第 3 章　Web 开发必会的客户端技术

客户端技术是 Web 程序中最重要的技术之一。客户端技术主要用来设置浏览器中页面元素的布局和显示效果，以及利用 JavaScript 技术对页面进行控制，与服务端进行通信等。常用的客户端技术主要包括 HTML、CSS、DOM、JavaScript 和 AJAX 等。读者通过学习这些客户端技术，可以很容易地编写具有良好用户体验的 Web 程序页面。本章将主要介绍 JavaScript 的基本功能，同时还涉及了 CSS、DOM、AJAX 等技术。本章的主要内容如下：

- 常用 JavaScript IDE 简介；
- JavaScript 的基本语法；
- JavaScript 中的内置对象；
- DOM 技术；
- 正则表达式；
- CSS 基础知识；
- AJAX 技术。

3.1　学习客户端技术的开发工具

工欲善其事，必先利其器。要想提高客户端技术的开发效率，选择一个合适的开发工具是非常必要的。现在几乎所有能开发 Web 程序的开发工具都支持客户端技术，如 MyEclipse、Eclipse、NetBeans、Dreamweaver 和 Visual Studio 2010 等。这些工具在编写客户端技术上各有千秋。在本节将讲解如何在 MyEclipse 开发工具中使用各种客户端技术。

3.1.1　在 MyEclipse 中使用 HTML 技术

MyEclipse 是本书主要使用的 HTML IDE。MyEclipse 不仅支持 Java EE 的开发，也支持 HTML 的开发。读者可右击 Web 工程，在弹出的快捷菜单中选择 New|Other 命令，打开 New 对话框。在该对话框中选择 MyEclipse|Web|HTML(Basic Templates)节点，如图 3.1 所示。

单击 Next 按钮，进入下一步操作。在 File name 文本框中输入一个文件名（不需要带文件扩展名），单击 Finish 按钮就会创建一个带有“.html”后缀名的 HTML 程序文件。

MyEclipse 中的 HTML 编辑器支持代码辅助功能，如图 3.2 所示。

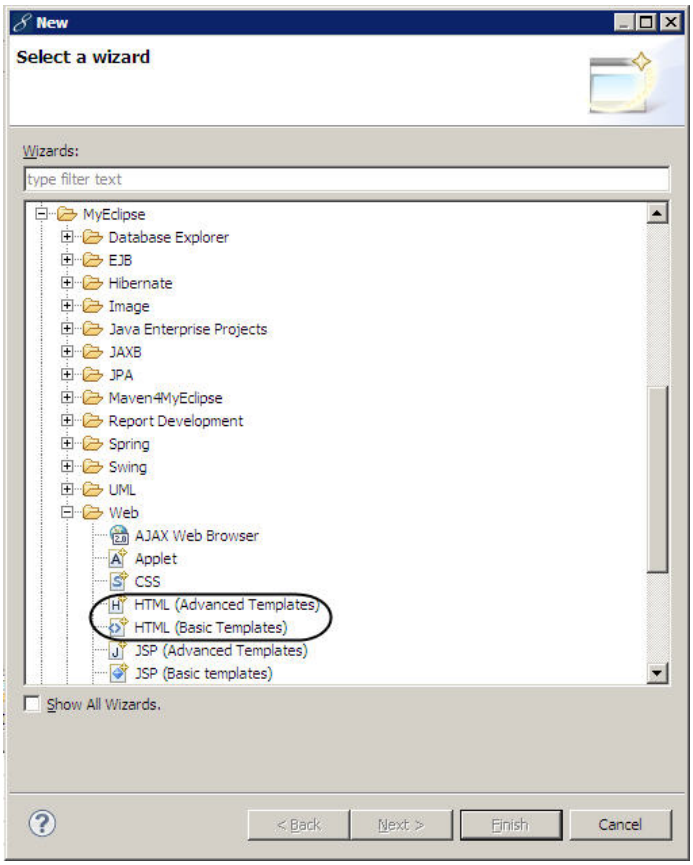

图 3.1　新建 HTML 文件

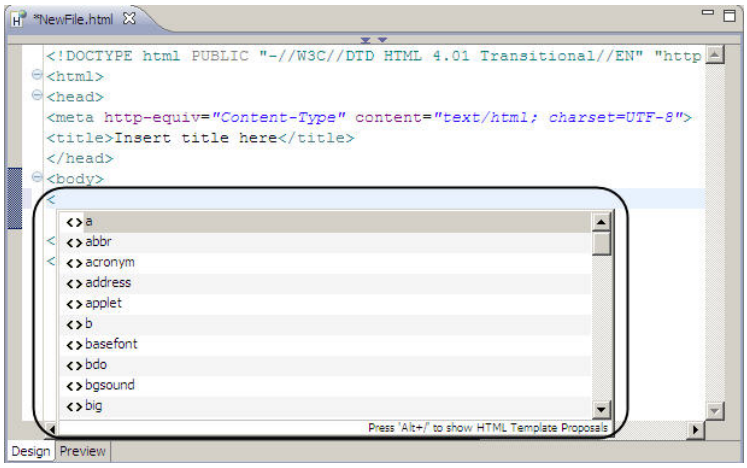

图 3.2　支持 HTML 语言

3.1.2　在 MyEclipse 中使用 JavaScript 技术

MyEclipse 是本书主要使用的 Java IDE。MyEclipse 不仅支持 Java EE 的开发，同时也

支持 JavaScript 开发。读者可右击 Web 工程，在弹出的快捷菜单中选择 New|Other 命令，打开 New 对话框。在该对话框中选择 JavaScript|JavaScript Source File 节点，如图 3.3 所示。

单击 Next 按钮，进入下一步操作。在 File name 文本框中输入一个文件名（不需要带文件扩展名），单击 Finish 按钮就会创建一个带有“.js”后缀名的 JavaScript 程序文件。

MyEclipse 中的 JavaScript 编辑器支持代码辅助功能，即在 JavaScript 的内置对象，如 window、document 等，在输入“.”后，都可以列出这些对象的内部成员。如在输入“window.”后，就会出现如图 3.4 所示的成员列表，同时还会显示支持这些成员的浏览器，如图 3.5 所示。

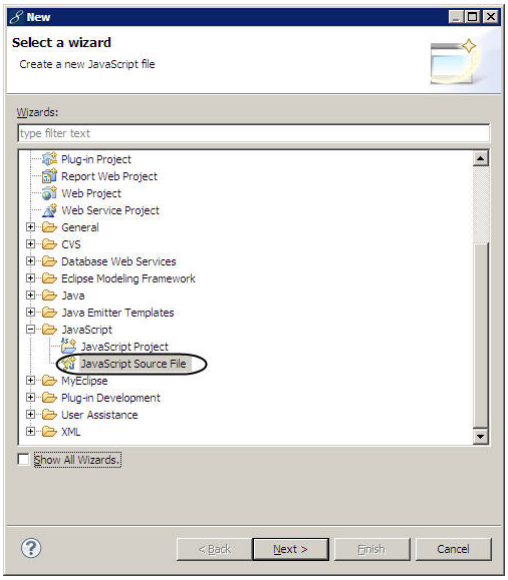

图 3.3　新建 JavaScript 文件

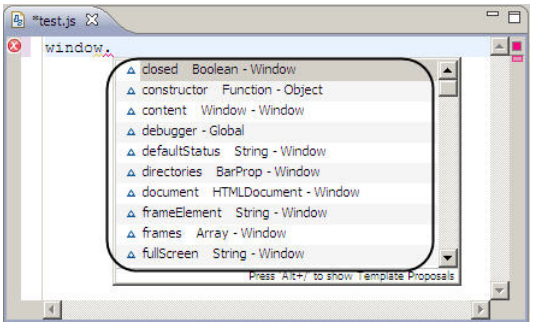

图 3.4　window 对象的成员列表

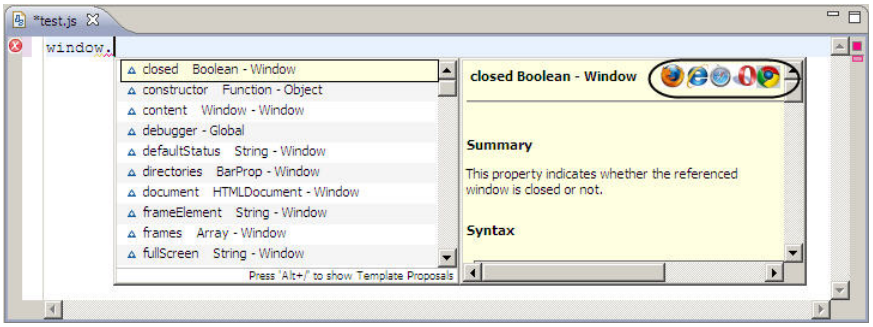

图 3.5　所支持的浏览器

3.1.3　在 MyEclipse 中使用 CSS 技术

MyEclipse 是本书主要使用的 HTML IDE。MyEclipse 不仅支持 Java EE 的开发，同时也支持 HTML 开发。读者可右击 Web 工程，在弹出的快捷菜单中选择 New|Other 命令，打开 New 对话框。在该对话框中选择 MyEclipse|Web|CSS 节点，如图 3.6 所示。

单击 Next 按钮，进入下一步操作。在 File name 文本框中输入一个文件名（不需要带文件扩展名），单击 Finish 按钮就会创建一个带有“.css”后缀名的 CSS 程序文件。

MyEclipse 中的 CSS 编辑器支持代码辅助功能，如图 3.7 所示。

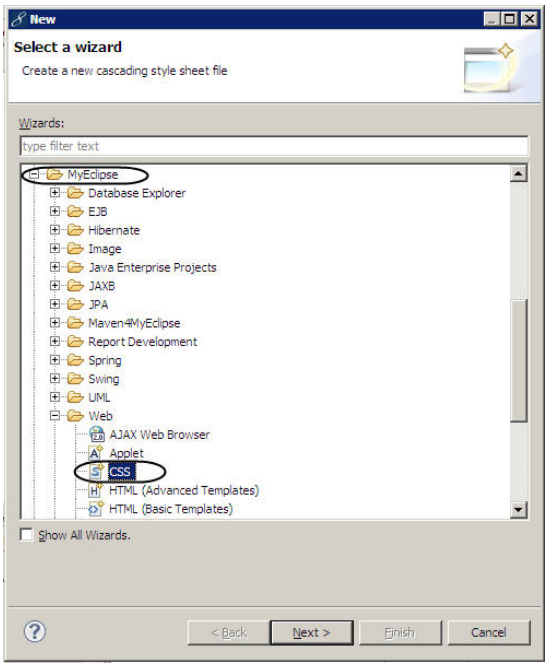

图 3.6　新建 CSS 文件

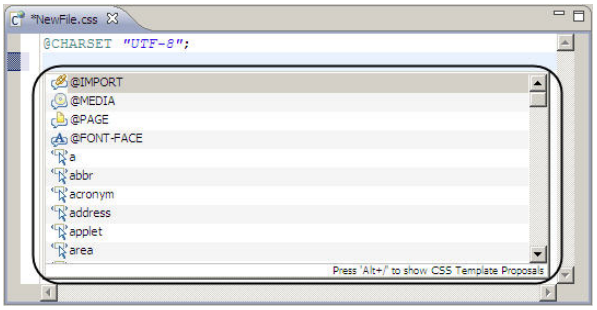

图 3.7　支持 CSS 语言

3.2　学习超文本标签语言 HTML

HTML 的全称为 Hypertext Markup Language，中文意思是超文本标签语言，是一种用来制作超文本文档的简单标签语言，也就是用于制作网页的内容。本节将介绍 HTML 的基本语法，内容包含该语言的基本结构和基本标签，分别是段落格式标签、超级链接标签、图像标签、表格标签、框架标签和表单标签。

3.2.1　HTML 基本构成

HTML 语言是 Web 用于创建和识别文档的标准语言，该语言是学习网页编程的基础，其主要应用就是对网页内容的排版。HTML 最大的优点就是在浏览器运行时有统一的规则和可用标准。所谓超文本，是因为其可以加入图片、声音、动画、影视等非单一文本内容。一个简单 HTML 文档的代码如下：

```
<!--  html1.html  -->
<HTML>                                          <!--  HTML 文档的开始标签  -->
    <HEAD>                                      <!--  HTML 文档的开头标签  -->
        <TITLE>第一个 HTML 文档：Hello</TITLE><!--  HTML 文档的标题标签  -->
    </HEAD>
    <BODY>                                      <!--  HTML 文档的主体标签  -->
        第一个 HTML 文档!
    </BODY>
</HTML>
```

代码说明：

保存上面的文档为 html1.html 文件，双击该文件，页面的显示如图 3.8 所示。上述代码中之所以会产生如图 3.8 所示的结果，最主要是因为代码中的一系列标签，下面将一一介绍这些标签。

（1）<HTML></HTML>

<HTML>标签用来标识 HTML 文档的开始，而</HTML>标签 HTML 文档的结束，两者必须同时出现配套使用，这是 HTML 语言的最基本标签。

（2）<HEAD></HEAD>

HTML 文档开始后，首先遇到的就是该文档的开头部分，其用<HEAD>和</HEAD>标签来标识。在该标签对中可以使用<title></title>和<script></script>等标签，用来描述 HTML 文档的相关信息。

