第3章

JavaScript

JavaScript 是一种脚本语言。脚本(Script)是一段可以嵌入到其他文档中的程序,用来完成某些特殊的功能。脚本既可以运行在浏览器端(称为客户端脚本),也可以运行在服务器端(称为服务器端脚本)。本章以 JavaScript 语言为基础介绍客户端脚本编程。

3.1 JavaScript 入门

客户端脚本经常用来检测浏览器,响应用户动作、验证表单数据及动态改变元素的 HTML 属性或 CSS 属性等,由浏览器对客户端脚本进行解释执行。由于脚本程序驻留在客户机上,因此响应用户动作时无须与 Web 服务器进行通信,从而降低了网络的传输量和 Web 服务器的负荷,目前的 RIA(Rich Internet Application,富集网络应用程序)技术提倡可以在客户端完成的功能都尽量放在客户端运行。

目前使用最广泛的两种脚本语言是 JavaScript 和 VBScript。需要说明的是,这两种语言都既可以作为客户端脚本也可以作为服务器端脚本。但 JavaScript 对于浏览器的兼容性比 VBScript 要好,所以已经成为客户端脚本事实上的标准,而 VBScript 由于是微软 ASP 默认的服务器端脚本语言,因此一般用作服务器端脚本。

3.1.1 JavaScript 的特点和功能

1. JavaScript 的特点

JavaScript 是一种基于对象的语言。基于对象的语言含有面向对象语言的编程思想,但比面向对象语言简单。

面向对象程序设计力图将程序设计为一些可以完成不同功能的独立部分(对象)的组合体。相同类型的对象作为一个类(class)被组合在一起,(例如,“小汽车”对象属于“汽车”类)。基于对象的语言与面向对象语言的不同之处在于,它自身已包含一些已创建完成的对象,通常情况下都是使用这些已创建好的对象,而不需要创建新的对象类型——“类”,来创建新对象。

JavaScript 是事件驱动的语言。当用户在网页中进行某种操作时,就产生了一个“事件”(event)。JavaScript 是事件驱动的,当事件发生时,它可以对其做出响应。具体如何响应由编写的事件处理程序完成。

JavaScript 是浏览器的编程语言,它与浏览器的结合使它成为最流行的编程语言之一。由于 JavaScript 依赖于浏览器本身,与操作系统无关,因此它具有跨平台性。

2. JavaScript 的功能

JavaScript 可以完成以下任务。

(1) JavaScript 为 HTML 提供了一种程序工具,弥补了 HTML 语言作为描述性语言不能编写程序的不足,JavaScript 和 HTML 可以很好地结合在一起。

(2) JavaScript 可以为 HTML 页面添加动态内容,如 document. write("< h1 >" + name + "</h1 >"),这条 JavaScript 可以向一个 HTML 页面写入一个动态的内容。其中,document 是 JavaScript 的内部对象,write 是方法,向其写入内容。

(3) JavaScript 能响应一定的事件,因为 JavaScript 是基于事件驱动机制的,所以若浏览器或用户的操作发生一定的变化,触发了事件,JavaScript 都可以做出相应的响应。

(4) JavaScript 可以动态地获取和改变 HTML 元素的属性或 CSS 属性,从而动态地创建网页内容或改变内容的显示,这是 JavaScript 应用最广泛的领域。

(5) JavaScript 可以检验数据,因此在客户端就能验证表单。

(6) JavaScript 可以检测用户的浏览器,从而为用户提供合适的页面。

(7) JavaScript 可以创建和读取 Cookie,从而为浏览者提供更加个性化的服务。

3. JavaScript 的限制功能

JavaScript 作为客户端语言使用时,设计它的目的是在用户的计算机上执行任务,而不是在服务器上。因此,JavaScript 有一些固有的限制,这些限制主要出于安全原因。

(1) JavaScript 不允许读写客户端计算机上的文件。唯一例外的是,JavaScript 可以写到浏览器的 Cookie 文件,但是也有一些限制。

(2) JavaScript 不允许读写服务器计算机上的文件,也不能访问本网站所在域外的脚本和资源。

(3) JavaScript 不能从来自另一个服务器的已经打开的网页中读取信息。也就是说,网页不能读取已经打开的其他窗口中的信息,因此无法探察访问这个站点的浏览者还在访问哪些其他站点。

(4) JavaScript 不能操纵不是由它自己打开的窗口。这是为了从避免一个站点关闭其他任何站点的窗口,从而独占浏览器。

(5) JavaScript 调整浏览器窗口大小和位置时也有一些限制,不能将浏览器窗口设置得过小或将窗口移出屏幕之外。

3.1.2 JavaScript 的代码结构

JavaScript 是事件驱动的语言。当用户在网页中进行某种操作时,就产生了一个“事件”。事件几乎可以是任何事情:单击一个网页元素、拖动鼠标指针等均可视为事件。JavaScript 是事件驱动的,当事件发生时,它可以对其做出响应。具体如何响应某个事件由编写的事件处理程序决定。

因此，一个 JavaScript 程序一般由“事件＋事件处理程序”组成。根据事件处理程序所在的位置，在 HTML 代码中嵌入 JavaScript 有 3 种方式。

1. 将脚本嵌入到 HTML 标记的事件中(行内式)

HTML 标记中可以添加“事件属性”，其属性名是事件名，属性值是 JavaScript 脚本代码。例如(3-1.html)：

```
<html><body>
    <p onclick = "alert('Hello,The Web World!');">Click Here</p>
</body></html>
```

其中，onclick 就是一个 JavaScript 事件名，表示单击鼠标事件。alert(…)；是事件处理代码，作用是弹出一个警告框。因此，当在这个 p 元素上单击时，就会弹出一个警告框，运行效果如图 3-1 所示。

图 3-1 3-1.html 和 3-2.html 的运行效果

2. 使用< script >标记将脚本嵌入到网页中

如果事件处理程序的代码很长，则一般把事件处理程序写在一个函数(称为事件处理函数)中，然后在事件属性中调用该函数。下面代码的运行效果与 3-1.html 完全相同。

```
<html><head>          <!-- 3-2.html -->
<title>第一个 JavaScript 程序</title>
<script>
    function msg () {                        //定义函数 msg
        alert ("Hello, the WEB world!") ;  }
</script></head>
<body>
   <p onclick = "msg()">Click Here</p>  <!-- 通过事件调用函数 -->
</body></html>
```

其中，“onclick＝"msg()"”表示调用函数 msg。可见，调用 JavaScript 函数可写在 HTML 标记的事件属性中，但函数的代码必须写在< script >和</script >标记之间。

将 JavaScript 代码写成函数的一个好处是，可以让多个 HTML 元素或不同事件调用同一个函数，从而提高了代码的重用性。

提示：< script >标记是专门用来在 HTML 中嵌入 JavaScript 代码的标记。建议将所有的 JavaScript 代码都写在< script >和</script >标记之间，而不要写在 HTML 标记的事件属性内。这可实现 HTML 代码与 JavaScript 代码的分离。

3. 使用< script >标记的 src 属性链接外部脚本文件

如果有多个网页文件需要共用一段 JavaScript，则可以把这段脚本保存成一个单独的.js 文件(JavaScript 外部脚本文件的扩展名为 js)，然后在网页中调用该文件，这样既提高了代码的重用性，也方便了维护，修改脚本时只需单独修改这个 js 文件的代码即可。

引用外部脚本文件的方法是使用< script >标记的 src 属性来指定外部文件的 URL。示例代码如下(3-3.html 和 3-3.js 位于同一目录下)，运行效果如图 3-1 所示。

```
------------------------------ 3-3.html 的代码 ------------------------------
<html><body>
<script type="text/JavaScript" src="3-3.js "></script>
<p onclick="msg()">Click Here</p>
</body></html>
------------------------------ 3-3.js 的代码 ------------------------------
function msg () {                    //定义函数 msg
    alert ("Hello,the WEB world!") ; }
```

从上面的几个例子可以看出,网页中引入 JavaScript 的方法其实与引入 CSS 的方法有很多相似之处,也有嵌入式、行内式和链接式。不同之处在于,用嵌入式和链接式引入 JavaScript 都是用同一个标记< script >,而 CSS 则分别使用了< style >和< link >标记。

3.1.3 JavaScript 开发和调试工具

编写 JavaScript 可以使用任何文本编辑器,但为了具有代码提示功能和程序调试功能,推荐使用下列 JavaScript 开发工具。

(1) Dreamweaver CS4:从 CS4 版本开始增加了对 JavaScript 的代码提示功能。

(2) Aptana:除了支持 JavaScript,还支持 jQuery、Dojo、Ajax 等开发框架。

(3) 1st JavaScript。

Google Chrome、IE、Firefox 等浏览器都具有 JavaScript 程序调试功能。以 Google Chrome 浏览器为例,只要在浏览器窗口中右击,在弹出的快捷菜单中选择"检查"选项,即可打开如图 3-2 所示的开发者工具,如果 JavaScript 程序在运行中发生错误,则在该窗口右上角会显示错误按钮和错误条数,单击"错误"按钮,则会弹出 Console(控制台)窗口,此时可看到具体的错误提示和错误所在的文件行数(调试之前最好单击"刷新"按钮将以前的错误提示清除)。

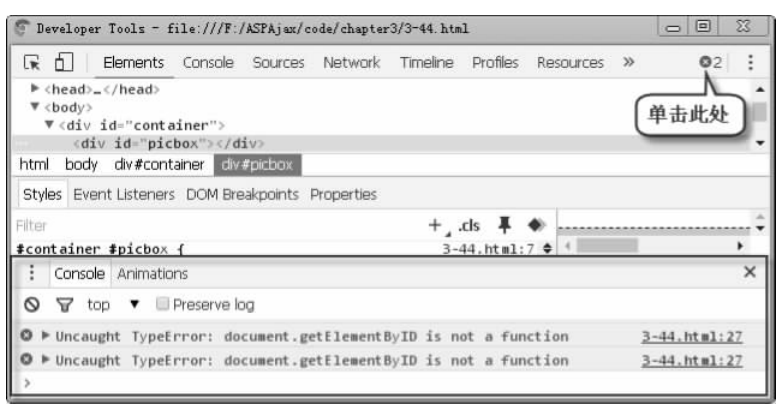

图 3-2 chrome 浏览器的控制台

如果要使用 IE 浏览器来调试 JavaScript 程序,可以执行"工具"→"Internet 选项"命令,在弹出的对话框中选择"高级"选项卡,选中其中的"显示每个脚本错误的通知"复选框,这样每次打开网页就会提示 JavaScript 程序的错误了。并且其错误提示的范围比 Firefox 的错误控制台更大。

提示：JavaScript 存在的浏览器兼容问题比 CSS 更加严重，因此 JavaScript 程序一定要通过 Google Chrome 和 IE 8 两种浏览器的测试检验。

3.2 JavaScript 语言基础

JavaScript 的语法类似于 Java 或 C 语言，但其独特之处在于，JavaScript 中的一切数据类型都是对象，并可以模拟类的实现，且功能并不简单。

3.2.1 JavaScript 的变量

JavaScript 的变量是一种弱类型变量。所谓弱类型变量，是指它的变量无特定类型，定义任何变量都是用 var 关键字，并可以将其初始化为任何值，而且可以随意改变变量中所存储的数据类型，当然为了程序规范应该避免这样操作。

JavaScript 的变量定义与赋值示例如下：

```
var name = "Six Tang";                    //定义了一个字符串变量
var age = 28;                             //定义了一个数值型变量
var male = True;                          //将变量赋值为布尔型
```

每行结尾的分号可有可无，而且 JavaScript 还可以不声明变量直接使用，它的解释程序会自动用该变量名创建一个全局变量，并初始化为指定的值。但用户应养成良好的编程习惯，变量在使用前都应当声明。另外，变量的名称必须遵循下面 5 条规则。

(1) 首字符必须是字母、下画线(_)或美元符号($)。

(2) 余下的字母可以是下画线、美元符号、任意字母或数字。

(3) 变量名不能是关键字或保留字。

(4) 变量名区分大小写。

(5) 变量名中不能有空格、回车符或其他标点字符。

例如，下面的变量名是非法的。

```
var 5zhao;                                //数字开头，非法
var tang-s, tang's;                       //对于变量名，中画线或单引号是非法字符
var this;                                 //不能使用关键字作为变量名
```

提示：为了符合编程规范，推荐变量的命名方式是：当变量名由多个英文单词组成时，第一个英文单词全部小写，以后每个英文单词的第一个字母大写，如 var myClassName。

3.2.2 JavaScript 的运算符

运算符是指完成操作的一系列符号，也称为操作符。运算符用于将一个或多个值运算成结果值，使用运算符的值称为算子或操作数。在 JavaScript 中，常用的运算符可分为四类。

1. 算术运算符

算术运算符所处理的对象都是数字类型的操作数。在对数值型的操作数进行处理之后，返回的还是一个数值型的值。算术运算符包括＋、－、＊、/、%(取模，即计算两个整数相

除的余数)、++(递增运算,递加 1 并返回数值或返回数值后递加 1,取决于运算符的位置)、--(递减运算)。

2. 比较运算符

基本关系运算符(表 3-1)通常用于检查两个操作数之间的关系,即两个操作数之间是相等、大于还是小于关系等。关系运算符可以根据是否满足该关系而返回 true 或 false。

表 3-1 基本关系(比较)运算符

运 算 符	说 明	例 子	结 果
==	是否相等(只检查值)	x=5, y="5"; x==y	true
===	是否全等(检查值和数据类型)	x=5,y="5"; x===y	false
!=	是否不等于	5!=8	true
!==	是否不全等于	x=5,y="5";x!==y	true
>、<、>=、<=	大于、小于、大于等于、小于等于	x=5, y=3; x>y	true

另外,还有两个特殊的关系运算符: in 和 instanceof。

(1) in 运算符用于判断对象中是否存在某个属性,例如:

```
var o = {title: "Informatics", author: "Tang"}
"title" in o                    //返回 ture,对象 o 具有 title 属性
"pub" in o                      //返回 false,对象 o 不具有 pub 属性
```

in 运算符对运算符左右两个操作数的要求比较严格。in 运算符要求左边的操作数必须是字符串类型或可以转换为字符串类型的其他类型,而右边的操作数必须是对象或数组。只有左边操作数的值是右边操作数的属性名,才会返回 true,否则返回 false。

(2) instanceof 运算符用于判断对象是否为某个类的实例,例如:

```
var d = new Date();
    d instanceof Date;        //返回 true
    d instanceof object;      //返回 true,因为 Date 类是 object 类的实例
```

3. 逻辑运算符

逻辑运算符(表 3-2)的运算结果只有 true 和 false 两种。

表 3-2 逻辑运算符

运 算 符	说 明	例 子	结 果
&&	逻辑与	x=6, y=3	(x < 10 && y > 1) returns true
||	逻辑或	x=6, y=3	(x==5 || y==5) returns false
!	逻辑非	x=6, y=3	!(x==y) returns true

4. 赋值运算符

JavaScript 基本的赋值运算符是=符号,它将等号右边的值赋给等号左边的变量,如 x=y,表示将 y 的值赋给 x。除此之外,JavaScript 还支持带操作的运算符,给定 x=10 和 y=5,表 3-3 所示的是各种赋值运算符的作用。

表 3-3 赋值运算符

运算符	例子	等价于	结果	运算符	例子	等价于	结果
=	x=y		x=5	*=	x*=y	x=x*y	x=50
+=	x+=y	x=x+y	x=15	/=	x/=y	x=x/y	x=2
-=	x-=y	x=x-y	x=5	%=	x%=y	x=x%y	x=0

5. 连接运算符

连接运算符“+”用于对字符串进行接合操作，例如：

```
txt1 = "What a very" ;  txt2 = "nice day!" ;
txt3 = txt1 + " " + txt2 ;
```

则变量 txt3 的值是："What a very nice day!"。

注意，连接运算符+和加法运算符+的符号相同，如果运算符左右的操作数中有一个是字符型或字符串类型，那么+表示连接运算符，如果所有操作数都为数值型，那么+表示加法运算符。例如：

```
var a = 1,b = 2;
var txt1 = "这个月是" + a + b + "月。";
var txt2 = "这个月是" + (a + b) + "月。";
document.write(txt1);                    //输出"这个月是 12 月。"
document.write(txt2);                    //输出"这个月是 3 月。"
```

