

## 第5章 编写JavaScript

### 5.1 简介

现在每个人都对JavaScript有所了解。它是面向对象的一种语言，由 Netscape公司开发出来，语言类似于Java，但并不完全一样，它在客户机上执行，它并不支持那种真正的面向对象的语言所支持的特别复杂的特性，如多态和继承，而且它几乎独立于 Web的开发工具存在。

如果你像我一样的话，你可能并不关心 JavaScript的历史。简要地说，Netscape公司开发了这种语言，开始命名为 LiveScript，当Java发布后改名为 JavaScript。微软公司在 IE中把JavaScript 做了进一步发展，命名为 Jscript，尽管在IE 3中，对JScript 并没有很强的支持功能，但从MSIE 4开始对Jscript开始有了很强的支持。

JavaScript赖以成名的特性是其将原来必须由服务器完成的功能交由客户机代替，但是这与Domino 又有什么关系呢？任何 Web服务器都能容纳包含 JavaScript代码的HTML网页。如果你使用了Domino，你就可以使用所有的Domino开发工具，包括由公式决定的隐藏属性、计算域、视图列公式、CGI域等，使用这些工具你可以创建动态 JavaScript。而且Domino 与JavaScript的优势联合起来可以使你的 Web应用程序功能更为强大。

考虑以下脚本，当用户点击某个按钮，根据用户的浏览器类型，你可以把用户转向到另一个URL。在此，我们可以使用下面的方法完成这个功能。

```
<script language="JavaScript">
function goThere() {
    browser = navigator.appName;
    if (browser == "Microsoft Internet Explorer") {
        url = "/msie.html";
    } else if (browser == "Netscape") {
        url = "/nn.html";
    } else {
        url = "/other.html";
    }
    self.location = url;
}
</script>
<input type="button" value="Go There" onClick="goThere();">
```

在这个例子中，所有的条件由JavaScript定义。现在，我们看第二个例子，由Domino开发：

```
<script language="JavaScript">
function goThere() {
    self.location = "[computed value]";
}
```

```
</script>
<input type="button" value="Go There" onClick="goThere();" >
```

在这个例子中，URL值由下面的公式返回：

```
app := HTTP_USER_AGENT;

@If(@Contains(app; "MSIE"); "/msie.html"; @Contains(app; "Mozilla");
"/nn.html"; "/other.html" );
```

由于我们使用Notes公式计算URL，JavaScript语句数量被大幅度减少。我们所关心的就是：首先，这意味着用户只需要下载较少的 JavaScript。在这个简单的例子中其区别并不明显，但随着例子复杂性的增加，更少的代码对加快下载速度至关重要。其次，这意味着我们只需要调试更少的源程序代码，从而使我们的 JavaScript程序拥有良好的可读性。

请注意这样一个事实，服务器端的脚本语句要比客户机端的脚本语句的安全性好得多。用户常常可以在其浏览器中关闭 JavaScript程序的运行，可以看到 JavaScript程序的源代码（并可以随意改变本地的副本）。但是用户不能访问服务器上的 JavaScript程序，更不能绕过这些程序的运行。

## 5.2 开始编写 JavaScript

如果说你对 JavaScript 完全不熟悉，我推荐你在看我们本书例子以前先看一本指南类型的 JavaScript 参考书，买一本能找到的最便宜、最薄的书。JavaScript学起来非常容易，你很快就可以掌握它。一旦掌握了 JavaScript的基本内容，以后，如果想要精通 JavaScript，访问 Netscape 公司的在线帮助，它会给你详细准确的解释。当你编写 JavaScript语句时可以把你的代码直接插入网页中的下面的标识符之间：

```
<script language="JavaScript">
. . .
</script>
```

通常，你的JavaScript代码使用预定义的对象，在 Web中可以通过编程修改这种对象的某些项目。图5-1显示了Netscape Navigator对象的层次结构。在这种层次结构中，每个对象的子节点都包含一些对象的属性。例如，为了获得一个表单中名为 LastName的文本域的值，你可以使用下面的代码：

```
var theValue = document.forms[0].LastName.value;
```

这个例子定义了一个名为 theValue的变量，这个变量赋值为文档中的第一个表单的 LastName域的值。

就像LotusScript 和Java中的对象一样，JavaScript中的对象有一些关于对象信息的属性和代表对象动作的方法。Web页中的对象也有一些事件可以触发 Javascript代码。例如，一个HTML连接有onMouseOver和onMouseOut两个事件，分别在鼠标移到和移出链接时触发。在一个对象的HTML中，你可以写某些JavaScript代码以使当事件被触发时运行这些代码。例如，你可以使用下面的代码使得当用户将鼠标移到某个链接时弹出一个对话框：

```
<a href="MyFile.html" onMouseOver="alert('Yo!');" >My File</a>
```

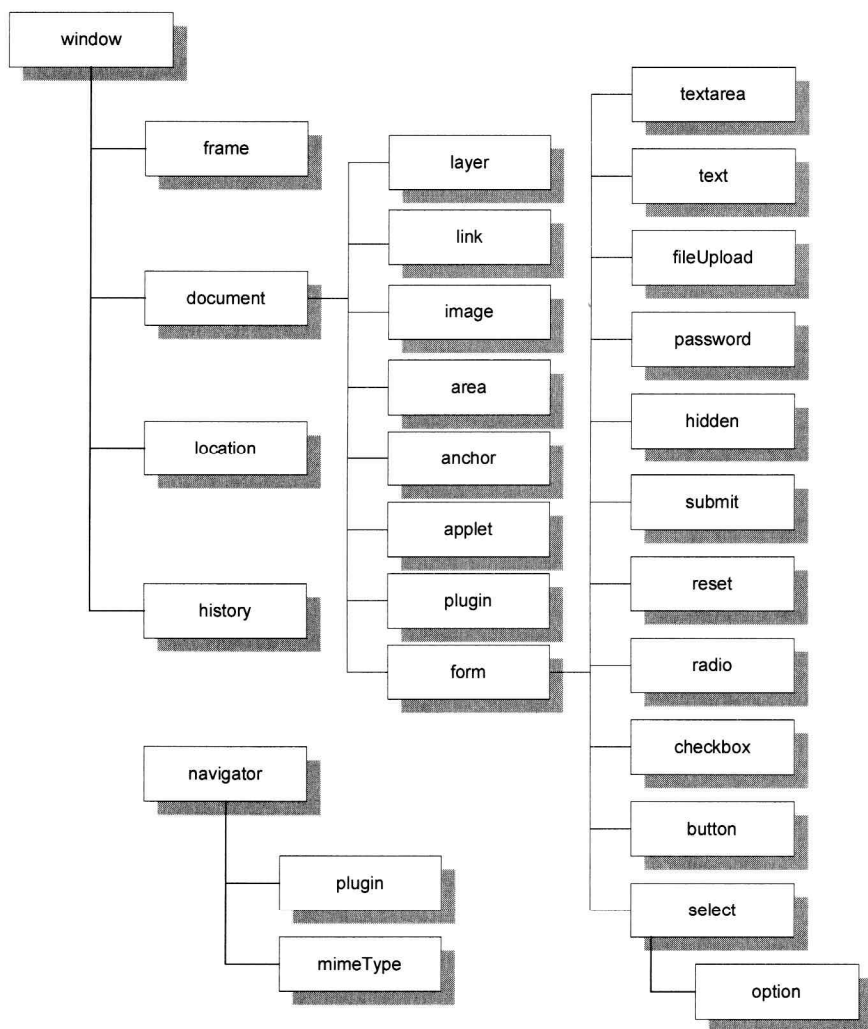


图5-1 在Web页中的大多数预定义对象的JavaScript代码操纵的对象

在这个例子中，`<a>`标签定义了两个属性。`href`属性确定了当用户点击该链接时激活的HTML，`onMouseOver`属性确定了当用户把鼠标移到该链接上触发 `onMouseOver`事件时所执行的JavaScript代码。当 `onMouseOver`事件发生时，将调用JavaScript 函数`alert()`，弹出一个包含特定文本的消息框。注意，尽管在此使用双引号引住 JavaScript，而`alert()`函数中的变量使用单引号，也可以反过来，如 `'alert("Yo!");'`。

`alert()`函数是在JavaScript中预定义的函数。你也可以在相应的事件中触发自己定义的函数。下面是常常被用来触发JavaScript函数的事件。

- 按钮对象的点击事件。
- 文档对象的下载事件。
- 选定对象的改变事件。

- 文本域的得到和失去焦点事件。

一个HTML按钮使用类型属性为按钮的 <INPUT>标签来定义。下面的 HTML定义了一个按钮标签，当点击时，触发 doSomething函数：

```
<input type="button" value="Click Here" onClick="doSomething();">
```

当网页被下载时，一个HTML文档的下载事件被触发。下载事件由 HTML的<BODY>标签定义，例如：

```
<html>
<head>
<script language="JavaScript">
function MyFunction() {
    alert("Yo!");
}
</script>
</head>
<body onLoad="MyFunction();">
</body>
</html>
```

你还可以在<script>和</script>标签之间调用Myfunction() 函数来完成同样的任务。例如

```
<html>
<body>
<script language="JavaScript">
function MyFunction() {
    alert("Yo!");
}
MyFunction();
</script>
</body>
</html>
```

选定对象可以使用<SELECT>标签定义。当你在 Domino表单中创建一个关键词域而且设置为列表框，则 Domino自动将其转换为<SELECT>标签。当用户改变这个域的值时将触发选定对象的改变事件可以触发一些 JavaScript语句。

文本对象，或一个文本域，在 HTML中使用<INPUT>标签并将其属性类型设置为文本。当文本域得到焦点时（即当用户将插入点移到文本域中时）触发文本对象的获得焦点事件，当文本域失去焦点时触发文本对象的失去焦点事件。

就像在LotusScript和Java中一样，还可以定义自己的类。这将使我们的目的有点难以理解，在本章的例子中将不使用这种功能。在这一章中我们将学习使用内嵌的 JavaScript类和Domino执行诸如域的验证以及对象列表框的动态填充符任务。

### 5.3 把JavaScript添加到Domino的表单中

把JavaScript添加到Domino的表单中或页面中有几种方式。最简单的方法是在表单中输入 JavaScript语句并选中它，然后将其标识为 HTML通用文本，这种文本属性告诉 Domino它不需

要把文本转换为HTML，因此Domino只是简单地把文本略过。

以前设置HTML通用文本的惟一方法是用方括号将其括住，就像 `[ body bgcolor="green" ]`。但由于JavaScript数组也使用方括号如：`document.forms[0]`。因此针对JavaScript，这种方法并不十分适合。后来，HTML段落格式被引入，尽管这比使用方括号要好得多，但要求整个段落被设置为HTML通用文本方式，而且有所限制（比如不能往隐藏条件中使用公式）。因此相对来讲HTML通用文本更为灵活。

在R5中，就像表单的Load事件一样，为JavaScript事件提供了特别的占位符。如果想在页首中插入JavaScript。还提供了特殊的JS Header占位符，占位符最大的优点是你可以直接键入JavaScript，而不必使用Notes公式或使你的代码页充斥混乱不堪的HTML通用文本，然而在某些情况下（如使用计算域产生JavaScript代码），仍然常常使用HTML通用文本。

## 5.4 验证域：Expense Report表单

Navigator3.04 <sup>TM</sup>      Navigator4.05 <sup>TM</sup>      Domino4.6.1 <sup>TM</sup>  
Explorer3.0 <sup>O</sup>      Explorer4.01 <sup>TM</sup>      Domino5.0 <sup>TM</sup>