（3）<BODY></BODY>

HTML 文档的开头部分结束后，就进入文档的主体部分，用<BODY>和</BODY>标签来标识。在此标签对之间可以使用<p><h1>
等标签，用来描述浏览器中显示出来的内容。同时<BODY>标签中还可以有以下属性。

- Bgcolor：设置背景颜色。
- Text：设置文本的颜色。
- Link：设置链接的颜色。
- Vlink：设置已使用的链接的颜色。
- Alink：设置被单击的链接的颜色。

（4）<title></title>

在使用<HEAD>和</HEAD>标签对时，会出现一对非常重要的标签<title></title>，用来修改浏览器窗口标题栏上的文本信息。

通过对基本标签的理解，可以把上述代码分成两部分，如图 3.9 所示。

图 3.8　HTML 文档的结构

图 3.9　运行结果

注意：标签不区分大小写，所以可以使用<html>代替<HTML>标签。

3.2.2　HTML 基本标签——段落格式设置标签

在编写一篇长文章或评论的时候，经常会把文章内容分成一系列段落，以显示作者的逻辑思想，在 HTML 文档中也可以把文本内容分成多个段落。段落标签可以是：

（1）

标签用来创建一个段落，在此标签对之间加入的文本，将按照段落的格式显示在浏览器中。另外，该标签还可以使用 align 属性来说明对齐方式，而属性值可以是 left、Center、和 Right 这 3 个中的任何一个。

注意：段落标签 用于标记段落的开始，段落结束标签 并非必须，是可以省略的。

（2）

是一个很简单的标签，它没有结束标签，因为它用来创建一个回车换行。

如果把
放在 外边，将创建一个大的回车换行，若放在 里将创建一个小的回车换行。

下面代码的具体内容为标签
在标签对 外。

```
<!--  html2.html  -->
<HTML>
    <HEAD>
        <TITLE>区别</TITLE>
    </HEAD>

    <BODY >
        <p>在标签对外</p><br>     <!--  <br>标签在<p>标签的外边  -->
        <p>在标签对外</p>
    </BODY>
</HTML>
```

下面代码的具体内容为标签
在标签对 里。

```
<!--  html3.html  -->
<HTML>
    <HEAD>
        <TITLE>区别</TITLE>
    </HEAD>

    <BODY >
```

```
        <p>在标签对里<br></p>      <!--  <br>标签在<p>标签的里边  -->
        <p>在标签对里</p>
    </BODY>
</HTML>
```

运行上面两段代码结果如图 3.10 和图 3.11 所示。

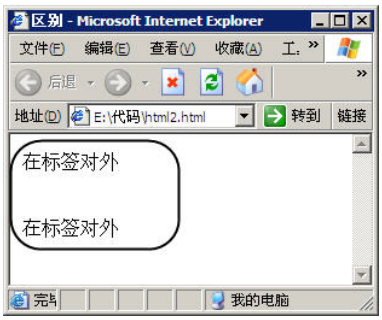

图 3.10　外边运行结果

图 3.11　里边运行结果

（3） 、

标签对用于文档分节，也用于格式化表，此标签对的用法与 标签对非常相似。标签用于在行内控制特定内容的显示，如果要为一行内容的某几个字设置特殊的格式，可以使用该标签将这几个文字包围起来，然后设置格式。

3.2.3　HTML 基本标签——超级链接标签

在 HTML 语言中超级链接用来实现页面的联系和转换，即当用户单击超级链接时就会进入链接中指定的文档或文件（图形、音频、视频）。超级链接的基本格式如下：

```
<a href=链接目标的位置>链接的名称</a>
```

在上述定义中链接名称是指向链接目标的链接指针，可以是文字或图片。而链接的目标位置可以使用 URL 指定。如果链接目标的网页位于同一站点下，可以使用相对 URL。如果链接目标的网页位于其他站点，需要使用绝对 URL。单击链接的名称，就会跳转到相应链接目标的网页。下面是具体的代码演示。

（1）链接到同一站点内的文档。该方式的例子如下：

```
<a href=myhtml.htm>示例</a>
```

当单击“示例”这个链接时，可以跳转到同一文件夹下的 myhtml.htm 文件。

（2）链接到其他站点内的文档。该方式的例子如下：

```
<a href=HTTP://www.myWeb.edu.cn >示例</a>
```

当单击“示例”这个链接时，页面就转到 HTTP://www.myWeb.edu.cn 网址的首页。

（3）E_mail 链接。该方式的例子如下：

```
<a href=mailto:mymail@126.com >示例</a>
```

当单击“示例”这个链接时，将调用系统设定的电子邮件程序，建立一个空白的邮件，

mailto 中指定的邮件地址被填写到“收件人”一栏中。

（4）页面内部链接。首先在页面中给需要链接的位置命名，方法如下：

```
<a name=myword></a>
```

用 name 属性把放置此标签的位置命名为 myword，<a>和之间没有任何内容。然后就可以设置指向该位置的链接了，方法如下：

```
<a href=#myword>转到</a>
```

当单击“转到”这个链接时，就从当前页的当前位置转到标记名为 myword 的位置。下面代码演示了超级链接的用法。

```
<!--  html4.html  -->
<HTML>
    <HEAD>
        <TITLE>超级链接</TITLE>
    </HEAD>

    <BODY >
        <a href=html1.htm>示例</a>        <!--  链接到同一站点内的文档  -->
        <!--  链接到其他站点  -->
        <a href=HTTP://www.myWeb.edu.cn >示例</a>
        <!--  E_mail 链接  -->
        <a href=mailto:mymail@126.com >示例</a>
        <a name=myword></a>
        标记的<a name=myword>位置</a>
        <a href=#myword>转到</a>          <!--  转到标记处  -->
    </BODY>
</HTML>
```

保存并使用浏览器浏览该页面，结果如图 3.12 所示。

图 3.12　网页中链接的运行结果

3.2.4　HTML 基本标签——图像标签

在网页中经常会用到图像，使用 img 标签就可以在网页中加入图像，常用的属性如表 3.1 所示。

表 3.1　标签img的属性

属　性	说　明
src	指明图形文件的文件路径和文件名

续表

属　　性	说　　明
alt	当鼠标移动到图像上时显示的文本。即在浏览器还没完全读入图像时，在图像位置上显示的备用文字
Height 属性和width属性	决定图像的高度和宽度，以像素为单位。利用这两个属性能提高图像的传输速度，原因是小的图像占用空间少，网上传输的时间短。所以可以创建一个比较小的图像，然后在浏览器上显示时按比例放大
border	决定边框线的宽度，0表示无边框

在使用 src 属性时，如果图片 pic.gif 放在 HTML 文档的同一个目录下，则可以将代码写成<img src="pic.gif">；如果图片 pic.gif 放在当前 HTML 文档所在目录的一个子目录（子目录名假设为 image）下，则代码写成<img src="images/pic.gif">；如果图片 pic.gif 放在当前 HTML 文档所在目录的上层目录（根目录下只有一个文件夹名 Web）下，则代码写成<img src="../Web/pic.gif">。

下面代码演示了图像的用法。

```
<!--  html5.html  -->
<HTML>
    <HEAD>
        <TITLE>图像</TITLE>
    </HEAD>

    <BODY >
        <img src=Sunset.jpg  alt=花>     <!--  图像标签的使用  -->
    </BODY>
</HTML>
```

保存并使用浏览器浏览页面，结果如图 3.13 所示。

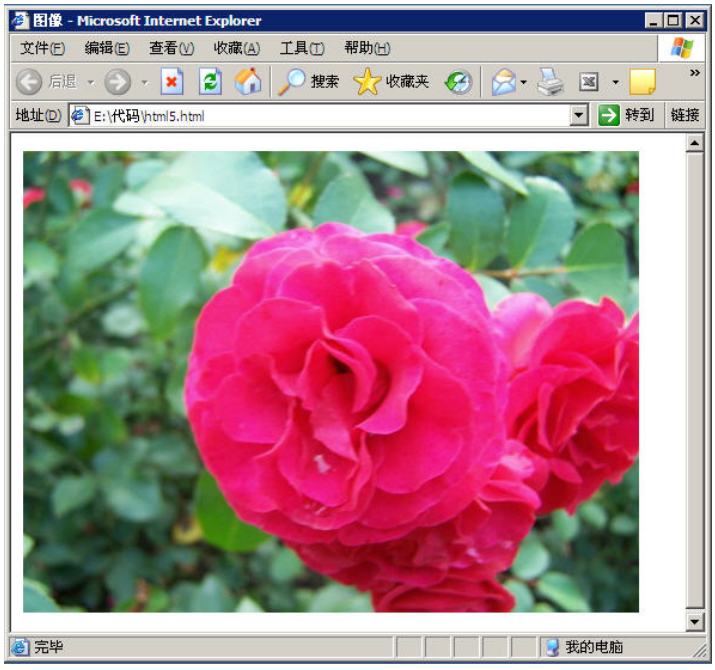

图 3.13　网页中图像的运行结果

3.2.5　HTML 基本标签——表格标签

表格标签对于制作网页是很重要的，因为其在一定程度上控制文本和图像在网页中的位置。表格是 HTML 高级构件之一，其由特定数目的行和列组成。关于表格的标签包含：

（1）<table></table>

该标签对用来创建一个表格，具有的属性如表 3.2 所示。

表 3.2　table标签的属性

属　　性	说　　明
bgcolor	设置表格的背景色
border	设置边框的宽度，其默认值为0
bordercolor	设置边框颜色
bordercolorlight	设置边框亮部分的颜色
bordercolordark	设置边框暗部分的颜色
cellspacing	设置表格格子之间空间的大小
cellpadding	设置表格格子边框与其内部内容之间空间的大小
width	设置表格的宽度

（2）<tr></tr>、td></td>和<th></th>

<tr></tr>标签对用于设置表格的一行，具有的属性如表 3.3 所示。

表 3.3　tr标签的属性

属　　性	说　　明
align	用来设置单元格中文字的水平方向的对齐方式
valign	用来设置单元格中文字的垂直方向的对齐方式

<td></td>标签对用于设置表格中除了标题行中单元格的文字格式。具有的属性如表 3.4 所示。

表 3.4　td标签的属性

属　　性	说　　明
colspan	可以设定跨多列的单元格
rowspan	可以设定跨多行的单元格

<th></th>标签对用于设置表格标题行中单元格的文字格式，通常默认的格式是黑体居中。

下面代码演示了表格的用法。

```
<!--  html6.html  -->
<HTML>
    <HEAD>
        <TITLE>表格</TITLE>
    </HEAD>

    <BODY >
```

```
        <table border="1" width="100%" >    <!--  表格  -->
            <tr>                            <!--  表格的第 1 行  -->
                <td>表格 1.1</td>           <!--  表格第 1 行第 1 列  -->
                <td>表格 1.2</td>
                <td>表格 1.3</td>
                <td>表格 1.4</td>
                <td>表格 1.5</td>
                <td>表格 1.6</td>           <!--  表格第 1 行第 6 列  -->
            </tr>
            <tr>                            <!--  表格的第 2 行  -->
                <td>表格 2.1</td>           <!--  表格第 2 行第 1 列  -->
                <td>表格 2.2</td>
                <td>表格 2.3</td>
                <td>表格 2.4</td>
                <td>表格 2.5</td>
                <td>表格 2.6</td>           <!--  表格第 2 行第 6 列  -->
            </tr>
            <tr>                            <!--  表格的第 3 行  -->
                <td>表格 3.1</td>           <!--  表格第 3 行第 1 列  -->
                <td>表格 3.2</td>
                <td>表格 3.3</td>
                <td>表格 3.4</td>
                <td>表格 3.5</td>
                <td>表格 3.6</td>           <!--  表格第 3 行第 6 列  -->
            </tr>
        </table>
    </BODY>
</HTML>
```

保存并使用浏览器浏览页面，结果如图 3.14 所示。

图 3.14　表格的运行结果

3.2.6　HTML 基本标签——框架标签

框架的标签为 Frame，它可以用来向浏览器窗口中装载多个 HTML 文件，即每个 HTML 文件占据一个框架，而多个框架可以同时显示在同一个浏览器窗口中，它们组成了一个最大的框架，即一个包含多个 HTML 文档的 HTML 文件。常用的方法是在一个框架中放置目录，然后将 HTML 文件显示在另一个框架中。

（1）<frameset></frameset>

<frameset></frameset>标签对放在框架主文档的<body></body>标签对的外边，也可以

嵌在其他框架文档中，并且可以嵌套使用。该标签对有两个属性，即 rows 属性和 cols 属性，当使用该标签时必须至少选择一个。

属性 rows 用来规定主文档中各个框架的行定位，而属性 cols 用来规定主文档中各个框架的列定义。这两个属性的取值可以是百分数、绝对像素值或星号（“*”），其中星号代表那些没被说明的空间，如果同一个属性中出现多个星号，则将剩下的没被说明的空间平均分配。同时，所有的框架按照 rows 和 cols 的值从左到右，然后从上到下排列。

