
   msg+="setMinutes(45)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setSeconds(59);
   msg+="setSeconds(59)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   MyDate.setTime(21234234);
   msg+="setTime(21234234)方法-->返回 : MyDate:"+MyDate.toString()+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
```

```
<br>
<center>
<form name=MyForm>
   <input type=button value=日期测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.26 所示。

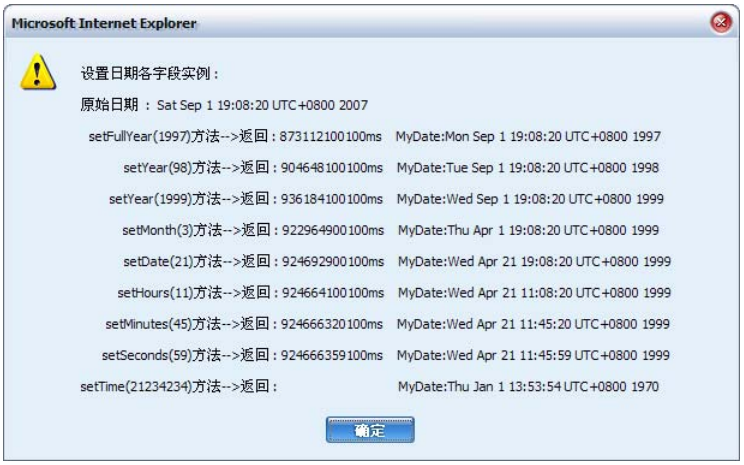

图 6.26 设置日期各字段的方法

使用 Date 对象的方法设置目标日期的指定字段之后，JavaScript 脚本更改目标日期的内容，同时将该新日期与 GMT 标准零点之间相距的毫秒数作为操作的结果返回。

注意：Date 对象的 setYear()方法可以接受 2 位或者 4 位的数字作为参数，其中 2 位的参数加上 1900 的结果作为设置的年份，但这种方法会带来二义性，一般应使用 4 位数值型参数。为解决此问题，Date 对象另外提供了 setFullYear()方法，该方法通过传入 4 位数值型参数实现同样的功能。

上述使用的都是本地时间，在 JavaScript 脚本中也可使用 UTC 标准世界时间作为操作的标准，同样存在诸如 setUTCDate()、setUTCMonth()等诸多的方法。

6.4.4 将日期转化为字符串

Date 对象提供如 toGMTString()、toLocalString()等方法将日期转换为字符串，而不需要开发人员编写专门的函数实现该功能。考察如下代码：

```
//源程序 6.21
<! 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">
<!--
function MyTest()
{
```

```
    var MyDate=new Date();
    var msg="日期转化为字符串实例 : \n\n";
    msg+="本地日期 toString()  :  "+MyDate.toString()+"\n";
    msg+="本地日期 toLocalString()  :  "+MyDate.toLocaleString()+"\n";
    msg+="GMT 世界时间 toGMTString()  :  "+MyDate.toGMTString()+"\n";
    msg+="UTC 世界时间 toUTCString()  :  "+MyDate.toUTCString()+"\n";
    alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
    <input type=button value=日期测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.27 所示。

图 6.27 日期转化为字符串实例

从程序结果可以看出，toString()和 toLocaleString()方法返回表示客户端日期和时间的字符串，但格式大不相同。实际上，toLocaleString()方法返回字符串的格式由客户设置的日期和时间格式决定，而 toString()方法返回的字符串遵循以下格式：

```
Fri Aug 3 22:49:00 UTC+0800 2007
```

由于目前 UTC 已经取代 GMT 作为新的世界时间标准，后面两种将日期转化为字符串的方法 toGMTString()和 toUTCString()返回的字符串格式、内容均相同。

同样，Date 对象提供了 parse()方法来将特定格式的字符串转化为毫秒数(目标日期与 GMT 标准零点的间隔)，后者可根据前面讲述的生成日期对象的第二种方法来生成表示该日期的 Date 对象，parse()方法的语法如下：

```
date.parse(date);
```

此方法与参数指定的对象而不是对象中的日期相联系，唯一的参数 date 应是使用 Date 对象的 toGMTString()方法生成的字符串格式：

```
Fri,3 August 2007 14:49:00 UTC
```

如果作为参数传入的表示日期的字符串不被 parse()方法认可，则 date.parse()方法返回 NaN 值。考察如下代码：