验证公式是Notes表单中的内嵌功能。当用户提交表单时可以使用验证公式验证用户输入的有效性。例如，为了保证Total域的值大于0且小于500，你可以为这个域创建一个有效性验证公式：

```
<If(Total > 0 & Total < 500; @Success; @Failure("The total must be  
greater than zero and less than 500."))
```

比如说，如果用户在Total域中输入800然后将表单提交，则@Failure消息被显示而且将不保存文档。不幸的是，这种验证需要一个从客户机到服务器的消息传递过程，这将增加服务器的工作量。

使用JavaScript，可以在客户端进行这种验证而完全不涉及服务器。图5-2显示了一个Expense Report表单。这个表单被包含在JavaScript的例程库（JSEExample.nsf）中。在图5-2中，用户输入的项目总计为\$550.00然后按下了Submit按钮。这将触发一些JavaScript代码去检测Total域，发现它是无效的，然后显示错误信息而不是真正提交表单。

这个例子是怎样工作的呢？正常情况下，在Domino表单中点击Submit按钮的结果是预先定义好的。因此在这里我们使用了一个小小的策略，这儿的Submit按钮并非真正的Submit按钮。它仅仅是一个普通的按钮，当它被点击时将触发JavaScript代码。这个按钮在创建的时候在表单中输入下面的HTML：

```
[<input type="button" value="Submit" onClick="doSubmit();">]
```

正常情况下，当Domino将Notes中的表单转换为HTML时，它自动产生一个Submit按钮。典型的Submit按钮的HTML标签如下：

```
<input type=submit value="Submit">
```

为了不使费用报表中出现两个Submit按钮，我们使用HTML注释标签`<!--和-->`把Domino自动产生的Submit按钮注释掉。

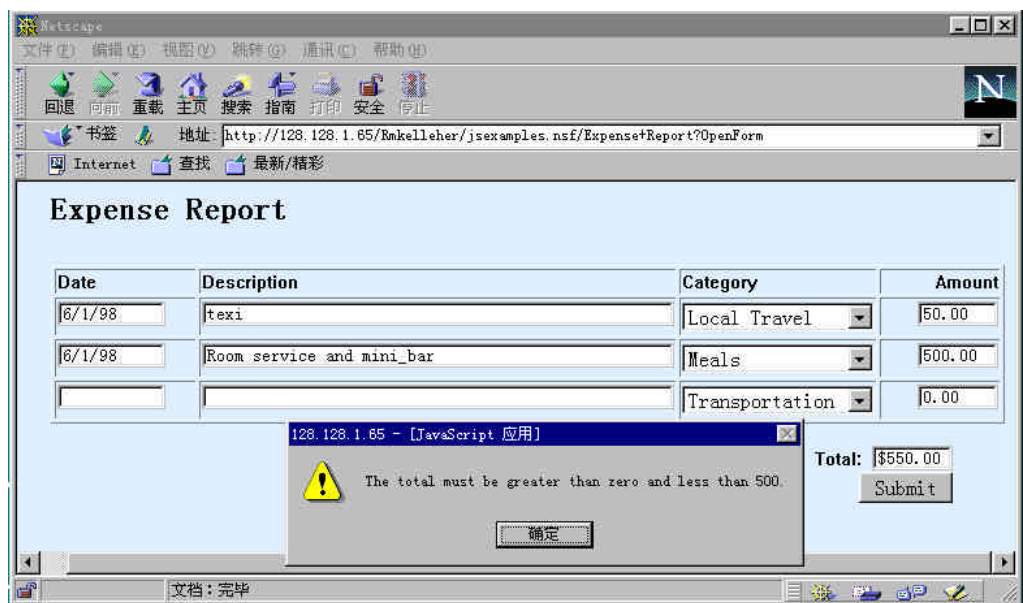


图5-2 启用JavaScript的表单验证Total域

提示 在某些版本中，我们可以将其隐藏公式设为 1，但在 R5 版本中不能使用这种方法。

新改进的 Submit 按钮在点击事件发生时触发用户自定义的 doSubmit 函数，该函数可以在 Expense Report 表单的 <script> 和 </script> 标签中：

```
<script language="JavaScript">
```

```
...
```

```
function doSubmit() {  
    form = document.forms[0];  
    newTotal(form);  
    len = form.Total.value.length;  
    total = form.Total.value.substring(1, len-1);  
    if (total < 1 || total > 499) {  
        alert("The total must be greater than zero and less than 500.");  
    } else {  
        form.submit();  
    }  
}  
</script>
```

doSubmit() 函数所要做的第一件事情是计算和显示 total 域（至于这个任务怎样完成，我们将在下一节“计算域的值”中讲述）。然后它将去除 total 域中的字符串的货币符号，最后使用公式检测以保证其值在规定范围内。这个函数利用了 JavaScript 的弱变量类型特性——total 域



变量在创建时作为字符串变量，但在使用时作为一个类似的数值变量。

如果total 域在有效范围之内，这个 doSubmit()函数调用表单的 submit()方法。在这个例子中，文档只容纳一个表单，被命名为 document.forms[0]。如果该文档中存在多个表单。那么，第二个表单被命名为 document.forms[1]，第三个表单被命名为 document.forms[2]，依此类推。你也可以通过在表单标签中的名称属性来指定表单，如 document.forms[ " Myform " ]。调用表单的submit()方法与点击以前的Submit按钮的功能相同。

## 5.5 计算域的值：Expense Report表单（续）

Navigator3.04 <sup>TM</sup>      Navigator4.05 <sup>TM</sup>      Domino4.6.1 <sup>TM</sup>  
Explorer3.0 <sup>TM</sup>      Explorer4.01 <sup>TM</sup>      Domino5.0 <sup>TM</sup>

我们的Expense Report例子还示范了另外一个 JavaScript的好用途：计算域中的值。其实我们完全可以轻松地把Total域设置为计算域并为其赋值：

```
Amt_1 + Amt_2 + Amt_3
```

但是，在默认情况下，这个公式在用户提交表单时才运行，该数值直接存到服务器上，用户并不一定能够看到结果。但是在我们的 Expense Report例子的功能设计中，我们希望在表单提交前Total域的值能够被计算出来，从而可以被用户看到和被验证。

为了完成这个任务，Expense Report表单在每个 Amt域中都使用 on Charge事件触发 JavaScript的计算公式，当用户改变任何一个 Amt域的时候都将发生 on Charge事件。你可以通过在域中增加适当的 HTML属性来描述域中的事件，为了得到你想要得到的属性在域中的 HTML属性区域中输入适当的公式；

```
"size=7 onChange=\"newTotal(this.form);\""
```

在此将Amt\_1域返回下面的HTML标签：

```
<INPUT NAME="Amt_1" VALUE="0.00" size=7  
onChange="newTotal(this.form);">
```

### 过去版本中的方法

我原来曾在Domino 4.5中开发过这个例子。在那个时候，在Domino中并无HTML属性这个功能。在4.5版本中，必须把HTML的属性标签包含在帮助描述域的方括号中，如：

```
[<size=7 onChange="newTotal(this.form);">]
```

与理想的方式相比，首先这种方法不论出于何种目的只能存在于帮助描述域中，其次它对输入的字符有所限制，而且它禁止使用公式计算各种属性。如果使用的是 4.6 或者更高的版本，那么可以使用HTML属性公式。

每个Amt域的改变事件都将触发用户定义的新Total（）函数，为了方便起见，我们把当前的表单以自变量形式传递到函数中。

```
function newTotal( form ) {  
    amt1 = format( getNumberValue( form.Amt_1 ), 2 ) ;  
    form.Amt_1.value = amt1;  
    amt2 = format( getNumberValue( form.Amt_2 ), 2 ) ;
```

```

form.Amt_2.value = amt2;
amt3 = format( getNumberValue( form.Amt_3 ), 2 );
form.Amt_3.value = amt3;
form.Total.value = dollarize(eval(amt1 + "+" + amt2 + "+" + amt3));
}

```

newTotal ( ) 函数调用了一些用户定义的函数用来对数据字符串进行格式化和检测：getNumberValue()、format()和dollarize()。现在，让我们把注意力集中在 newTotal ( ) 的基本动作。

- 它保证每个 Amt域包含一个数值。
- 它将三个值累加起来。
- 它对总计值进行格式化并把结果输入 Total域。

这个函数把 Amt\_1、Amt\_2、Amt\_3、和Total变量，设置为字符值。而在 Expense Report 表单中相应的域设置为数值域，当表单被提交时由于有函数保证可以把字符值转化为数值。因此，Domino中不会出现麻烦。

当然，有些类型的验证域要求一些特殊的 Notes信息，比如，从视图或简要表文档中得到的某些值。在大部分情况下，你仍然可以使用 JavaScript去验证这些域。把需要被 Domino计算的域放在计算显示域（或计算文本热点）并插入 JavaScript代码。计算显示域将在文档被转换为HTML以前计算并将计算后的值传递到空白文本中，最后将转换为 JavaScript代码。例如；你想使用JavaScript 中的Alert ( ) 函数显示用户的 Domino用户名。你可以创建一个名为 User的计算显示域，并将其赋值为 @Name([CN]; UserName)并把其包含在对 alert ( ) 函数的JavaScript调用中。在被调用域两边使用引号为限定符，比如：

```
alert('[Notes field]');
```

当该函数被触发时，在产生的消息框中将包含 User域的计算值。

这种技术非常强大，在后面的例子中，你将看到怎样使用这种技术产生整个 JavaScript语句，产生动态仿真系统。

## 5.6 显示域相关的帮助

Navigator3.04 <sup>TM</sup>	Navigator4.05 <sup>TM</sup>	Domino4.6.1 <sup>TM</sup>
Explorer3.0 <sup>O</sup>	Explorer4.01 <sup>TM</sup>	Domino5.0 <sup>TM</sup>

Notes客户端的一个非常好的功能就是域相关的帮助，当设计一个表单的时候，可以为每个域提供一个简短的帮助信息，当那个域获得焦点的时候，它的帮助显示在状态条上，使用JavaScript可以轻松地把这种功能复制到Web表单。

图5-3显示了在JSEamples.nsf数据库中的Field help表单当用户点击 Description域时的情形，注意在浏览器的状态条中有域相关的帮助信息。当点击另外一个域时，另外一个域的帮助信息被显示出来。当点击表单中空白区域时，状态条被清空。

这个表单使用了两个JavaScript 函数来获得上述功能：showhelp ( ) 和clearHelp ( )。三个提供帮助的域都被赋予了下面的HTML

```
"onFocus=\"showHelp(this);\" onBlur=\"clearHelp();\""
```



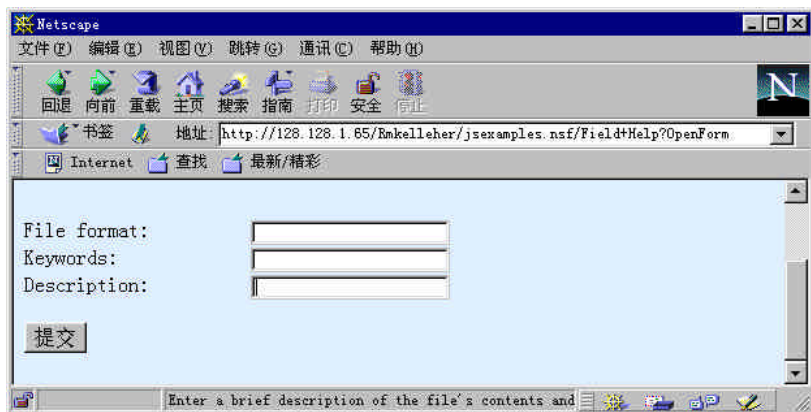


图5-3 这个表单使用JavaScript显示域特定的帮助

当点击一个可编辑域时，该域获得焦点，则该域的获得焦点事件发生。当在可编辑域外点击时，该域失去焦点，则该域的失去焦点事件发生。在这个例子中，域的获得焦点事件发生时调用 showHelp函数，这个函数以域本身为自变量。域的失去焦点事件发生时调用 clearHelp函数。这两个函数被包含在表单顶部的 <script>和</script>标签之间：

```
<script language="JavaScript">

function showHelp(obj) {
    var item = obj.name;
    var help = "";
    if (item == "Format")
        help = "Graphics must be valid GIF or JPEG files.";
    else if (item == "Keywords")
        help = "Enter one or more identifying keywords to help _
        users search for this file.";
    else if (item == "Description")
        help = "Enter a brief description of the file's contents _
        and purpose.";
    window.status = help;
    return true;
}

function clearHelp() {
    window.status = "";
}
</script>
```

showHelp函数首先认定被传递给它的对象名称，以便决定要使用哪一个帮助字符串，然后使用帮助字符串设置在浏览器底部的窗口的状态属性值。clearHelp函数仅仅把窗口的状态属性值设为空字符串。