（2）<frame>

<frame>标签放在<frameset></frameset>之间，用来定义某一个具体的框架。<frame>标签具有 src 属性和 name 属性，这两个属性都是必须赋值的。

- src 是此框架的源 HTML 文件名，浏览器将会在此框架中显示 src 指定的 HTML 文件。
- name 是此框架的名字，这个名字是用来供超文本链接标签<a href="target=">中的 target 属性，来指定链接的 HTML 文件将显示在哪一个框架中。

注意：<frame>标签还有 scrolling 和 noresize 属性，scrolling 用来指定是否显示滚动条，取值可以是 yes、no 或 auto；noresize 属性直接加入标签中即可使用，不需赋值，用来禁止调整一个框架的大小。

（3）<noframesets></noframesets>

<noframesets></noframesets>标签对也是放在<frameset></frameset>标签对之间，用来在那些不支持框架的浏览器中显示文本或图像信息。在此标签对之间先紧跟<body></body>标签对，然后才可以使用以前讲过的任何标签。

下面代码演示了框架的用法。

```
<!--  html7.html  -->
<HTML>
    <HEAD>
        <TITLE>框架</TITLE>
    </HEAD>

    <frameset col ="25%,*">     <!--  定义一个带有两列的框架  -->
        <!--设置左框架的内容-->
        <frame src="left.html" scrolling="no" name="left">
        <!--设置右框架的内容-->
        <frame src="page1.html" scrolling="auto" name="Main">
        <noframes>              <!--  浏览器不支持框架显示的内容  -->
            <BODY >
                <p> 你的浏览器不支持框架!</p>
            </BODY>
        </noframes>
    </frameset>
</HTML>

<!--  left.html  -->
<HTML>
    <HEAD>
        <TITLE>导航</TITLE>
    </HEAD>

    <BODY >
```

```
        <p> 导航</p>
        <!--  设置“第一页”超链接  -->
        <p><a href="page1.html" target="Main">第一页</a></p>
        <!--  设置“第二页”超链接  -->
        <p><a href="page2.html" target="Main">第二页</a></p>
    </BODY>
</HTML>

<!--  page1.html  -->
<HTML>
    <HEAD>
        <TITLE>第一页</TITLE>
    </HEAD>

    <BODY >              <!--  定义第一页的内容  -->
        <p>这是第一页</p>
    </BODY>
</HTML>

<!--  page2.html  -->
<HTML>
    <HEAD>
        <TITLE>第二页</TITLE>
    </HEAD>

    <BODY >              <!--  定义第二页的内容  -->
        <p>这是第二页</p>
    </BODY>
</HTML>
```

3.2.7　HTML 基本标签——表单标签

表单是用来输入信息的区域，它是使网页具有交互功能的关键。利用表单可以接收用户输入，以便向服务器传送。服务器端程序接收并处理这些信息，然后动态产生网页。

（1）文本框。文本框分为单行文本框和多行文本框，密码框可以认为是一种特殊的文本框。单行文本框的格式如下：

```
<input type=text name=控件名称 value=传出值>
```

密码文本框的格式如下：

```
<input type=password name=控件名称 value=传出值>
```

多行文本框的格式如下：

```
<textarea row=行数 cols=列数 name=控件名称></textarea>
```

（2）按钮。按钮有 3 种类型，分别是提交按钮、重置按钮和普通按钮。提交按钮的格式如下：

```
<input type=submit name=控件名称 value=显示值>
```

重置按钮的格式如下：

```
<input type=reset name=控件名称 value=显示值>
```

提交按钮的格式如下：

```
<input type=button name=控件名称 value=显示值>
```

（3）单选按钮。一组单选按钮只能有一个单选按钮被选中，同一组内的多个单选按钮必须使用一个名称，格式如下：

```
<input type=radio name=控件名称 value=传出值>
```

（4）复选框。跟单选按钮相反，可以有多项被选中。格式如下：

```
<input type=checkbox name=控件名称 value=传出值>
```

（5）下拉菜单。下拉菜单格式如下：

```
<select name=控件名称 size=显示项数>
<option>选项 1</option>
<option>选项 2</option>
<option>选项 3</option>
</select>
```

下面代码演示了表单的用法。

```
<!--  html8.html  -->
<HTML>
    <HEAD>
        <TITLE>表单</TITLE>
    </HEAD>

    <BODY >
        <form method="post" >
            <table>
                <tr>    <!--  定义了输入文本框  -->
                    <td>用户名：</td>
                    <td><input  name="UserName" type="text" size="10">
                    </td>
                </tr>
                <tr>    <!--  定义了密码文本框  -->
                    <td>密 码：</td>
                    <td><input  name="Password1" type="password" size=
                    "10"></td>
                </tr>
                <tr>    <!--  定义了下拉菜单  -->
                    <td>选 择</td>
                    <td>
                        <select name="myselect" multiple size="4">
                            <option value="1" selected>选项 1</option>
                            <option value="2">选项 2</option>
                            <option value="3">选项 3</option>
                            <option value="4">选项 4</option>
                        </ select >
                    </td>
                </tr>
                <tr>    <!--  定义了单选按钮  -->
                    <td>性 别</td>
                    <td><input name="myradio" type="radio" value="M">
                    男；
```

```
                    </td>
                    <td><input name="myradio" type="radio" value="F">
                    女;
                    </td>
                </tr>
            </table>
        </form>
    </BODY>
</HTML>
```

运行结果如图 3.15 所示。

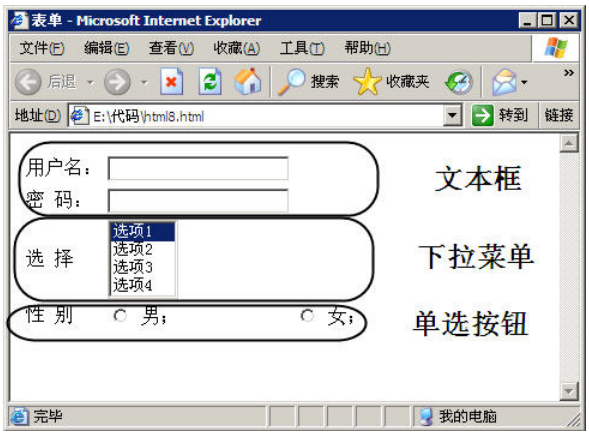

图 3.15　表单的运行结果

3.3　学习 JavaScript 技术

Web 开发中的客户端技术，除了 3.2 节介绍的 HTML 语言外，还有 JavaScript 技术。本节将介绍 JavaScript 的基本语法，内容包括 JavaScript 的基本数据类型、引用类型，以及类型之间的转换。在 JavaScript 中还经常会使用到函数和类，因此本节还介绍了函数的基本使用方法，以及类的编写、使用和继承。

3.3.1　实例：编写第一个 JavaScript 程序：Greet

本节的例子是一个非常简单的 JavaScript 程序。这个程序通过 DOM 技术获得了 name 文本框，并从 name 文本框中读出用户输入的值，最后使用 alert 函数将这个值以对话框形式显示，该程序的代码如下：

```
<!--  greet.html  -->
<html>
    <head>
        <title>Greet</title>
        <!--  开始编写 JavaScript 代码  -->
        <script type="text/javascript">
            //  由按钮单击事件调用的函数
            function greet()
```

```
        {
            var name = document.getElementById("name");
                                                    //  得到 name 文本框
            //  如果找到 name 文本框
            if(name)
                alert("Hello " + name.value);       //  显示对话框
        }
    </script>
</head>
<body>
    <!--  用于输入消息的文本框  -->
    <input type="text" id="name" />
    <!--  调用 greet 方法的按钮  -->
    <input type="button" value="Greet" onclick="greet()"/>
</body>
</html>
```

上面的 JavaScript 代码嵌入到了 HTML 中，在这种情况下，必须将 JavaScript 代码写在<script>标签中，也可以将 JavaScript 代码写在一个单独的文件中，如 greet.js，代码如下：

```
//  greet.js
//  显示问候语的 JavaScript 函数
function greet()
{
    var name = document.getElementById("name");     //  得到 name 文本框
    //  如果找到 name 文本框
    if(name)
        alert("Hello " + name.value);               //  显示对话框
}
```

在 HTML 中也可以通过<script>标签来引用 greet.js，代码如下：

```
<script type="text/javascript" src="greet.js">
</script>
```

注意：不能使用<script type="text/javascript" src="greet.js"/>来引用 JavaScript 文件，否则在 IE 和 Firefox 3 中将无法正常工作，只有在 Firefox 2 中可以正常运行。

JavaScript 程序可以不依赖 Web 服务器运行，但在本章中将所有的 JavaScript 程序文件放到了 webdemo 工程 WebRoot\chapter3 目录中，并通过 Tomcat 将其发送到客户端浏览器运行。由于本章的程序并未涉及任何的服务端技术，因此读者也可以使用其他的 Web 服务器（如 IIS、Apache 等）来运行本章提供的例子。

3.3.2　学习变量

学习任何一门语言或技术，一般都会从变量开始，对于 JavaScript 技术同样如此。本节将详细介绍关于 JavaScript 的变量，其有如下两种定义变量的方法：

- 在为变量第一次赋值时定义。
- 使用关键字 var 定义变量。

第一种方法实现起来非常容易，代码如下：

```
name = "姓名";                  //  定义 name 变量
age = 23;                       //  定义 age 变量
```

```
alert(name);                    //  显示 name 变量的值
alert(age);                     //  显示 age 变量的值
```

使用 var 来定义变量和第一种方法差不多，代码如下：

```
var product = "自行车";          //  使用 var 关键字定义 product 变量
alert(product);                 //  显示 product 变量的值
```

虽然这两种定义变量的方法类似，但是通过 var 定义变量会有一些不同，如在 MyEclipse 中通过 Content Assist 功能显示当前可用的变量时就是根据 var 来寻找这些变量的。如果不使用 var 定义变量，这些变量将不会在列表框中显示。因此笔者建议使用 var 来定义 JavaScript 变量。使用 var 可以在一行定义多个变量，中间使用逗号（,）分隔，代码如下：

```
var p1 = "abc", p2 = 1234;      //  定义两个变量，中间使用逗号分隔
```

由于 JavaScript 是弱类型语言，因此在第一次为变量赋值时，JavaScript 解析器就会为变量创建一个相应类型的值，如为 p1 创建一个字符串值，为 p2 创建一个整型的值。与 Java 不同的是，JavaScript 变量还可以存放不同类型的值，变量的当前值类型就是最后一次为这个变量赋的值的类型，如下面的代码所示。

```
var p = "abc";                  //  当前值类型是 String
alert(p);                       //  显示 p 变量的值
p = 1234;                       //  当前值类型是 Integer
alert(p);                       //  显示改变数据类型的 p 变量的值
```

JavaScript 变量名需要遵循如下两条规则：

- 第一个字符必须是字母、下划线（_）和美元符号（$）；
- 其他的字符可以是下划线、美元符号、任何字母或数字字符。

注意：本节中涉及的代码存放到了 webdemo 工程的 WebRoot\chapter3\var.html 文件里。

3.3.3　学习原始类型

JavaScript 有 5 种原始类型，即 Undefined、Null、Boolean、Number 和 String。每一种原始类型都定义了自身的取值范围和表示形式，在 JavaScript 中提供 typeof 运算符获得一个变量的类型，可以用 typeof 判断一个变量是否属于原始类型，以及属于哪一个原始类型。下面是使用 typeof 获得变量类型的一个例子。

```
var iValue = 20;                //  定义整型变量 iValue
var sValue = "字符串";           //  定义字符串变量
alert(typeof iValue);           //  输出 number
alert(typeof sValue);           //  输出 string
```

在运行上面的代码后，将弹出两个对话框，分别显示 number 和 string。这是上述 5 种原始类型中的两个。typeof 运算符可以返回如下 5 个值中的一个。

- undefined：变量是 Undefined 型。
- boolean：变量是 Boolean 型。

- number：变量是 Number 型。
- string：变量是 String 型。
- object：变量是引用类型或 Null 类型。

下面就分别介绍上面的 5 种返回值。

1．Undefined类型

Undefined 类型只有一个值，也就是 undefined，如果要判断未使用关键字 var 定义的变量是否为 undefined，需要编写如下的代码：

```
alert(typeof abc);              //  显示 undefined
//  条件为 true，弹出"未定义"对话框
if(typeof abc == "undefined")
        alert("abc 未定义");
```

由于 typeof 返回的是字符串，因此，需要使用 undefined 的字符串形式，不能使用如下的代码来判断 abc 是否为 Undefined 类型。

```
//  无法使用 undefined 来判断 abc 是否定义
if(typeof abc == undefined)
        alert("abc 未定义");
```

如果变量是使用关键字 var 来定义的，并且未初始化，这个变量的初始值就是 undefined，因此，可以使用如下代码判断变量是否为 Undefined 类型。

```
var name;           //  定义 name 变量
//  通过 undefined 值判断 name 变量是否为 Undefined 类型
if(name == undefined)
        alert("name 未初始化");
```

当然，用 var 定义的变量也可以使用如下代码判断变量类型是否为 Undefined。

```
var name;           //  定义 name 变量
//  通过"undefined"值判断 name 变量是否为 Undefined 类型
if(typeof name == "undefined")
        alert("name 未初始化");
```