从上例可以看出，只要表达式中有字符串或字符串变量，那么所有的+就都会变成连接运算符，表达式中的数值型数据也会自动转换为字符串。如果希望数值型数据中的“+”仍为加法运算符，可以为它们添加括号，使加法运算符的优先级增高。

6. 其他运算符

JavaScript 还支持一些其他运算符，主要有以下几种。

(1) 条件操作符“?:”。条件运算符是 JavaScript 中唯一的三目运算符，即它的操作数有 3 个，用法如下：

```
x = (condition)? 100: 200;
```

它实际上是 if 语句的一种简写形式，例如上述表达式等价于：

```
if (condition) x = 100;
else x = 200;
```

(2) typeof 运算符。typeof 运算符返回一个用来表示表达式的数据类型的字符串，如“string”“number”“object”等。例如：

```
var a = "abc";     alert(typeof a);     //返回 string
var b = true;     alert(typeof b);      //返回 boolean
```

(3) 下标运算符[]。下标运算符[]用来引用数组中的元素。例如：

```
arr[3]
```

(4) 逗号运算符“,”。逗号运算符“,”用来分开不同的值。例如：

```
var a, b
```

(5) 函数调用运算符“()”。JavaScript 的函数运算符是“()”之前是被调用的函数名，括号内部是由逗号分隔的参数列表，如果被调用的函数没有参数，则括号内为空。例如：

```
function f (x, y)  {return x + y;}
alert (f (2,3));                                    //返回值为 5
(function (x, y)  {return x + y;})                  //匿名函数
alert ((2,3));                                      //调用匿名函数,返回值为 5
```

(6) new 运算符。new 运算符用来创建一个对象或生成一个对象的实例。例如：

```
var a = new Object;           //创建一个 Object 对象,对于无参数的构造函数括号可省略
var dt = new Date();          //创建一个新的 Date 对象
```

7. 运算符的优先级

JavaScript 中的运算符优先级是一套规则。该规则在计算表达式时控制运算符执行的顺序。具有较高优先级的运算符先于较低优先级的运算符执行。例如，乘法的执行先于加法。

圆括号可用来改变运算符优先级所决定的求值顺序。这意味着圆括号中的表达式应在其用于表达式的其余部分之前全部被求值。例如：

```
var x = 5, y = 7;
z = (x + 4 > y)?x++ :++y;          //返回值为 5
```

在对 z 赋值的表达式中有 5 个运算符：=，+，>，()，++，?:。根据运算符优先级的规则，它们将按下面的顺序求值：()，+，>，++，?:，=。

首先对圆括号内的表达式求值。先将 x 和 4 相加得 9，然后将其与 7 比较大小，是否大于 7，得到 true，接着执行 x++，得到 5，最后把 x 的值赋给 z，所以 z 返回值为 5。

8. 表达式

表达式是运算符和操作数的组合。表达式是以运算符为基础的，表达式的值是对操作数实施运算符所确定的运算后产生的结果。表达式可分为算术表达式、字符串表达式、赋值表达式及逻辑表达式等。

3.2.3 JavaScript 数据类型

JavaScript 支持字符串、数值型和布尔型 3 种基本数据类型，支持数组、对象两种复合数据类型，还支持未定义、空、引用、列表和完成。其中后 3 种类型仅仅作为 JavaScript 运行

时的中间结果的数据类型，因此不能在代码中使用。本节介绍一些常用的数据类型及其属性和方法。

1. 字符串(String)

字符串由零个或多个字符构成，字符可以是字母、数字、标点符号或空格。字符串必须放在单引号或双引号中。例如：

```
var course = "data structure"
```

但如果一个字符串中含有双引号，则只能将该字符串放在单引号中，例如：

```
var case = 'the birthday"19801106"'
```

更通用的方法是使用转义(escaping)字符“\”实现特殊字符按原样输出，例如：

```
var score = " run time 3\'15\""   //输出 3'15"
```

2. JavaScript 中的转义字符

在 JavaScript 中，字符串都必须用引号引起来，但有些特殊字符是不能写在引号中的，如(")，如果字符串中含有这些特殊字符就需要利用转义字符来表示，转义字符以反斜杠开始表示。表 3-4 所示的是一些常见的转义字符。

表 3-4 JavaScript 的转义字符

代 码	输 出	代 码	输 出	代 码	输 出
\'	单引号	\\	反斜杠\	\t	tab，制表符
\"	双引号	\n	换行符	\b	后退一格
\&	&	\r	返回，esc	\f	换页

如果要测试这些转义字符的具体含义，可以用下面的语句将它们输出在页面上。

```
document.write ("<pre>\&\'\"\\\n\r\tabc\b</pre>");
```

3. 字符串的常见属性和方法

字符串(String)对象具有下列属性和方法，下面先定义一个示例字符串 myString：

```
var myString = "This is a sample";
```

(1) length 属性：它是字符串对象唯一的一个属性，将返回字符串中字符的个数，例如：

```
alert (myString.length);      //返回 16
```

即使字符串中包含中文(双字节)，每个中文也只算一个字符。

(2) charAt 方法：它返回字符串对象在指定位置处的字符，第一个字符位置是 0。例如：

```
myString.charAt(2);          //返回 i
```

(3) charCodeAt：返回字符串对象在指定位置处字符的十进制的 ASCII 码。

```
myString.charCodeAt(2);       //返回 i 的 ASCII 码 105
```

(4) indexOf：用于查找和定位子串，它返回要查找的子串在字符串对象中的位置。

```
myString.indexOf("is");     //返回 2
```

还可以加参数，指定从第几个字符开始查找。如果找不到则返回－1。

```
myString.indexOf("i",2);        //从索引为 2 的位置"i"后面的第一个字符开始向后查找，返回 2
```

(5) lastIndexOf：要查找的子串在字符串对象中的倒数位置。

```
myString.lastIndexOf("is");           //返回 5
myString.lastIndexOf("is",2)          //返回 2
```

(6) substr 方法：根据开始位置和长度截取子串。

```
myString.substr(10,3);        //返回 sam,10 表示开始位置,3 表示长度
```

(7) substring 方法：根据起始位置截取子串。

```
myString.substring(5,9);        //返回 is a, 5 表示开始位置,9 表示结束位置
```

(8) split 方法：根据指定的符号将字符串分割成一个数组。

```
  var a = myString.split(" ");
//a[0] = "This" a[1] = "is" a[2] = "a" a[3] = "sample"
```

(9) replace 方法：替换子串。

```
myString.replace("sample","apple");       //结果"This is a apple"
```

(10) toLowerCase 方法：将字符串变成小写字母。

```
myString.toLowerCase();        //this is a sample
```

(11) toUpperCase 方法：将字符串变成大写字母。

```
myString. toUpperCase();       //THIS IS A SAMPLE
```

4. 数值型(number)

在 Javascript 中，数值型数据不区分整型和浮点型，数值型数据和字符型数据的区别是

数值型数据不需要用引号括起来。例如，下面都是正确的数值表示法。

```
var num1 = 23.45;
var num2 = 76;
var num3 =- 9e5;              //科学计数法，即 - 900000
```

5. 布尔型(boolean)

布尔型数据的取值只有两个：true 和 false。布尔型数据不能用引号引起来，否则就变成字符串了。用方法 typeof()可以很清楚地看到这点，typeof()返回一个字符串，这个字符串的内容是变量的数据类型名称。

```
var married = true;
document.write(typeof(married) + "<br />");       //输出 boolean
```

6. 数据类型转换

在 JavaScript 中，除了可以隐式转换数据类型之外(将变量赋予另一种数据类型的值)，还可以显式转换数据类型。显式转换数据类型，可以增强代码的可读性。显式转换数据类型的方法有以下两种：将对象转换为字符串和基本数据类型转换。

(1) 数值转换为字符串。常见的数据类型转换是将数值转化为字符串，这可以通过 toString()方法，或者直接用加号在数值后加上一个长度为空的字符串。例如：

```
var a = 4;     var b = a + "";   var c = a.toString();   var d = "stu" + a;
alert(typeof(a) + " " + typeof(b) + " " + typeof(c) + " " + typeof(d));
                                                    //返回"number string string string"
var a = b = c = 5;
alert(a + b + c.toString());                        //返回 105
```

(2) 字符串转换为数值。字符串转换为数值是通过 parseInt()和 parseFloat()方法实现的，前者将字符串转换为整数，后者将字符串转换为浮点数。如果字符串中不存在数字，则返回 NaN。例如：

```
document.write(parseInt("4567red") + "<br>");          //返回 4567
document.write(parseInt("53.5") + "<br>");             //返回 53
document.write(parseInt("0xC") + "<br>");              //直接进制转换，返回 12
document.write(parseInt("tang@tom.com") + "<br>");     //返回 NaN
```

parseFloat()方法与 parseInt()方法的处理方式类似，只是会转换为浮点数(带小数)，读者可把上例中的 parseInt()都改为 parseFloat()测试验证。

3.2.4 数组

数组(array)是由名称相同的多个值构成的一个集合，集合中的每个值都是这个数组的元素。例如，可以使用数组变量 rank 来存储论坛用户所有可能的级别。

1. 数组的定义

在 JavaScript 中，数组使用关键字 Array 来声明，同时还可以指定这个数组元素的个

数,也就是数组的长度(length),例如:

```
var rank = new Array(12);                                    //第 1 种定义方法
```

如果无法预知某个数组中元素的最终个数,定义数组时也可以不指定长度,例如:

```
var myColor = new Array();
myColor[0] = "blue";
myColor[1] = "yellow";
myColor[2] = "purple";
```

以上代码创建了数组 myColor,并定义了 3 个数组项,如果以后还需要增加其他的颜色,则可以继续定义 myColor[3]、myColor[4]等,每增加一个数组项,数组长度就会动态的增长。另外还可以用参数创建数组,例如:

```
var Map = new Array("China", "USA", "Britain");       //第 2 种定义方法
Map[4] = "Iraq";
```

则此时动态数组的长度为 5,其中 Map[3]的值为 undefined。

除了用 Array 对象定义数组外,数组还可以用方括号直接定义,例如:

```
var Map = ["China", "USA", "Britain"];                //第 3 种定义方法
```

2. 数组的常用属性和方法

(1) length 属性:用来获取数组的长度,数组的位置同样是从 0 开始的。例如:

```
var Map = new Array("China", "USA", "Britain");
alert(Map.length + " " + Map[2]);                    //返回 3 Britain
```

(2) toString 方法:将数组转化为字符串。例如:

```
var Map = new Array("China", "USA", "Britain");
alert(Map.toString() + " " + typeof(Map.toString()));
```

(3) concat 方法:在数组中附加新的元素或将多个数组元素连接起来构成新数组。例如:

```
var a = new Array(1,2,3);
var b = new Array(4,5,6);
alert(a.concat(b));                                  //输出 1,2,3,4,5,6
alert(a.length);                                     //长度不变,仍为 3
```

也可以直接连接新的元素。例如:

```
a.concat(4,5,6);                                     //a 变成 1,2,3,4,5,6
```

(4) join 方法：将数组的内容连接起来，返回字符串，默认为“,”连接。例如：

```
var a = new Array(1,2,3);
alert(a.join());                          //输出 1,2,3
```

也可用指定的符号连接。例如：

```
alert(a.join("-"));                       //输出 1-2-3
```

(5) push 方法：在数组的结尾添加一个或多个项，同时更改数组的长度。例如：

```
var a = new Array(1,2,3);
a.push(4,5,6);
alert(a.length);                          //输出为 6
```

(6) pop 方法：返回数组的最后一个元素，并将其从数组中删除。例如：

```
var a1 = new Array(1,2,3);
alert(a1.pop());                          //输出 3
alert(a1.length);                         //输出 2
```

(7) shift 方法：返回数组的第一个元素，并将其从数组中删除。例如：

```
var a1 = new Array(1,2,3);
alert(a1.shift());                        //输出 1
alert(a1.length);                         //输出 2
```

(8) unshift 方法：在数组开始位置插入元素，返回新数组的长度。例如：

```
var a1 = new Array(1,2,3);
a1.unshift(4,5,6)
alert(a1);                                //输出 4,5,6,1,2,3
```

(9) slice 方法：返回数组的片断(或子数组)。该方法有两个参数，分别指定开始和结束的索引(不包括第二个参数索引本身)；如果只有一个参数该方法返回从该位置开始到数组结尾的所有项。如果任意一个参数为负，则表示是从尾部向前的索引计数。例如：－1 表示最后一个，－3 表示倒数第三个。例如：

```
var a1 = new Array(1,2,3,4,5);
alert(a1.slice(1,3));                     //输出 2,3
alert(a1.slice(1));                       //输出 2,3,4,5
alert(a1.slice(1,-1));                    //输出 2,3,4
alert(a1.slice(-3,-2));                   //输出 3
```

(10) splice 方法：从数组中替换或删除元素。第一个参数指定删除或插入将发生的位置。第二个参数指定将要删除的元素数目，如果省略该参数，则从第一个参数的位置到最后都会被删除。splice()会返回被删除元素的数组。如果没有元素被删，则返回空数组。

例如：

```
var a1 = new Array(1,2,3,4,5);
alert(a1.splice(3));                    //输出 4,5
alert(a1.length);                       //输出 3
var a1 = new Array(1,2,3,4,5);
alert(a1.splice(1,3));                  //输出 2,3,4
alert(a1.length);                       //输出 2
```

(11) sort 方法：对数组中的元素进行排序，默认是按照 ASCII 字符顺序进行升序排列。例如：

```
var a1 = new Array(1,4,23,3,5);
alert(a1.sort());                       //输出 1,23,3,4,5
var a2 = ["HTML","CSS","JavaScript","DOM"];
alert(a2.sort());                       //输出 CSS,DOM,HTML,JavaScript
```

如果要使数组中的数值型元素按大小进行排列，可以对 sort 方法指定其比较函数 compare(a,b)，根据比较函数进行排序，例如：

```
function compare(a,b) {
    return (b-a);       }          //b-a 是正数,表示逆序排列
var a1 = new Array(1,4,23,3,5);
alert(a1.sort(compare));            //输出 23,5,4,3,1
```

(12) reverse 方法：将数组中的元素逆序排列。

```
var a1 = new Array(1,4,23,3,5);
alert(a1.reverse());              //输出 5,3,23,4,1
```

3.2.5 JavaScript 语句

在任何一种编程语言中，程序的逻辑结构都是通过语句来实现的，JavaScript 也具有一套完整的编程语句用来在流程上进行判断循环等。总的来说，JavaScript 的语法与 C 或 Java 语法很相似，如果学习过这些编程语言，可以很快掌握 JavaScript 语句。

1. 条件语句

条件语句可以使程序按照预先指定的条件进行判断，从而选择需要执行的任务。在 JavaScript 中提供了 if 语句、if else 语句和 switch 语句 3 种条件判断语句。

(1) if 语句。if 语句是最基本的条件语句，它的格式为：

```
if (表达式)          //if 的判断语句,括号里是条件
  {   语句块;  }
```

如果要执行的语句只有一条，可以省略大括号把整个 if 语句写在一行。例如：

```
if(a == 1) a++;
```

如果要执行的语句有多条，就不能省略大括号，因为这些语句构成了一个语句块。例如：

```
if(a == 1) {a++; b --}
```

(2) if else 语句。如果还需要在表达式值为假时执行另外一个语句块，则可以使用 else 关键字扩展 if 语句。if else 语句的格式为：

```
if (表达式)
     {    语句块 1;     }
else
     {     语句块 2;    }
```

实际上，语句块 1 和语句块 2 中又可以再包含条件语句，这样就实现了条件语句的嵌套，程序设计中经常需要这样的语句嵌套结构。

(3) if…else if …else 语句。除了用条件语句的嵌套表示多种选择外，还可以直接用 else if 语句获得这种效果，格式为：

```
if (表达式 1)
        {  语句块 1;         }
else if (表达式 2)
        {   语句块 2;         }
else if (表达式 3)
        {   语句块 3;         }
       ⋮
else{   语句块 n;          }
```