当然，这是一个使用JavaScript在表单中添加域相关的帮助的非常简单的例子。对于一个

拥有大量的域的表单来说，这个简单的办法将使 JavaScript语句变得极其冗长。因此，也可以采取另一种办法，使用一个允许多值的域将域名映射为帮助文本，使用计算域产生相应的 JavaScript。这儿显示了Notes公式语句无可比拟的强大功能。例如，你可以通过使用加号“+”来将列表联合起来。假如 ListA为“ABC”：“DEF”：“GHI”而ListB为“123”：“456”：“789”，则ListA+ListB等于：

```
"ABC123" : "DEF456" : "GHI789"
```

由于这个小秘密的存在，我们可以在 Notes允许多值的域的基础上创建一些公式并返回 JavaScript的if-then语句。现在假设你有两个允许多值的域 FieldName 和Description，并分别包含下列值：

```
FieldName      " LastName " : " FirstName " : " MiddleInit "
```

```
Description    " Enter your last name " : " Enter your first name " : " Enter your middle  
initial "
```

根据这个信息，下面的计算显示域通过公式生成 JavaScript语句把域名映射为帮助文本：

```
"if (item == \"\" + FieldName + "\") help=\"\" + Description + "\";"
```

计算域将自动转换为下面的结果：

```
if (item == "LastName") help = "Enter your last name";  
if (item == "FirstName") help = "Enter your first name";  
if (item == "MiddleInit") help = "Enter your middle initial";
```

你可以在showHelp函数中插入域以便把解决问题的途径从使用复杂的代码转换为使用简单的配置。

**提示** 在JavaScript函数中当使用计算显示域或计算文本时，请记住 Domino对域的计算为从上到下，从左到右，因此，上面例子中的 FieldName 和Description域应被插在 showHelp函数以前。

## 5.7 控制帧的使用

Navigator3.04 o Navigator4.05 ™ Domino4.6.1 ™

Explorer3.0 o Explorer4.01 ™ Domino5.0 ™

我曾和我的伙伴为了创建用户的界面而使用 7个帧，但愿此生我再也不要再有第二次那样的经历。帧常常是用户感到界面不友好的根源，尤其是对那些视力上有缺陷的人来说更是这样。就我个人的观点来说，应该视那种包含大量内容经常改变的帧的界面如瘟疫。帧使一些简单的任务如打印或书签变得困难重重，使导航设计难以实现。为了保证每个帧在每种浏览器下都具有适当的宽度，适当的字符大小与风格，还需要相当多的专业知识与多次测试。

当然，在此我仍然提供了一个具有多个帧的例子，因为，说不定有一天你也会遇到我当年的遭遇，被迫使用多个帧去创建用户界面。另外，教你怎样设计界面并非本书的任务。可能你是一位狂热的帧爱好者，能找到大量必须使用帧的理由，这是你的权利。

在第3章中，我们说明了怎样使用简单的两个帧的界面创建一个视图。一个帧包含文档的

链接，另一个帧显示选定的文档。通过在每个链接中使用HTML目标属性可以完成这个工作。目标属性确定了在帧中将显示那个页的内容。

然而，你不能确定两个目标。图 5-4显示了在JSEexamples.nsf数据库中的帧的例子。

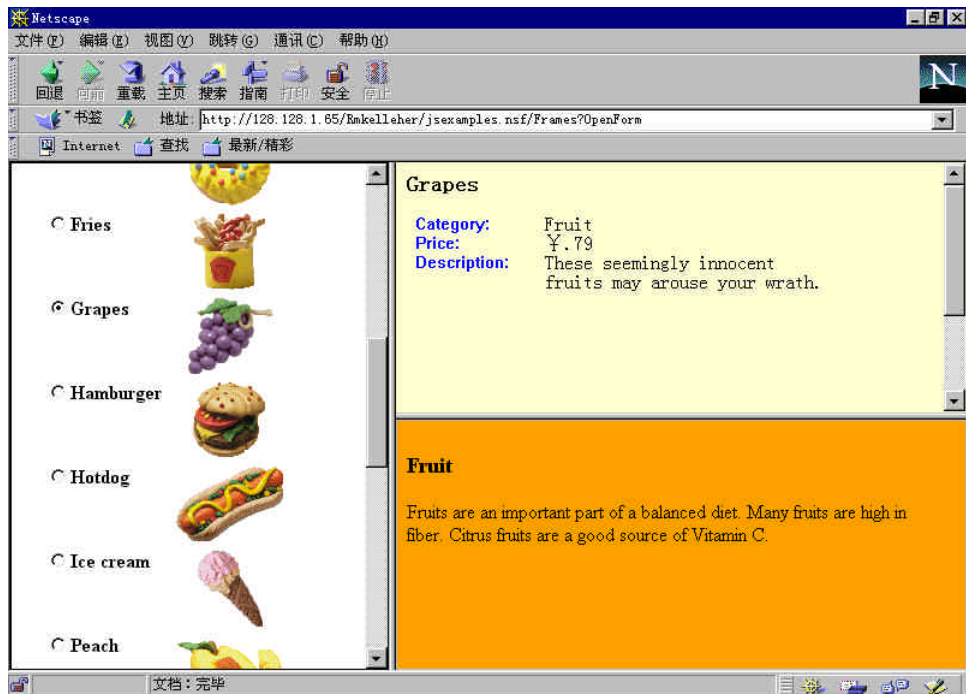


图5-4 一个简单的单击可以引发两个帧被更新

这个例子使用三个帧显示信息。在左边的帧中，显示 See Food视图，这个视图显示的每种不同食物的文档包括：名称、种类、价格、描述。而视图本身只显示了食物的名称并将其作为单选按钮显示。

See Food视图有两个列：第一个列显示了一个单选按钮，而第二个列显示了视图中的每个文档中存储的图形。下面是第一个列的公式：

```
"[</a><input type=\"radio\" name=\"food\" value=\"\" + Name + "\""
onClick=\"fillFrames(\" + Name + "\", \"\" + Category + "\");\">]" +
Name
```

上述公式为每个文档产生一个单选按钮，每个按钮都有自己的点击事件，该事件将调用 fillFrames方法。例如，如果食物名称为 Banana,则公式变为：

```
[</a><input type="radio" name="food" value="Banana"
onClick="fillFrames('Banana', 'Fruit');">]Banana
```

由于你不会使用当 Domino把视图转换为HTML时自动产生的<A HREF>链接，因此</a>标签的用途是终止该链接。这将使用户点击按钮时不是触发文档链接而是触发 fillFrames方法。

提示 控制Domino把视图转换为HTML的两种方便的技术为

在链接中显示一系列的值。视图中列的属性允许你控制 Domino把视图中的哪一列转换为<A HREF>链接。

把整个视图作为 HTML的内容。在视图的属性中可以使 Domino把整个视图当作 HTML通用文本。

FillFrames函数被放在显示视图的表单中（对于 See Food是\$\$ViewTemplate视图）：

```
<script language="JavaScript">
function fillFrames(item, category) {
    item = item.replace(' ', '+');
    window.open("/RMKelleher/JSEexamples.nsf/Food/" + item +
        "?OpenDocument", "Frame2");
    window.open("/RMKelleher/JSEexamples.nsf/FoodCategories/" +
        category + "?OpenDocument", "Frame3");
}
</script>
```

FillFrames函数根据item（如Banana）计算URL并在右上角的Frame2中应用其所得内容，然后再根据category（如Fruit）计算URL并在右下角的Frame3中应用其所得内容。每当用户选择按钮时都将对两个帧进行更新，显示该食物的项目和种类信息。

注意 这个例子要求已经把示例数据库拷贝到你的 Notes数据目录下的RMKelleher子目录下。最好使用公式返回数据库文件名而不是硬编码，例如：

```
@ReplaceSubstring(@Subset(@>DbName; -1); " " : "\\\"; "+" : "/" )
```

## 5.8 填充动态列表框

Navigator3.04 <sup>TM</sup>      Navigator4.05 <sup>TM</sup>      Domino4.6.1 <sup>TM</sup>

Explorer3.0 o      Explorer4.01 <sup>TM</sup>      Domino5.0 <sup>TM</sup>

使用JavaScript填充一个动态列表框事实上是一件非常简单的事情。如果你已经有一个字符串数组，可以简单地在数组中使用一个循环使每一个字符串对应一个列表框选项。例如下面的JavaScript使用字符串填充一个名为ProductCategory的列表框：

```
list = new Array("Hardware", "Software", "Other");
select = document.forms[0].ProductCategory;
len = list.length;
select.length = len;
for (i = 0; i < len; i++) {
    ProductCategory.options[i].text = list[i];
    ProductCategory.options[i].value = list[i];
}
```

现在假设你已经有了一个名为Product的列表框，当用户选择一个产品类别时，Product列表框应该显示属于该类别的产品。解决问题的办法看起来非常简单，当ProductCategory列表框的改变事件发生时，它触发了一个根据所选内容填充一个列表框的函数。例如：

```
if (selection == "Hardware") {  
    array = new Array("PC", "CD-ROM drive", "Modem");  
} else if (selection == "Software") {  
    array = new Array("Window 95", "Lotus Domino", "MS Word");  
} else if (selection == "Other") {  
    array = new Array("Toner", "Zip disks", "Floppies");  
}  
... use array to populate Product listbox ...
```

如果你想通过使用固定代码直接编写脚本语句，那么 Domino不会显示任何特别的优势，你可能想使用固定的HTML文件，然后每当有新的产品或产品类别时就更新一次脚本。

如果你的产品列表经常变化，那么代码的不断升级很显然将变成一种很烦人的负担，因此我们在这儿引入了Domino。使用Domino，你可以把生成产品和产品类别的列表的工作放在最终用户端。用户可以通过文档来增加、删除或修改显示在视图中的产品。你作为开发者使用这些视图来生成可以自行更新的Javascript代码。

图5-5显示了JSExamples.nsf数据库的产品表单。在这个例子中，根据用户在 Product Category域中的选择来改变Product域的选项。两个列表的内容由数据库的一个视图决定。



图5-5 根据选定的产品类的不同，表单中的 Product域显示不同的列表

如果这是一个Notes客户端的应用，可以使用公式计算 ProductCategory域的值并将其作为 ProductName域的选项的确定基础。在ProductCategory域中简单地选择“按关键字变化刷新域”（当该域的值发生变化时驱动所有的域初始化）。在ProductName域中选择“在刷新文档时刷新选项”（当发生某些行为时该域被驱动）。