注意：未使用 var 定义的变量不能使用 if(name = undefined)形式判断变量类型是否为 Undefined。

2．Boolean类型

Boolean 类型是 JavaScript 中最常用的类型之一，它只有两个值（true 和 false），也可以用 1 表示 true，0 表示 false，如下代码所示。

```
var bYes = true;        //  定义 bYes 变量
var bNo = false;        //  定义 bNo 变量
alert(bYes);            //  显示 bYes 变量
alert(bNo);             //  显示 bNo 变量
//  true 相当于 1，所以条件为 true
if(bYes == 1)
```

```
    alert("bYes 的值是 true");
//  false 相当于 0，所以条件为 true
if(bNo == 0)
    alert("bNo 的值是 false");
```

3．Number类型

Number 类型是 JavaScript 中最特殊的原始类型。这种类型既可以表示 32 位整数，也可以表示 64 位的浮点数。如下面的代码声明了一个存放整数值的变量，这个变量的值是 120。

```
var iNum=120;
```

整数也可以被表示成八进制和十六进制的数。八进制数必须以 0 开头，十六进制数必须以 0x 开头，如下面代码所示。

```
var iOctalNum = 0213;   //  八进制数
var iHexNum = 0xFE;     //  十六进制数
alert(iOctalNum);       //  显示十进制数（139）
alert(iHexNum);         //  显示十进制数（254）
```

注意：虽然可以使用八进制和十六进制表示 Number 类型的值，但所有的数学运算返回的值都是十进制结果。

要表示浮点数，必须包括小数点和小数点后的至少一位数字（如要使用 1.0，而不是 1），如下面的代码所示。

```
var fNum1 = 23.0;       //  定义浮点类型变量 fNum1
var fNum2 = 12.45;      //  定义浮点类型变量 fNum2
```

如果表示的浮点数非常大，也可以采用科学计数法来表示浮点数，如 432450000，可以将这个数表示成 4.3245*10^8。在科学计数法中，使用 e 或 E 表示 10^，因此，可以使用下面的科学计数法来表示这个大数：

```
var fNum = 4.3245e8;    //  用科学计数法表示 432450000
alert(fNum);            //  显示 432450000
```

也可以用科学计数法表示非常小的数，如 0.00000567，可以将这个数表示成 5.67e-8，如下面的代码所示。

```
var iNum = 5.67e-8;     //  用科学计数法表示小数
alert(iNum);            //  仍然会显示 5.67e-8
```

JavaScript 默认会将具有 6 个或 6 个以上前导 0 的浮点数自动用科学计数法表示。如表 3.5 所列的是 Number 类型的几个特殊值。

表 3.5　Number类型的特殊值

特　殊　值	含　　义
Number.MAX_VALUE	表示 Number 类型所能存储的最大值
Number.MIN_VALUE	表示 Number 类型所能存储的最小值
Number.POSITIVE_INFINITY	表示正无穷大，可以使用 isFinite 函数判断一个 Number 类型变量是否为正无穷大。这个值不能参加算术运算

续表

特　殊　值	含　　义
Number.NEGATIVE_INFINITY	表示负无穷大，可以使用 isFinite 函数判断一个 Number 类型变量是否为负无穷大。这个值不能参加算术运算
Number.NaN	表示某个值是否可以被转换成 Number 类型的值，可以使用 isNaN 函数判断一个变量或一个值是否为 NaN，如 isNaN（"12"）返回 false，而 isNan（"12a"）返回 true，这说明“12”可以被转换成 Number 类型的值，而“12a”无法转换成 Number 类型的值，如果将“12a”改为十六进制形式“0x12a”，则 isNAN（"0x12a"）返回 false。这个值不能参加算术运算

4．String类型

String 类型是 JavaScript 中唯一没有固定大小的原始类型。它可以存储 0 个或更多的 UCS2 编码的字符（UCS2 是两个字节长度的 Unicode 编码，Unicode 编码是一种国际通用的字符集，可以表示世界上所有语言）。String 类型的值可以使用双引号（"）和单引号（'）表示，如下面的代码所示。

```
var sName1 = "未来";          //  定义字符串变量 sName1
var sName2 = '希望';          //  定义字符串变量 sName2
```

如果要获得 String 类型值的长度，可以使用如下的代码：

```
var sName = "聪慧";           //  定义字符串变量 sName
alert(sName.length);          //  显示 2
```

由于 String 类型值是以 UCS2 编码格式保存的，因此所有的字符的长度都是 1，如中文的每一个汉字的长度是 1。如果要以字节为单位获得字符串的长度，可以编写如下的代码：

```
//  使用 String 的 prototype 为 String 对象添加新方法
String.prototype.lenB = function ()
{
     //  将每一个中文替换成##
     return this.replace(/[^\x0-\xf]/g, "##").length;
}
var sName = " a 聪慧 b";                //  定义包含中文和英文的字符串变量 sName
alert(sName.lenB());                    //  显示 6
```

在上面的代码中使用了原始的方式向 String 类（String 类是 string 原始类型的对象表示形式）中添加了一个 lenB 方法，用来以字节为单位获得字符串的长度。基本原理是将每一个中文使用正则表达式替换成“##”，这样一个汉字就变成了两个“##”，因此，也就能以字节为单位得到字符串的长度了。

由于在 JavaScript 中有一些特殊的符号，如果这些特殊的符号要想在字符串中表示的话，就需要使用如表 3.6 所示的转义符号。

表 3.6　String类型中的转义符号

转 义 符 号	含　　义
\n	换行
\t	制表符

续表

转 义 符 号	含　义
\b	空格
\r	回车
\f	换页符
\\	反斜杠
\'	单引号
\"	双引号
\0nnn	八进制数 nnn（n 的值从 0～7）表示的字符
\xnn	十六进制数 nn（n 的值从 0～F）表示的字符
\unnnn	十六进制数 nnnn（n 的值从 0～F）表示的 Unicode 字符

注意：本节中涉及的代码存放到了 webdemo 工程的 WebRoot\chapter3\primitive.html 文件里。

3.3.4　掌握类型转换

在类型转换中，最常用的是将其他类型的值转换成 String 类型的值。任何类型的变量都有一个 toString 方法，通过这个方法，可以将相应类型的值转换成字符串，如下面的代码所示。

```
var iNum = 123;                          //  定义整型变量
var sStr = iNum.toString();              //  将整数转换成字符串
alert(sStr);                             //  显示 123
```

如果被转换的是八进制或十六进制数，使用 toString 方法仍然以十进制输出这些数，如下面的代码所示。

```
var iOctalNum = 0345;                    //  定义八进制整数
var iHexNum = 0xF1;                      //  定义十六进制整数
var sOctalNum = iOctalNum.toString();    //  sOctalNum 的值是 229
var sHexNum = iHexNum.toString();        //  sHexNum 的值是 241
alert(sOctalNum);                        //  显示 229
alert(sHexNum);                          //  显示 241
```

如果想直接获得二进制、八进制和十六进制的变量值，可以使用如下代码：

```
var iNum = 123;
alert(iNum.toString(2));                 //  显示二进制数 1111011
alert(iNum.toString(8));                 //  显示八进制数 173
alert(iNum.toString(16));                //  显示十六进制数 7b
```

在 JavaScript 中提供了两个函数用来将字符串转换成数字，这两个函数是 parseInt 和 parseFloat，其中 parseInt 可以将字符串转换成整型值，parseFloat 函数可以将字符串转换成浮点值。使用 parseInt 函数的示例代码如下：

```
var iNum1 = parseInt("1234xyz");         //  返回 1234
var iNum2 = parseInt("0123");            //  返回 83
var iNum3 = parseInt("43.4");            //  返回 43
```

```
var iNum4 = parseInt("false");          //  返回 NaN
alert(iNum1);                           //  显示 1234
alert(iNum2);                           //  显示 83
alert(iNum3);                           //  显示 43
alert(iNum4);                           //  显示 NaN
```

还可以使用 parseInt 函数的基模式，将二进制、八进制、十六进制或其他进制的字符串转换成整数，即是由 parseInt 方法的第 2 个参数指定的，代码如下：

```
var iNum1 = parseInt("110101", 2);      //  按二进制转换，返回 53
var iNum2 = parseInt("110101", 8);      //  按八进制转换，返回 36929
var iNum3 = parseInt("110101", 16);     //  按十六进制转换，返回 1114369
alert(iNum1);                           //  显示 53
alert(iNum2);                           //  显示 36929
alert(iNum3);                           //  显示 1114369
```

parseFloat 函数和 parseInt 函数的使用方法类似，所不同的是 parseFloat 函数所转换的字符串必须以十进制形式表示浮点数，而不能以二进制、八进制、十六进制或其他进制表示浮点数，对于十六进制的数，如 0xAB，parseFloat 函数将返回 0。下面的代码是一个使用 parseFloat 函数的例子。

```
var fNum1 = parseFloat("1234xyz");      //  返回 1234.0
var fNum2 = parseFloat("0xAB");         //  返回 0
var fNum3 = parseFloat("22.4");         //  返回 22.4
var fNum4 = parseFloat("22.6.12");      //  返回 22.6
var fNum5 = parseFloat("0123");         //  返回 123
var fNum6 = parseFloat("xyz");          //  返回 NaN
alert(fNum1);                           //  显示 1234
alert(fNum2);                           //  显示 0
alert(fNum3);                           //  显示 22.4
alert(fNum4);                           //  显示 22.6
alert(fNum5);                           //  显示 123
alert(fNum6);                           //  显示 NaN
```

在 JavaScript 中还可以使用强制类型转换来处理变量值的类型。下面是 JavaScript 支持的 3 种强制类型转换。

- Boolean（value）：把 value 中的值转换成 Boolean 类型。
- Number（value）：把 value 中的值转换成数字（整数或浮点数）。
- String（value）：把 value 中的值转换成字符串。

下面的代码演示了如何使用 Boolean（value）强制将 value 转换成 Boolean 类型的值：

```
// 强制转换成 Boolean 类型
var b1 = Boolean("");                   //  返回 false
var b2 = Boolean("abc");                //  返回 true（非空字符串都返回 true）
var b3 = Boolean(100);                  //  返回 true（非 0 数都返回 true）
var b4 = Boolean(0);                    //  返回 false
var b5 = Boolean(null);                 //  返回 false
var b6 = Boolean(new String());         //  返回 true
```

Number()的强制类型转换与 parseInt()及 parseFloat()的处理方式类似，只是 Number()转换的是整个值，而不是部分值，如“12ab”，使用 parseInt()转换后返回 12，而使用 Number()

转换后返回 NaN。下面的代码演示了 Number()强制类型转换的各种情况。

```
//  强制转换成 Number 类型
var n1 = Number(false);              //  返回 0
var n2 = Number(true);               //  返回 1
var n3 = Number(undefined);          //  返回 NaN
var n4 = Number(null);               //  返回 0
var n5 = Number("12.1");             //  返回 12.1
var n6 = Number("66");               //  返回 66
var n7 = Number("12ab");             //  返回 NaN
var n8 = Number(new Object());       //  返回 NaN
```

String()是这 3 种强制类型转换中最简单的一种，它和 toString 方法唯一不同的是可以将 null 和 undefined 值强制转换成相应的字符串（"null"和"undefined"）而不引发错误，如下面的代码所示。

```
var s1 = String(null);               //  返回"null"
var s2 = String(undefined);          //  返回"undefined"
var s3 = String(true);               //  返回"true"
alert(s1);                           //  显示 null
alert(s2);                           //  显示 undefined
alert(s3);                           //  显示 true
```

注意：本节中涉及的代码存放到了 webdemo 工程的 WebRoot\chapter3\conversion.html 文件里。

3.3.5　学习函数与函数调用

对于 JavaScript 技术来说，除了要掌握上面章节所介绍的变量、类型外，函数也是其最重要的技术之一。函数需要使用关键字 function 来声明，函数的基本语法如下：

```
function funName(arg0, arg1, ...,argN)
{
    statements
}
```

函数可以加多个参数，中间使用逗号（,）分隔。下面的代码是一个具体的函数声明。

```
//  定义并实现一个 JavaScript 函数
function greet(sName)
{
    alert("你好 " + sName);
}
```

调用 greet 函数的代码如下：

```
greet("bill");
```

执行上面的代码后，将会弹出如图 3.16 所示的警告对话框。

图 3.16　调用 greet 函数弹出的警告框

JavaScript 中的函数没有返回类型，但也可以使用 return 语句来返回值，代码如下：

```
//  求 n1 和 n2 的和
function sum(n1, n2)
{
    return n1 + n2;          //  返回 n1 和 n2 之和
}
```

下面的代码把 sum 函数返回的值赋给一个变量，并在警告对话框中显示这个值。

```
var iSum = sum(12, 22);     //  返回 12 和 22 的和
alert(iSum);                //  显示 34
```

由于 JavaScript 是弱类型语言，因此，在 JavaScript 中的函数不能重载，如果函数重名，后面的会覆盖前面的，代码如下：

```
function fun1()
{
    alert(100);
}
//  这个 fun1 覆盖了上一个 fun1
function fun1()
{
    alert(10);
}
fun1();          //  显示 10
```