这种格式表示只要满足任何一个条件，则执行相应的语句块，否则执行最后一条语句。下面的例子根据不同的时间显示不同的问候语。

```
<script>
    var d = new Date();  var time = d.getHours();
    if (time < 10)
    {document.write("<b>Good morning</b>");}
    else if (time > 10 && time < 16)
    {document.write("<b>Good day</b>");}
    else document.write("<b>Good afternoon</b>");
</script>
```

(4) switch 语句。在实际应用当中，很多情况下要对一个表达式进行多次判断，每一种结果都需要执行不同的操作，这种情况下使用 switch 语句比较方便。switch 语句的格式为：

```
switch (表达式)
{   case 值 1: 语句 1;          break;
    case 值 2: 语句 2;          break;
```

```
        ⋮
    case 值 n: 语句 n;          break;
    default: 语句;          }
```

每个 case 都表示如果表达式的值等于某个 case 的值,就执行相应的语句,关键字 break 会使代码跳出 switch 语句。如果没有 break,代码就会继续进入下一个 case,把下面所有 case 分支的语句都执行一遍。关键字 default 表示表达式不等于其中任何一个 case 的值时所进行的操作。

2. 循环语句

循环语句用于在一定条件下重复执行某段代码。在 JavaScript 中提供了一些与其他编程语言相似的循环语句,包括 for 循环语句、for…in 语句、while 循环语句及 do while 循环语句,同时还提供了 break 语句用于跳出循环,continue 语句用于终止当次循环并继续执行下一次循环,以及 label 语句用于标记一个语句。下面分别介绍这些循环语句。

(1) for 语句。for 循环语句是不断地执行一段程序,直到相应条件不满足,并且在每次循环后处理计数器。for 语句的格式为:

```
for (初始表达式; 循环条件表达式; 计数器表达式)
     {  语句块 }
```

for 循环最常用的形式是 for(var i=0; i<n; i++){statement},它表示循环一共执行 n 次,非常适合于已知循环次数的运算。下面是 for 循环的一个例子:九九乘法表(3-4.html)。

```
<table cellpadding="6" cellspacing="0" style="border-collapse:collapse; border:none;">
<script>
for(var i=1;i<10;i++){                          //乘法表一共 9 行
    document.write("<tr>");                     //每行是 table 的一行
     for(j=1;j<10;j++)                          //每行都有 9 个单元格
        if(j<=i)                                //有内容的单元格
           document.write("<td style='border:2px solid #004B8A; background: white;'>" +
i+"*"+j+"="+(i*j)+"</td>");
        else                                    //没有内容的单元格
            document.write("<td style='border:none;'></td>");
    document.write("</tr>");
}
</script></table>
```

(2) for…in 语句。在有些情况下,开发者根本没有办法预知对象的任何信息,更谈不上控制循环的次数。这个时候用 for…in 语句可以很好地解决这个问题。for…in 语句通常用来枚举数组的元素或对象的属性,下面是使用 for…in 循环遍历数组的例子。

```
var mycars = new Array();
mycars[0] = "Audi";mycars[1] = "Volvo";mycars[2] = "BMW";
for (x in mycars)  {
    document.write(mycars[x] + "<br />");  }
```

(3) while 语句。while 循环是前测试循环,也就是说,是否终止循环的条件判断是在执行内部代码之前,因此循环的主体可能根本不会被执行,其语法格式为:

```
while(循环条件表达式){语句块}
```

下面是 while 循环的运算示例程序,请读者思考它的输出结果。

```
var i = iSum = 0;
while(i <= 100){
    iSum += i;
    i++;  }
document.write(iSum);
```

(4) do…while 语句。与 while 循环不同,do…while 语句将条件判断放在循环之后,这就保证了循环体中的语句块至少会被执行一次,在很多时候这是非常实用的。例如:

```
var aNumbers = new Array();
var sMessage = "你输入了:\n";
var iTotal = 0, i = 0, userInput;
do{
   userInput = prompt("输入一个数字,或者'0'退出","0");
   aNumbers[i] = userInput;
   i++;
   iTotal += Number(userInput);
   sMessage += userInput + "\n";
}while(userInput != 0)          //当输入为 0(默认值)时退出循环体
sMessage += "总数:" + iTotal;
alert(sMessage);
```

(5) break 和 continue 语句。break 和 continue 语句为循环中的代码执行提供了退出循环的方法,使用 break 语句将立即退出循环体,阻止再次执行循环体中的任何代码。continue 语句只是退出当前这一次循环,根据控制表达式还允许进行下一次循环。

在上例中,没有对用户的输入做容错判断,实际上,如果用户输入了英文或非法字符,可以利用 break 语句退出整个循环。修改后的代码为:

```
do{
if(isNaN(userInput)){                  //如果不是数字
        document.write("输入错误,将立即退出<br>");
        break;  }                      //输入错误直接退出整个 do 循环体
    userInput = prompt("输入一个数字,或者'0'退出","0");
    aNumbers[i] = userInput;
    i++;
    iTotal += Number(userInput);
    sMessage += userInput + "\n";
}while(userInput != 0)                 //当输入为 0(默认值)时退出循环体
```

但上例中只要用户输入错误就马上退出了循环,而有时用户可能只是不小心按错了键,

导致输入错误,此时用户可能并不想退出,而希望继续输入,这个时候就可以用 continue 语句来退出当次循环,即用户输入的非法字符不被接受,但用户还能继续下次输入。

```
do{
if(isNaN(userInput)){
        document.write("输入错误,请重新输入<br>");
        continue;    }              //输入错误则退出当次循环,但继续下一次循环
    userInput = prompt("输入一个数字,或者'0'退出","0");
    aNumbers[i] = userInput;
    i++;
    iTotal += Number(userInput);
    sMessage += userInput + "\n";
}while(userInput != 0)               //当输入为 0(默认值)时则退出循环体
```

3.2.6 函数

函数是一个可重用的代码块,可用来完成某个特定功能。每当需要反复执行一段代码时,可以利用函数来避免重复书写相同代码。不过,函数的真正威力体现在,用户可以把不同的数据传递给它们,而它们将使用实际传递给它们的数据去完成预定的操作。在把数据传递给函数时,用户把那些数据称为参数(argument)。如图 3-3 所示,函数就像一台机器,它可以对输入的数据进行加工再输出需要的数据(只能输出唯一的值)。当这个函数被调用时或被事件触发时这个函数会执行。

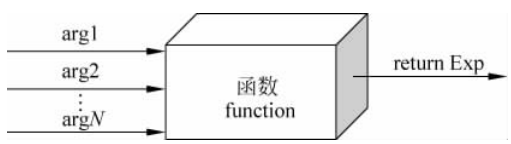

图 3-3 函数示意图

1. 函数的基本语法

函数的基本语法如下:

```
function[函数名](arg1,arg2,...,argN){
    语句
    [return[表达式]]    }
```

其中,function 是定义函数的关键字,argX 是函数的形参列表,各个参数之间用逗号隔开,参数可以为空,表示没有输入参数的函数; return 语句用来返回函数值,同样为可选项。

(1) 下面的函数接受一个输入参数 sName,不返回值。

```
function myName(sName){
    alert("Hello " + sName);    }
```

如果要调用该函数,只需把形参换成实参即可。调用它的代码如下:

```
myName("six-tang");       //弹出框显示"Hello six-tang"
```

如果希望函数返回一个值，则需使用 return 关键字接一个表达式即可。例如：

```
function fnSum(a, b){
    return a + b;  }
iResult = fnSum(52, 14);              //调用函数
alert(iResult);
```

可见，调用函数的返回值只需将函数赋给一个变量即可。

另外，与其他编程语言一样，函数在执行过程中只要执行完 return 语句就会停止继续执行函数体中的代码，因此 return 语句后的代码都不会执行。

(2) 下例函数中的 alert()语句就永远都不会执行。

```
function fnSum(iNum1, iNum2){
    return iNum1 + iNum2;
    alert (iNum1 + iNum2);            //永远不会被执行
}
```

如果函数本身没有返回值，但又希望在某些时候退出函数体，则可以调用无参数的 return 语句来随时返回函数体。例如：

```
function myName(sName){
    if (myName == "bye")
        return;                       //退出函数体
    alert("Hello" + sName);   }
```

2. 定义匿名函数

实际上，定义函数时，函数名有时都可以省略，这种函数称为匿名函数。例如：

```
function(a,b)
{ return  a + b;  }
```

但是一个函数没有了函数名，怎样调用该函数呢？调用该函数有两种方法。一种是将函数赋给一个变量(给函数赋一个名称)，那么该变量就成为这个函数对象的实例，就可以像对函数赋值一样对该变量赋予实参调用函数了。例如：

```
var sum =  function(a,b) { return  a + b;  }
 sum(3,5);                           //返回 8
```

另一种方法是函数的自运行方式。例如：

```
(function(a,b){return a + b;})(5,9)
```

为什么将函数写在一个小括号内，就能调用它呢？这是因为小括号能把表达式组合分块，并且每一块(也就是每一对小括号)都有一个返回值。这个返回值就是小括号中表达式的返回值。那么，当用一对小括号将函数括起来时，则它的返回值就是这个函数对象的实

例,假设函数对象实例为 sum,那么上面的写法就等价于 sum(5,9)。

实际上,定义函数还可以用创建函数对象的实例方法定义。例如:

```
var sum = new Function ("a","b","return a + b;")
```

这句代码的意思就是创建一个 Function 对象的实例 sum,而 Function 对象的实例就是一个函数。但这种方法显然复杂些,因此很少用。

3. 用 arguments 对象来访问函数的参数

JavaScript 的函数有个特殊的对象 arguments,主要用来访问函数的参数。通过 arguments 对象,无须指出参数的名称就能直接访问它们。例如,用 arguments[0]可以访问函数第一个参数的值,刚才的 myName 函数可以重写如下:

```
function myName(sName){
if (arguments[0] == "bye")            //如果第一个参数是“bye”
    return;
alert("Hello" + sName);   }
```

有了 arguments 对象,便可以根据参数个数的不同分别执行不同的命令,模拟面向对象程序设计中函数的重载。示例代码如下:

```
function fnAdd(){
    if(arguments.length == 0)
        return;
    else if(arguments.length == 1)
        return arguments[0] + 6;
    else{   var iSum = 0;
        for(var i = 0;i < arguments.length;i++)
            iSum += arguments[i];
        return iSum;  }}
document.write(fnAdd(44) + "<br>");              //输出结果为 50
document.write(fnAdd(45,50) + "<br>");           //输出结果为 95
document.write(fnAdd(45,50,55,70) + "<br>");     //输出结果为 220
```

3.3 对象

在客观世界中,对象是指一个特定的实体。一个人就是一个典型的对象,他包含身高、体重、年龄等特性,又包含吃饭、走路、睡觉等动作。同样,一辆汽车也是一个对象,它包含型号、颜色、种类等特性,还包含加速、拐弯等动作。

3.3.1 JavaScript 对象

在 JavaScript 中,其本身具有并能自定义各种各样的对象。例如,一个浏览器窗口可看成是一个对象,它包含窗口大小、窗口位置等属性,又具有打开新窗口、关闭窗口等方法。网页上的一个表单也可以看成一个对象,它包含表单内控件的个数、表单名称等属性,又有表

单提交(submit())和表单重设(reset())等方法。

1. JavaScript 中的对象分类

在 JavaScript 中,使用对象可分为 3 种情况。

(1) 使用自定义对象,方法是使用 new 运算符创建新对象。例如:

```
var university = new Object();          //Object 对象可用于创建一个通用的对象
```

(2) 使用 JavaScript 内置对象,如 Date、Math、Array 等。例如:

```
var today = new Date();
```

实际上,JavaScript 中的一切数据类型都是它的内置对象。

(3) 使用由浏览器提供的内置对象,如 window、document、location 等。在浏览器对象模型(BOM)将详细讲述这些内置对象的使用。

2. 对象的属性和方法

定义了对象之后,就可以对对象进行操作了,在实际中对对象的操作主要有引用对象的属性和调用对象的方法。

引用对象属性的常见方式是通过点运算符(.)实现引用。例如:

```
university.province = "湖南省";
university.name = "衡阳师范学院";
university.date = "1904";
```

university 是一个已经存在的对象,province、name 和 date 是它的 3 个属性。

从上面的例子可以看出,对象包含以下两个要素。

(1) 用来描述对象特性的一组数据,也就是若干变量,通常称为属性。

(2) 用来操作对象特性的若干动作,也就是若干函数,通常称为方法。

在 JavaScript 中,如果要访问对象的属性或方法,可使用“点”运算符来访问。

例如,假设汽车这个对象为 Car,具有品牌(brand)、颜色(color)等属性,就可以使用 Car. brand Car. color 来访问这些属性。

又如,假设 Car 关联着一些诸如 move()、stop()、accelerate(level)之类的函数,这些函数就是 Car 对象的方法,可以使用 Car. move()、Car. stop()语句来调用这些方法。

把这些属性和方法集合在一起,就得到了一个 Car 对象。也就是说,可以把 Car 对象看作是所有这些属性和方法的主体。

3. 创建对象的实例

为了使 Car 对象能够描述一辆特定的汽车,需要创建一个 Car 对象的实例(instance)。实例是对象的具体表现。对象是统称,而实例是个体。

在 JavaScript 中给对象创建新的实例也采用 new 关键字。例如:

```
var myCar = new Car();
```

这样就创建了一个 Car 对象的新实例 myCar,通过这个实例就可以利用 Car 的属性、方

法来设置关于 myCar 的属性或方法了,代码如下:

```
myCar.brand = Fiat;
myCar.accelerate(3);
```

在 JavaScript 中,字符串、数组等都是对象,严格地说所有的一切都是对象。而一个字符串变量、数组变量可看成是这些对象的实例。例如:

```
var iRank = new Array();                    //定义数组的另一种方式
var myString = new String("web design");    //定义字符串的另一种方式
```

3.3.2 with 语句

对对象的操作还经常使用 with 语句和 this 关键字,下面来讲述它们的用途。

with 语句的作用是,在该语句体内,任何对变量的引用都被认为该变量是这个对象的属性,以节省一些代码。语法如下:

```
with object{  …  }
```

所有在 with 语句后的花括号中的语句,都是在 object 对象的作用域中。例如:

```
today = new Date();
with today {
    year = getYear();                       //等价于 year = today.getYear();
    month = getMonth();                     //等价于 year = today. getMonth();
    hour = getHours();       }
```

3.3.3 this 关键字

this 是面向对象语言中的一个重要概念,在 Java、C#等大型语言中,this 固定指向运行时的当前对象。但是在 JavaScript 中,由于 JavaScript 的动态性(解释执行,当然也有简单的预编译过程),this 的指向在运行时才确定。

1. this 指代当前元素

(1) 在 JavaScript 中,如果 this 位于 HTML 标记内,即采用行内式的方式通过事件触发调用的代码中含有 this,那么 this 指代当前元素。例如:

```
<div id = "div2" onmouseover = "this.align = 'right'" onmouseout = "this.align = 'left'" >
会逃跑的文字</div>
```

此时 this 指代当前这个 div 元素。

(2) 如果将该程序改为引用函数的形式,this 作为函数的参数,则可以写成如下格式:

```
<script>
function move(obj)  {
    if(obj.align == "left"){obj.align = "right";}
```

```
    else if (obj.align == "right"){obj.align = "left";}   }
</script>
<div align = "left" onmouseover = "move(this)"> 会逃跑的文字</div>
```

此时 this 作为参数传递给 move(obj)函数，根据运行时谁调用函数指向谁的原则，this 仍然会指向当前这个 div 元素，因此运行结果和上面行内式的方式完全相同。

(3) 如果将 this 放置在由事件触发的函数体内，那么 this 也会指向事件前的元素，因为是事件前的元素调用了该函数。例如，上面的例子还可以改写成如下格式，执行效果相同。

```
<script>
stat = function(){
    var taoId = document.getElementById('div2');
    taoId.onmouseover = function(){
    this.align = "right";}                    //this 指代 taoId
    taoId.onmouseout = function(){
    this.align = "left";     }}
window.onload = stat;
</script>
<div id = "div2">会逃跑的文字</div>
```