但是在Web应用程序中，上述技术并不生效：ProductName列表框只是当表单下载时进行一次计算，当类别改变时需要我们来保证适时刷新。

在ProductCategory域中显示了一个由 Computer Products视图导出的类别列表。如果你熟悉Notes和Domino开发，应该明白这不是一个太难的问题，由公式生成关键字域是一个标准的功能。公式使用 @DbColumn得到类别列表：

```
" " : @Unique(@DbColumn("Notes":"NoCache"; "" ; "Products"; 1))
```

默认值为空，这意味着在选择任何合法有效的值的时候必须改变域的值，触发改变事件。

ProductName域，尽管也是一个关键字域，但并未使用由公式生成关键字域功能而是列出了固定选项：

一系列的连字符只代表了域的初始宽度，以便当以后选项被添加进来时可以被全部显示出来。

下一步工作是保证当选择一个产品类别时会触发某些 JavaScript。为此，ProductCategory域的HTML属性公式为：

```
"onChange=\"changeMenu(this.form);\""
```

改变事件将触发在页面头部<script>和</script>标签之间的changeMenu函数：

```
<script language="JavaScript">
function getMenuOptions(category) {
    var s = "";
    [makeString field]
    s = s.substring(1, s.length);
    var myArray = s.split("~");
    return myArray;
}
function changeMenu(form) {
    var category =
form.ProductCategory.options[form.ProductCategory.selectedIndex].text;
    var myArray = getMenuOptions(category);
    form.ProductName.options.length = myArray.length;
    for (var i = 0; i < myArray.length; i++) {
        form.ProductName.options[i].value = myArray[i];
        form.ProductName.options[i].text = myArray[i];
    }
}
</script>
```

changeMenu函数调用getMenuOption得到所选类别的产品列表。详细检查 getMenuOption函数发现我们漏掉某些过程。创建必要的产品选项数组的 JavaScript语句在哪里呢？在你看到的[makeString field]中，插入一个名字叫做makeString的显示时计算域，遗漏的JavaScript语句事实上藏在这个域中。这个域中的公式从 Computer Products视图中得到值并由此产生一系列的JavaScript的if-then语句。这个允许多值存在的域被设置为以空格为界定符。下面的公式为：

```
list1 := @DbColumn("Notes":"NoCache"; "" ; "Products"; 1);
list2 := @DbColumn("Notes":"NoCache"; "" ; "Products"; 2);
"if (category == \" + list1 + "\") s += \"~\" + list2 + "\"; "
```

假设视图容纳的值如表5-1。

makeString域返回如下JavaScript语句：

```
if (category == 'Hardware') s += '~Conner Tape-Store 150';
(category == 'Hardware') s += '~IBM 350-Plus 3(category ==
'Hardware') s += '~Iomega 100 MB ZIP drive (category ==
```



```
'Software') s += '~Lotus Notes 4.1a'(category == 'Software')
s += '~Windows 95'if (category == 'Software') s += '~Windows NT
4.0';
```

表5-1 Computer Products视图中的值

Column1	Column2
Hardware	Conner Tape-Stor 250
Hardware	IBM 350-p133
Hardware	Iomega 100 MB ZIP drive
Software	Lotus Notes 4.5
Software	Windows 95
Software	Windows NT 4.0

结果得到由适当的产品与连字符“~”组成的字符串。然后这个字符串被 JavaScript函数 split()（类似于 Notes函数 @Explode）转换为数组，最终，函数向 changeMenu函数返回数组，并使用该数组填充 ProductName列表框。

对于 @DbColumn，该技术的限制为 64K 字节，对于文本域，其字符不能超过 15K 字节长度。事实证明，当你要为用户提供上百个选项从而超出了上述限制的时，你将面临非常严峻的问题，因此你应该考虑更改你的设计思路。

5.9 例子：动态下载图像

```
Navigator3.04  TM   Navigator4.05  TM   Domino4.6.1  TM
Explorer3.0   o    Explorer4.01   TM   Domino5.0    TM
```

直到本章为止，我们已经使用 JavaScript 处理过许多种类型的对象：窗口、帧、列表框（被叫做选择型元素）、单选按钮。但我们从来没有处理过图像对象。这个例子描述了你怎样使用 JavaScript 和 Domino 根据用户选择联合处理动态下载图像问题。

图 5-6 显示了在 JSExamples.nsf 数据库中的 Grocery List 表单在用户从列表框中选择了 Peach 后的情况。在表单中显示了桃子的图形，同时在 Price 和 Description 域显示关于 Peach 的有关信息。

FoodItem 域显示了食物的列表，用户可以选择食物。这个域的改变事件触发了一个对 showDetail 函数的调用，该函数在表单头的 <script> 和 </script> 标签之间。FoodItem 域包含以下 HTML 属性公式：

```
"size=5 onChange=\"showDetails(this);\""
```

选择对象本身作为自变量传递到 showDetails 函数中：

```
function showDetails(obj) {
    var text = obj.options[obj.selectedIndex].text;
    var value = obj.options[obj.selectedIndex].value;
    var foodImage = document.images["FoodImage"];
    [ComputedJavaScript field]
}
```



图5-6 根据用户的选择表单下载适当的图形

在showDetails函数的结尾，一个名为ComputedJavaScript的计算显示域被插入。这个域返回JavaScript代码，这些代码显示了每个选项的图像、价格以及描述等。域的公式如下：

```
"if (text == \"\" + NameList + "\") { " +
  " foodImage.src = \"\" + ImageList + "\";" +
  " obj.form.Price.value = \"\" + @Text(PriceList; "C") + "\";" +
  " obj.form.Description.value = \"\" + DescriptionList + "\";" + "}"
```

NameList、ImageList、PriceList、DescriptionList 是一些计算域，使用 @DbColumn函数返回视图列的内容。NameList列出了项目名称：如 Banana\Broccoli等等。ImageList列出每个图像的URL，这些URL作为附件保存在相应的Food文档中。PriceList列出了每个项目的价格，以现金格式保存。DescriptionList 列出了每个项目的简短描述。当页面下载时，JavaScript计算域返回的结果如下：

```
if (text == "Banana") { foodImage.src =
"/RMKelleher/JSEExamples.nsf/Food/Banana/$File/banana.gif";
obj.form.Price.value = "$0.50"; obj.form.Description.value = "A
luscious
tropical fruit."; } if (text == "Broccoli") { foodImage.src =
"/RMKelleher/JSEExamples.nsf/Food/Broccoli/$File/broccoli.gif";
obj.form.Price.value = "$0.60"; obj.form.Description.value = "A
green
vegetable rich in Vitamin D."; } if (text == "Cake")
{ foodImage.src =
"/RMKelleher/JSEExamples.nsf/Food/Cake/$File/cake.gif"; obj.form.Price.value
"$1.19"; obj.form.Description.value = "A little slice of heaven."; }
```

...

这样一来，当食物被选定以后，这个脚本将改变名为 FoodImage 的图像对象的 src 属性。当页面被下载时，用默认值创建对象 FoodImage。更新它的 src 属性可使 Food Image 显示不同的图像。可以把 src 属性看做 HTML 的 <IMG> 标签的属性：

```

```

然后，这个脚本根据从 Food 视图中返回的值设置 Price 和 Description 域的值。

如果你已经看过了上面的几个例子，可能已经得到了一个基本模式：Notes 公式语句的列表处理特性对根据视图内容计算重复的 JavaScript 语句有相当强大的作用。

## 5.10 例子：在一个滚动按钮中预先下载图片

Navigator3.04 <sup>TM</sup>      Navigator4.05 <sup>TM</sup>      Domino4.6.1 <sup>TM</sup>

Explorer3.0 o      Explorer4.01 <sup>TM</sup>      Domino5.0 <sup>TM</sup>

在上面的例子中，选择一个项目将导致动态下载图片。在这个例子中，我们在页面下载时就下载所有的图片，以使当需要使用它们时能够立即显示出来。这是非常重要的，比如在这个例子中的滚动按钮，它由一系列的按钮组成，当鼠标移到哪个按钮上时哪个按钮的显示将改变。

图 5-7 显示了当你在 Netscape 中打开 Auto Parts 数据库时 (AutoParts.nsf) 的图像。在图 5-7 中，当用户把鼠标的指针移到 Part by Part Name 按钮上时，这个按钮与其他按钮看起来有所不同，似乎这个按钮已经被按下去。



图5-7 当你的鼠标移向某个按钮的时候该按钮的外表改变

在此使用了两个图像：GreenButton1.gif(未选的按钮图像)和GreenButton2.gif(选择的按钮图像)。这些在视图中显示的图像被保存在文档的附件中。这些图像的关键字为文件名，因此我们可以使用附件的URL加上关键词来访问每个图像：

```
/RMKelleher/AutoParts.nsf/GreenButton1.gif/$FILE/GreenButton1.gif
```

下面的脚本表示当页面被下载的同时下载图像。这个 JavaScript脚本创建了两个 Image对象，并设置了它们的src属性，但是并未显示它们。在此，需要一个HTML的<IMG>标签。

```
var selectedImage = new Image();
selectedImage.src = "[Db]/Images/GreenButton2.gif/$FILE/
GreenButto2.gif";
var deselectedImage = new Image();
deselectedImage.src = "[Db]/Images/GreenButton1.gif/$FILE/
GreenButton1.gif";
```

注意在这儿的JavaScript脚本中使用了[Db]，一个已经被插入页面中的被计算文本热点，这个热点返回数据库文件名：

```
@ReplaceSubstring(@Subset(@DbName; -1); "\\\"; "/" )
```

计算文本热点类似于一个计算显示域，使用文本热点的好处在于你不必像域那样设置属性。计算文本热点从4.6版本开始引入，因此，早期版本不能使用。

下一个难题是<IMG>标签，默认情况下，每个图像被设置为未选时的图像 GreenButton1.gif。但是我们并未简单地把该图像传递到文档中，而是使用 HTML通用文本保证每个<IMG>标签被赋予惟一的名称：Image1、Image2、Image3或Image4。

```
[<a href = "/RMKelleher/AutoParts.nsf/BodyGroups"
onMouseOver="selectImage('Image1');"
onMouseOut="deselectImage('Image1');" ></a>]
[<a href="/RMKelleher/AutoParts.nsf/PartsByGroup"
onMouseOver="selectImage('Image2');"
onMouseOut="deselectImage('Image2');" ></a>]
```

```
[<a href="/RMKelleher/AutoParts.nsf/PartsByName"
onMouseOver="selectImage('Image3');"
onMouseOut="deselectImage('Image3');" ></a>]
```

```
[<a href="/RMKelleher/AutoParts.nsf/OrderForm"
```

```
onMouseOver="selectImage('Image4');"  
onMouseOut="deselectImage('Image4');"></a>]
```

在上面的HTML中，通过使用<A HREF>链接使每个<IMG>标签链接一个不同的 Domino 视图。这些链接之所以重要是由于下述两个原因：

我们需要显示选定视图的方法。当用户点击这个链接时将显示链接的视图。

链接对象需要一个获得鼠标和失去鼠标事件以便编写脚本。

当用户将鼠标指针移到链接上时，获得鼠标事件发生。当用户将鼠标指针从链接上移开时，失去鼠标事件发生。于是，获得鼠标事件触发 selectImage函数，而失去鼠标事件触发 deselectImage函数。

```
function selectImage(imageName) {  
    document.images[imageName].src = selectedImage.src;  
}  
function deselectImage(imageName) {  
    document.images[imageName].src = deselectedImage.src;  
}
```

每当selectImage函数或deselectImage函数被调用时，一个现存的图像的名称被作为自变量传递到函数中去。函数可以把图像的 src属性设置为适当的URL，引起显示不同的URL。由于selectedImage和deselectedImage对象已经被下载，因此速度非常快。

## 5.11 例子：显示对话框

Navigator3.04 <sup>TM</sup>      Navigator4.05 <sup>TM</sup>      Domino4.6.1 <sup>TM</sup>