在 JavaScript 中，函数还可以使用 arguments 对象实现动态参数，也就是在声明函数时并不需要定义参数，而在函数体内使用 arguments 对象来获得当前函数的参数值，如下面的代码所示。

```
//  求不定数目的 Number 类型值的和
function sum()
{
    var n = 0;
    //  从 arguments 对象中取出 sum 函数的参数值，并将这些参数值相加
    for(var i = 0; i < arguments.length; i++)
    {
        n += arguments[i];           //  迭代求和
    }
    return n;
}
```

下面的代码调用了这个 sum 函数。

```
alert(sum(1,2,3));                  //  显示 6
alert(sum(-1,4,5,6));               //  显示 14
```

在 JavaScript 中，一个函数实际上相当于一个对象。也就是说，可以使用 Function 类来建立任何的函数，使用 Function 类创建函数的语法如下：

```
var funName = new Function(arg1, arg2,...,argN, functionBody);
```

下面的代码使用 Function 类创建了上面的 fnSum 函数。

```
var fnSum = new Function("n1", "n2", "return n1 + n2");
```

下面的代码调用了 fnSum 函数。

```
alert(fnSum(-1, 20));       //  显示 19
```

注意：本节中涉及的代码存放到了 webdemo 工程的 WebRoot\chapter3\fun.html 文件里。

3.3.6　学习类和对象

类在JavaScript中的使用非常灵活。在JavaScript中有一些预定义的类，如Object、Array、String 和 Number 等。在创建对象时，要使用 new 关键字，如下面的代码创建了一个数组对象。

```
var aValues = new Array();      //  定义一个数组变量
aValues[0] = "v1";
aValues[1] = "v2";
aValues[2] = "v3";
```

如果类的构造方法没有参数，也可以将括号省略，代码如下：

```
var aValues = new Array;        //  用省略括号的方式定义一个数组变量
```

在 JavaScript 中自定义类是非常灵活的，可以使用下面的 3 种方法来自定义 JavaScript 类。

1．工厂方式（动态添加类成员）

由于 JavaScript 对象的属性可以动态加入，因此可以使用下面的代码创建一个对象，并调用其中的 study 方法。

```
var oStudent = new Object;      //  创建一个对象变量
oStudent.id = '01';             //  定义 id 属性
oStudent.xm = '赵子龙';          //  定义 xm 属性
oStudent.age = 16;              //  定义 age 属性
//  定义 study 方法
oStudent.study = function()
{
   alert(this.xm + '开始学习');
};
oStudent.study();               //  调用 study 方法
```

为了使创建对象更方便，可以建立一个对象工厂函数来建立 Student 对象，代码如下：

```
//  建立 Student 对象的工厂函数
function createStudent(id, xm, age)
{
   var oStudent = new Object;  //  创建一个对象变量
    oStudent.id = id;          //  定义 id 属性
    oStudent.xm = xm;          //  定义 xm 属性
    oStudent.age = age;        //  定义 age 属性
    //  定义 study 方法
    oStudent.study = function()
    {
        alert(this.xm + '开始学习');
    };
   return oStudent;            //  返回 oStudent 对象
}
```

下面的代码通过 createStudent 函数创建了两个对象，并分别调用了这两个对象的 study 方法。

```
//  使用 createStudent 函数创建 oStudent1 对象
var oStudent1 = createStudent('02', '赵明', 12);
//  使用 createStudent 函数创建 oStudent2 对象
var oStudent2 = createStudent('03', 'bill', 33);
oStudent1.study();          //  调用 oStudent1 对象的 study 方法
oStudent2.study();          //  调用 oStudent2 对象的 study 方法
```

虽然上面的代码可以方便地建立 Student 对象，但是每次建立 Student 对象时，都要重新创建 study 方法。实际上，也可以使多个对象可以共享同一个 study 函数，如下面的代码所示。

```
//  多个对象共享的 study 方法
function study()
{
    alert(this.xm + "开始学习");
}
//  创建 Student 对象的工厂函数，可以共享 study 方法
function createStudent(id, xm, age)
{
    var oStudent = new Object;
    oStudent.id = id;       //  定义 id 属性
    oStudent.xm = xm;       //  定义 xm 属性
    oStudent.age = age;     //  定义 age 属性
    oStudent.study = study; //  将全局的 study 方法赋给 Student 对象的 study 方法
    return oStudent;        //  返回 oStudent 对象
}
```

2．构造函数方式

用构造函数方式创建对象和工厂方式非常像，如下面的代码所示。

```
//  多个对象共享的 study 方法
function study()
{
    alert(this.xm + "开始学习");
}
//  Student 类的构造方法
function Student(id, xm, age)
{
    this.id = id;           //  定义 id 属性
    this.xm = xm;           //  定义 xm 属性
    this.age = age;         //  定义 age 属性
    this.study = study;     //  将全局的 study 方法赋给 Student 对象的 study 方法
}
//  使用构造方法创建 oStudent1 对象
var oStudent1 = new Student('10', 'Mike', 22);
//  使用构造方法创建 oStudent2 对象
var oStudent2 = new Student('20', '比尔', 123);
oStudent1.study();          //  调用 oStudent1 对象的 study 方法
oStudent2.study();          //  调用 oStudent2 对象的 study 方法
```

从上面的代码可以看出，构造函数方式和工厂方式的最大区别就是不在 Student 函数

内部建立对象，而是直接使用 this 来代表当前的对象实例。实际上 Student 函数就是 Student 类的构造方法。

3．原型方式

该方式使用了对象的 prototype 属性，可以把 prototype 属性看成是创建新对象所依赖的原型。可以使用空构造函数来创建一个空类，然后使用 prototype 属性为其添加成员（属性和方法），如下面的代码使用原型方式重写了 Student 类。

```
//  建立一个空的构造方法
function Student()
{
}
//  使用 prototype 为 Student 类添加属性
Student.prototype.id = '12';
Student.prototype.xm = 'bill';
Student.prototype.age = 20;
//  使用 prototype 为 Student 类添加方法
Student.prototype.study = function()
{
   alert(this.xm + '开始学习');
};
var oStudent  = new Student();
oStudent.study();
```

使用原型方式的另一个好处是可以为已经存在的类添加新的成员，如下面的代码为 String 类添加一个可以获得字节长度的 lenB 方法。

```
String.prototype.lenB = function ()
{
     //  将每一个中文替换成##
     return this.replace(/[^\x0-\xf]/g, "##").length;
}
var s = "超人 abc";                          //  定义一个包含中文和英文的字符串变量
alert(s.lenB());                             //  显示 7，如果使用 length，则输出 5
```

注意：本节中涉及的代码存放到了 webdemo 工程的 WebRoot\chapter3\class.html 文件里。

3.4　其他客户端技术

在本节将介绍一些其他客户端技术，如操作 DOM 技术、JavaScript 的内置对象、表单对象的使用、正则表达式等，还给出了一些比较实用的例子，如图像自动切换、用正则表达式验证客户端输入数据、表格排序等。

3.4.1　了解 DOM

DOM 是为了方便处理层次型文档（如 XML、HTML）的一种技术。DOM 还提供了一套 API，使开发人员可以用面向对象的方式来处理这些文档。对于 XML 文档来说，有专

门处理 XML 文档的 XML DOM，然而在本节中将主要介绍处理 HTML 文档的 HTML DOM。在后面的内容中，如果不特别说明，DOM 指的都是 HTML DOM。

DOM 主要的功能是获得 HTML 语言中的各个元素（如 div、form 等），从而可以很容易地获得这些元素的信息，或动态向这些元素中添加新的元素。操作 DOM 的对象实际上也需要使用 JavaScript，也就是说，调用 DOM API 也要编写 JavaScript 代码。在 JavaScript 中描述 DOM 的对象是 document，其实 document 不仅是 HTML DOM，它也是 XML DOM，如果要直接操作 HTML 文档，可以使用 documentElement 属性，如下面代码所示。

```
var oHtml = document.documentElement;
```

oHtml 对象表示了当前的整个 HTML 文档。在 HTML 文档的最下一层有两个元素，即 head 和 body，这两个元素可以分别使用 firstChild 属性和 lastChild 属性获得，代码如下：

```
var oHead = oHtml.firstChild;          //  获得 head 对象
var oBody = oHtml.lastChild;           //  获得 body 对象
```

当然，除了使用 firstChild 和 lastChild 属性外，还可以使用 childNodes 属性来获得同样的对象，代码如下：

```
var oHead = oHtml.childNodes[0];       //  获得 head 对象
var oBody = oHtml.childNodes[1];       //  获得 body 对象
```

通过 HTML 元素对象的 outerHTML 和 innerHTML 属性，可以分别得到当前元素的包括元素本身和不包括元素本身的 HTML 代码，代码如下：

```
alert(oHead.outerHTML);                //  显示包括 head 标签本身的内容
alert(oBody.innerHTML);                //  显示不包括 body 标签本身的内容
```

下面的代码是使用 innerHTML 和 outerHTML 属性显示<head>及<body>标签的内容的一个例子。

```
<!--  dom1.html  -->
<html>
   <head>
      <title>dom</title>
      <script type="text/javascript">
         //  显示 head 标签的内容
         function showHead()
         {
            //  得到 Html 对象
            var oHtml = document.documentElement;
            //  到 Head 对象
            var oHead = oHtml.firstChild;
            alert(oHead.outerHTML);          //  显示 outerHTML 属性的值
            alert(oHead.innerHTML);          //  显示 innerHTML 属性的值
         }
         //  显示 body 标签的内容
         function showBody()
         {
            var oHtml = document.documentElement;
            var oBody = oHtml.childNodes[1]; //  得到 body 对象
            alert(oBody.outerHTML);          //  显示 outerHTML 属性值
            alert(oBody.innerHTML);          //  显示 innerHTML 属性值
         }
```

```
        </script>
    </head>
    <body>
        <input type="button" onclick="showHead()" value="显示 head 标签的内
        容" />
         <p/>
        <input type="button" onclick="showBody()" value="显示 body 标签的内
        容" />
   </body>
</html>
```

在运行上面的代码后，单击“显示 head 标签的内容”按钮，就会弹出一个显示<head>标签内容的对话框，如图 3.17 所示。

在关闭图 3.17 所示的对话框后，又会弹出如图 3.18 所示的显示 innerHTML 属性值的对话框。如果单击“显示 body 标签的内容”按钮，显示的内容类似。从上面的描述可以看出，使用 DOM 技术来操作 HTML 文档是非常容易的。

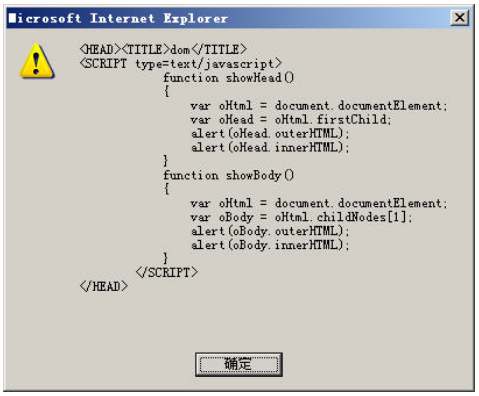

图 3.17　outerHTML 属性的值

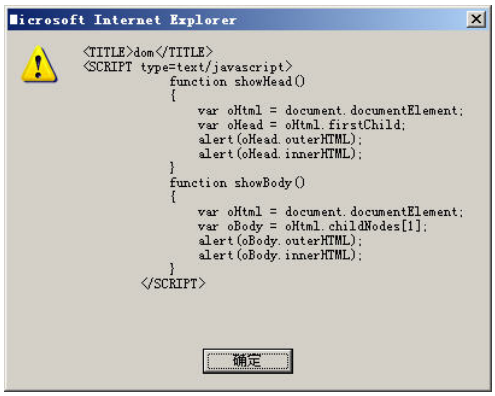

图 3.18　innerHTML 属性的值

3.4.2　获得 HTML 元素的 3 种方法

在 DOM 中有 3 种方法可以获得当前 HTML 文档中的任意一个 HTML 元素，实际上，这 3 种方法也就是 HTML Document 的 3 个方法，分别是 getElementById、getElementByName 和 getElementByTagName。

1．getElementById方法

getElementById 方法是这 3 个方法中最简单的一个，这个方法可以根据 HTML 元素的 id 属性值得到 HTML 元素。在 HTML 文档中，id 属性值是唯一的，也就是说，没有两个 HTML 元素的 id 属性值是相同的。假设有如下的 HTML 代码：

```
<!--  dom2.html  -->
<html>
    <head>
        <title>dom</title>
    </head>
    <body>
      <!--  文本输入框  -->
```

```
    <input type="text" id = "my_text"/>
  </body>
</html>
```

在其中有一个 id 值为 my_test 的文本框。现在通过 getElementById 方法可以使用下面的代码来获得这个文本框元素，并向其中赋一个值，然后再得到并显示这个值。

```
var oText = document.getElementById("my_text");//  得到文本框元素
oText.value = "小超人";                          //  向文本框中赋值
alert(oText.value);                              //  得到并显示文本框中的值
```