所以，this 指代当前元素主要包括以上 3 种情况，可以简单地认为，哪个元素直接调用了 this 所在的函数，则 this 指代当前元素，如果没有元素直接调用，则 this 指代 window 对象，这是下面要讲的内容。

2. 作为普通函数直接调用时，this 指代 window 对象

(1) 如果 this 位于普通函数内，那么 this 指代 window 对象，因为普通函数实际上都是属于 window 对象的。如果直接调用，根据“this 总是指代其所有者”的原则，那么函数中的 this 就是 window。例如：

```
<script>
function doSomething()  {
    this.status = "在这里 this 指代 window 对象";  }
</script>
```

可以看到状态栏中的文字改变了，说明在这里 this 确实是指 window 对象。

(2) 如果 this 位于普通函数内，通过行内式的事件调用普通函数，又没有为该函数指定参数，那么 this 会指代 window 对象。例如，如果将“this 指代当前元素”中的函数改成如下格式，则会出错。

```
<Script>
function move()  {            //注意：该程序为典型错误写法
    if(this.align == "left"){this.align = "right";}
    else if (this.align == "right"){ this.align = "left";} }
</script>
<div align = "left" onmouseover = "move()"> 会逃跑的文字</div>
```

在这里,位于普通函数 move()中的 this 指代 window 对象,而 window 对象并没有 align 属性,所以程序会出错,当然 div 中的文字也不会移动。

3.3.4 JavaScript 的内置对象

作为一种基于对象的编程语言,JavaScript 提供了很多内置的对象,这些对象不需要用 Object()方法创建就可以直接使用。实际上,JavaScript 提供的一切数据类型都可以看成是它的内置对象,如函数、字符串等都是对象。下面将介绍两类最常用的对象,即 Date 对象和 Math 对象。

1. 时间日期:Date 对象

时间、日期是程序设计中经常需要使用的对象,在 JavaScript 中,使用 Date 对象既可以获取当前的日期和时间,也可以设置日期和时间。例如:

```
var toDate = new Date();
document.write(new Date());                    //返回当前日期和时间
```

如果 new Date()带有参数,那么就可以设置当前时间。例如:

```
new Date("July 7, 2009 15:28:30");             //设置当前日期和时间
new Date("July 7, 2009");
```

通过 new Date()显示的时间格式在不同的浏览器中是不同的。这就意味着要直接分析 new Date()输出的字符串会相当麻烦。幸好 JavaScript 还提供了很多获取时间细节的方法,如 getFullYear()、getMonth()、getDate()、getDay()、getHours()等。另外,也可以通过 toLocaleString()函数将时间日期转化为本地格式,如(new Date()).toLocaleString()。

2. 数学计算:Math 对象

Math 对象用来做复杂的数学计算。它提供了很多属性和方法,其中常用的方法有以下几种。

(1) floor(x):取不大于参数的整数。

(2) ceil(x):取不小于参数的整数。

(3) round(x):四舍五入。

(4) random(x):返回随机数(为 0～1 的任意浮点数)。

(5) pow(x, y):返回 x 的 y 次方。

3.4 浏览器对象模型 BOM

JavaScript 是运行在浏览器中的,因此提供了一系列对象用于与浏览器窗口进行交互。这些对象主要有 window、document、location、navigator 和 screen 等,把它们统称为 BOM (Browser Object Model,浏览器对象模型)。

BOM 提供了独立于页面内容而与浏览器窗口进行交互的对象。Window 对象是整个 BOM 的核心,所有对象和集合都以某种方式与 window 对象关联。BOM 中的对象关系如图 3-4 所示。下面分别介绍几个最常用对象的含义和用途。

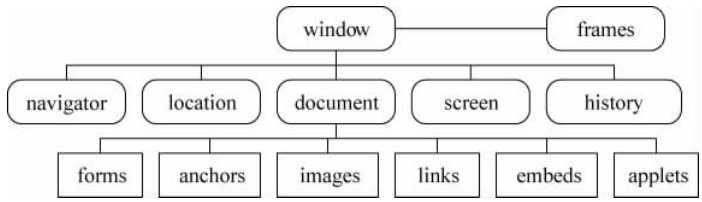

图 3-4 BOM 对象关系图

3.4.1 window 对象

window 对象表示整个浏览器窗口，但不包括其中的页面内容。window 对象可以用于移动或调整其对应的浏览器窗口的大小，或者对它产生其他影响。

在浏览器宿主环境下，window 对象就是 JavaScript 的 Global 对象，因此使用 window 对象的属性和方法是不需要特别指明的。例如，用户经常使用的 alert 方法，实际上完整的形式是 window.alert，在代码中可省略 window 对象的声明，直接使用其方法。

window 对象对应浏览器的窗口，使用它可以直接对浏览器窗口进行各种操作。window 对象提供的主要功能可以分为以下五类。

1. 调整窗口的大小和位置

window 对象有如下 4 个方法用来调整窗口的位置或大小。

(1) window.moveBy(dx, dy)：该方法将浏览器窗口相对于当前的位置移动指定的距离(相对定位)，当 dx 和 dy 为负数时则向反方向移动。

(2) window.moveTo(x, y)：该方法将浏览器窗口移动到屏幕指定的位置(x、y 处)(绝对定位)。同样可使用负数，只不过这样会把窗口移出屏幕。

(3) window.resizeBy(dw, dh)：相对于浏览器窗口的当前大小，把宽度增加 dw 个像素，高度增加 dh 个像素。两个参数也可以使用负数来缩小窗口。

(4) window.resizeTo(w, h)：把窗口大小调整为 w 像素宽、h 像素高，不能使用负数。

2. 打开新窗口和关闭窗口

打开新窗口的方法是 window.open，这个方法在 Web 编程中经常使用，但有些恶意站点滥用了该方法，频繁在用户浏览器中弹出新窗口。它的用法如下：

```
window.open([url] [, target] [, options])
```

例如：

```
window.open("pop.html", "new", "width=400, height=300");
//表示在新窗口打开 pop.html,新窗口的宽和高分别是 400 像素和 300 像素
```

target 参数除了可以使用“_self”“_blank”等常用属性值外，还可以利用 target 参数为窗口命名。例如：

```
window.open("pop.html", "myTarget");
```

这样可以让其他链接将目标文件指定在该窗口中打开。

```
<a href = "iframe.html" target = "myTarget">在指定名称为 myTarget 窗口打开</a>
<form target = "myTarget">     <! -- 表单提交的结果将会在 myTarget 窗口显示 -->
```

window.open ()方法会返回新建窗口的 window 对象,利用这个对象就可以轻松操作新打开的窗口了,代码如下:

```
var oWin = window.open("pop.html", "new", "width = 400,height = 300");
oWin.resizeTo(600,400);
oWin.moveTo(100,100);
```

注意,如果要关闭当前窗口,可使用 window.close ()。

3. 通过 opener 属性实现与父窗口交互

通过 window.open ()方法打开子窗口后,还可以让父窗口与子窗口之间进行交互。opener 属性存放的是打开它的父窗口,通过 opener 属性,子窗口可以与父窗口发生联系;而通过 open()方法的返回值,父窗口也可以与子窗口发生联系(如关闭子窗口),从而实现两者之间的互相通信和参数传递。例如:

(1) 显示父窗口名称。在子窗口中加入如下代码:

```
alert(opener.name);
```

(2) 判断一个窗口的父窗口是否已经关闭。子窗口中的代码如下:

```
if(window.opener.closed){alert("不能关闭父窗口")}
```

其中,closed 属性用来判断一个窗口是否已经关闭。

(3) 获取父窗口中的信息。在子窗口中的网页内添加如下函数:

```
function getNews() {
var parent = window.opener;
      if (!parent) return;          //如果没有父窗口则退出
//从父窗口中获取 id 为 title 的文本框中输入的内容,把它填入子窗口相关位置
    var sonTitle = document.getElementById("sonTitle");
      sonTitle.value = parent.document.getElementById("title").value;   }
```

(4) 单击父窗口中的按钮关闭子窗口(其中,oWin 是子窗口名)。

```
<input type = "button" value = "关闭子窗口" onclick = "oWin.close()" />
```

4. 系统对话框

JavaScript 可产生 3 种类型的系统对话框,即弹出对话框、确认提示框和消息提示框。它们都是通过 window 对象的方法产生的,具体方法如下。

(1) alert([message])。alert()方法前面已经反复使用,它只接受一个参数,即弹出对话框要显示的内容。调用 alert()语句后浏览器将创建一个带有“确定”按钮的消息框。

(2) confirm([message])。confirm 方法将显示一个确认提示框,包括“确定”和“取消”按钮。

用户单击“确定”按钮时,window. confirm 返回 true;单击“取消”按钮时,window. confirm 返回 false。例如:

```
if (confirm("确实要删除这张图片吗?"))                    //弹出确认提示框
    alert("图片正在删除…");
else
    alert("已取消删除!");
```

(3) prompt([message] [, default])。prompt 方法将显示一个消息提示框,其中包含一个输入框。能够接受用户输入的信息,从而实现进一步的交互。该方法接受两个参数,第一个参数是显示给用户的文本,第二个参数为文本框中的默认值(可以为空)。整个方法返回字符串,值即为用户的输入。例如:

```
var nInput = prompt ("请输入:\n 你的名字","");          //弹出消息提示框
if(nInput!= null)
document.write("Hello! " + nInput);
```

以上代码运行时弹出如图 3-5 所示的对话框,提示用户输入,并将用户输入的字符串作为 prompt()方法(函数)的返回值赋给 nInput。将该值显示在网页上,如图 3-6 所示。

图 3-5 消息提示框 prompt()

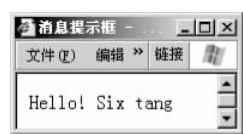

图 3-6 返回 prompt()函数的值

5. 状态栏控制(status 属性)

浏览器状态栏显示的信息可以通过 window. status 属性直接进行修改。例如:

```
window.status = "看看状态栏中的文字变化了吗?";
```

3.4.2 定时操作函数

定时操作通常有两种使用目的,一种是周期性地执行脚本,如在页面上显示时钟,需要每隔一秒钟更新一次时间的显示;另一种则是将某个操作延时一段时间执行,如迫使用户等待一段时间才能进行操作,可以使用 setTimeout()函数使其延时执行,但后面的脚本在延时期间可以继续运行不受影响。

定时操作函数还是利用 JavaScript 制作网页动画效果的基础,如网页上的漂浮广告,就是每隔几毫秒更新一下漂浮广告的显示位置。其他的定时操作函数,如打字机效果、图片轮转显示等,可以说一切动画效果都离不开定时操作函数。JavaScript 中的定时操作函数有 setTimeout()和 setInterval(),下面分别来介绍。

1. setTimeout()函数

setTimeout()函数用于设置定时器,在一段时间之后执行指定的代码。下面是 setTimeout 函数的应用实例——显示时钟(3-6.html),它的运行效果如图 3-7 所示。

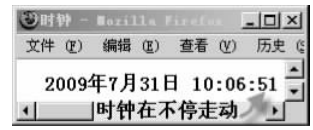

图 3-7 时钟显示效果

```
  <script>
    function $(id) {                    //根据元素 id 获取元素
      return document.getElementById(id);     }
      function dispTime() {
       $("clock").innerHTML = "<b>" + (new Date()).toLocaleString() + "</b>"; }  //将时间
加粗显示在 clock 的 div 中,new Date()获取系统时间,并转换为本地格式
    function init() {                   //启动时钟显示
      dispTime();                       //显示时间
      setTimeout(init, 1000);           //过 1 秒钟后执行一次 init()
    }
  </script>
<body onload="init()">
    <div id="clock"></div>
</body>
```

由于 setTimeout()函数的作用是过 1 秒钟之后执行指定的代码,执行完一次代码后就不会再重复地执行代码。因此 3-6.html 是通过 setTimeout()函数递归调用 init()实现每隔一秒执行一次 dispTime()函数的。

想一想:把 setTimeout(init, 1000);中的 1000 改成 200 还可以吗?

如果要清除 setTimeout()函数设置的定时器,可以使用 clearTimeout()函数,方法是将 setTimeout(init, 1000)改写成 sec = setTimeout(init, 1000),然后再使用 clearTimeout(sec)即可。

2. setInterval()函数

setInterval()函数用于设置定时器,每隔一段时间执行指定的代码。需要注意的是,它会创建间隔 ID,若不取消将一直执行,直到页面卸载为止。因此如果不需要了应使用 clearInterval 取消该函数,这样能防止它占用不必要的系统资源。它的用法如下:

```
setInterval(code, interval)
```

(1) 由于 setInterval 函数可以每隔一段时间就重复执行代码,因此 3-6.html 中的"setTimeout(init, 1000);"可以改写成如下格式:

```
setInterval(dispTime, 1000);          //每隔 1 秒钟执行一次 dispTime ()
```

(2) 这样不用递归也能实现每隔 1 秒钟刷新一次时间。下面是一个 setInterval 函数的例子,它可实现每隔 0.1 秒改变窗口大小并移动窗口位置。

```
<script>
function init() {
```

```
      window.moveTo(10,10);
      window.resizeTo(100,100);}
function move() {
      window.moveBy(10,10);              //每次向右下移动 10px
      window.resizeBy(10,10);   }        //每次变大 10px
  </script>
<body onLoad = "init()">
<b onClick = "setInterval(move, 100);">这个窗口会移动还会变大</b></body>
```

(3) 如果要清除 setInterval()函数设置的定时器,可以使用 clearInterval()函数。

3.4.3 定时操作函数的应用举例

1. 打字机效果

下面的例子用来实现打印效果,它将使变量 str 中的文字一个接一个的出现。

```
<body onLoad = "setInterval(trim, 100)">
<p id = "exp"></p>
<script>
var exp = document.getElementById("exp");
var str = "函数就像是一台机器,它对输入的数据进行加工再输出需要的数据";
y = 1;
function trim(){                          //用来定时执行的函数
    var trimstr = str.substring(0,y);     //截取从 0 到 y 的子字符串
    exp.innerHTML = trimstr;
    if(y < str.length) y++;
    else clearInterval(setInterval(trim, 100));}
</script></body>
```

2. 漂浮广告

漂浮广告的原理是向网页中添加一个绝对定位的元素,由于绝对定位元素不占据网页空间,因此会浮在网页上。下面的代码将一个 div 设置为绝对定位元素,并为它设置了 id,方便通过 JavaScript 程序操纵它。在 div 中放置了一张图片,并对这张图片设置了链接。

```
<div id = "Ad" style = "position:absolute">
<a href = "http://www.163.com" target = "_blank">
    <img src = " logo.jpg" border = "0"/></a></div>
```

接下来通过 JavaScript 脚本每隔 10 毫秒改变该 div 元素的位置,代码如下:

```
<script>
var x = 50,y = 60;                        //设置元素在浏览器窗口中的初始位置
var xin = true, yin = true;               //设置 xin、yin 用于判断元素是否在窗口范围内
var step = 1 ;                            //可设置每次移动几像素
var obj = document.getElementById("Ad") ; //通过 id 获取 div 元素
function floatAd() {
var L = T = 0;
```

```
var R = document.body.clientWidth - obj.offsetWidth;  //浏览器的宽度减 div 对象占据的空间宽度就是元素可以到达的窗口最右边的位置
var B = document.body.clientHeight - obj.offsetHeight;
obj.style.left = x + document.body.scrollLeft;        //设置 div 对象的初始位置
//当没有拉动滚动条时,document.body.scrollTop 的值是 0,当拉动滚动条时,为了让 div 对象在屏幕中的位置保持不变,就需要加上滚动的网页的高度
obj.style.top = y + document.body.scrollTop;
x = x + step*(xin?1:-1);                              //水平移动对象,每次判断左移还是右移
if (x<L) { xin = true; x = L;}
if (x>R){ xin = false; x = R;} //当 div 移动到最右边,x 大于 R 时,设置 xin = false,让 x 每次都减 1,即向左移动,直到 x<L 时,再将 xin 的值设为 true,让对象向右移动
y = y + step*(yin?1:-1)
if (y<T) { yin = true; y = T; }
if (y>B) { yin = false; y = B;}
}
var itl = setInterval("floatAd()", 10)                //每隔 10 毫秒执行一次 floatAd()
obj.onmouseover = function(){clearInterval(itl)}      //鼠标指针滑过时,让漂浮广告停止
obj.onmouseout = function(){itl = setInterval("floatAd()", 10)}  //鼠标指针离开时,继续移动
</script>
```