Explorer3.0 <sup>O</sup>      Explorer4.01 <sup>TM</sup>      Domino5.0 <sup>TM</sup>

对于Notes客户端的开发者来说，另外一个漂亮的组件是对话框。对话框在使数据输入表单尽量简单的情况下提供一组域与按钮群帮助用户轻松输入数据。联合使用设计区域和 LotusScript 的 DialogBox方法，可以轻松创建对话框从而简化表单界面。Domino不能使用这种功能，主要是由于它将任何东西都转换为HTML形式，但是HTML中没有类似对话框那样的组件。然而，仍然可以通过JavaScript创建对话框。

如果打开AutoParts数据库（AutoParts.nsf）并点击Order Parts链接，将会看到一个简单的订购表单。选择 Select Parts按钮将会得到如图5-8所示的对话框。

在对话框的左边显示了一个零件列表如锁扣或锁眼等。你选择某个项目时，将会显示零件编号和价格。然后可以输入数量并选择 Add按钮就会在你的订购单上增加相应的订购项目。订购项目列表在对话框右边显示。

如果要从订购单中删除某个项目，选择该项目并选择 Remove按钮。如果要删除所有订购项目重新开始的话，选择 Remove All按钮。当你对订购项目满意以后选择 OK按钮返回订购表单，现在该表单显示你订购的零件列表（见图5-9）。

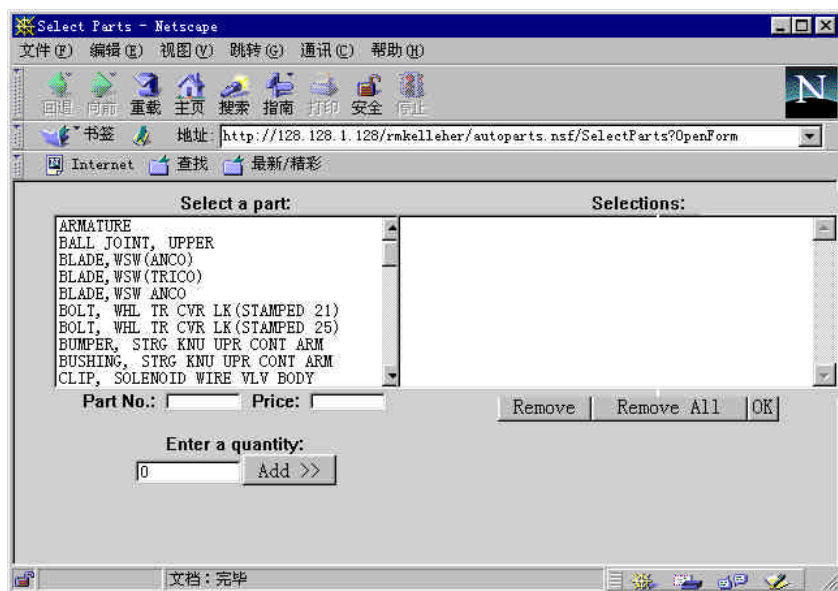


图5-8 这个对话框帮助用户订购自动零件

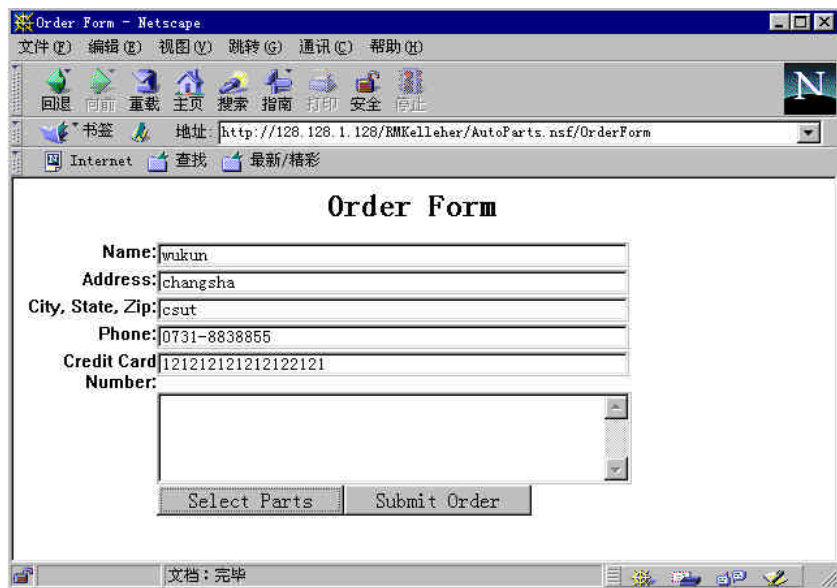


图5-9 Select Parts对话框用来填充这个订购表单

最后，选择Submit Order按钮提交你的表单。要看到提交过的表单，打开 Orders 视图，完成的订购表单如图 5-10所示。

与以前的例子不同，这个例子由几个部分协同工作，在本节中我们将分别讨论以下几部分内容：



- Order Form 表单。
- Select Parts 表单。
- Read-Only Order Form 表单。
- Orders 视图。

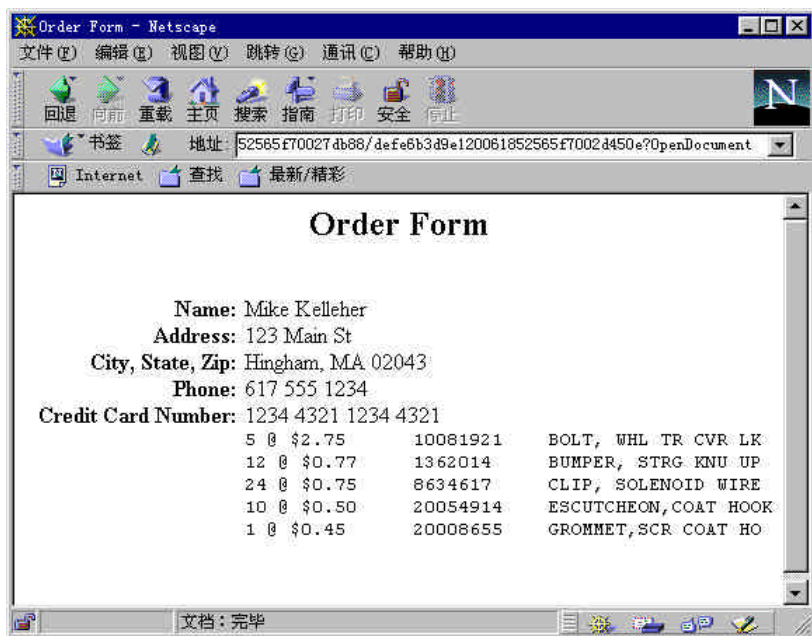


图5-10 使用特定的只读表单显示整个订购表单

### 5.11.1 Order Form 表单

Order Form 表单包含了几个简单的数据输入域，一个列表框，一些 JavaScript 按钮。列表框实际上是一个名为 OrderDetails 的关键字域，在这个域中存储了用户在对话框中的选择。与表单中的其他域不同，OrderDetails 域的字体为一种固定宽度的字体 Courier。

可以使用 HTML 通用文本添加 Select Parts 按钮和 Submit Order 按钮：

```
<input type="button" value="Select Parts" onClick="selectParts();">  
<input type="button" value="Submit Order" onClick="doSubmit();">
```

Submit Order 按钮代替了由 Domino 自动产生的真正的 Submit 按钮。真正的 Submit 按钮已经由 HTML 命令抑制其产生。按惯例 Submit 按钮完成当点击事件发生时提交表单的功能。但由于我们的应用程序除了提交表单以外需要更多的功能，因此我们必须创建自己的 Submit Order 按钮。

下面看一下 Select Parts 按钮。这个按钮触发用户自定义的 JavaScript 函数 selectParts()。selectParts 函数以名为 Parts 的窗口为目标调用 Select Parts 表单的 URL。如果这儿已经有一个名为 Parts 的窗口或帧被打开，那么 Select Parts 表单将显示在该窗口中。如果没有名为 Parts 的窗

口或帧被打开，则调用 Window.open()函数来创建一个叫做 Parts的窗口并以给定的常量来定义该窗口的宽和高。

```
// do this when Select Parts button is clicked
function selectParts() {
    url = "/RMKelleher/AutoParts.nsf/Select1Parts?OpenForm";
    options = "width=700,height=310";
    popup = window.open(url, "Parts", options);
}
```

点击Submit Order 按钮触发doSubmit()函数，doSubmit()函数完成如下两个功能：

- 选择所有在 OrderDetails域中列出的选项。
- 提交表单。

因为这些值需要在被提交的文档中记录，所以第一个功能，选择所有在 OrderDetails域中列出的选项是非常必要的。Select Parts对话框仅仅填充列表而并未选择某个选项。请记住这个关键字域的值由所选选项的值来确定，而不是由可以选择的选项的值来确定。doSubmit()函数在选项列表中循环将被选择属性设为真。这样一来，OrderDetails域中的值就与对话框创建的列表值相同了。

```
// do this when submitting
function doSubmit() {
    form = document.forms[0];
    OrderDetails = form.OrderDetails;
    len = OrderDetails.options.length;
    for (i = 0; i < len; i++) {
        OrderDetails.options[i].selected = true;
    }
    form.submit();
}
</script>
```

### 5.11.2 Select Parts表单

在实际的对话框中我们使用了 Select Parts表单，该表单设计为 700×310像素大小。本表单中的大部分JavaScript脚本由列表框选择或按钮点击事件触发，但也有一个函数当页面下载时自动触发，这个函数为init()：

```
function init() {
    PartList = document.forms[0].PartList;
    Selections = document.forms[0].Selections;
    PartNumber = document.forms[0].PartNumber;
    Price = document.forms[0].Price;
    Qty = document.forms[0].Qty;
    PartList.options[0] = null;
    Selections.options[0] = null;
    OrderDetails = window.opener.document.forms[0].OrderDetails;
    len = OrderDetails.options.length;
```

```

        Selections.options.length = len;
        for (i = 0; i < len; i++) {
            Selections.options[i].text = OrderDetails.options[i].text;
        }
    }
}

```

init()函数初始化项目的列表以便反映出原来输入的值，这允许我们在关闭并重新打开对话框时不需要再次输入原来的订单单。init()函数通过访问 window.opener属性得到 OrderDetails域的值。window.opener指向打开对话框窗口的窗口。使用这个属性，对话框可以从启动对话框的对象中得到或返回值。

当你选择了某个零件时，触发 selectPart()函数。这个函数可以取得选项的值。注意选项的值与选项在对话框中显示的文本可能并不相同。这类似于 Notes中的关键字域中的别名，事实上，选项的实际值是零件的编号与零件的价格以 ~ 连字符连接而成的字符串。selectPart()函数取得这个值并通过调用 JavaScript 函数split()将其转换为一个数组。这样就可以得到每个产品的编号和价格的值，并将这些值添入表单的 PartNumber和Price域。

```

function selectPart() {
    i = PartList.options.selectedIndex;
    tmp= PartList.options[i].value.split('~');
    partNumber = tmp[0];
    price = tmp[1];
    PartNumber.value = partNumber;
    Price.value = price;
}

```

Add、Remove、Remove All和OK按钮通过使用HTML通用文本创建。每个这样的按钮都以点击事件触发相应的JavaScript函数：

```

<input type="button" value="Add >>" onClick="addButton();">
<input type="button" value="Remove" onClick="removeButton();">
<input type="button" value="Remove All" onClick="removeAllButton();">
<input type="button" value="OK" onClick="okButton();">

```

addButton()按钮在 Selections列表中添加一项。首先它通过调用用户自定义的函数 validateQty()保证在数量域中输入的为数字。然后建立一个由数量、价格、零件编号、描述组成的项目列字符串。可以通过对它的列进行格式化使其与其他条目对齐。如果值太短，就在字符串上加空格补齐，太长则将其截短。