在运行上面的代码后，文本框中的值变成了“小超人”，然后会显示一个对话框，如图 3.19 所示。

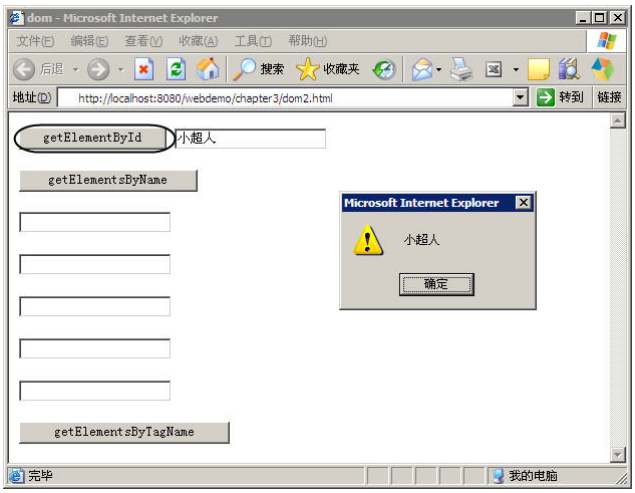

图 3.19　用 DOM 技术修改文本框中的值

2．getElementsByName方法

这个方法可以通过 HTML 元素的 name 属性获得相应的 HTML 元素集合。由于 HTML 元素的 name 属性值并不唯一，因此，使用 getElementsByName 方法有可能得到多个相同 name 属性值的 HTML 元素。假设有如下的 HTML 文档：

```
<!--  dom2.html  -->
<html>
   <head>
      <title>dom</title>
   </head>
  <body>
     <!--  5 个文本框  -->
     <input type="text" id = "text"/> <p/>
     <input type="text" id = "text"/> <p/>
     <input type="text" id = "text"/> <p/>
     <input type="text" id = "text"/> <p/>
     <input type="text" id = "text"/> <p/>
  </body>
</html>
```

在上面的代码中有 5 个 name 属性值为 text 的文本框，可以通过如下的代码为这 5 个

文本框赋值。

```
var oTexts = document.getElementsByName("text"); //  获得一个文本框对象数组
//  循环为这 5 个文本框赋值
for(var i = 0; i < oTexts.length; i++)
   oTexts[i].value = i;
```

如图 3.20 所示为上面代码的运行效果。

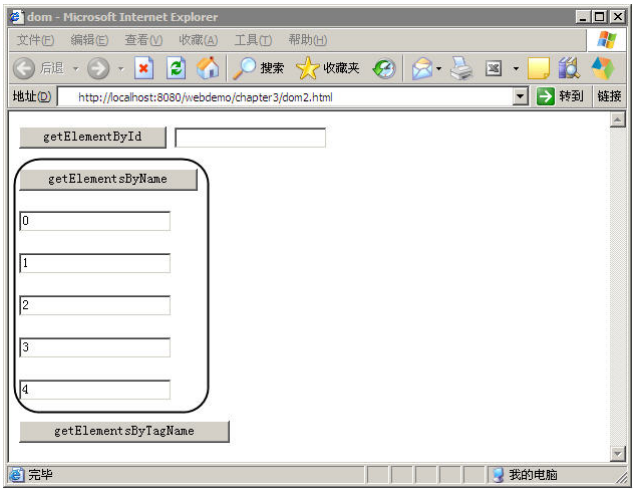

图 3.20　用 DOM 技术修改多个文本框的内容

3．getElementsByTagName方法

getElementsByTagName 方法获得的 HTML 元素的范围最大，它可以根据 HTML 元素的标签类型来获得一个相同 HTML 元素的数组，代码如下：

```
<!--  dom2.html  -->
var oInputs = document.getElementsByTagName("input");//  获得所有 Input 对象
//  循环显示 Input 对象的 type 值
for(var i = 0; i < oInputs.length; i++)
   alert(oInputs[i].type);
```

上面的代码可获得当前 HTML 文档内所有的<input>标签。如果想获得所有的 HTML 元素，可以使用“*”，代码如下：

```
var oAllElements = document.getElementsByTagName("*");
```

在运行上面的代码后，就会显示出 from 组件的类型，如图 3.21 所示，显示第一个组件为 button 类型。

3.4.3　实例：图像自动切换

图像自动切换一般可以通过 JavaScript 的定时器按照一定的时间间隔更新<img>标签中的 src 属性。JavaScript 中的定时器可通过 setInterval 函数实现，代码如下：

```
setInterval("function()", interval);
```

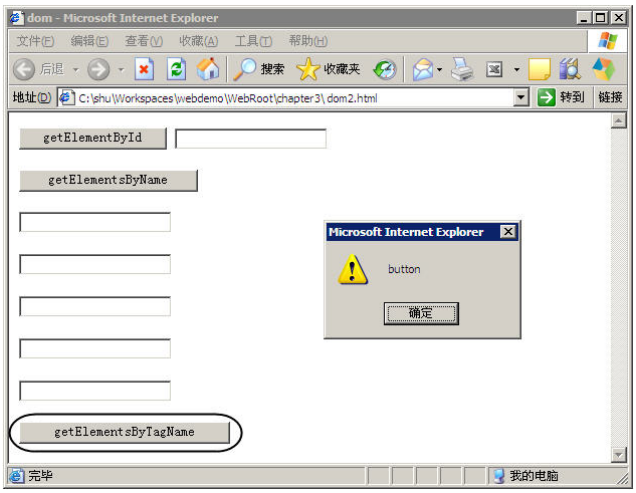

图 3.21　用 DOM 技术获取组件类型

其中第一个参数是定时器要调用的函数名（带括号），interval 表示时间间隔，单位是毫秒。在本节将给出一个可循环切换图像的例子，时间间隔为 3 秒。读者可以自己准备 5 个 jpg 图像文件，也可以使用随书光盘中的 jpg 图像文件，将这些图像文件放到 WebRoot\images 目录中，并取名为 01.jpg、02.jpg、...、05.jpg，然后编写下面的代码：

```
<!--  changeimage.html  -->
<html>
   <head>
      <title>自动切换图像</title>
      <script type="text/javascript">
         setInterval("loadImage()",3000);      //  启动定时器
         var images = ['01.jpg', '02.jpg', '03.jpg', '04.jpg', '05.jpg'];
                                               //  指定图像文件名
         var i = 0;                            //  从第一个图像文件开始显示
         //  装载图像文件（定时器调用）
         function loadImage()
         {
              i++;
              //  当显示到第 5 个图像文件时，再从第 1 个图像开始循环
              if(i == 5)
                  i = 0;
              //  得到 Img 标签
              var oImage = document.getElementById('image');
              //  得到 Label 标签
              var oLabel = document.getElementById('info');
              //  为 img 标签的 src 属性赋值
              oImage.src = '../images/' + images[i];
              oLabel.innerText = images[i];    //  显示当前的图像文件名
         }
      </script>
   </head>
   <body>
      当前图像名：<label id="info"></label>
      <p/>
      <img src="../images/01.jpg" id="image"  width="320" height =
      "240"/>
      <script type="text/javascript">
```

```
            var oLabel = document.getElementById('info');
            //  在初始化时显示第一个图像文件
            oLabel.innerText = images[i];
        </script>
    </body>
</html>
```

在上面的代码中，定时器每隔 3 秒就会调用 loadImage 函数装载图像文件，并显示当前的图像文件名。由于<head>标签中的 JavaScript 执行时，<label>标签还没有创建，因此，要想在初始化时为<label>标签赋值，就要将 JavaScript 代码放到<label>标签的后面，在运行这段代码后，显示的效果如图 3.22 所示。

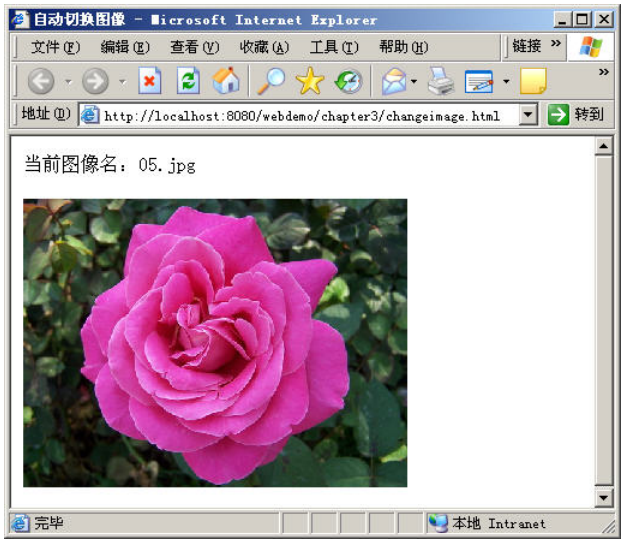

图 3.22　自动切换图像

3.4.4　了解正则表达式

正则表达式是具有特殊语法的字符串，用来表示一组字符串的规则。如要描述所有以“i”开头的字符串，可以使用正则表达式“^i\w*”。在 JavaScript 中使用以下两种方式定义正则表达式：

- 通过 RegExp 类进行定义。
- 通过两个斜杠（/）进行定义。

RegExp 类的构造方法可以有一个或两个参数。构造方法的第一个参数表示正则表达式字符串，代码如下：

```
//  使用正则表达式和 RegExp 类创建一个 RegExp 对象
var reExp = new RegExp("^i\w*");
```

构造方法的第 2 个参数是一些控制正则表达式的指令。在默认情况下，正则表达式是对大小写敏感的，但可以通过 i 指令使其不区分大小写，也可以使用 g 指令将正则表达式应用于整个字符串，如下面的代码所示。

```
var reExp = new RegExp("abc", "gi");   //  在构造方法中指定指令
```

在 JavaScript 中还可以使用两个斜杠来定义正则表达式，代码如下：

```
var reg = /a\w*b/;                    //  使用两个斜杠来定义正则表达式变量
```

其中“*”表示 0 次或多次出现，“\w”表示单词字符（所有的字母、数字和下划线）。这个正则表达式的含义就是以 a 开头，以 b 结尾，中间是单词字符的字符串。可以使用 test 方法来判断一个字符串是否满足这个正则表达式，代码如下：

```
var reg = /a\w*b/;
alert(reg.test("axyzb"));             //  显示 true
alert(reg.test("axyzc"));             //  显示 false
```

也可以使用如下的形式指定一些特殊的指令。

```
var reg1 = /a\wb/gi;                  //  以 a 开头，以 b 结尾，中间是一个单词字符
```

在 String 类中有一个 replace 方法，可以根据正则表达式来替换字符串中所有满足条件的子串，代码如下：

```
var reg1 = /a\wb/gi;
var s = "axbcdaybx";              //  被替换的字符串
//  使用"ok"来替换所有满足条件的子串，替换结果是 okcdokx
alert(s.replace(reg1, "ok"));
```

正则表达式经常被用来进行客户端验证，如电话号、E-mail 等，如下面是验证电话号码和 E-mail 的代码：

```
var regPhone = /^0\d{2,3}\-\d{7,8}$/;       //  验证电话号码的正则表达式
alert(regPhone.test("024-12345678"));       //  显示 true
var regEmail = /^(\w+\.?)*\w+@\w+\.\w+/;    //  验证 E-mail 的正则表达式
alert(regEmail.test("abc.xyz@sun.com"));    //  显示 true
```

在上面的代码中，假设电话号码的规则是“区号-电话号”，其中区号第 1 位是 0，后面可跟两位或三位数字，电话号是 7 位或 8 位的数字。E-mail 的规则只有如下两种：

```
abc@sun.com
abc.xyz@sun.com
```

第一种规则是“@”前面的部分没有“.”，另一种规则是“@”前面的部分有“.”。验证 E-mail 的正则表达式中的“?”表示出现 0 或 1 次，“+”表示至少出现 1 次，“*”表示出现 0 或多次。由于“.”在正则表达式中有特殊含义，因此，使用转义符“\.”来表示“.”。

注意：本节中涉及的代码存放到了 webdemo 工程的 WebRoot\chapter3\regexp.html 文件里。

3.4.5　实例：表格排序

表格排序是 Web 程序中最常用的功能之一，按照通常的做法，是要依赖服务端将排序好的数据发送到客户端，然后，客户端浏览器将这些已被排序的数据显示出来。虽然这样可以解决排序问题，但是会造成客户端和服务端之间频繁交换数据，从而导致大量带宽被

占用等问题。但如果使用 JavaScript 对表格进行排序，这些问题就会迎刃而解。

表格排序要依赖于 JavaScript 中的 Array 对象的 sort 方法。sort 方法可以不带参数，也可以带一个参数。如果不带参数，sort 方法会将数组中的数据按字符串排序规则进行排序，对于表格中的字符串列，这么做是可以的，但如果表格中的某一列是数字，如序号，就必须用 sort 方法的第二种形式。