上述代码中,scrollTop 是获取 body 对象在网页中当拉动滚动条后网页被滚动的距离。由于 x 和 y 每次都是减 1 或加 1,因此漂浮广告总是以 45°角飘动,碰到边框后再反弹回来。

图 3-8 简单图片轮显效果

3. 简单图片轮显效果

通过每隔 2 秒钟修改 img 元素的 src 属性,就能制作出如图 3-8 所示的图片轮显效果,相应的代码如下:

```
<script>
var  n=1;
function changePic(m){
   return n=m;  }                                     //强行将 n 值改变成当前图片的 m 值
function change(){
   var myImg = document.getElementsByTagName("img")[0];  //获取图片
   myImg.src="images/0"+n+".jpg";                     //修改元素的 src 属性
   if(n<5) n++;                                       //定时函数每执行一次 n 值加 1
   else n=1;    }
</script>
<body onload="setInterval(change,2000);">
<img src="images/01.jpg" width="200" />
<div><a href="#" onclick="changePic(1)">屋檐</a>
<a href="#" onclick="changePic(2)">旅途</a><a href="#" onclick="changePic(3)">红墙</a><a href="#" onclick="changePic(4)">梅花</a><a href="#" onclick="changePic(5)">宫殿</a></div>
```

4. 用定时操作函数制作动画效果总结

(1) 首先获取需要实现动画效果的 HTML 元素,一般用 getElementById()方法。

(2) 将实现动画效果的代码写在一个函数里,如需要移动元素位置,则代码里要有改变元素位置的语句;如需改变元素属性,则代码里要有设置元素属性的语句,这样每执行一次函数就会改变对象的某些属性。

(3) 通过 setInterval()调用实现动画的函数或 setTimeout()递归调用实现动画函数的父函数,使其重复执行。

3.4.4 location 对象

location 对象的主要作用是分析和设置页面中的 URL 地址,它是 window 对象和 document 对象的属性。location 对象表示窗口地址栏中的 URL,其常用属性如表 3-5 所示。

表 3-5 location 对象的常用属性

属性	说 明	示 例
hash	URL 中的锚点部分("#"号后的部分)	#sec1
host	服务器名称和端口部分(域名或 IP 地址)	www.hynu.cn
href	当前载入的完整 URL	http://www.hynu.cn/web/123.htm
pathname	URL 中主机名后的部分	/web/123.htm
port	URL 中的端口号	8080
protocol	URL 使用的协议	http
search	执行 GET 请求的 URL 中问号(?)后的部分	?id=134&name=sxtang

其中 location.href 是最常用的属性,用于获得或设置窗口的 URL,类似于 document 的 URL 属性。改变该属性的值就可以导航到新的页面,代码如下:

```
location.href = "http://ec.hynu.cn/index.htm";
```

实际上,DW 中的跳转菜单就是使用下拉菜单结合 location 对象的 href 属性实现的。下面是跳转菜单的代码:

```
<select name = "select" onchange = "location.href = this.options[this.selectedIndex].value">
        <option>请选择需要的网址</option>
        <option value = "http://www.sohu.com">搜狐</option>
        <option value = "http://www.sina.com">新浪</option>
</select>
```

location.href 对各个浏览器的兼容性都很好,但依然会在执行该语句后执行其他代码。采用这种导航方式,新地址会被加入到浏览器的历史栈中,放在前一个页面之后,这意味着可以通过浏览器的"后退"按钮访问之前的页面。

如果不希望用户用"后退"按钮返回原来的页面,可以使用 replace()方法,该方法也能转到指定的页面,但不能返回到原来的页面了,这常用在注册成功后禁止用户后退到填写注册资料的页面。例如:

```
<p onclick = "location.replace('http://www.sohu.com');">搜狐</p>
```

可以发现转到新页面后,“后退”按钮是灰色的了。

3.4.5 document 对象

document 对象实际上又是 window 对象的子对象,document 对象的独特之处在于,它既属于 BOM 又属于 DOM。

从 BOM 角度来看,document 对象由一系列集合构成,这些集合可以访问文档的各个部分,并提供页面自身的信息。

document 对象最初是用来处理页面文档的,但很多属性已经不推荐继续使用了,如改变页面的背景颜色(document. bgColor)、前景颜色(document. fgColor)和链接颜色(document. linkColor)等,因为这些可以使用 DOM 动态操纵 CSS 属性实现。如果一定要使用这些属性,应该把它们放在 body 部分,否则对 Firefox 浏览器无效。

由于 BOM 没有统一的标准,各种浏览器中的 document 对象特性并不完全相同,因此在使用 document 对象时需要特别注意,尽量要使用各类浏览器都支持的通用属性和方法。表 3-6 所示的是 document 对象的一些常用属性。

表 3-6 document 对象的属性

集 合	说 明
anchors	页面中所有锚点的集合(设置了 id 或 name 属性的 a 标记)
embeds	页面中所有嵌入式对象的集合(由< embed >标记表示)
forms	页面中所有表单的集合
images	页面中所有图像的集合
links	页面中所有超链接的集合(设置了 href 属性的 a 标记)
cookie	用于设置或读取 cookie 的值
body	指定页面主体的开始和结束
all	页面中所有对象的集合(IE 浏览器独有)

下面是 document 对象的一些典型应用的例子。

1. 获得页面的标题和最后修改时间

document 对象的 lastModified 属性可以输出网页的最后更新时间;而它的 title 属性可以获取或更改页面的标题,下列代码的效果如图 3-9 所示。

图 3-9 获得页面的标题和最后修改时间

```
document.write(document.title + "< br />");
document.write(document.lastModified);
```

2. 将页面中所有超链接的打开方式都设置为新窗口打开

如果希望网页中所有的窗口自动在新窗口打开,除了通过网页头部的< base >标记设置外,还可以通过设置 document 对象中 links 集合的 href 属性来实现。例如:

```
< body onload = "newwin()">
< script type = "text/JavaScript">
function newwin(){
```

```
for (i = 0;i <= document.links.length - 1;i++)
    document.links[i].target = "_blank";
}</script>
<a href = "01.htm">测试 1 </a> <a href = "02.htm">测试 2 </a></ /body >
```

3. 改变超链接中原来的链接地址

在有些下载网站上,要求只有注册会员才能下载软件,会员单击下载软件的链接会转到下载页面,而其他浏览者单击该链接却是转到要求注册的页面。这可以通过改变超链接中原有链接地址的方式实现,把要求注册的链接写到 href 属性中,而如果发现是会员,就通过 JavaScript 改变该链接的地址为下载软件的页面。代码如下:

```
<body><a href  = "register.asp" > 会员可以下载 </ a >
<script type  = "text/JavaScript" >
  if( member = true )  {                                  //如果是会员
    document.links[0].href =  "download.asp" ;  }        //转到下载页面
  </script ></body >
```

当然,一般情况是通过服务器端脚本改变原来的链接地址,这样可防止用户查看源代码找到改变后的链接地址。但不管哪种方式,都是要通过 document.links 对象来实现的。

4. 用 document 对象的集合属性访问 HTML 元素

document 对象的集合属性能简便地访问网页中某些类型的元素,它是通过元素的 name 属性定位的,由于多个元素可以具有相同的 name 属性,因此这种方法访问得到的是一个元素的集合数组,可以通过添加数组下标的方式精确访问某一个元素。

例如,对于下面的 HTML 代码:

```
<img src =  "logo.gif" name =  "home"/>
<form method =  "post" action = "" name = "data">
    <input type = "text" name = "txtEmail"/>
    <input type = "submit" value = "提交"/>
</form >
```

要访问 name 属性为 home 的 img 图像,可使用 document.images["home"],但如果网页中有多个 img 元素的 name 属性相同,那么 IE 浏览器中获取到的将是最后一个 img 元素,而 Firefox 获取到的是第一个 img 元素。

访问该表单中元素可使用 document.forms["data"].txtEmail。而 document.forms[0].title.value 表示网页第一个表单中 name 属性为 title 的元素的 value 值。

但如果要访问 table、div 等 html 元素,由于 document 对象没有 tables、divs 这些集合,就不能这样访问了,要用 3.5.2 节介绍的 DOM 中访问指定结点的方法访问。

5. document 对象的 write 和 writeln 方法

document 对象有很多方法,但大部分是操纵元素的,如 document.getElementById(ID)。这些在 DOM 中再介绍,这里只介绍最简单的用 document 动态输出文本的方法。

(1) write 和 writeln 方法的用法。write 和 writeln 方法都接受一个字符串参数,在当前 HTML 文档中输出字符串,唯一的区别是 writeln 在字串末尾加一个换行符(\n)。但是

writeln 只是在 HTML 代码中添加一个换行符，由于浏览器会忽略代码中的换行符，因此以下两种方式都不会使内容在浏览器中产生换行。

```
document.write("这是第一行" + "\n");
document.writeln("这是第一行");          //等效于上一行的代码
```

要在浏览器中换行，只能再输出一个换行标记< br />，即：

```
document.write ("这是第一行" + "< br />");
```

(2) 用 document. write 方法动态引入外部 js 文件。如果要动态引入一个 js 文件，即根据条件判断，通过 document. write 输出< script >元素，就必须这样写才对：

```
if (prompt("是否链接外部脚本(1 表示是)","") == 1)
document.write("< script type = 'text/JavaScript' src = '1.js'>" + "</scr" + "ipt>");
```

注意，要将</script >分成两部分，因为 JavaScript 脚本是写在< script ></script >标记对中的，如果浏览器遇到</script >就会认为这段脚本在这里就结束了，而忽略后面的脚本代码。

3.4.6 history 和 screen 对象

history 对象主要用来控制浏览器后退和前进。它可以访问历史页面，但不能获取到历史页面的 URL。下面是 history 对象的一些用法：

```
history.go( - 1);                    //浏览器后退一页，等价于 history.back()
history.go(1);                       //浏览器前进一页，等价于 history.forward();
history.go(0);                       //浏览器刷新当前页，等价于 location.reload();
document.write(history.length);      //输出浏览历史的记录总数
```

screen 对象主要用来获取用户计算机的屏幕信息，包括屏幕的分辨率、屏幕的颜色位数、窗口可显示的最大尺寸。有时可以利用 screen 对象根据用户计算机的屏幕分辨率打开适合该分辨率显示的网页。表 3-7 所示的是 screen 对象的常用属性。

表 3-7 screen 对象的常用属性

属　性	说　明
availHeight	窗口可以使用的屏幕高度，一般是屏幕高度减去任务栏的高度
availWidth	窗口可以使用的屏幕宽度
colorDepth	屏幕的颜色位数
height、width	屏幕的高度、宽度(单位是像素)

1. 根据屏幕分辨率打开适合的网页

下面的代码首先获取用户的屏幕分辨率，然后根据不同的分辨率打开不同的网页。

```
if (screen.width == 800) { location.href = '800 * 600.htm'  }
 else if (screen.width == 1024) { location.href = '1024 * 768.htm' }
 else {self.location.href = 'else.htm' }
```

2. 使浏览器窗口自动满屏显示

在网页中加入下面的脚本代码，可保证网页打开时总是满屏幕显示。

```
<script>
    window.moveTo(0,0);
    window.resizeTo(screen.availWidth,screen.availHeight);
</script>
```

3.5 文档对象模型 DOM

文档对象模型 DOM(Document Object Module)定义了用户操纵文档对象的接口，DOM 的本质是建立了 HTML 元素与脚本语言沟通的桥梁，它使得用户对 HTML 文档有了空前的访问能力。

3.5.1 网页中的 DOM 模型

一段 HTML 代码实际上对应一棵 DOM 树。每个 HTML 元素就是 DOM 树中的一个结点，如 body 元素就是 HTML 元素的子结点。整个 DOM 模型都是由元素结点(Element Node)构成的。通过 HTML 的 DOM 模型可以获取并操纵 DOM 树中的结点，即 HTML 元素。

对于每一个 DOM 结点 node，都有一系列的属性、方法可以使用，表 3-8 所示的是结点的常用属性和方法，供读者需要时查询。

表 3-8 node 的常用属性和方法

属性/方法	返回类型/类型	说 明
nodeName	String	结点名称，元素结点的名称都是大写形式
nodeValue	String	结点的值
nodeType	Number	结点类型，数值表示
firstChild	Node	指向 childNodes 列表中的第一个结点
lastChild	Node	指向 childNodes 列表中的最后一个结点
childNodes	NodeList	所有子结点列表，方法 item(i)可以访问第 i+1 个结点
parentNode	Node	指向结点的父结点，如果已是根结点，则返回 null
previousSibling	Node	指向前一个兄弟结点，如果已是第一个结点，则返回 null
nextSibling	Node	指向后一个兄弟结点，如果已是最后一个结点，则返回 null
hasChildNodes()	Bolean	当 childNodes 包含一个或多个结点时，返回 true
attributes	NameNodeMap	包含一个元素的 Attr 对象，仅用于元素结点
appendChild(node)	Node	将 node 结点添加到 childNodes 的末尾
removeChild(node)	Node	从 childNodes 中删除 node 结点
replaceChild(newnode, oldnode)	Node	将 childNodes 中的 oldnode 结点替换成 newnode 结点
insertBefore(newnode, refnode)	Node	在 childNodes 中的 refnode 结点前插入 newnode 结点

总的来说,利用 DOM 编程在 HTML 页面中的应用可分为以下几类:

(1) 访问指定结点;

(2) 访问相关结点;

(3) 访问结点属性;

(4) 检查结点类型;

(5) 创建结点;

(6) 操作结点。

3.5.2 访问指定结点

"访问指定结点"的含义是已知结点的某个属性(如 id 属性、name 属性或结点的标记名),在 DOM 树中寻找符合条件的结点。对于 HTML 的 DOM 模型来说,就是根据 HTML 元素的 id 或 name 属性或标记名,找到指定的元素。相关方法包括 getElementById()、getElementsByName()和 getElementsByTagName()。

1. 通过元素 ID 访问元素——getElementById()方法

getElementById 方法可以根据传入的 id 参数返回指定的元素结点。在 HTML 文档中,元素的 id 属性是该元素对象的唯一标识,因此 getElementById 方法是最直接的结点访问方法。例如:

```
<body onclick = "searchDOM()">
<ul><li id = "wuli">统计物理</li></ul>
<script language = "JavaScript">
function searchDOM(){
    var wuli = document.getElementById("wuli");          //产生一个 DOM 对象 wuli
    alert(wuli.tagName + " " + wuli.childNodes[0].nodeValue); }
</script></body>
```

说明:

① 当单击网页时,将弹出如图 3-10 所示的对话框,注意元素结点名称 LI 是大写的形式。因为默认元素结点的 nodeName 都是大写的,这是 W3C 规定的。因此元素结点名并不完全等价于标记名。

图 3-10 getElementById()方法

② getElementById()方法将返回一个对象,该对象被称为 DOM 对象,它与那个有着指定 id 属性值的 HTML 元素相对应。例如,本例中变量"wuli"就是一个 DOM 对象。wuli.childNodes[0]返回该对象的第一个子结点,即"统计物理"这个文本结点。

注意,如果给定的 id 匹配某个或多个表单元素的 name 属性,那么 IE 也会返回这些元素中的第一个,这是 IE 一个非常严重的 bug,也是开发者需要注意的,因此在写 HTML 代码时应尽量避免某个表单元素的 name 属性值与其他元素的 id 属性值重复。

提示: IE 浏览器可以根据元素的 ID 直接返回该元素的 DOM 对象,如将上例中的"var wuli = document.getElementById("wuli");"删除后,IE 中仍然有相同效果。

2. 通过元素 name 访问元素——getElementsByName 方法

getElementsByName 方法也查找所有元素对象,只是返回 name 属性为指定值的所有

元素对象组成的数组，但可以通过添加数组下标使其返回这些元素对象中的一个。例如：

```
var tj = document.getElementsByName("tongji")[0];
```