```
var MyDate=new Date();
var msg="字符串转化为时间实例 : \n\n";
msg+="转化为 GMT 世界时间的字符串 : \n"+MyDate.toGMTString()+"\n\n";
```

```
msg+="Date.parse(s1)方法返回毫秒数 : \n"+Date.parse(MyDate.toGMTString())+"ms\n\n";
var newDate=new Date(Date.parse(MyDate.toGMTString()));
msg+="通过返回的毫秒数生成的日期 : \n"+newDate.toString()+"\n\n";
var str="Friday,2002";
msg+="传入字符串 str : \n              "+str+"\n\n";
msg+="Date.parse(str)方法返回 : \n              "+Date.parse(str)+"\n\n";
alert(msg);
```

程序运行结果如图 6.28 所示。

图 6.28 字符串转化为时间实例

在老版本的浏览器中，toUTCString()方法和 toGMTString()方法返回的字符串不同，parse()方法只能识别 toGMTString()方法返回的字符串（也可接受缺失所有或部分时间或时区部分的字符串）。由于 UTC 世界时间取代 GMT 世界时间（实际上两者在某种意义上等同）成为世界时间标准，目前上述两种方法产生的字符串（或其子串）都可作为 parse()方法的参数传入以实现生成新的日期对象等功能。

6.4.5 常见属性和方法汇总

Date 对象提供成熟的操作日期和时间的诸多方法，方便脚本开发过程中程序员简单、快捷操纵日期和时间。表 6.6 列出了其常用的属性、方法以及脚本版本支持情况：

表 6.6 Date对象常用的属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	prototype	允许在Date对象中增加新的属性和方法	③⑥
方法	getdate()	返回月中的某一天	①②
	getday()	返回星期中的某一天（星期几）	①②
	getFullyear()	返回用4位数表示的当地时间的年	④⑥
	getHours()	返回小时	①②
	getMillseconds()	返回毫秒	④⑥
	getMinutes()	返回分钟	①②
	getMonth()	返回月份	①②
	getseconds()	返回秒	①②
	getTime()	返回以毫秒表示的日期和时间	①②
	getTimezoneoffset()	返回以GMT为基准的时区偏差，以分计算	①②
	getUTCDate()	返回转换成世界时间的月中某一天	④⑥
	getUTCDay()	返回转换成世界时间的星期中的某一天（星期几）	④⑥
	getUTCFullyear()	返回转换成世界时间的4位数表示的当地时间的年	④⑥
	getUTCHours()	返回转换成世界时间的小时	④⑥

	getUTCMillseconds()	返回转换成世界时间的毫秒	④⑥
	getUTCSeconds()	返回转换成世界时间的分钟	④⑥
	getYear()	返回转换成世界时间的月份	①②
	parse()	返回转换成世界时间的秒	①②
	setDate()	设置月中的某一天	③⑥
	setFullyear()	按以参数传入的4位数设置年	④⑥
	setHours()	设置小时	①②
	setMillseconds()	设置毫秒	④⑥
	setMinutes()	设置分钟	①②
	setMonth()	设置月份	①②
	setseconds()	设置秒	①②
	setTime()	从一个表示日期和时间的毫秒数来设置日期和时间	①②
	setUTCDate()	按世界时间设置月中的某一天	④⑥
	setUTCFullyear()	按世界时间按以参数传入的4位数设置年	④⑥
	setUTCHours()	按世界时间设置小时	①②
	setUTCMillseconds()	按世界时间设置毫秒	④⑥
	setUTCMinute()	按世界时间设置分钟	④⑥
	setUTCMonth()	按世界时间设置月份	④⑥
	setUTCSeconds()	按世界时间设置秒	④⑥
	setYear()	以2位数或4位数来设置年	①②
	toGMTString()	返回表示GMT世界时间的日期和时间的字符串	①②
	toLocaleString()	返回表示当地时间的日期和时间的字符串	①②
	toSource()	返回Date对象的源代码	①⑥
	toString()	返回表示当地时间的日期和时间的字符串	①②
	toUTCString()	返回表示UTC世界时间的日期和时间的字符串	④⑥
	toUTC()	将世界时间的日期和时间转换位毫秒	①②

Date 对象提供大量的方法来方便脚本程序开发人员快捷操作日期对象，但在早期的浏览器及其版本（特别是 Netscape Navigator）中的支持较差，经常出现莫名其妙的错误。但在 Netscape 4+和 Internet Explorer 4+后逐步得到较为完善的支持，有效减少了错误，还可以处理 GMT 标准零点之前或之后成千上百年的日期，足以解决大部分的问题。

6.5 Number 对象

Number 对象对应于原始数值类型和提供数值常数的对象，可通过为 Number 对象的构造函数指定参数值的方式来创建一个 Number 对象的实例。其中的参数符合 IEEE 754-1985 标准，采用双精度浮点数格式表示，占用 64 字节，允许浮点量级不大于$\pm 1.7976\times 10^{308}$同时不小于$\pm 2.2250*10^{-308}$。

6.5.1 创建 Number 对象的实例

创建 Number 对象的实例

语法如下：