一旦该条目字符串已经准备好，则 addButton()函数把它加入到列表中。为了完成这个操作，它必须通过增加选择数组的长度给该字符串存储空间，然后将该字符串赋于数组的最后一个元素。

```

function addButton() {
    validateQty();
    if (Qty.value == 0 || PartNumber.value == "" || Price.value == "")
        return;
    entry = pad(Qty.value + " @ " + Price.value, 15) +
        pad(PartNumber.value, 12);
}

```

```

    entry += pad(PartList.options[PartList.options.selectedIndex].
text, 20);
    len = Selections.options.length;
    Selections.options.length = ++len;
    Selections.options[len-1].text = entry;
}
function pad(string, newLen) {
    oldLen = string.length;
    if (oldLen == newLen) return string;
    else if (oldLen < newLen) {
        diff = newLen - oldLen;
        for (i = 0; i < diff; i++)
            string += " ";
        return string;
    } else {
        return string.substring(0, newLen-1);
    }
}
function validateQty() {
    if (Qty.value == "") Qty.value = 0;
    else if (isNaN(parseInt(Qty.value))) Qty.value = 0;
}

```

removeButton()函数通过判定选定的是哪个零件并将其设为空值，从而把选定的零件从原选择项目的列表中移出。从选择数组中把选定的零件删除并把选择数组的大小相应减少。

```

function removeButton() {
    i = Selections.options.selectedIndex;
    Selections.options[i] = null;
}

```

在此，removeAllButton函数通过将选择数组的长度设为0而删除所有的选择零件。

```

function removeAllButton() {
    Selections.options.length = 0;
}

```

okButton()函数把选择列表中的值拷贝到 OrderDetails域中。就像init()函数一样，这个函数使用 window.opener属性指向本窗口的父窗口，只是这一次不是从该窗口中得到信息而是向该窗口中输出值。

```

function okButton() {
    orderDetails = window.opener.document.forms[0].OrderDetails;
    len = Selections.length;
    orderDetails.length = len;
    for (i = 0; i < len; i++) {
        orderDetails.options[i].text =
            Selections.options[i].text;
        orderDetails.options[i].value =

```

```

        Selections.options[i].text;
    }
    self.close();
}

```

最后，okButton()函数通过调用close()方法关闭对话框。

### 5.11.3 Read-Only Order Form 表单

Read-Only Order Form 表单的目的是显示原来提交的定单，但并不显示一些不必要的数据控件以免使界面非常杂乱。这个表单由一些显示定单的内容的域组成。

### 5.11.4 Orders 视图

为了显示 Read-Only Order Form 表单中存在的定单，Orders 视图使用了 Notes 的特殊功能叫做表单公式。这种类型的公式决定在视图中使用哪种表单显示文档。通过在表单公式中增加条件可以得到相当奇妙的结果。但在此我们只是根据需要添加了一个非常简单的公式：

```
"Read-Only Order Form"
```

## 5.12 例子：操纵 cookie

Navigator3.04 <sup>TM</sup>	Navigator4.05 <sup>TM</sup>	Domino4.6.1 <sup>TM</sup>
Explorer3.0 <sup>O</sup>	Explorer4.01 <sup>TM</sup>	Domino5.0 <sup>TM</sup>

cookie 数据库 (Cookie.nsf) 包含了一个名为 Grocery Store 的例子，这个例子使用 JavaScript 操纵 cookie。一个 cookie 是指由浏览器控制的存储在文件中的一小段信息。根据用户对浏览器的设置，一个脚本可以使用同一个域中创建的 cookie。在记录用户信息，尤其是用户的偏好（例如是否使用帧）的时候，cookie 可以将这种信息保存在客户端而不是服务器端。cookie 与浏览器的安装程序有关，因此，对于同一台计算机的不同用户来说，他们将使用同样的 cookie。同样地，当一台计算机在以不同的 IP 地址连接到 Internet 时应该保持同样的 cookie。

得到和设置 cookie 的最方便的途径是使用 JavaScript。例程 Grocery Store 使用 JavaScript 按钮触发增加和减少 cookie 的函数。在某个项目上单击加号 (+) 将在数量上增加一。在某个项目上单击减号 (-) 将在数量上减少一。

```

// increment quantity of this item
function increment(item) {
    var n = getCookie(item);
    if (n == "")
        n = 1;
    else
        n = parseInt(n) + 1;
    var expDate = new Date();
    expDate.setTime(expDate.getTime() + (24 * 60 * 60 * 1000));
    setCookie(item, n, expDate);
}

```

```
// decrement quantity of this item
function decrement(item) {
    var n = getCookie(item);
    if (n == "") return;
    n = parseInt(n) - 1;
    if (n <= 0) {
        deleteCookie(item);
        return;
    }
    var expDate = new Date();
    expDate.setTime(expDate.getTime() + (24 * 60 * 60 * 1000));
    setCookie(item, n, expDate);
}
```

getCookie()、setCookie()、deleteCookie函数是存储在 Cookie Functions 子表单中的一般 cookie 函数。为了便于在表单中插入这些函数，我们将这些函数保存在子表单中以免在使用时不得不使用拷贝粘贴命令或者重新键入这些函数。

当你提交 Grocery Store 表单时，LotusScript 代理接管该表单并显示你的总数。在第 6 章“编写 LotusScript”中我们再详细讲述 LotusScript 代理的问题。

### 5.13 例子：ActiveX 对象脚本

Navigator3.04 o Navigator4.05 o Domino4.6.1 ™  
Explorer3.0 ™ Explorer4.01 ™ Domino5.0 ™

微软的 IE 支持一种叫做 ActiveX 的技术，可以认为它是 OLE（对象的链接和嵌入）的另一个名字。一个 ActiveX 控件就是一个在浏览器中运行的一段小程序。使用 IE 中的 <OBJECT> 标签或者 Netscape Navitagor 的 <EMBED> 标签可以在 HTML 页中添加一个 ActiveX 控件。

**提示** 通过安装 NCompass 公司（<http://www.ncompasslabs.com>）的一个插件，你可以在 Netscape 中添加对 ActiveX 的支持。

在理论上，当在 Notes 文档中嵌入一个 ActiveX 控件时，在该文档被转换为 HTML 时 Domino 自动产生一个 <OBJECT> 标签或者 <EMBED> 标签。但事实上，根据笔者的测试这种功能并不能可靠地工作。作为一个编程环境，你可以使用微软公司的 ActiveX Control Pad（一个可以从 <http://www.microsoft.com> 下载的自由软件）。由于包含 Windows 注册表中 class ID 的标签相当长，因此使用 ActiveX Control Pad 要比手工编写 HTML 标签容易得多。然后你可以将这些标签拷贝到 Domino 的 Web 页中。

当你在 Web 页中嵌入 ActiveX 控件时，我们假设用户已经把这些控件安装在客户端的计算机里。根据在用户的浏览器中的设置和许可证中对 ActiveX 控件的限制，有些时候我们可以下载一些在用户的计算机上没有安装的 ActiveX 控件。

由于嵌入 ActiveX 控件对客户工作站有一些特殊的要求，同时由于嵌入 ActiveX 控件会带来一些安全上的问题，因此 ActiveX 控件更多的是使用在 intranet 上面而不是使用在公众网站中。



使用JavaScript，可以编写一些脚本来操作在 Web页中嵌入的ActiveX对象。在本节中我们提供了一个关于微软的Chart组件的例子，为了这个例子的正常运行必须在IE中安装适当的ActiveX控件。

图5-11显示了ActiveX例程数据库（ActiveX.nsf）中的一个表单在IE中的显示。这个表单包含5个ActiveX控件：1个Microsoft Chart控件和4个spinbutton按钮。当你单击箭头时可以改变spinbutton的值。在这个例子中，向左的箭头使值减小而向右的箭头使值增加。值事实上是spinbutton按钮自身的一个属性。

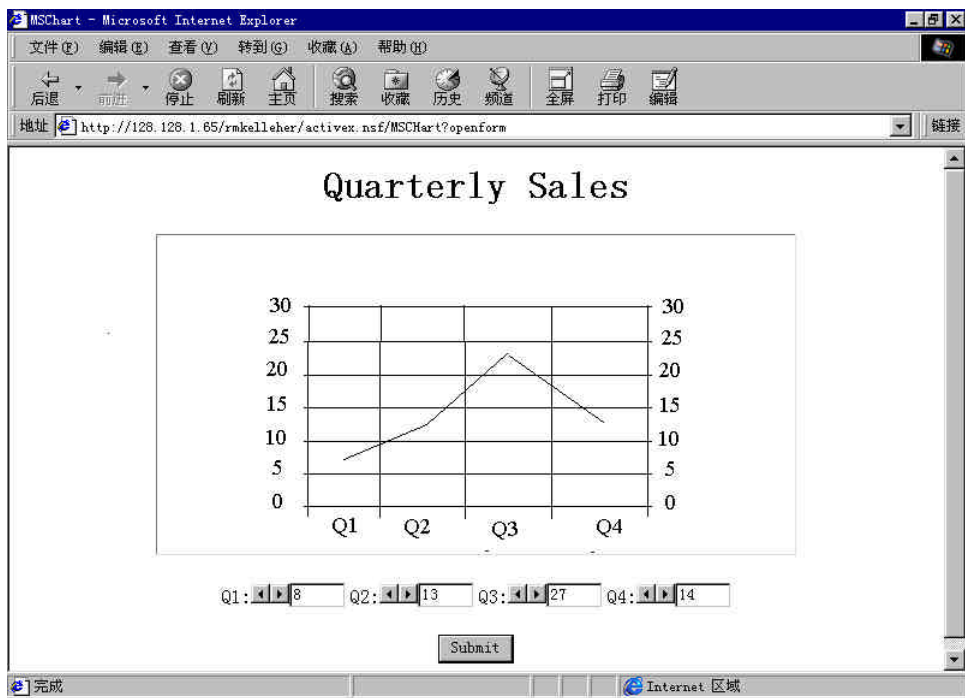


图5-11 当用户输入新数据的时候MS Chart控件立即更新

当用户改变spinbutton的值时，这个值立即在文本域中记录下来，文本域的改变立即触发Microsoft Chart控件的更新，改变了图表的显示。你也可以不使用spinbutton而是直接编辑文本域的值，这可以得到相同的结果。

现在，让我们揭开显示图形的面纱。图5-12显示了设计模式下的Chart表单。

由ActiveX Control Pad产生的<OBJECT>标签在页面中产生了一个MS Chart对象：

```
<OBJECT ID="MSChart1" WIDTH=533 HEIGHT=267  
CLASSID="CLSID:31291E80-728C-11CF-93D5-0020AF99504A">  
</OBJECT>
```

通过对象的ID属性命名chart对象可以在以后使用时能方便地得到它。在这个例子中，引用对象时可以使用document.MSChart1，或者更为简单地使用MSChart1。width和height属性可以控制对象的大小。Class ID是保存在Windows注册表中的用来识别特定的ActiveX控件的独特的值。