3. 通过元素的标记名访问元素——getElementsByTagName 方法

getElementsByTagName 是通过元素的标记名来访问元素的，它将返回一个具有某个标记名的所有元素对象组成的数组。例如，下面的代码将返回文档中 li 元素的集合。

```
<body onclick = "searchDOM()">
<ul>客户端编程
        <li><a href = "#">HTML</a></li>
        <li>CSS</li>
        <li>JavaScript</li>
    </ul>
<script>
function searchDOM(){
    var myul = document.getElementsByTagName("ul")[0];          //获取第一个 ul
    alert(myul.tagName + " " + myul.childNodes[0].nodeValue);
    var sib = myul.getElementsByTagName("*")                    //获取该 ul 的所有子孙元素
    alert(sib[1].tagName + " " + sib[2].childNodes[0].nodeValue);
}</script></body>
```

上述代码运行时将先后弹出两个警告框，第一个警告框中显示“UL 客户端编程”，因为网页中第一个 ul 元素的标记名显然是 UL，而 ul 元素的第一个子结点是文本结点，它的值是“客户端编程”。

第二个警告框中会显示 A CSS，因为 ul 元素总共有 4 个子元素，即 li、a、li、li。其中，sib[1].tagName 指其中第 2 个元素的元素名，即 A；而第 3 个元素的子结点是文本结点，它的值是 CSS。

注意，getElementById()获取到的是单个元素，所以 Element 没有 s，而 getElementsByTagName 和 getElementsByName 获取到的是一组元素，所以是“Elements”。它们要获取单个元素必须添加数组下标，切记。

4. 访问指定结点的子结点

如果要访问一个指定结点的子结点，那么第一步是要找到这个指定结点，然后可以通过以下两种方法找到它的子结点。例如，有下列 HTML 代码：

```
<ul id = "nav">
  <li><a href = "">E-cash</a></li>
  <li><a href = "">微支付</a></li>
 </ul>
```

如果要访问 #nav 下的所有 li 元素，那么首先可以用 getElementById()方法找到 #nav 元素，代码如下：

```
var navRoot = document.getElementById("nav");
```

(1) 第一种方法是在 DOM 对象 navRoot 中再次使用 getElementsByTagName 搜寻它的子结点。代码如下：

```
var navli = navRoot.getElementsByTagName("li");
```

(2) 第二种方法是使用 childNodes 集合获取 navRoot 对象的子结点。

```
var navli = navRoot.childNodes;
```

这两种方法返回的 navli 都是一个数组，可以使用循环语句输出该数组中的所有元素，例如：

```
for(var i = 0;i < navli.length;i++)          //逐一查找
    DOMString += navli[i].nodeName + "\n";
    alert(DOMString);
```

第一种方法在 IE7-和 Firefox 浏览器中的输出结果完全相同，都输出两个子结点“LI”；而第二种方法在 IE7-和 Firefox 中的运行结果分别如图 3-11 和图 3-12 所示。

图 3-11 IE 中有两个子结点

图 3-12 Firefox 中的子结点

这种差异是因为 Firefox 在计算元素的子结点时，不只计算它下面的元素子结点，连元素之间的回车符也被当成文本子结点计算进来了，由此计算出的子结点个数是 5。因此，如果要找一个元素中所有同一标记的子元素，应尽量使用第一种方法，这样可避免 Firefox 把回车符当成文本子结点计算的麻烦。

当然，如果一定要用第二种方法，并且兼容 IE7-和 Firefox 浏览器，可以在获取子结点前加一条判断语句。代码如下：

```
for(var i = 0;i < navli.length;i++)
    if (navli[i].nodeType == 1) DOMString += navli[i].nodeName + "\n";
    alert(DOMString);
```

如果要访问第一个子结点，还可以使用 firstChild，访问最后一个结点则是 lastChild。例如，var navli = navRoot.firstChild，它将返回一个元素对象，但对于 IE 浏览器来说，第一个子结点是“LI”，而在 Firefox 中，第一个子结点是“#text”，同样存在不一致的问题。

5. 访问某些特殊结点

如果要访问文档中的 html 结点或 body 结点等特殊结点，以及 BOM 中具有的某些元素集合，除了使用上面的通用方法外，还可以使用表 3-9 中所示的方法。

表 3-9 访问特殊元素结点的方法

要访问的元素	方 法
html	var htmlnode = document. documentElement
head	varbodynode = document. documentElement. firstChild;
body	document. body
超链接元素	var nava= document. links[n]
img 元素	var img = document. images[n]
form 元素	var reg= document. forms ["reg"]
form 中的表单域元素	var email= document. forms["reg"]. txtEmail

说明：

① 访问 html 元素应该使用 document. documentElement，而不是 document. html。对 html 元素使用 firstChild 方法就可以得到 head 元素，在 Firefox 中也是如此。这说明 Firefox 只是在求 body 结点及其下级结点的子结点时才会计算文本子结点(如回车符)。

② 由于 document 对象具有 links、images 和 forms 等集合，因此访问这类元素可以使用相应的集合名带数组下标找到指定的元素，或者使用“集合名["name 属性"]”方法。例如，表 3-9 中的 reg、txtEmail 都是指定元素的 name 属性值。

3.5.3 访问和设置元素的 HTML 属性

在找到需要的结点(元素)之后通常希望对其属性进行读取或修改。DOM 定义了 3 个方法来访问和设置结点的 HTML 属性，它们是 getAttribute(name)、setAttribute(name, value)和 removeAttribute(name)，它们的作用如表 3-10 所示。

表 3-10 访问和设置元素 HTML 属性的 DOM 方法

方 法	功 能	举 例
getAttribute(name)	读取元素属性	myImg. getAttribute("src")
setAttribute(name, value)	修改元素属性	myImg. setAttribute("src","02. jpg")
removeAttribute(name)	删除元素属性	myImg. removeAttribute("title")

实际上，用户也可以不使用以上 3 种方法，直接通过(DOM 元素. 属性名)获取元素的 HTML 属性，通过(DOM 元素. 属性名="属性值")设置或删除元素的 HTML 属性。这种方法和表 3-10 中方法的区别在于，表 3-10 中的方法可以访问和设置元素自定义的属性(如对< img >标记自定义一个 author 属性)，而这种方法只能访问和设置 HTML 语言中已有的属性，但一般都不会去自定义 HTML 属性，因此这种方法完全够用，本节中主要采用这种方法。

1. 读取元素的 HTML 属性

下面的代码首先获取一个 img 图像元素，然后读取该元素的各种属性并输出。

```
<body onload = "init()">
  <img src = "images/01. jpg" alt = "沙漠古堡" class = "west"/>
<script>
  function init() {
```

```
        var myImg = document.getElementsByTagName("img")[0];  //获取元素
        alert(myImg.src);
        alert(myImg.alt);                              //等价于 alert(myImg.getAttribute("alt"));
        alert(myImg.className);                                  //输出"west"
      }
</script></body>
```

说明：使用 myImg.alt 就可以读取 myImg 元素的 alt 属性，它和 myImg.getAttribute("alt")有等价的效果。对于 class 属性，由于 class 是 JavaScript 的关键字，因此访问该属性时必须将它改写成 className。

2. 设置元素的 HTML 属性

图 3-13 所示的是一个图像依据鼠标指针指向文字的不同而变换的效果。当鼠标指针滑动到某个 li 元素上时，就动态地改变左边 img 元素的 src 属性，使其切换显示图片。该实例的代码如下(在与该代码的同级目录下有 3 个图像文件 pic1.jpg、pic2.jpg 和 pic3.jpg)。

图 3-13 图片跟随文字变换的效果

```
<style type="text/css">
#container ul {
      margin:8px;  padding:0;  list-style:none;   border:1px dashed red;}
#container li {
      font:24px/2 "黑体"; }
</style>
<div id="container"><img src="pic1.jpg" id="picbox" style="float:left;" />
<ul><li onmouseover="changePic(1)">沙漠古堡</li>
      <li onmouseover="changePic(2)">天山冰湖</li>
      <li onmouseover="changePic(3)">自然村落</li></ul>
</div>
<script>
function changePic(n){                                  //必须写在#picbox 元素后面
      var myImg = document.getElementById("picbox"); //获取 img 元素
      myImg.src="pic"+n+".jpg";  }                      //设置 myImg 的 src 属性为某个 jpg 文件
</script>
```

3. 删除元素的 HTML 属性

通过(DOM 元素.属性名=="")就可以删除一个元素的 HTML 属性值，例如：

```
<img src="pic1.jpg" title="沙漠古堡" onclick="changePic()" width="200" class="bk" />
```

```
<script>
function changePic(){
    var myImg = document.getElementsByTagName("img")[0];  //获取图片
    myImg.title = "";                                      //删除 title 属性
    myImg.className = "";                                  //删除 class 属性
    myImg.removeAttribute("width");                        //删除 width 属性
}</script>
```

提示：

① 由于 width 属性和 CSS 中的 width 属性同名，因此不能用 myImg.width=""删除。

② removeAttribute()可以删除元素的任何 HTML 属性，只是与 getAttribute()一样，对于 class 属性在 IE 中必须把“class”写成“className”，而在 Firefox 中“class”又只能写成“class”，因此，解决的办法是把两条都写上或使用 myImg.className=""来删除。

3.5.4 访问和设置元素的内容

如果要访问或设置元素的内容，一般使用 innerHTML 属性。innerHTML 可以将元素的内容（起始标记和结束标记之间）改变成其他任何内容（如文本或 HTML 元素）。innerHTML 虽然不是 DOM 标准中定义的属性，但大多数浏览器都支持，因此不必担心浏览器兼容问题。下面的例子当鼠标指针滑到 span 元素上时，将读取和改变该元素的内容。

```
<span id = "a" onmouseover = "change()"><b>把鼠标指针移过来，我会变</b></span>
<script>
function change(){
var a = document.getElementById("a");
alert(a.innerHTML)                    //读取元素中的 HTML 内容，输出"<B>把鼠标…</B>"
a.innerHTML = "看见变化了吗?";        //设置元素中的 HTML 内容
}
</script>
```

下面是一个例子。当选中表单中的复选框后，将为 span 元素添加内容，取消选中则清空 span 元素的内容。运行效果如图 3-14 所示。

```
<form name = "userInfo" method = "post" action = "">您有小孩吗?有:
<input type = "checkbox" name = "hasBoy" id = "hasBoy" value = "1" onclick = "check()" />
  <span id = "add"> </span></form>
<script>
function check(){
var hasboy = document.forms["userInfo"].hasBoy;
var add = document.getElementById("add");              //去掉 var 后 IE 将出错
if(hasboy.checked)
    add.innerHTML = "有几个<input type = 'text' name = 'textfield' />";
else
    add.innerHTML = "";
}</script>
```

图 3-14 利用 innerHTML 改变元素的内容

提示：对于要设置 innerHTML 属性的 DOM 元素来说，最好要对它进行显式定义，不能去掉“var”，或者确保 DOM 对象名和元素 id 不同名(如将变量 add 改成 add2)，否则在 IE 中将会出错。因为 IE 有时会把元素 id 直接当在 DOM 元素对象来使用。

innerHTML 属性可更改元素中的 HTML 内容，如果只需要更改元素中的文本内容，可以使用 innerText 方法，它只能更改标记中文本的内容，但它只支持 IE 浏览器，在 Firefox 浏览器中要使用 textContent 属性才能实现相同的效果。

3.5.5 访问和设置元素的 CSS 属性

在 JavaScript 中，除了能够访问元素的 HTML 属性外，还能够访问和设置元素的 CSS 属性，访问和设置元素的 CSS 属性可分为以下两种方法。

1. 使用 style 对象访问和设置元素的行内 CSS 属性

style 对象是 DOM 对象的子对象，在建立了一个 DOM 对象后，可以使用 style 对象来访问和设置元素的行内 CSS 属性。语法为：DOM 元素. style. CSS 属性名。可以看出用 DOM 访问 CSS 属性和访问 HTML 属性的区别在于 CSS 属性名前要有“style.”。例如：

```
  <p id = "test" onclick = " $ ()" style = "font - size:14px; color: # 000;">内容</p>
<script>
  function $ (){
  var test = document.getElementById("test");
  alert (test.style.fontSize);              //访问 CSS 属性 font - size,输出 14px
  test.style.color = " # f00";              //修改 CSS 属性 color
  alert (test.style.color);                 //访问 CSS 属性 color,IE 中输出 # f00
}</script>
```

说明：

① 样式设置必须符号 CSS 规范，否则该样式会被忽略。

② 如果样式属性名称中不带-号，如 color，则直接使用 style. color 就可访问该属性值；如果样式属性名称中带有-号，如 font-size，则对应的 style 对象属性名称为 fontSize。转换规则是去掉属性名称中的-号，再把后面单词的第一个字母大写即可。例如，border-left-style 对应的 style 对象属性名为 borderLeftStyle。

③ 对于 CSS 属性 float，不能使用 style. float 访问，因为 float 是 JavaScript 的保留字，要访问该 CSS 属性，在 IE 浏览器中应使用 style. styleFloat，在 Firefox 浏览器中应使用 style. cssFloat。

④ 使用 style 对象只能读取到元素的行内样式，而不能获得元素所有的 CSS 样式。如果将上例中 p 元素的 CSS 样式改为嵌入式的形式，那么 style 对象是访问不到的。因此 style 对象获取的属性与元素的最终显示效果并不一定相同，因为可能还有非行内样式作用

于元素。

⑤ 如果使用 style 对象设置元素的 CSS 属性，而设置的 CSS 属性和元素原有的任何 CSS 属性冲突，由于 style 会对元素增加一个行内 CSS 样式属性，而行内 CSS 样式的优先级最高，因此通过 style 设置的样式一般为元素的最终样式。

下面的例子通过修改 div 元素的 CSS 背景图片属性实现图 3-13 中图像随文字切换效果。

```
#container #picbox {width:150px;    height:150px;    float:left;
                    background:url(pic1.jpg) no-repeat;  }
<div id="container">
      <div id="picbox"></div>       <!--用 div 放置供切换的背景图像-->
      <ul><li onmouseover="changePic(1)">沙漠古堡</li>
        <li onmouseover="changePic(2)">天山冰湖</li>
        <li onmouseover="changePic(3)">自然村落</li>
      </ul></div>
<script>
function changePic(str){
 var myImg = document.getElementById("picbox");              //获取图像元素
 myImg.style.backgroundImage="url(pic"+str+".jpg)"; }     //设置 CSS 背景属性
</script>
```

如果要为当前元素设置多条 CSS 属性，可以使用 style 对象的 cssText 方法，例如：

```
var a = document.getElementById("a");
a.style.cssText="border:1px dotted;width:300px;height:200px;background:#c6c6c6;"
```

2. 使用 className 属性切换元素的类名

为元素同时设置多条 CSS 属性还可以将该元素原来的 CSS 属性和修改后的 CSS 属性分别写到两个类选择器中，再修改该元素的 class 类名以调用修改后的类选择器。例如，下面的例子同样可用来实现图 3-13 中的图片切换效果。

```
<style type="text/css">
.pic1{background:url(pic1.jpg)}                /*将要修改的 CSS 属性放在一个类选择器中*/
.pic2{background:url(pic2.jpg)}
.pic3{background:url(pic3.jpg)}
</style>
<div id="container">
<div id="picbox" class="pic1"></div>
<ul><li onmouseover="changePic(1)">沙漠古堡</li>
  <li onmouseover="changePic(2)">天山冰湖</li>
  <li onmouseover="changePic(3)">自然村落</li>
</ul></div>
<script>
function changePic(str){
 var myImg = document.getElementById("picbox");  //获取图片
 myImg.className="pic"+str;  }                   //切换#picbox 元素的类名
</script>
```

提示：如果要删除元素的所有类名，设置 DOM 元素.className=""即可。

3. 使用 className 属性追加元素的类名

有时元素可能已经应用了一个类选择器中的样式，如果想要使元素应用一个新的类选择器，但又不能去掉原有的类选择器中的样式，则可以使用追加类名的方法，当然这种情况也可以通过 style 对象添加行内样式实现同样效果。

但是，当追加元素的类名不是为了控制该元素的样式，而是为了控制其子元素的样式(如下拉菜单)时，就只能用这种方法实现。下面是一个追加元素类别的例子：

```
className + = " over"
```