```
var MyData=new Number();
var MyData=new Number(value);
```

第一句构造一个空的 Number 对象实例 MyData；第二句构造一个 Number 对象的实例 MyData，同时用通过参数传入的参数 value 初始化。考察如下代码：

```
//源程序 6.22
<! 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">
<!--
function MyTest()
{
   var MyData1=new Number();
   var MyData2=new Number(312);
   var msg="构造 Number 对象的实例 : \n\n";
   msg+="构造方法 : \n";
   msg+="          语句 1 : var MyData1=new Number();\n";
   msg+="          语句 2 : var MyData1=new Number(312);          \n";
   msg+="构造结果 : \n";
   msg+="          MyData1   :   "+MyData1+"\n";
   msg+="          MyData2   :   "+MyData2+"\n";
   MyData1=1200;
   MyData2=2400;
   msg+="更改内容方法 : \n";
   msg+="          语句 1 : MyData1=1200;\n";
   msg+="          语句 2 : MyData2=2400;\n";
   msg+="更改内容结果 : \n";
   msg+="          MyData1   :   "+MyData1+"\n";
   msg+="          MyData2   :   "+MyData2+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
   <input type=button value=日期测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.29 所示。

图 6.29 构造 Number 对象实例

通过第一种方法构造空的 Number 对象实例后，JavaScript 语言给其赋默认值 0；第二种方法中构造 Number 对象实例，并用参数 312 初始化。在后续操作中，都可通过直接赋值的方式更改其内容。

6.5.2 将 Number 对象转化为字符串

使用上述的方法构造 Number 对象的实例后，可调用 Number 对象的 toString()方法将其转化为字符串。考察如下代码：

```
var MyData=new Number(312);
var str=MyData.toString();
var msg="Number 兑现实例转化为字符串 : \n\n";
msg+="转化方法 : \n";
msg+="          语句 : var str=MyData.toString();\n";
msg+="转化结果 : \n";
msg+="          MyData1   :   "+str+"\n";
msg+="          str 类型   :   "+typeof(str)+"\n";
alert(msg);
```

程序运行结果如图 6.30 所示。

图 6.30 Number 对象实例转化为字符串

事实上，这种转换必要性不大，因为在需要转换的时候，JavaScript 会自动将该实例转换为对应的字符串。

6.5.3 通过 prototype 属性添加属性和方法

在实际应用中，经常要为同一种数据类型定义一种临时、通用的方法，如 Date 对象的 setYear()方法可以接受 2 位和 4 位数字来修改年份，方法识别传入的参数，如果是 2 位数字，则自动加上 1900，然后将结果返回构造函数，以便正确生成目标年份。可以想象在其中存在一个函数 add1900：

```
function add1900(value)
{
   var reNum;
   var errorMsg="error!";
   if(value.isNumber==1)
   {
      if(value.length==2)
          reNum=value+1900;
      else
          reNum=value;
      return reNum;
```

```
  }
  else
    return errorMsg;
}
```

Number 对象除了 toString()方法外，不支持任何方法进行数值运算，但开发者能够通过其 prototype 属性来扩充其属性和方法。下面的代码演示如何给 Number 对象添加新的方法 newFunc()并将其指向其具体实现函数，具体功能是返回 Number 对象的实例代表的数值的正弦值并加 1，然后将结果返回：

```
//源程序 6.23
<! 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 MySin(num)
{
  var result=Math.sin(num*Math.PI/180)+1;
  return result;
}
function MyTest()
{
  var MyData=new Number();
  Number.prototype.newFunc=MySin;
  var ivalue=document.MyForm.MyText.value;
  var tempData=MyData.newFunc(ivalue);
  var msg="使用 prototype 属性添加方法实例 : \n\n";
  msg+="添加方法语句 : \n";
  msg+="          Number.prototype.newFunc=MySin;\n";
  msg+="触发函数 : MySin()\n";
  msg+="          结果 : sin(" +ivalue+" • )+1="+MyData.newFunc(ivalue)+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<p>文本框数值代表目标角度,单击按钮得到其正弦值</p>
<form name=MyForm>
  <input type=text name=MyText size=20>
  <input type=button value=测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.31 所示。

图 6.31 通过 prototype 属性添加对象的方法实例

语句：