可以通过 sort 方法的参数指定一个比较函数，至于按什么规则来排序，就由开发人员在这个比较函数中指定。在本节中将采用比较函数对表格进行排序。假设有如下的表格代码：

```
<!--  tablesort.html  -->
<!--  表格标签  -->
<table border = "1" id = "tblSort">
    <thead>                          <!--  表头标签  -->
        <tr>
            <th>序号</th>
            <th>姓名</th>
        </tr>
    </thead>
    <tbody>                          <!--  表体标签  -->
        <tr>
            <td>1</td>
            <td>王明</td>
        </tr>
        <tr>
            <td>12</td>
            <td>超人</td>
        </tr>
        <tr>
            <td>3</td>
            <td>张三</td>
        </tr>
        <tr>
            <td>4</td>
            <td>李四</td>
        </tr>
    </tbody>
</table>
```

对于要排序的表格，最好的方法是使用<thead>和<tbody>标签，因为这两个标签可以将表头和表体的数据分开。

由于本节的例子需要对两列进行排序，为了避免重复编写代码，需要编写一个产生比较函数的函数，在这个函数中指定要对哪一列进行排序，代码如下：

```
<!--  tablesort.html  -->
//  产生比较函数的函数
function generateCompareTRs(iCol)
{
   //  比较函数
    return function compare(tr1, tr2)
    {
       var v1 = tr1.cells[iCol].firstChild.nodeValue;
```

```
                                                //  获得上一个单元格的值
        var v2 = tr2.cells[iCol].firstChild.nodeValue;
                                                //  获得下一个单元格的值
        //  序号列，降序
        if(iCol == 0)
        {
           //  当 v1 大于 v2 时返回-1 为降序
           if(parseInt(v1) > parseInt(v2))
              return -1;
           //  当 v1 小于 v2 时返回 1，为升序
            else if(parseInt(v1) < parseInt(v2))
                   return 1;
           //  当 v1 等于 v2 时，返回 0
            else
                  return 0;
        }
        //  姓名列，升序
        else
        {
            //  当 v1 大于 v2 时返回 1，为降序
            if(v1 > v2)
               return 1;
            //  当 v1 小于 v2 时返回-1，为升序
            else if(v1 < v2)
               return -1;
            //  当 v1 等于 v2 时返回 0
            else
               return 0;
        }
    };
}
```

下面的代码通过调用 generateCompareTRs 函数获得一个比较函数，并通过获取的比较函数对指定列进行升序或降序排序。

```
<!--  tablesort.html  -->
//  排序表格的函数
function sortTable(iCol)
{
   var oTable = document.getElementById("tblSort");//  获得等待排序的表格
   var oTBody = oTable.tBodies[0];                  //  得到<tbody>标签对象
   var aRows = oTBody.rows;                         //  得到表格体的行对象
   //  创建一个数组，用于保存 aRows 中的行引用
   var aTRs = new Array;
    //  循环将行引用放到 aTRs 数组中
   for(var i = 0; i < aRows.length; i++)
   {
       aTRs.push(aRows[i]);
    }
    //  对 sTRs 进行排序
    aTRs.sort(generateCompareTRs(iCol));
    //  创建一个文档碎片
    var oFragment = document.createDocumentFragment();
```

```
    for(var i = 0; i < aTRs.length; i++)
    {
        //  向文档碎片中加入每一行数据
        oFragment.appendChild(aTRs[i]);
    }
    //  删除原来表中的数据，并添加新的已排序后的数据
    oTBody.appendChild(oFragment);
}
```

如图 3.23 所示为排序的效果。

图 3.23　按姓名升序排序的结果

3.5　学习 CSS 技术

CSS 是 Cascading Style Sheet（层叠样式化表单）的简称，是一种格式化网页的语言。这是由 W3C 协会（World Wide Web Consortium）为了弥补 HTML 在样式编排上的不足而制定的一套扩展样式标准。在以前做网页时，网页的内容和样式都混在一起，这将使网页很难维护，而 CSS 的出现解决了这个问题，它专门用于网页的样式设置，使网页内容和样式分开。

3.5.1　了解 CSS

对于初学者来说，如果想掌握 CSS 语言，需要从该语言的基本语法开始。关于 CSS 语言的基本语法格式如下：

```
H3{color:red}
```

其格式分为两部分：选择器（Selector）和样式规则（Rule）。在上例中，H3 为选择器，{}中的内容为样式规则。样式规则用于设置样式内容，选择器用来指定哪些 HTML 元素采用该样式规则。如上面的代码中，指定所有在<H3>标签中的内容都显示为红色。如果有多个样式规则，中间用分号（;）隔开，代码如下：

```
H3{font-family:Arial; text-align:center; color:red}
```

为了增加可读性，可以将上面的代码分行编写，如下所示。

```
H3
{
    font-family:Arial;
    text-align:center;
    color:red
}
```

如果要为一个属性赋多个值，中间使用逗号（,）分隔，代码如下：

```
H3
{
    font-family:Arial, sans-serif;
    text-align:center;
    color:red
}
```

上面的 font-family 属性提供了两个字体，浏览器会依次选择，直到遇见可识别的字体为止。

3.5.2　在 Style 属性中定义样式

最简单的 CSS 使用方法就是直接设置 HTML 元素的 style 属性，如下面的代码所示。

```
<!--  css1.html  -->
<html>
    <head>
        <title>css</title>
    </head>
    <!--  关于 body 的样式  -->
    <body  style="background-color: '#0000FF'">
        <!--  关于 a 的样式  -->
        <a href="http://nokiaguy.cnblogs.com" style="color:
        red;font-size: 40px">
        nokiaguy.cnblogs.com</a>
        <!--  关于 h3 的样式  -->
        <h3 style="font-size:50px">1234</h3>
    </body>
</html>
```

上面的代码设置了<body>的背景颜色、<a>的字体颜色和文字大小、<h3>的文字大小。虽然看上去这么设置很方便，但是如果代码很多的话，修改起来就不太方便了，而且如果多个 HTML 元素使用了相同的样式，那就会产生大量的重复代码。为了解决这个问题，就需要将要经常使用到的样式集中写在一起，就像函数一样，在需要的地方只要引用这些事先定义好的样式就可以了，这就是 3.5.3 节要讲的在 HTML 中定义样式。

3.5.3　在 HTML 中定义样式

在 HTML 中通过<style>标签可以将 HTML 元素中的样式提炼出来，并且可以通过 3 种方式指定哪些 HTML 元素可以使用这些样式，这 3 种方式如下：

- 指定 HTML 元素的 id。
- 通过 HTML 元素的 class 属性。
- 指定 HTML 元素的标签名。

在选择器前面加“井号”（#）表示这个选择器就是一个 id 属性值，任何一个 HTML 元素，只要它的 id 属性值为选择器名，就会应用这个样式，如下面的代码所示：

```
#link{color: red;font-size: 40px}
```

如果一个<a>标签的 id 属性值是 link，那么这个<a>标签就会应用 link 样式，代码如下：

```
<a href="http://nokiaguy.cnblogs.com" id="link">nokiaguy.cnblogs.com</a>
```

在选择器前加实心点（.）表示这个选择器的名可以放在 HTML 元素的 class 属性中，代码如下：

```
.bg{background-color: '#0000FF'};
```

当<body>标签的 class 属性值为 bg 时，会自动应用 bg 样式，代码如下：

```
<body  class="bg">...</body>
```

当选择器名正好是一个 HTML 元素名的话，所有相应的 HTML 元素都会应用这个样式，代码如下：

```
h3{font-size:50px}
```

下面的例子演示了如何将 3.5.2 节的样式放到<style>标签中，然后通过选择器来应用样式。

```
<html>
    <head>
        <title>css</title>
        <!--  定义样式  -->
        <style type="text/css">
            .bg{background-color: '#0000FF'};
            h3{font-size:50px}
            #link{color: red;font-size: 40px}
        </style>
    </head>
    <!--  使用类选择器  -->
    <body class="bg">
        <a href="http://nokiaguy.cnblogs.com" id="link">nokiaguy.cnblogs.
        com</a>
        <h3 >1234</h3>
    </body>
</html>
```

3.5.4　在外部文件中定义样式

虽然在 HTML 中定义样式可以在一定范围上重用，但在不同的 HTML 页面之间，却无法共享样式，因此，CSS 标准中允许将样式单独写在一个.css 文件中，然后通过<link>标签引用这个文件，从而达到多个 HTML 页面共享样式的目的。假设 3.5.3 节中的样式写在了一个 style.css 文件中（与 html 页面在同一个目录下），引用 style.css 文件的 HTML 代码如下：

```
<!--  css2.html  -->
<html>
    <head>
        <title>css</title>
        <!--  引用 style.css 文件  -->
        <link type="text/css" rel="stylesheet" href="style.css"/>
    </head>
    <!--  使用类选择器  -->
    <body class="bg">
        <!--  应用在 style.css 文件中定义的样式  -->
        <a href="http://nokiaguy.cnblogs.com" id="link">nokiaguy.
        cnblogs.com</a>
        <h3 >1234</h3>
    </body>
</html>
```

3.5.5　实现样式的继承

继承样式在 CSS 中非常容易实现。所谓继承，就是如果 HTML 元素未设置某些样式，但在其父元素中设置了，在子元素中就会继承父元素中的样式，如下面的代码所示。

```
<h3 style="font-size:50px">
   <!--  继承 h3 标签的样式  -->
   <a href="http://nokiaguy.cnblogs.com" style="color:red">
      nokiaguy.cnblogs.com
   </a>
</h3>
```

在上面的代码中，<a>标签未设置 font-size 样式，而其父元素<h3>设置了 font-size 样式，因此<a>也会应用 font-size 样式。

3.6　学习 AJAX 技术

AJAX 是目前最流行的 Web 技术之一。通过 AJAX 技术，可以实现以无刷新的方式更新 HTML 元素中的内容。因此，在本节将介绍一下 AJAX 技术的基本原理和一些常用的技巧，如通过 XMLHttpRequest 访问服务端资源，跨域访问及信息传输的几种方法。

3.6.1　了解 AJAX 技术

实际上，AJAX 技术并不是一种新的技术，它只是由 4 种技术组成的结合体，这 4 种技术是 JavaScript、CSS、DOM 和 XMLHttpRequest。其中前 3 种技术在前面已经讲过了，这 3 种技术都是客户端技术，它们和服务端一点关系都没有。然而，XMLHttpRequest 组件和服务端的关系却密不可分。可以说，AJAX 技术中最核心同时也是最简单的部分就是 XMLHttpRequest。因为如果没有 XMLHttpRequest，AJAX 就变成了 DHTML，虽然可以利用 JavaScript 做出非常酷的效果，但不管多酷，使用的也只是客户端的数据。

有了 XMLHttpRequest，客户端就可以使用 DHTML 原有的技术，并利用从服务端获

得数据做出具有更好的用户体验的系统来。XMLHttpRequest 的工作原理也非常简单，其实它只是一个发送 HTTP 请求的客户端组件。开发人员可以根据不同的情况选择以同步或异步的方式来发送 HTTP 请求，并获得服务端的响应消息。

如果读者曾编写过 C/S 模式的程序，就会发觉使用 XMLHttpRequest 组件和服务端通信的方式来编写 Web 程序和编写 C/S 模式的程序的方式非常类似。在 C/S 程序中，客户端一般可直接运行.exe 文件，界面叫做 form。在更新 form 中控件的数据时，整个 form 并不需要刷新和重新装载，而只需要触发某个事件（可能是单击一个按钮或选择一个菜单，也可能按了一个快捷键），在这个事件代码中负责取数据的代码就会从服务端获得相应的数据，并按照某些规则更新 form 中控件的数据。

在 Web 客户端程序中使用 XMLHttpRequest 组件和上述的方法类似，通过同步或异步的方式从服务端获得相应的数据，然后使用 DOM 技术找到要更新的某些 HTML 元素（相当于在 form 中通过控件名来引用相应的控件），并用这些数据进行更新。因此，在一般情况下，编写过基于 C/S 系统的开发人员会更容易适应 AJAX 的开发方式。

注意：在本书中如未特殊指明，使用 XMLHttpRequest 开发 Web 应用程序，都只适合于 IE 浏览器。如果要开发可跨浏览器的基于 AJAX 的 Web 程序，可以使用 JSON 等客户端框架。这些基于 AJAX 的框架一般都考虑到了 AJAX 跨平台的问题。

3.6.2　实例：使用 XMLHttpRequest 获得 Web 资源

XMLHttpRequest 组件是以 COM 组件形式发布的，因此，在客户端需要使用如下的代码创建一个 XMLHttpRequest 对象。

```
var myRequest;
//  创建 XMLHttpRequest 对象
myRequest = new ActiveXObject("Microsoft.XMLHTTP");
```

在 XMLHttpRequest 对象中有一个 open 方法，负责向服务端发送 HTTP 请求消息，这个方法有 3 个参数，第 1 个参数是 HTTP 请求方法（GET、POST 等），第 2 个参数是服务端的 URL，第 3 个参数指定了 XMLHttpRequest 对象是以同步，还是以异步的方式发送请求消息。如果为 true，表示以异步的方式发送，如果为 false，表示以同步的方式发送。XMLHttpRequest 的 send 方法负责向服务端发送数据。下面的代码演示了如何用同步的方式发送 HTTP 请求消息，并接收服务端的响应消息。

```
var myRequest = getXMLHTTPRequest();            //  获得 XMLHTTPRequest 对象
//  如果 XMLHttpRequest 对象创建成功，以同步的方式向服务端发送请求，并接收响应消息
if (myRequest)
{
   myRequest.open("POST", "url", false);        //  同步发送 HTTP 请求消息
   myRequest.send(null);                        //  向服务端发送空数据
   alert(myRequest.responseText);               //  获得并显示 HTTP 响应消息
}
```