提示：在"与 over 之间的空格一定不能省略，因为 CSS 中为元素设置多个类别名的语法是：class="test over"(多个类名间用空格隔开)，因此添加一个类名一定要在前面加空格，否则就变成了 class="testover"，这显然不对。

4. 使用 replace 方法去掉元素的某一个类名

如果要在元素已经应用的几个类名中去掉其中的一个时，则可以使用 replace 方法，将类名替换为空即可。例如：

```
this.className = this.className.replace(/over/,"");     //用两斜杠/将 over 括起来
```

假设元素的类名原来是 class="test over"，则去掉后变成了 class="test"。要去掉的类名一定要用两斜杠/括起来，如果用引号，则在 Firefox 中会不起作用。

下面是追加和删除某一特定类名的例子，当单击导航项时，将显示折叠菜单，再次单击导航项时又将隐藏折叠菜单。

```
<style type = "text/css">
.test{   width: 160px;    border: 1px solid #ccc; }
li ul { display: none;  }
li.over ul { display: block;}
</style>
<ul id = "nav">
  <li class = "test" onclick = "toggle(this)"><a href = "#">文 章</a>
    <ul><li><a href = "#"> Ajax 教程</a></li>
      <li><a href = "#"> Flex 教程</a></li></ul>
    </li></ul>
<script>
function toggle(obj){
if (obj.className.indexOf("over") ==- 1)                    //如果类名中没有"over"
     obj.className += " over";                              //追加类名"over"
else   obj.className = obj.className.replace(/over/,"");  }   //去除类名"over"
</script>
```

3.5.6 创建和替换元素结点

1. DOM 结点的类型

DOM 中的结点主要有 3 种类型，分别是元素结点、属性结点和文本结点。例如，一个 a

元素：

```
<a href = "iframe.html" target = "myTarget">在指定窗口打开</a>
```

则该 a 元素中的各种结点如图 3-15 所示。

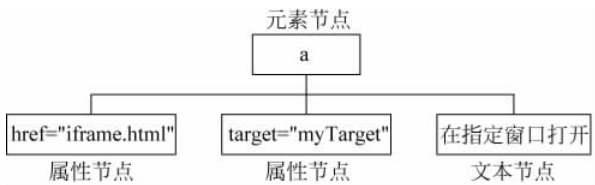

图 3-15 各种结点的关系

在 DOM 中可以使用结点的 nodeType 和 nodeName 属性检查结点的类型，其值的含义如表 3-11 所示。

表 3-11 DOM 结点的 nodeType 和 nodeName

DOM 结点的属性	元 素 节 点	属 性 节 点	文 本 节 点
nodeType	1	2	3
nodeName	元素标记名的大写	属性名称	#text

2. 创建结点

除了查找结点并处理结点的属性外，DOM 同样提供了很多便捷的方法来管理结点，包括创建、删除、替换和插入等操作。在 DOM 中创建元素结点采用 creatElement()，创建文本结点采用 createTextNode()，创建文档碎片结点采用 createDocumentFragment()等。

(1) createElement 方法：创建 HTML 元素。使用该方法可以在文档中动态创建新的元素。例如，希望在网页中动态添加如下代码：

```
<p>这是一条感人的新闻</p>
```

则首先可以利用 createElement()创建< p >元素，代码如下：

```
var oP = document.createElement("p");
```

然后利用 createTextNode()方法创建文本结点，并利用 appendChild()方法将其添加到 oP 结点的 childNodes 列表的最后，代码如下：

```
var oCont = document.createTextNode("这是一条感人的新闻");
oP.appendChild(oCont);
```

最后再将已经包含了文本结点的元素< p >结点添加到< body >中，同样可采用 appendChild()方法，代码如下：

```
document.body.appendChild(oP);
```

这样便完成了< body >中< p >元素的创建，appendChild()方法是向元素的尾部追加结点，因此创建的 p 元素总是位于 body 元素的尾部。

(2) createTextNode 方法：创建文本结点。

```
var txt = document.createTextNode("some text");
```

可以首先创建一个“模板”结点，创建新结点时首先调用 cloneNode 方法获得“模板”结点的副本，然后根据实际应用的需要对该副本结点进行局部内容的修改。

3. 操作结点

操作 DOM 结点可以使用标准的 DOM 方法，如 appendChild()、removeChild()等，也可以使用非标准的 innerHTML 属性。DOM 中可以使结点发生变化的常用方法包括以下几种。

(1) appendChild()：为当前结点新增一个子结点，并且将其作为最后一个子结点。

(2) insertBefore()：为当前结点新增一个子结点，将其插入到指定的子结点之前。

(3) replaceChild()：将当前结点的某个子结点替换为其他结点。

(4) removeChild()：删除当前结点的某个子结点。

这里以 replaceChild()替换结点方法来展示用 DOM 操作结点的方法。下列代码的功能是在单击文本时，将文本所在的 p 结点替换成 h1 结点。

```
<p onclick="replaceP()">这行文字被替换了</p>
<script>
function replaceP(){
    var oOldP = document.getElementsByTagName("p")[0];
    var oNewP = document.createElement("h1");              //新建元素结点
    var oText = document.createTextNode("这是一个感人至深的故事");
    oNewP.appendChild(oText);
    oOldP.parentNode.replaceChild(oNewP,oOldP);  }         //替换结点
</script>
```

3.5.7 用 DOM 控制表单

1. 访问表单中的元素

每个表单中的元素，无论是文本框、单选按钮、下拉列表框还是其他内容，都包含在 form 的 elements 集合中，可以利用元素在集合中的位置或元素的 name 属性获得对该元素的引用。代码如下：

```
var oForm = document.forms["user"];          //user 为该 form 元素的 name 属性
var oTextName = oForm.elements[0];           //该 form 中的第一个表单域元素
var passwd = oForm.elements["passwd"];       //passwd 为该表单域元素的 name 属性
```

另外，还有一种最简便的方法，就是直接通过表单元素的 name 属性来访问，例如：

```
var oComments = oForm.elements. passwd;        //获取 name 属性为 comments 的元素
```

经验：虽然也可以用 document.getElementById()和表单元素的 id 值来访问某个特定的元素。但由于表单中的元素要向服务器传送数据，一般都具有 name 属性，因此用 name 属性的方法来访问更加方便，除了像单选按钮组各项之间的 name 值相同，而 id 值不同。

2. 表单中元素的共同属性和方法

所有表单中的元素(除了隐藏元素)都有一些共同的属性和方法，其中常用的部分列在表 3-12 中。

表 3-12 表单中元素的共同属性和方法

属性/方法	说 明
checked	对于单选按钮和复选框而言，选中则为 true
defaultChecked	对于单选按钮和复选框而言，如果初始时是选中的则为 true
value	除下拉菜单外，所有元素的 value 属性值
defaultValue	对于文本框和多行文本框而言，初始设定的 value 值
form	指向元素所在的< form >
name	元素的 name 属性
type	元素的类型
blur()	使焦点离开某个元素
focus()	聚焦到某个元素
click()	模拟用户单击该元素
select()	对于文本框、多行文本框而言，选中并高亮显示其中的文本

对于表 3-12 中的各个属性和方法，读者可以逐一试验。例如：

```
var oForm = document.forms["myForm1"];                    //获取表单 myForm1
var oComments = oForm.elements.comments;
alert(oComments.type) ;                                   //返回元素类型(输出 text)
var oTextPasswd = oForm.elements["passwd"];
oTextPasswd.focus();                                      //聚焦到 passwd 元素上
```

3. 用表单的 submit()方法代替提交按钮

在 HTML 中，表单的提交必须采用提交按钮或具有提交功能的图像按钮才能够实现，例如：

```
< input type = "submit" name = "Submit" value = "登 录" />
```

用户单击“提交”按钮就可提交表单。但在很多场合中用其他方法提交却显得更为便捷，如选中某个单选按钮，选择了下拉列表框中某一项后就让表单立即提交。只要在相应的元素事件中加入下面这条事件处理代码即可。

```
document.formName.submit();
```

或

```
document.forms[index].submit();
```

这两条语句使用了表单对象的 submit()方法,该方法等效于单击 submit 按钮。

通过采用 submit()方法提交表单,还可以把验证表单的程序写在提交表单之前。下面是用一个超链接(a 元素)模拟提交按钮实现表单提交的例子。在提交之前还验证了用户名是否为空。

```
<script>
   function checkvalue() {
     if (document.welcome.username.value == "" )
       { alert( " 用户名不能为空! " );
        return ( false );     }
     document.welcome.submit();
    return ( true );     }
  </script>
<form name = " welcome"  method = "post"  action = "" >
      <input type = "text" name = "username" />
      <input type = "text" name = "password"/>
      <a href = " # " onclick = "checkvalue();return false: " >登 录</a>
  </form>
```

在 2.3.6 节曾提到提交按钮是表单的三要素之一,但这个观点现在需要改变了。利用 submit()方法代替提交按钮的功能,可以使表单不再需要提交按钮。

3.6 事件处理

事件是 JavaScript 和 DOM 之间进行交互的桥梁,当某个事件发生时,通过它的处理函数执行相应的 JavaScript 代码。例如,页面加载完毕后,会触发 load 事件,用户单击元素时,会触发 click 事件。通过编写这些事件的处理函数,可以实现对事件的响应,如向用户显示提示信息,改变这个元素或其他元素的 CSS 属性。

3.6.1 事件流

浏览器中的事件模型分为两种,即捕获型事件和冒泡型事件。捕获型事件是指事件从最不特定的事件目标传播到最特定的事件目标。例如下面的代码中,如果单击 p 元素,那么捕获型事件模型的触发顺序是 body→div→p。早期的 NN 浏览器采用这种模型。

```
<script>
function add(sText){
    var oDiv = document.getElementById("display");
    oDiv.innerHTML += sText;  }       //输出发生事件的元素顺序
</script>
<body onclick = "add('body<br>');">
    <div onclick = "add('div<br>');">
        <p onclick = "add('p<br>');"> Click Me </p>
    </div>
    <div id = "display"></div>
</body>
```

而 IE 等浏览器采用了事件冒泡的方式，即事件从最特定的事件目标传播到最不特定的事件目标。而且目前大部分浏览器都是采用了冒泡型事件模型，上例中的代码在 IE 和 Firefox 浏览器中的显示结果如图 3-16 所示，可看到它们都是采用事件冒泡的方式。因此这里主要讲解冒泡型事件。但是 DOM 标准则吸取了两者的优点，采用了捕获＋冒泡的方式。

图 3-16 IE 和 Firefox 浏览器均采用冒泡型事件

3.6.2 处理事件的两种方法

1. 事件处理函数

用于响应某个事件而调用的函数称为事件处理函数，事件处理函数既可以通过 JavaScript 进行分配，也可以在 HTML 中指定。因此事件处理函数出现的形式可分为以下两类。

(1) HTML 标记事件处理程序：这是最常见的一种事件处理形式，它直接在 HTML 标记中的事件名后书写事件处理函数。形式为：

```
<Tag eventhandler = "JavaScript Code">
```

例如：

```
<p onclick = "alert('我的内容是' + this.innerHTML);"> Click Me </p>
<button id = "btn" onclick = "alert('你好')" > Click Me </button>
```

这种方法简单，而且在各种浏览器中的兼容性也很好。

(2) 以对象属性的形式出现的事件监听程序：这种方法没有把 JavaScript 代码写在 HTML 标记内，实现了结构和行为的分离，它的形式为：

```
object.eventhandler = function;
```

例如：

```
<script>
window.onload = function(){
    var oP = document.getElementById("myP");          //找到对象
    oP.onclick = function(){                          //设置事件监听函数
        alert('我被点击了');
    }}</script>
<p id = "myP"> Click Me </p>
```

注意：

① 这种形式的“＝”后只能跟函数名或匿名函数。例如，上述 oP.onclick = function(){…} 还可以写为 oP.onclick =msg，但绝不能写成 oP.onclick =msg()。msg 是下面函数的函数名：

```
function msg(){ alert('我被点击了'); }
```

② 将这段程序放在 window 对象的 onload 事件中,保证了 DOM 结构完全加载后再搜索< p >结点。如果去掉 window.onload = function(){},就必须保证这段代码放在 # myP 元素的下面,否则就会出现找不到元素的错误。

③ 这种方法最大的优点是可以统一对很多元素设置事件处理程序。假设页面中很多元素对同一事件都会采用相同的处理方式,如果在每个元素的标记内都添加一条事件处理程序就会有很多代码冗余。下面是一个用 JavaScript 模仿 a 标记 hover 伪类效果的例子:

```
< p onmouseover = " this. style. textDecoration = ' underline ' " onmouseout = " this. style.
textDecoration = 'none'">第一段</p>
<p onmouseover = " this. style. textDecoration = ' underline ' " onmouseout = " this. style.
textDecoration = 'none'">第二段</p>
<p onmouseover = " this. style. textDecoration = ' underline ' " onmouseout = " this. style.
textDecoration = 'none'">第三段</p>
```

从代码中可以看出,如果使用 HTML 标记事件处理程序的话,那么每个标记内都要写一段相同的事件处理代码。而使用事件监听程序,就可以把上述代码改为:

```
<script>
window.onload = function()
{   var ps = document.getElementsByTagName("p");
    for (var p in ps)    {
           ps[p].onmouseover = function()
            {  this.style.textDecoration = "underline"    };
            ps[p].onmouseout = function()
            {  this.style.textDecoration = "none"     };
        }  };
</script>
```

这样所有< p >标记中的 onmouseover="…"就可以去掉了,而运行效果完全一样。

2. 通用事件监听程序

事件处理函数使用便捷,但是这种传统的方法不能为一个事件指定多个事件处理函数,事件属性只能赋值一种方法,考虑下面的代码:

```
button1.onclick = function() { alert('你好'); };
button1.onclick = function() { alert('欢迎'); };
```

则后面的 onclick 事件处理函数就会将前面的事件处理函数覆盖了。在浏览器中预览只会弹出一个显示“欢迎”的警告框。

正是由于事件处理函数存在上述功能上的缺陷,就需要通用事件监听函数。事件监听函数可以作用于多个元素,不需要为每个元素重复书写,同时事件监听函数可以为一个事件添加多个事件处理方法。

(1) IE 中的事件监听函数。在 IE 浏览器中,有两个函数来处理事件监听,分别是

attachEvent()和 detachEvent()。其中,attachEvent()用来给某个元素添加事件处理函数,而 detachEvent()则是用来删除元素上的事件处理函数。例如:

```
<script>
function fnClick1(){
    alert("我被点击了");
    oP.detachEvent("onclick",fnClick1);}                    //单击了一次后删除监听函数
function fnClick2(){
    alert("我的内容是" + myP.innerHTML);}
window.onload = function(){
    oP = document.getElementById("myP");                    //找到对象
    oP.attachEvent("onclick",fnClick1);                     //添加监听函数
    oP.attachEvent("onclick",fnClick2);  }
</script>
<p id = "myP">Click Me</p>
```

通过以上代码可以看出 attachEvent()和 detachEvent()的使用方法,它们都接受两个参数,前一个参数表示事件名,而后一个参数是事件处理函数的名称。

这种方法可以为同一个元素添加多个监听函数。在 IE 浏览器中运行时,当用户第一次单击 p 元素会接连弹出两个对话框,而单击了一次以后,监听函数 fnClick1()被删除,再单击就只会弹出一个对话框了,这也是前面的方法所无法实现的。

(2) Firefox 中的事件监听函数(标准 DOM 的监听方法)。Firefox 等其他非 IE 浏览器采用标准 DOM 监听函数进行事件监听,即 addEventListener()和 removeEventListener()。与 IE 不同之处在于,这两个函数接受 3 个参数,即事件名、事件处理的函数名和是用于冒泡阶段还是捕获阶段。

这两个函数接受的第一个参数"事件名"与 IE 也有区别,事件名是 click mouseover 等,而不是 IE 中的 onclick 或 onmouseover,即事件名没有 on 开头。另外,第 3 个参数通常设置为 false,即冒泡阶段。例如:

```
<script>
function fnClick1(){
    alert("我被 fnClick1 监听了");
    oP.removeEventListener("click",fnClick1, false);  //删除监听函数 1  }
function fnClick2(){
    alert("我被 fnClick2 监听了");  }
var oP;
window.onload = function(){
    oP = document.getElementById("myP");              //找到对象
    oP.addEventListener("click",fnClick1, false);     //添加监听函数 1
    oP.addEventListener("click",fnClick2, false);     //添加监听函数 2  }
</script>
<p id = "myP">Click Me</p>
```