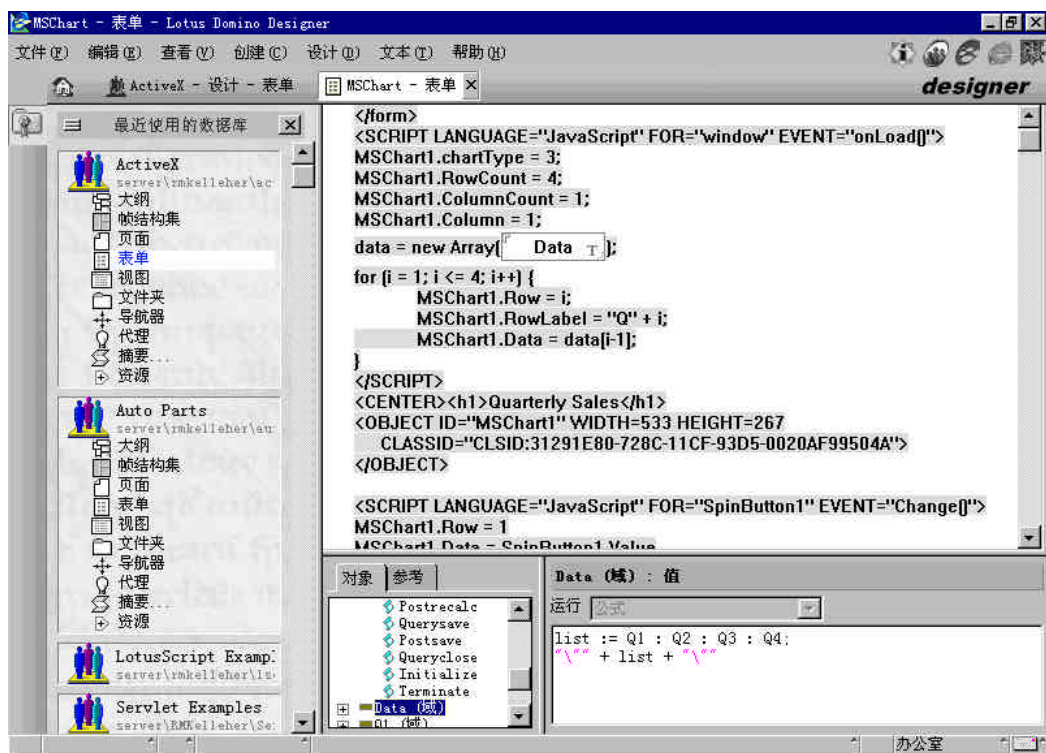


图5-12 MSChart表单使用JavaScript操纵ActiveX控件并把结果保存在域中

在表单设计的顶部使用下面的脚本在页面下载时填充 chart对象：

```
<SCRIPT LANGUAGE="JavaScript" FOR="window" EVENT="onLoad()">
MSChart1.chartType = 3;
MSChart1.RowCount = 4;
MSChart1.ColumnCount = 1;
MSChart1.Column = 1;
data = new Array([Computed-for-Display Field]);
for (i = 1; i <= 4; i++) {
    MSChart1.Row = i;
    MSChart1.RowLabel = "Q" + i;
    MSChart1.Data = data[i-1];
}
</SCRIPT>
```

<SCRIPT>的FOR属性指定触发脚本的事件的发生对象。EVENT属性指定触发脚本的事件。在这个例子中，当窗口对象的下载事件发生时触发脚本。然后脚本设置了图表对象的属性，比如图表类型（2D的线性表）、行数、数据列等。然后根据4个域的值填充表单：Q1、Q2、Q3、Q4。图表数据的改变将导致图表立即更新，因此该图表提供了域中数据的图形表示。

当用户单击spinbutton的箭头或者说当spinbutton的改变事件发生时，将触发下面的4个脚本。每个脚本都会做下面两件事情：

1) 改变图表单元格的值，触发图表的显示更新。

2) 改变相应的文本域的值。

```
<SCRIPT LANGUAGE="JavaScript" FOR="SpinButton1" EVENT="Change()">
MSChart1.Row = 1
MSChart1.Data = SpinButton1.Value
document.forms["MyForm"].Q1.value = SpinButton1.Value;
</SCRIPT>
```

```
<SCRIPT LANGUAGE="JavaScript" FOR="SpinButton2" EVENT="Change()">
MSChart1.Row = 2
MSChart1.Data = SpinButton2.Value
document.forms["MyForm"].Q2.value = SpinButton2.Value;
</SCRIPT>
```

```
<SCRIPT LANGUAGE="JavaScript" FOR="SpinButton3" EVENT="Change()">
MSChart1.Row = 3
MSChart1.Data = SpinButton3.Value
document.forms["MyForm"].Q3.value = SpinButton3.Value;
</SCRIPT>
```

```
<SCRIPT LANGUAGE="JavaScript" FOR="SpinButton4" EVENT="Change()">
MSChart1.Row = 4
MSChart1.Data = SpinButton4.Value
document.forms["MyForm"].Q4.value = SpinButton4.Value;
</SCRIPT>
```

对于每个spinButton1来说，表单中包含了类似下面的<OBJECT>标签：

```
<OBJECT ID="SpinButton1" WIDTH=32 HEIGHT=16
CLASSID="CLSID:79176FB0-B7F2-11CE-97EF-00AA006D2776">
<PARAM NAME="Size" VALUE="847;423">
<PARAM NAME="Orientation" VALUE="1">
</OBJECT>
```

在spinButton的<OBJECT>标签后面是相应的文本域；如针对 spinButton1的Q1。每个文本域有一个HTML公式属性使得当文本域更新时引发相应的 spinButton值更新，随后引发图表对象的更新。

```
" size=\"5\" onChange=\"SpinButton1.Value=this.value;\""
```

“现在等一下，”你可能想，“这不就成为一个死循环了？当 spinButton的值改变时引发文本域值的改变而文本域值的改变又会引发 SpinButton值的改变。在这儿工作流程到底应该是什么样子呢？”

谢天谢地，通过程序对文本域进行修改并不会触发文本域的的改变事件。只有用户手动修改才能触发文本域的的改变事件。

使用上述方法，我们可以随时保持表单中图表与文本域的同步。因此当表单被提交时 Q1、Q2、Q3、Q4域中的值反映了当前图表的状态，反之亦然。

如果你仅仅是想看一下已经创建的图表时，又该怎么做呢？在这种情况下，你并不需要

表单、文本域或者 spinbutton，你只是想要看到一个图表对象，在该图表对象中反映出存储在文档中的数据。我们有好多种方法来完成这个任务。我们可以为查看图表对象创建第二个表单，可以使用一个表单公式确定在什么时候使用什么表单。但是有一种最懒惰的方法，作者在此将使用这种最为快捷但并不漂亮的方法：将图表对象与其他对象放在两个不同的表单中。在设计表单的顶部，一个 `</FORM>` 标签使当表单被转换为 HTML 时由 Domino 自动产生的 `<FORM>` 标签不出现。在图表对象的 `<OBJECT>` 标签下面，我们插入我们自己的 `<FORM>` 标签。

```
<FORM METHOD=post ACTION="/RMKelleher/ActiveX.nsf/MSChart?
CreateDocument"
ENCTYPE="multipart/form-data" NAME="MyForm">
```

当文档不处于编辑模式时，在 `<FORM>` 标签下面的其他对象被设置为隐藏。这样一来，当文档以只读模式打开时，`onload` 脚本将根据当前的域值更新图表对象。这就是实现的全部过程，不显示 spinbutton，不显示文本域，也不显示提交按钮（见图 5-13）。

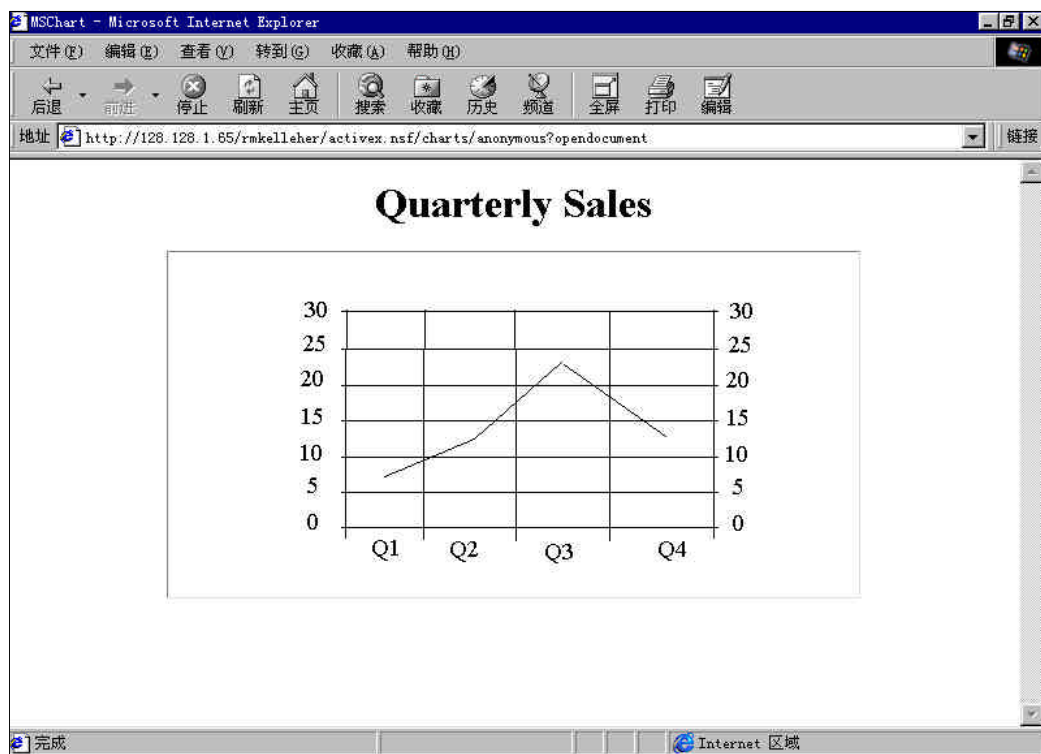


图5-13 当一个存在的MSChart文档以只读方式打开的时候，  
一个脚本根据当前域值自动填充它

现在，对 ActiveX 令人激动的特长你应该有所了解。但是我们仍然不得不在这儿多嘴插入一些细节问题。首先，这个特别的例子要求必须使用以某种方式设置的 IE 作为浏览器，以及一台安装有 ActiveX 控件的 Windows 操作系统的计算机。由于这种非常严格的要求，这个应

用程序不适用于可能被使用多种操作系统和多种浏览器访问的公共站点。

另一方面，由于在大部分客户端已经存在这些软件，用户不必下载它，因此，嵌入一个 ActiveX 比下载一个 Java 小应用程序要快得多。如果你为那些使用安装有 Windows 操作系统和 IE 的计算机的用户开发 Intranet，ActiveX 可能正是你所需要的开发组件。它可以使下载更快并且不会造成大量不必要的通信负担。

## 5.14 例子：Domino 产生的 JavaScript

Navigator3.04 <sup>TM</sup>      Navigator4.05 <sup>TM</sup>      Domino4.6.1 <sup>TM</sup>  
Explorer3.0 <sup>o</sup>      Explorer4.01 <sup>TM</sup>      Domino5.0 <sup>TM</sup>

在“数据库属性”对话框中，有一个默认情况下为禁用的属性：“Web Access: Use JavaScript when generating pages.”，将这个属性启用将导致 Domino 将表单转换为 HTML 的方法有非常重大的变化。最明显的变化就是 Domino 将不再为你自动生成 Submit 按钮。这个变化主要是由该属性改变导致的其他变化产生的。默认情况下，Domino 把任何不能转换为 Domino URL 的活动按钮忽略掉。例如，在表 5-2 中的操作公式可以被直接转换为 Domino URL。

表5-2 操作公式

活动公式	URL
@Command([Compose]; "Order Form");	/db.nsf/Order+Form?OpenForm
@Command([EditDocument]);	/db.nsf/view/unid?EditDocument
@UrlOpen("http://www.ibm.com")	http://www.ibm.com