```
Number.prototype.newFunc=MySin;
```

通过访问 Number 对象的 prototype 属性添加 newFunc()方法，并将该方法的具体实现指向自定义的函数 MySin()；然后通过调用该方法实现相应的功能。通过 Number 对象的 prototype 属性给 Number 对象添加属性的过程更简单，只需设定新属性的名称并直接赋值即可。

> 注意：使用 Number 对象的 prototype 属性给 Number 对象添加的属性和方法作用范围仅限于当前代码范围，超出该代码范围新的属性和方法将失效，此特性也适用于其他 JavaScript 内置核心对象。

6.5.4 常见属性和方法汇总

Number 对象为 JavaScript 核心对象中表示数值类型的对象，拥有较少的属性和方法，表 6.7 列出了其常用的属性、方法以及脚本版本支持情况：

表 6.7 Number对象常见属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	MAX_VALUE	指定脚本支持的最大值	③⑦
	MIN_VALUE	指定脚本支持的最小值	③⑦
	NaN	为Not a Number的简写，表示一个不等于任何数的值	③⑦
	NEGTTIVE_INFINITY	表示负无穷大的特殊值	③⑦
	POSITIVE_INFINITY	表示正无穷大的特殊值	③⑦
	prototype	允许在Number对象中增加新的属性和方法	③⑦
方法	toSource()	返回表示当前Number对象实例的字符串	⑤
	toString()	得到当前Number对象实例的字符串表示	③⑦
	valueOf()	得到一个Number对象实例的原始值	③⑥

> 注意：在脚本版本支持一栏，⑦指 JScript 1.0+，其余代号如前述。

Number 对象为 JavaScript 脚本开发人员提供了一系列常用于数值判断的常量，同时可通过其 prototype 属性扩展 Number 对象的属性和方法。

6.6 Boolean 对象

Boolean 对象是对应于原始逻辑数据类型的内置对象，它具有原始的 Boolean 值，只有 true 和 false 两个状态，在 JavaScript 脚本中，1 代表 true 状态，0 代表 false 状态。

6.6.1 创建 Boolean 对象的实例

Boolean 对象的实例可通过使用 Boolean 对象的构造函数、new 操作符或 Boolean()函数来创建：

```
var MyBool=new Boolean();
var MyBool=new Boolean(value);
var MyBool=Boolean(value);
```

第一句通过 Boolean 对象的构造函数创建对象的实例 MyBool，并用 Boolean 对象的默认值 false 将其初始化；第二句通过 Boolean 对象的构造函数创建对象的实例 MyBool，并用以参数传入的 value 值将其初始化；第三句使用 Boolean()函数创建 Boolean 对象的实例，并用以参数传入的 value 值将其初始化。

下面的代码用不同的方式创建 Boolean 对象的实例，并使用 typeof 操作符返回其类型：

```
//源程序 6.24
<! 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 MyTest()
{
   var MyBoolA=new Boolean();
   var MyBoolB=new Boolean(false);
   var MyBoolC=new Boolean("false");
   var MyBool=Boolean(false);
   var msg="创建 Boolean 对象实例 : \n\n";
   msg+="生成语句 : \n";
   msg+="          var MyBoolA=new Boolean();\n";
   msg+="返回结果 : \n";
   msg+="          MyBoolA = " +MyBoolA+ "          类型 : "+typeof(MyBoolA)+"\n\n";
   msg+="生成语句 : \n";
   msg+="          var MyBoolB=new Boolean(false);\n";
   msg+="返回结果 : \n";
   msg+="          MyBoolB = " +MyBoolB+ "          类型 : "+typeof(MyBoolB)+"\n\n";
   msg+="生成语句 : \n";
   msg+="          var MyBoolC=new Boolean("false");\n";
   msg+="返回结果 : \n";
   msg+="          MyBoolC = " +MyBoolC+ "          类型 : "+typeof(MyBoolC)+"\n\n";
   msg+="生成语句 : \n";
   msg+="          var MyBool=new Boolean(false);\n";
   msg+="返回结果 : \n";
```