在 Firefox 浏览器中运行该程序时,当第一次单击 p 元素时,会接连弹出两个对话框,顺序是"我被 fnClick1 监听了"和"我被 fnClick2 监听了"。当以后再次单击时,由于第一次单击后删除了监听函数 1,就只会弹出一个对话框了,内容是"我被 fnClick2 监听了"。

3.6.3 浏览器中的常用事件

1. 事件的分类

对于用户而言,常用的事件无非是鼠标事件、HTML 事件和键盘事件,其中鼠标事件的种类如表 3-13 所示。

表 3-13 鼠标事件的种类

事 件 名	描 述	事 件 名	描 述
onclick	单击鼠标左键时触发	onmouseover	鼠标指针移动到元素上时触发
ondbclick	双击鼠标左键时触发	onmouseout	鼠标指针移出该元素边界时触发
onmousedown	鼠标任意一个按键按下时触发	onmousemove	鼠标指针在某个元素上移动时持续触发
onmouseup	松开鼠标任意一个按键时触发		

常用的 HTML 事件如表 3-14 所示。

表 3-14 常用的 HTML 事件

事 件 名	描 述
onload	页面完全加载后在 window 对象上触发,图片加载完成后在其上触发
onunload	页面完全卸载后在 window 对象上触发,图片卸载完成后在其上触发
onerror	脚本出错时在 window 对象上触发,图像无法载入时在其上触发
onselect	选择了文本框的某些字符或下拉列表框的某项后触发
onchange	文本框或下拉列表框内容改变时触发
onsubmit	单击“提交”按钮时在表单 form 上触发
onblur	任何元素或窗口失去焦点时触发
onfocus	任何元素或窗口获得焦点时触发

对于某些元素来说,还存在一些特殊的事件。例如,body 元素就有 onresize(当窗口改变大小时触发)和 onscroll(当窗口滚动时触发)这样的特殊事件。

键盘事件相对来说用得较少,主要有 keydown(按下键盘上某个按键触发)、keypress(按下某个按键并且产生字符时触发,即忽略 Shift、Alt 等功能键)和 keyup(释放按键时触发)。通常键盘事件只有在文本框中才有实际意义。

2. 事件的应用举例——设置鼠标指针经过时自动选择表单中文本

有时希望当鼠标指针经过文本框时,文本框能自动聚焦,并能选中其中的文本以便用户直接输入就可修改。其中实现鼠标指针经过时自动聚焦的代码如下:

```
<input name="user" type="text" onmouseover="this.focus()" />
```

其次是聚焦后自动选中文本框中的文本,代码如下:

```
<input name="user" value="tang" type="text" onfocus="this.select()"/>
```

如果表单中有很多文本框,不希望在每个文本框标记中都写上这些事件处理代码,则可

改写成如下的通用事件处理函数。

```
<script>
function myFocus(){
    this.focus();  }
function mySelect(){
    this.select();  }
window.onload = function(){
    var elements = document.getElementsByTagName("input");
     for (var i = 0; i < elements.length; i++) {
        var type = elements[i].type;          //获取 input 标记的 type 属性值
        if (type == "text") {
          elements[i].onmouseover = myFocus;
          elements[i].onfocus = mySelect;
        } } }
</script>
```

3. 事件的应用举例——利用 onBlur 事件自动校验表单

过去,表单验证都是在表单提交时进行验证,即当用户输入完表单后单击“提交”按钮时再进行验证。随着 Ajax 技术的兴起,现在表单的输入验证一般在用户输入完一项转到下一项时,对刚输入的一项进行验证。即输完一项验证一项,也就是在前一输入项失去焦点(onBlur)时进行验证。这样的好处很明显,在用户输入错误后可马上提示用户进行修改,还可防止提交表单后如果有错误要求用户重新输入所有的信息。这种效果的制作步骤如下。

(1) 写结构代码。该例的结构代码是一个包含有文本框、密码框和提交按钮的表单,考虑到失去焦点时要返回提示信息,在各个文本框后面添加一个用于显示提示信息的标记。表单<form>的 HTML 代码如下:

```
<form name="register">
<table cellpadding="5" cellspacing="0" border="0">
    <tr><td>用户名:</td><td><input type="text" name="User"></td>
    <td><span id="UserResult"></span></td></tr>
    <tr><td>输入密码:</td><td><input type="password" name="passwd1"></td>
<td></td></tr>
    <tr><td>确认密码:</td><td><input type="password" name="passwd2"></td>
<td><span id="pwdResult"></span></td></tr>
    <tr><td colspan="2" align="center">
        <input type="submit" value="注册"><input type="reset" value="重置">
        </td><td></td></tr></table>
</form>
```

(2) 当文本框或密码框获得焦点时改变其背景色,以便突出显示,失去焦点时其背景色又恢复为原来的背景色。代码如下:

```
<script>
function myFocus(){
    this.style.backgroundColor="#fdd";  }
function myBlur(){
```

```
    this.style.backgroundColor = "#fff";     }
window.onload = function(){
    var elements = document.getElementsByTagName("input");
     for (var i = 0;i < elements.length;i++) {
        var type = elements[i].type;
       if (type == "text" || type == "password") {
          elements[i].onfocus = myFocus;
          elements[i].onblur = myBlur;
        } } }
```

(3) 当文本框或密码框失去焦点时开始验证该文本框中的输入是否合法,在这里仅验证文本框的输入是否为空,以及两次输入的密码必须相同。

① 由于要在失去焦点时验证,因此在函数 myBlur()中添加执行验证函数的代码,将上述代码中的 myBlur()修改为:

```
function myBlur(){
    this.style.backgroundColor = "#ffffff";
    startCheck(this);}                              //这一句是新增的验证表单的代码
```

② 然后编写验证函数 startCheck()的代码,它的代码如下:

```
function startCheck(oInput){
    if(oInput.name == "User"){                      //如果是用户名的输入框
      if(!oInput.value){                            //如果值为空
        oInput.focus();                             //聚焦到用户名的输入框
        document.getElementById("UserResult").innerHTML = "用户名不能为空";
        return;  }
      else
        document.getElementById("UserResult").innerHTML = "";     }
      if(oInput.name == "passwd2"){                 //如果是第二个密码输入框
        if(document.getElementsByName("passwd1")[0].value!= document.getElementsByName("
passwd2")[0].value)                                 //如果两个密码框值不相等
document.getElementById("pwdResult").innerHTML = "两次输入的密码不一致";
      else
        document.getElementById("pwdResult").innerHTML = "";}}
```

这个在 onBlur 事件中验证表单输入的程序最终效果如图 3-17 所示。如果能够添加与服务器交互的服务器端脚本,就能实现验证“用户名是否已经被注册”等功能。

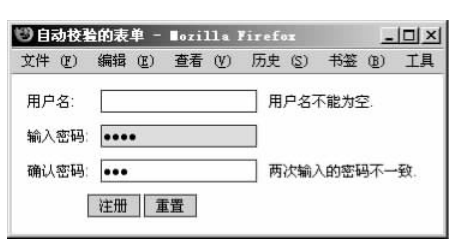

图 3-17 利用 onBlur 事件自动校验的表单

3.6.4 事件对象

1. IE 和 DOM 中的事件对象

当在 IE 浏览器中发生一个事件时，浏览器将会自动创建一个名称为“event”的事件对象，在事件处理函数中可以通过访问该对象来获取所发生事件时的各种信息，包括触发事件的 HTML 元素、鼠标指针位置及鼠标按钮状态等。在 IE 浏览器中，event 对象实际上又是 window 对象的一个属性 event，因此在代码中可以通过 window.event 或 event 形式来访问该对象。

尽管它是 window 对象的属性，但 event 对象还是只能在事件发生时被访问，所有的事件处理函数执行完之后，该对象就自动消失了。而标准 DOM 中规定，event 对象必须作为唯一的参数传给事件处理函数。因此在类似 Firefox 浏览器中访问事件对象通常将其作为参数，代码如下：

```
oP.onclick = function(oEvent){
}
```

因此为了兼容这两种浏览器，通常采用下面的方法。

```
oP.onclick = function(oEvent){
oEvent = oEvent || window.event;    }
```

浏览器在获取了事件对象后就可以通过它的一系列属性和方法来处理各种具体事件了，如鼠标事件、键盘事件和浏览器事件等。对于鼠标事件来说，其常用的属性是它的位置信息属性，主要有以下两类。

(1) screenX/screenY：事件发生时，鼠标指针在计算机屏幕中的坐标。

(2) clientX/cilentY：事件发生时，鼠标指针在浏览器窗口中的坐标。

通过鼠标指针的位置属性，可以随时获取到鼠标指针的位置信息。例如，有些电子商务网站可以将商品用鼠标指针拖放到购物篮中，这就需要获取鼠标事件的位置，才能让商品跟着鼠标指针移动。

2. 键盘事件对象的应用举例——验证用户输入的是否为数字

如果要判断用户在文本框中输入的内容是否为数字，最简单的办法就是用键盘事件对象来检测按下键的键盘码是否是为 48～57，当用户按下的不是数字键时，会发现根本无法输入。示例代码如下：

```
<script>
function IsDigit()
{  return ((event.keyCode >= 48) && (event.keyCode <= 57));  }
</script>
请输入手机号码:
<input type = "text" name = "phone" onkeypress = "event.returnValue = IsDigit();"/>
```

3. 鼠标事件对象的应用举例——制作跟随鼠标指针移动的图片放大效果

本例中，当鼠标指针滑动到某张图片上时，鼠标指针的旁边就会显示这张图片的放

大图片，而且放大的图片会跟随鼠标指针移动，如图 3-18 所示。在整个例子中，原图和放大的图片采用的都是同一张图片，只不过对原图设置了 width 和 height 属性，使它缩小显示，而放大图片就显示图片的真实大小。制作步骤如下。

图 3-18 跟随鼠标指针移动的图片放大效果

(1) 把几张要放大的图片放到一个 div 容器中，然后再添加一个 div 的空容器用来放置当鼠标指针经过时显示的放大图像。结构代码如下：

```
<div id = "demo">
    <img src = "pic1.jpg" /> <img src = "pic2.jpg" /> <img src = "pic3.jpg" />
</div>
<div id = "enlarge_img"></div>     <! -- 用来放置放大的图片 -->
```

当然，严格来说，把这几幅图片放到一个列表中结构会更清晰些。

(2) 写 CSS 代码，对于 img 元素来说，只要定义它在小图时的宽和高，并给它添加一条边框以显得美观。对于 enlarge_img 元素，它应该是一个浮在网页上的绝对定位元素，在默认时不显示，并设置它的 z-index 值很大，防止被其他元素遮盖。

```
    #demo img{
    width:90px;   height:90px;              /* 页面中小图的大小 */
    border:5px solid #f4f4f4; }
#enlarge_img{
    position:absolute;
    display:none;                           /* 默认状态不显示 */
    z-index:999;                            /* 位于网页的最上层 */
    border:5px solid #f4f4f4  }
```

(3) 对鼠标指针在图片上移动这一事件对象进行编程。首先获取到 img 元素，当鼠标指针滑动到它们上面时，使 #enlarge_img 元素显示，并且通过 innerHTML 往该元素中添加一个图像元素作为大图。大图在网页上的纵向位置(即距离页面顶端的距离 top)应该是鼠标指针到窗口顶端的距离(event.clientY)加上网页滚动过的距离(document.body.scrollTop)。代码如下：

```
<script>
var demo = document.getElementById("demo");
var gg = demo.getElementsByTagName("img");              //获取 #demo 中的 img 元素集合
var ei = document.getElementById("enlarge_img");
for(i = 0; i<gg.length; i++){
    var ts = gg[i];
    ts.onmousemove = function(event){                   //鼠标指针在某个 img 元素上移动时
        event = event || window.event;                  //兼容 IE 和标准 DOM 事件
        ei.style.display = "block";                     //显示装大图的盒子
```

```
        ei.innerHTML = '<img src="' + this.src + '" />';  //设置大图盒子中的图像路径
        ei.style.top = document.body.scrollTop + event.clientY + 10 + "px";
                                                          //大图在页面上的位置
        ei.style.left = document.body.scrollLeft + event.clientX + 10 + "px";
    }
    ts.onmouseout = function(){                           //鼠标指针离开时
        ei.innerHTML = "";
        ei.style.display = "none";  }
    ts.onclick = function(){  window.open( this.src );    //单击大图时在新窗口打开图片
}}
</script>
```

这样该实例就制作好了，注意 JavaScript 代码在这里只能放在结构代码的后面，当然也可以把这些 Javascript 代码作为一个函数放在 Window.onload 事件中。

习题 3

一、选择题

1. 下列定义数组的方法中(　　)是不正确的。

   A. var x=new Array["item1" , "item2" , "item3" , "item4"];

   B. var x=new Array("item1" , "item2" , "item3" , "item4");

   C. var x= ["item1" , "item2" , "item3" , "item4"];

   D. var x=new Array(4);

2. 计算一个数组 x 的长度的语句是(　　)。

   A. var aLen=x.length();　　B. var aLen=x.len ();

   C. var aLen=x.length;　　D. var aLen=x.len;

3. 下列 JavaScript 语句将显示(　　)结果。

```
var a1 = 10;
var a2 = 20;
alert("a1 + a2 = " + a1 + a2);
```

   A. a1+a2=30　　B. a1+a2=1020

   C. a1+a2=a1+a2　　D. "a1+a2="1020

4. 产生当前日期的方法是(　　)。

   A. Now();　　B. date();　　C. new Date();　　D. new Now();

5. 下列(　　)可以得到文档对象中的一个元素对象。

   A. document.getElementById("元素 id 名")

   B. document.getElementByName("元素名")

   C. document.getElementByTagName("元素标签名")

   D. 以上都可以

6. 如果要制作一个图像按钮，用于提交表单，方法是(　　)。

   A. 不可能的

B. <input type="button" image="image.gif">

C. <input type="submit" image="image.gif">

D. <img src="image.gif" onclick="document.forms[0].submit()">

7. 如果要改变元素 … 的背景颜色为蓝色,代码是(　　)。

A. document.getElementById("userInput").style.color = "blue";

B. document.getElementById("userInput").style.divColor = "blue";

C. document.getElementById("userInput").style.background-color = "blue";

D. document.getElementById("userInput").style.backgroundColor = "blue";

8. 通过 innerHTML 的方法改变某一 div 元素中的内容,(　　)。

A. 只能改变元素中的文字内容　　B. 只能改变元素中的图像内容

C. 只能改变元素中的文字和图像内容　　D. 可以改变元素中的任何内容

9. 下列选项中,(　　)不是网页中的事件。

A. onclick　　B. onmouseover　　C. onsubmit　　D. onmouseclick

10. JavaScript 中自定义对象时使用关键字(　　)。

A. Object　　B. Function

C. Define　　D. 以上 3 种都可以

11. 以下语句不能为对象 obj 定义值为 22 的属性 age 的是(　　)。

A. obj."age"=22;　　B. obj.age=22;

C. obj["age"]=22;　　D. obj={age:22};

12. 下面语句不能定义函数 *f*()的是(　　)。

A. function f(){};　　B. var f=new Function("{}");

C. var f=function(){};　　D. f(){};

二、填空题

1. _____对象表示浏览器的窗口,可用于检索关于该窗口状态的信息。

2. Navigator 对象的_____属性用于检索操作系统平台。

3. "var a = 10; var b = 20; var c = 10; alert(a = b); alert(a == b); alert(a == c);"结果是_____。

三、简答题

试说明以下代码输出结果的顺序,并解释其原因,最后在浏览器中验证。

```
<script>
    setTimeout(function(){
        alert("A");
        },0);
    alert("B");
</script>
```

四、实操题

编写代码实现以下效果:打开一个新窗口,原始大小为 400×300px,然后将窗口逐渐增大到 600×450px,保持窗口的左上角位置不变。