然而，其他一些活动公式，比如在下面将要提到的，不能被转换为 Domino URL，因此在默认情况下 Domino 将仅仅忽略激发这些公式的按钮或热点。

```
@Command([FileCloseWindow]);  
@Command([FileSave]);  
@Command([ViewRefreshFields]);
```

当你将“Web Access: Use JavaScript when generating pages.”这个属性启用时，Domino 将把这种按钮转换为 JavaScript 按钮，这种按钮将激活 doClick() 函数。doClick() 方法将做以下两件事情：

- 设置一个 hidden\_Click 域的值去确定哪一个按钮被点击，然后将这个表单提交。
- 激活带有 Click= 参数的 URL 表明哪一个按钮被点击。

当 Domino 接收到这个提交的表单时，它将不自动保存文档，而是执行在 hidden\_Click 域中标明的活动公式。如果这个活动公式恰巧包含 @Command([FileSave]) 函数，则文档将被保存，并根据公式返回值。为了弄清楚工作原理，打开 Generated JavaScript 数据库 (GenJS.nsf)，这个数据库将“Web Access: Use JavaScript when generating pages.”属性启用。

图 5-14 显示了 GenJS.nsf 中的 Colors 表单。这个表单中有两个按钮和一个热点使用了 Notes

的公式语句，见表5-3。



图5-14 表单使用JavaScript处理按钮点击

表5-3 在Colors表单中的按钮的Notes公式

按 钮	公 式
Save	@Command([FileSave])
Save&Exit	@Command([FileSave]) @Command([FileCloseWindow])
Refresh	@Command([ViewRefreshFields])

Colors表单也包含三个域。第一个域 BgColor以HTML通用文本格式插入<BODY>标签内。这个计算显示域使用下面的公式：

```
@If(Color = ""; "white"; Color)
```

当表单第一次被下载时，Color 域为空白，因此背景色彩设置为白色。第二个域 Color是一个可编辑的关键字域，该域允许用户选择背景颜色。第三个域 \$\$Return详细说明了当用户提交表单时返回的HTML文本（注意这里用了“用户”这个词，以后你将知道它的重要意义。）

```
"<h1>Thank you</h1>"
```

图5-15显示Colors表单第一次下载时的情形。表单中的所有按钮都是可以使用的，而且在此没有Submit按钮。如果我们没有将“Web Access: Use JavaScript when generating pages.”属性启用，Save、Save&Exit和Refresh将消失，同时Domino将产生一个Submit按钮。



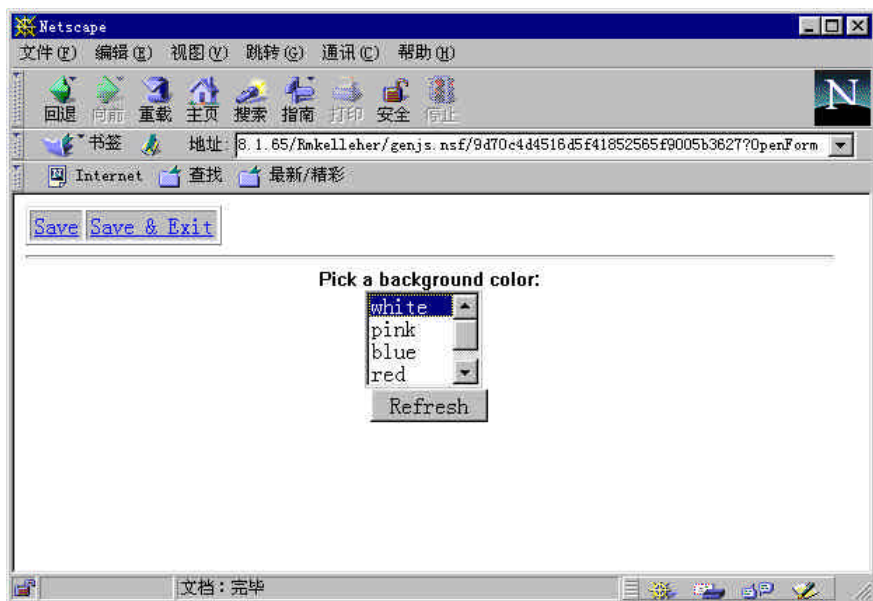


图5-15 Colors表单在显示时不包含提交按钮

现在选择查看 | 源文件去看一下由Domino为这个表单产生的HTML语句。每个按钮都有一个点击事件触发onClick()方法，该方法接收一个指针，指向在表单设计与公式链接的位置。

```
<HTML>
<!-- Lotus-Domino (Release 4.6.1 - March 3, 1998 on Windows
NT/Intel) -->
<HEAD>

<SCRIPT LANGUAGE="JavaScript">
<!--
function _doClick(v) {
  document._Colors.__Click.value=v;
  document._Colors.submit();
}
// -->

</SCRIPT>
</HEAD>
<BODY>

<FORM METHOD=post
ACTION="/RMKelleher/GenJS.nsf/9d70c4d4516d5f41852565f9005b3627?Open
Form&Seq=1"
ENCTYPE="multipart/form-data" NAME="_Colors">
<INPUT TYPE=hidden NAME="__Click" VALUE="0">
<TABLE BORDER=1 CELSPACING=2 CELLPADDING=2>
<TR VALIGN=middle BGCOLOR="#C0C0C0"><TD><A
HREF="javascript:_doClick('9d70c4d4516d5f41852565f9005b3627/$ACTION
```

```

S/0.1e')">Save</A></TD><TD><A
HREF="javascript:_doClick('9d70c4d4516d5f41852565f9005b3627/$ACTION
S/0.58')">Save & Exit</A></TD></TR>
</TABLE>
<HR><body bgcolor="white">
<BR>
<CENTER><B>Pick a background color:</B><BR>

<SELECT NAME="Color" size=4>
<OPTION SELECTED>white
<OPTION>pink
<OPTION>blue
<OPTION>red
<OPTION>yellow</SELECT>
<BR>

<INPUT TYPE=button
onClick="_doClick('9d70c4d4516d5f41852565f9005b3627/$Body/0.25c')"
VALUE="Refresh"></CENTER></FORM>
</BODY>
</HTML>

```

当单击Save按钮时，Domino执行这个按钮的公式，保存文档。然后返回同一个文档，只有这时，文档已被保存，同时每个计算域已被更新。

当单击Save&Exit按钮时，Domino执行的动作就像单击了一个Submit按钮：它保存文档而且返回\$\$Return域中的内容。

当单击Refresh按钮时，Domino只是更新计算域而不保存文档，只是在所有的计算域上添加新变化然后返回原文档。图 5-16显示了当在 Colors 表单中选择粉红色为背景色并点击 Refresh按钮后产生的效果。

作为一个例子，将“Web Access: Use JavaScript when generating pages.”这个属性启用将使你可以在不编写任何JavaScript语句的情况下添加一些功能性按钮。缺点是每次单击按钮都将触发一次到服务器的通信和一些服务器的处理过程。基本上，每个按钮的点击事件都相当于一个CGI程序。使用JavaScript的两个基本原因是：首先，它在客户端执行，节省了服务器与网络的资源。其次，它使你的应用程序的易用性大大提高。将“Web Access: Use JavaScript when generating pages.”这个属性启用可以得到第二个优点但无法得到第一个。

值得指出的是将“Web Access: Use JavaScript when generating pages.”这个属性启用将导致编写自己的JavaScript语句变得非常困难。考虑一下下面的热点公式：

```

js := "javascript:alert('\Hello!\');"
@URLOpen(js)

```

在正常情况下，在Netscape Navigator中点击热点将激发下面的JavaScript URL，在消息框中显示“Hello!”：

```

javascript:alert('Hello!');

```

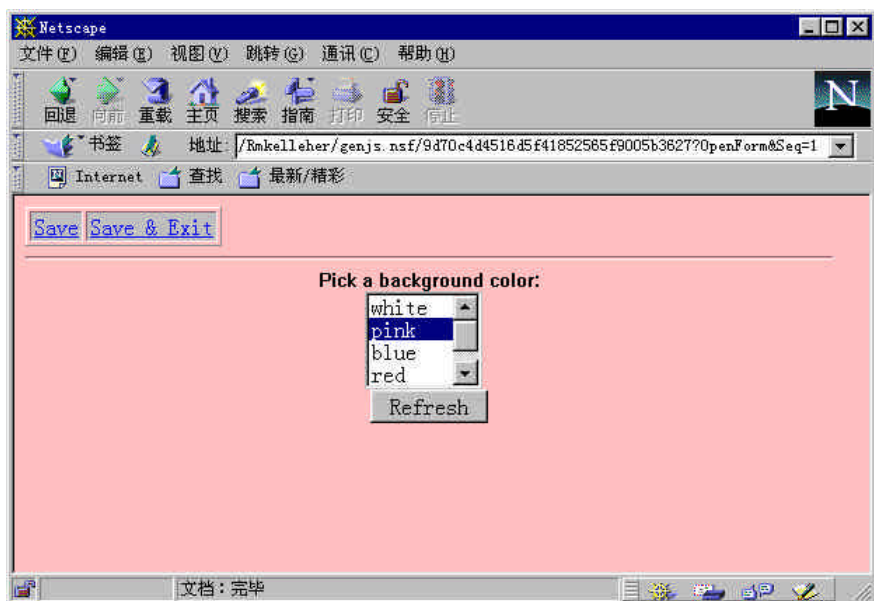


图5-16 这个表单已经被使用JavaScript刷新，但并未保存

然而当将“Web Access: Use JavaScript when generating pages.”这个属性启用的时候，如下事件将发生：

- 在链接上的单击触发JavaScript的doClick()函数。
- 表单被提交，然后Domino执行公式。
- Domino自动添加http://和主机名到JavaScript的URL中，并使浏览器指向一个不正确的URL，比如http://kelleher2/javascript:alert( ' Hello! ' )。
- 服务器返回一个未发现的错误信息。

总而言之，如果你能轻松地编写你自己的JavaScript，最好将“Web Access: Use JavaScript when generating pages.”这个属性禁用。

## 参考信息

关于JavaScript还有许多我们并未提及的东西。关于ActiveX我们只是看到了冰山的一角，甚至没有提及服务器端的JavaScript、JavaScript对象或者使用Netscape公司的LiveConnect技术在JavaScript和Java小应用程序之间通信。幸运的是，在下面这些站点上包含大量关于这些方面的有关信息。

要得到完全的、最新的JavaScript文档资料，请访问Netscape的在线Java指导：<http://developer.netscape.com/library/documentation/communicator/jsguide4/index.htm>

请教技术问题可以访问JavaScript的新闻组news:comp.lang.javascript

Iris 联合公司（Lotus Notes 和Domino的开发者）在“*Iris Today*”上以“Domino and JavaScript: Dynamic Partners”为名发表了大量的文章，在<http://www.notes.net>中也包含大量

有用的文章。

## 本章小结

你可以使用 Domino 独特的功能结合 JavaScript 建立一个功能强大的动态主页应用程序。特别是使用诸如在表单中的 JavaScript 中插入计算显示域（或计算文本），使用视图和多值域去产生一系列的 JavaScript 语句。在这一章中，学习了使用 JavaScript 技术验证域，计算域的值，显示域范围内的帮助，控制多个帧，填充列表框，下载图片，显示对话框，得到和设置 cookie，使用 ActiveX 对象编写脚本。

在数据库中将 “ Web Access: Use JavaScript when generating pages. ” 这个属性启用时将导致 Domino 改变把表单转换为 HTML 的方法。Domino 将不再自动产生 Submit 按钮，它将把每一个用户定义的按钮转换为将导致文档被提交的 JavaScript 函数。对于那些不想自己编写 JavaScript 语句的开发者来说这是一个很有用的功能，但是将以增加服务器负担为代价，同时会把自己编写的 JavaScript 语句改得一塌糊涂。