```
   msg+="          MyBool   =   " +MyBool+ "          类型 : "+typeof(MyBool)+"\n\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
   <input type=button value=测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.32 所示。

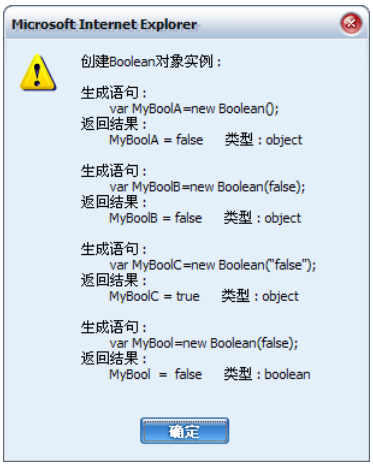

图 6.32 创建 Boolean 对象的实例

以下两点值得特别注意：

- 在第三种构造方式中，首先判断字符串“false”是否为 null，结果返回 true，并将其作为参数通过 Boolean 构造函数创建对象，故其返回 MyBoolC=true；
- 在第四种构造方式中，生成的 MyBool 仅仅为一个包含 Boolean 值的变量，其类型与前面三种不同，为 boolean 而不是 object。

Boolean 对象构造完成后，可通过直接对实例赋值的方式修改其内容。在实际构造过程中，要灵活运用这几种构造的方法，并理解其间的不同点和相似之处。

> 注意：在创建 Boolean 对象实例过程中，如果传入的参数为 null、NaN、""或者 0 将自动变成 false，其余的将变成 true。

6.6.2 将 Boolean 对象转化为字符串

由于 Boolean 对象继承自 Object 对象，后者为其提供 toString()方法将其代表的状态转化为字符串“true”或“false”，进行后续操作。考察如下代码：

```
//源程序 6.25
<! 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 MyTest()
{
   var MyBool=new Boolean(false);
   var msg="转换 Boolean 对象为字符串实例 : \n\n";
   msg+="生成语句 : \n";
   msg+="          var MyBool=new Boolean(); \n";
   msg+="返回结果 : \n";
   msg+="          MyBool = " +MyBool+ "          类型 : "+typeof(MyBool)+"\n\n";
   var MyStr=MyBool.toString();
   msg+="转换语句 : \n";
   msg+="          var MyStr=MyBool.toString();              \n";
   msg+="转换结果 : \n";
   msg+="          MyStr = " +MyStr+ "          类型 : "+typeof(MyStr)+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
   <input type=button value=测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.33 所示。

图 6.33 将 Boolean 对象转化为字符串实例

可以明显看出，使用 toString()方法，返回的 MyStr="false"为字符串。

在 Boolean 对象中，还有一种方法将 Boolean 对象转为字符串：toSource()方法，该方法返回一个表示对象创建代码的字符串，并应括号括起。如：

```
var MyBool=new Boolean(true);
var str=MyBool.toSource();
```

```
document.write("str="+str+"<br>type:"+typeof(str));
```

代码执行后，返回字符串：

```
str=(new Boolean(true))
type:string
```

可见，toSource()方法返回的是用括号括起来的表示 Boolean 对象的字符串，与 toString()方法返回的字符串存在根本的不同。

6.6.3 常见属性和方法汇总

Boolean 对象为 JavaScript 脚本语言的封装对象，表示原始的逻辑状态 true 和 false，表 6.8 列出了其常用的属性、方法以及脚本版本支持情况：

表 6.8 Boolean对象常见属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	prototype	允许在Boolean对象中增加新的属性和方法	③⑥
方法	toSource()	返回表示当前Boolean对象实例创建代码的字符串	⑤⑥
	toString()	返回当前Boolean对象实例的字符串（"true"或"false"）	③⑥
	valueOf()	得到一个Boolean对象实例的原始Boolean值	③⑥

在实际应用中，常通过 prototype 属性扩展 Boolean 对象的属性和方法，开发者要十分注意几种创建 Boolean 对象实例的方式的相似点和不同之处。

6.7 Function 对象

JavaScript 核心对象 Funtion 为构造函数的对象，由于开发者一般直接定义函数而不是通过使用 Function 对象创建实例的方式来生成函数，对于实际编程而言，Function 对象很少涉及到，但正确地理解它有助于开发者加深对 javaScript 脚本中函数概念的理解。

6.7.1 两个概念：Function 与 function

简而言之，Function 是对象而 function 是函数。实际上，在 JavaScript 中声明一个函数本质上为创建 Function 对象的一个实例，而函数名则为实例名。先看如下的函数：