下面的代码演示了以异步方式发送请求的方法。

```
var myRequest = getXMLHTTPRequest();            //  获得 XMLHTTPRequest 对象
//  如果 XMLHttpRequest 对象创建成功，以异步的方式向服务端发送请求，并接收响应消息
if (myRequest)
{
    //  建立一个用于接收异步响应消息的方法
    myRequest.onreadystatechange = function ()
    {
        //  状态为 4 时表示响应消息成功返回
        if (myRequest.readyState == 4)
        {
            alert(myRequest.responseText);      //  获得响应消息并显示这些消息
        }
    };
    //  异步发送 HTTP 请求消息
    myRequest.open("POST", "/webdemo/servlet/AjaxEncode", true);
    myRequest.send(null);                       //  向服务端发送空数据
}
```

由于使用异步方式发送 HTTP 请求后，send 方法会立即返回，因此，不能直接在 open 方法后访问 responseText 属性，而是使用 XMLHttpRequest 对象的一个 onreadystatechange 方法，这个方法在 XMLHttpRequest 访问服务端资源的过程中在不同的状态下调用。其中当 readState 的状态是 4 时，表示成功获得了响应消息。

3.6.3　实例：使用 XMLHttpRequest 跨域访问 Web 资源

为了安全起见，在默认情况下，XMLHttpRequest 不允许跨域访问 Web 资源，但可以通过某些方法屏蔽这些安全措施。在 Firefox 中，可以通过如下两步打开跨域访问 Web 资源的功能。

（1）打开 Firefox，在地址栏中输入 about:config，将会输出如图 3.24 所示的配置项列表。

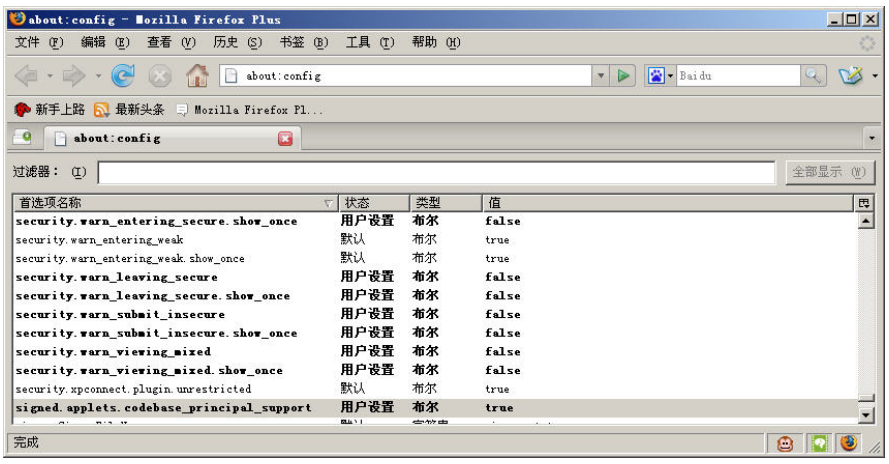

图 3.24　Firefox 的配置项列表

在“过滤器”文本框中输入 signed.applets.codebase_principal_support，Firefox 会自动找到这个配置，如图 3.25 所示。

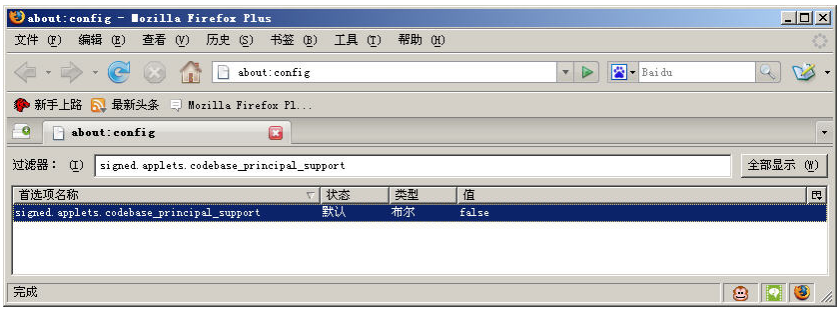

图 3.25　signed.applets.codebase_principal_support 的配置

选中 signed.applets.codebase_principal_support 选项，使它的值为 true（如果已经是 true 了，就继续执行下一步）。

（2）在调用 XMLHttpRequest 对象的 open 方法之前加上如下的代码：

```
//  只适用于 Firefox
if(window.XMLHttpRequest)
{
    try
    {
        //  打开跨域访问权限
        netscape.security.PrivilegeManager.enablePrivilege
        ("UniversalBrowserRead");
    }
    catch (exception)
    {
        alert(exception);
    }
}
```

执行完上述两步后，再次使用 open 方法跨域访问 Web 资源，Firefox 就会弹出一个如图 3.26 所示的提示对话框。

单击“是”按钮，就可以跨域访问了。如果是 IE 6 浏览器，并不需要上面的设置和代码，但在跨域访问时会出现一个如图 3.27 所示的提示对话框。

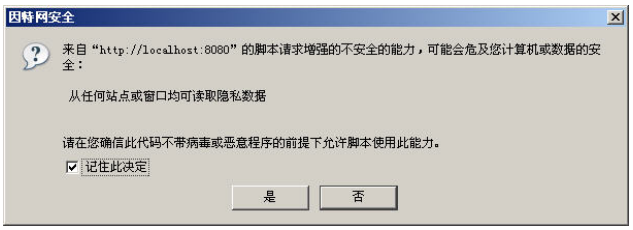

图 3.26　Firefox 的安全询问对话框

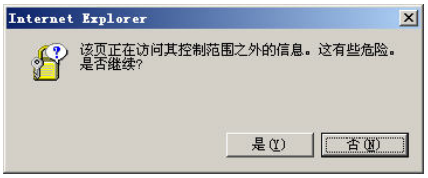

图 3.27　IE 6 的安全提示对话框

单击“是(Y)”按钮后，就可以跨域访问了。

3.6.4　实例：AJAX 的 3 种交换数据方法

通过 XMLHttpRequest 组件和服务端交换数据可以使用多种数据格式，在本节将介绍其中的 3 种，分别是 XML、HTML 和 JavaScript 代码。

客户端从服务端获得 XML 格式的数据是一种比较常用的交换数据的方式。当 XMLHttpRequest 对象获得 XML 格式的响应消息后，就可以通过 responseXML 属性直接获得 XML DOM 对象，然后从中取出相应的数据进行处理，例如，下面的 HTML 代码包含一个<select>元素，在后面的代码中，将使用 JavaScript 和 DOM 技术从服务端获得 XML 数据，并动态为该<select>元素添加选项。

```
<html>
   <head>
      <title>AJAX 传输数据的三种方式</title>
    </head>
    <body>
        <select id="opt">          <!--  选择标签  -->
        </select>
    </body>
</html>
```

下面的代码将从服务端获得 XML 格式的数据，然后将数据动态地加入到<select>标签中。

```
var myRequest = getXMLHTTPRequest();   //  获得 XMLHTTPRequest 对象
if (myRequest)
{
    //  同步发送 HTTP 请求消息
    myRequest.open("GET", "data.xml", false);
    myRequest.send(null);              //  向服务端发送空的请求内容
    var xml = myRequest.responseXML;   //  获得 XML DOM 对象
    var select = document.getElementById("opt");
    var html = "";
    html = "<select id='opt'><option>" + xml.childNodes[0].childNodes[0].
    text + "</option>";
    html += "<option>" + xml.childNodes[0].childNodes[1].text + "</option>
    </select>";
    //  将生成的 html 代码赋给 outerHTML 属性
    select.outerHTML = html;
}
```

在上面的代码中，通过从 XML DOM 对象（Xml 变量）中获得相应的数据，然后组合成 HTML 代码，再赋给<select>标签的 outerHTML 属性。由于本章未涉及服务端程序，因此，在本例中通过一个静态文件 data.xml 来模拟服务端发送过来的 XML 格式的数据，这个文件的内容如下：

```
<data>
   <value1>bike</value1>
   <value2>car</value2>
</data>
```

下面的例子是从服务端获得一个 HTML 格式的数据，返回的 HTML 代码正好是<select>标签里的内容，这些代码保存在 data.html 中，这个文件的内容如下：

```
<option>clothing</option>
<option>shoes</option>
```

例子代码如下：

```
var myRequest = getXMLHTTPRequest();          //  获得 XMLHTTPRequest 对象
if (myRequest)
```

```
{
    myRequest.open("GET", "data.html", false); //  同步发送 HTTP 请求消息
    myRequest.send(null);                      //  向服务端发送空的请求内容
    var html = myRequest.responseText;         //  获得 HTML 代码
    var select = document.getElementById("opt");
    select.outerHTML = "<select id='opt'>" + html + "</option>";
}
```

上面的代码在获得 HTML 代码后，直接和<select>标签组合成了完整的<select>标签，并赋给了<select>标签的 outerHTML 属性。

最后一种方法是从服务端返回 JavaScript 代码，并通过 eval 函数执行这些代码。在这个例子中涉及一个 data.js 文件，里面保存了 JavaScript 代码，内容如下：

```
var aList = new Array;
aList[0] = 'basketball';
aList[1] = 'football';
```

下面的代码从服务端获得 JavaScript 代码（data.js 文件里的内容），并通过 eval 函数执行这些 JavaScript 代码，然后从 aList 数组中读取相应的数据进行处理。

```
var myRequest = getXMLHTTPRequest();           //  获得 XMLHTTPRequest 对象
if (myRequest)
{
    myRequest.open("GET", "data.js", false);   //  同步发送 HTTP 请求消息
    myRequest.send(null);                      //  向服务端发送空的请求内容
    var javascript = myRequest.responseText;   // 获得 JavaScript 代码
    var select =  document.getElementById("opt");
    eval(javascript);                          //  执行 JavaScript 代码
    html = "<select id='opt'><option>" + aList[0] + "</option>";
    html += "<option>" + aList[1] + "</option></select>";
    select.outerHTML = html;
}
```

上述的 3 种方法各有千秋，并不能简单地说哪一种是好是坏。如使用 HTML 格式交换数据时，在客户端写的代码比较少，但这种方法不灵活。而采用 JavaScript 代码的方式来交换数据，客户端的代码显得更容易理解，但服务端如果用程序来控制生成 JavaScript 代码，有时会比较麻烦。至于采用哪种方法，读者可根据实际情况而定。

注意：本节中涉及的代码存放到了 webdemo 工程的 WebRoot\chapter3\transmit.html 文件里。

3.7　小　　结

本章讲解了 Web 系统中的客户端技术，其中 HTML 是客户端技术的基础，而 JavaScript 是客户端技术的核心。通过 JavaScript，可以进行逻辑处理、使用 DOM 技术来控制 HTML 元素。还可以通过正则表达式对客户端的输入进行验证，而且还可以进行更高级的应用，如通过 Array 对象对表格进行排序。虽然 JavaScript 功能强大，但长久以来，功能强大的 JavaScript IDE 却并不多。幸好在最近几年，出现了一些不错的 JavaScript IDE，如 MyEclipse

开发工具就属于其中的佼佼者。

除了 JavaScript 外，CSS 在客户端代码中的作用也不可小视。通过 CSS 的应用，可以使 HTML 中的样式达到最大限度的重用，从而有效地减少了客户端的代码量，当然这也会在一定程度上节省网络带宽。虽然 AJAX 主要依赖于 JavaScript，但 AJAX 的核心却是 XMLHttpRequest 组件，正是因为有了这个组件，客户端浏览器才能通过无刷新的方式更新页面的内容。

3.8　实 战 练 习

编码题

1．在项目开发中，经常为了多显示内容而使用树形结构，利用本章所学的知识，实现简单的树形结构。

【提示】主要涉及 JavaScript 和 DOM 技术，关键代码如下：

```
//通过层对象的显示和隐藏实现树形结构的展开和收缩
function show(d1){
    if(document.getElementById(d1).style.display=='none'){
    //如果触动的层处于隐藏状态，即显示
    document.getElementById(d1).style.display='block';
    }
    //如果触动的层处于显示状态，即隐藏
    else{
    document.getElementById(d1).style.display='none';
    }
}
```

2．利用本章所学的客户端技术，首先设计“简易购物车”界面（如图 3.28 所示），然后实现“合计”按钮功能。

图 3.28　“简易购物车”界面

【提示】主要涉及 HTML、JavaScript 和 DOM 技术，关键代码如下：

```
//简易购物车的计算方式：商品数量*单价+运费=合计
function cal(){
    var num= parseInt(document.myform.num.value);      //获得商品数量
    var price=parseFloat(document.myform.price.value); //获得商品单价
    var cost=parseFloat(document.myform.cost.value);   //获得运费
    var amount=num*price+cost;
    //计算费用并向文本框复制赋值
    document.myform.amount.value=amount;
}
```