```
function sayHello(username)
{
   alert("Hello "+name);
}
```

输入参数“NUDT!”，返回警告框如图 6.34 所示。

图 6.34 sayHello()函数的返回警告框

如果通过创建 Function 对象的实例的方式来实现该功能，代码如下：

```
var sayHello = new Function("name","alert('Hello '+name)");
```

在该方式中,第一个参数是函数 sayHello()的参数，第二个参数是函数 sayHello()的函数体。定义之后，可通过调用 sayHello("NUDT!")的方式获得上述的结果。

通过两种构造方式的对比，可以看出所谓的函数只不过是 Function 对象的一个实例，而函数名为实例的名称。

既然函数名为实例的名称，那么就可以将函数名作为变量来使用。考察如下的代码：

```
function sayHello()
{
   alert("Hello");
}
function sayBye()
{
   alert("Bye");
}
sayHello = sayBye;
```

上述代码运行后，再次调用 sayHello()函数，返回的是“Bye”而不是“Hello”。

6.7.2 使用 Function 对象构造函数

在 JavaScript 中，构造函数常用如下的两种方法：

- 函数的原始构造方法：

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

- 创建 Function 对象实例的方法：

```
functionName = new Function( [arg1, [... argN,]],body );
```

其中 functionName 是创建的目标函数名称，为必选项；arg1，...，argN 是函数接收的参数列表，为可选项。函数接收的参数列表；body 是函数体，包含调用此函数时执行的代码，为可选项。

举个执行两个数加法的程序，使用第一种构造方法：

```
function add(x, y)
{
    return(x + y);
}
```

如果采用创建 Function 对象实例的方式实现同样的功能，如下代码：

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

在这两种情况下，都通过 add(4, 5)的方式调用目标函数。第二种构造方式适用于参数较少、函数代码比较简单的情形，而第一种方式代码层次感较强，且对代码的复杂程度和参数多少并无特别的规定。

鉴于以上原因，脚本开发人员主要使用第二种构造方法来构造函数，但理解 Function 对象对开发者深入理解函数的本质有很大的帮助。

6.7.3 常见属性和方法汇总

Function 对象在脚本编程中使用不是很广泛，表 6.9 列出了其常用的属性、方法以及脚

本版本支持情况：

表 6.9 Function对象常见属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	arguments	包含传给函数的参数，只能在函数内部使用	③⑦
	arity	表示一个函数期望接收的参数数目	④
	caller	用来访问调用当前正在执行的函数的函数	③⑦
	prototype	允许在Function对象中增加新的属性和方法	③⑦
方法	apply()	将一个Function对象的方法使用在其他Function对象上	⑤⑦
	call()	该方法允许当前对象调用另外一个Function对象的方法	
	toSource()	允许创建一个Function对象的拷贝	⑤⑦
	toString()	将定义函数的JavaScript源代码转换为字符串并将其作为调用此方法的结果返回	③⑥

在实际应用中，经常使用 apply()和 call()方法将一个 Function 对象的方法使用在其他对象上，实现脚本代码的重用。

6.8 Object 对象

所有的 JavaScript 对象都继承自 Object 对象，后者为前者提供基本的属性（如 prototype 属性等）和方法（如 toString()方法等）。而前者也在这些属性和方法基础上进行扩展，以支持特定的某些操作。

6.8.1 创建 Object 对象的实例

Object 对象的实例构造方法如下：

```
var MyObject=new Object(string);
```

上述语句构造 object 对象的实例 MyObject，同时用以参数传入的 string 初始化对象实例，该实例能继承 object 对象提供的几个方法进行相关处理。参数 string 为要转为对象的数字、布尔值或字符串，此参数可选，若无此参数，则构建一个未定义属性的新对象。

JavaScript 脚本支持另外一种构造 Object 对象实例的方法：

```
var MyObject={name1:value1,name2:value2,...,nameN:valueN};
```

该方法构造一个新对象，并使用指定的 name1，name2，…，nameN 指定其属性列表，使用 value1，value2，…，valueN 初始化该属性列表。

考察如下代码：

```
//源程序 6.26
<! 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 MyTest()
{
   //第一种构造方法
   var myObject1=new Object();
```

```
  //为新对象添加属性
  myObject1.name="小李";
  myObject1.gender="女";
  myObject1.age=28;
  var msg="创建 Object 对象实例 : \n\n";
  msg+="创建语句 : \n";
  msg+="          var myObject1=new Object();\n";
  msg+="          myObject1.name="小李";\n";
  msg+="          myObject1.gender="女";\n";
  msg+="          myObject1.age=28;\n";
  msg+="创建结果 : \n";
  msg+="          类型 : " +typeof(myObject1)+ "\n";
  msg+="          属性 name : " +myObject1.name+ "\n";
  msg+="          属性 gender : " +myObject1.gender+ "\n";
  msg+="          属性 age : " +myObject1.age+ "\n\n";
  //第二种构造方法
  var myObject2={name:"小王",gender:"男",age:32};
  msg+="创建语句 : \n";
  msg+="          var myObject2={name:"小王",gender:"男",age:32};          \n";
  msg+="创建结果 : \n";
  msg+="          类型 : " +typeof(myObject2)+ "\n";
  msg+="          属性 name : " +myObject2.name+ "\n";
  msg+="          属性 gender : " +myObject2.gender+ "\n";
  msg+="          属性 age : " +myObject2.age+ "\n\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
  <input type=button value=测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.35 所示。

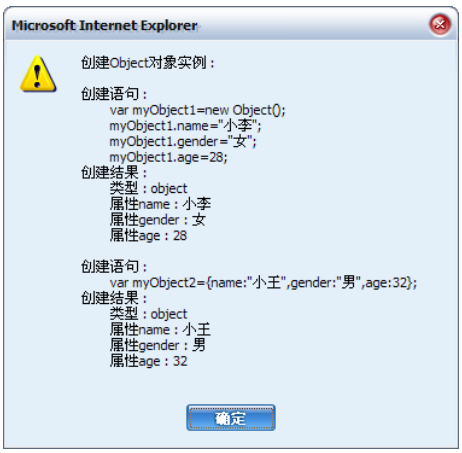

图 6.35 Object 对象两种不同的构造方式实例

由程序结果可见，这两种构造 Object 对象实例的方法得到相同的结果，相比较而言，第一种方法结构清晰、层次感强，而第二种方法代码简单、编程效率高。

6.8.2 常见属性和方法列表

通过从 Object 对象继承产生后，String、Math、Array 等对象获得了 Object 对象所有的属性和方法，同时扩充了之只属于自身的属性和方法，以对特定的目标进行处理。表 6.10 列出了其常用的属性、方法以及脚本版本支持情况：

表 6.10 Object对象常见属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	constructor	指定对象的构造函数	③⑥
	prototype	允许在Object对象中增加新的属性和方法	③⑥
方法	eval()	通过当前对象执行一个表示JavaScript脚本代码的字符串	③⑥
	toSource()	返回创建当前对象的源代码	④
	toString()	返回表示对象的字符串	③⑥
	valueOf()	返回目标对象的值	③⑥

在 JavaScript 脚本编程实践中，由于 Object 对象的属性和方法都比较少，直接使用 Object 对象进行相关操作的情况很少，一般情况下使用由其派生的 Array、Math 等其他核心对象或者使用程序员自己定义、由 Object 对象派生的对象，并为其添加特定的属性和方法的方式来实现既定的功能。

6.9 本章小结

JavaScript 语言的核心对象由于其构成了脚本编程的基础，显得尤为重要。本章主要介绍了除 RegExp 对象之外的其余 JavaScript 核心对象，论述了其创建策略、常见操作并与实际需求相结合，列举了很多常用的操作实例，如邮箱地址验证、任意范围随机数发生器等。

很多核心对象与 JavaScript 支持的各种数据类型相关，程序员经常使用与数组、字符串等复杂数据类型相关的内置变量的属性和方法，但大多数程序员会忽略“基本类型也是对象”的事实。理解它们之间的联系可以使读者成为优秀的 JavaScript 脚本程序员，而不是普通的使用者。

前几章了解了 JavaScript 脚本的概念、语法、事件处理和内置的核心数据对象等基础知识，下一章将全面进入奇妙的文档结构模型（DOM）世界。