jQuery 基础教程

JavaScript 作为 Web 开发的客户端标准语言，逐渐被人们重视。由于 JavaScript 仅定义了基本的语法规范和逻辑结构，对于浏览器端的各种应用只能够借助 API 应用扩展来实现（如 DOM、XMLHttpRequest 等组件）。但是当掌握 JavaScript 语言的基本用法之后，就会发现 JavaScript 用法比较繁琐，直接使用 DOM 来控制文档，直接调用 XMLHttpRequest 组件来操作数据都显得很麻烦，于是在 JavaScript 语言和扩展组件的基础上诞生了很多 Ajax 技术框架。

Ajax 技术框架是在 Web 2.0 大背景下产生的，这些技术框架封装 JavaScript、DOM 和 XMLHttpRequest 等技术的功能，为人们提供了一种快捷、强大的应用方式，降低了 Web 开发的门槛。借助这些技术框架，用户只需要简单地调用框架对象的方法或属性即可实现各种复杂的功能。如果直接使用 JavaScript、DOM 和 XMLHttpRequest 等技术来开发这些复杂的功能，可能会占用大量的时间，最关键的是这种原始开发方式对于广大初级技术人员来说是望尘莫及的。

随着 Ajax 概念的快速传播，互联网上出现了很多优秀的技术框架，比较著名的有 Prototype、YUI、jQuery、mootools、Bindows 以及国内的 JSVM 框架等，通过将这些 JS 框架应用到项目中，能够使程序员从繁杂的 Web 开发应用中解脱出来，将关注点转向应用项目的实现上，而非底层技术的开发上，从而提高项目的开发速度。

1　jQuery 技术框架概述

jQuery 是继 Prototype 之后又一个优秀的 JavaScript 框架，由 John Resig 于 2006 年初创建，目前最新版本为 1.3.2，官方地址为 :http://jquery.com/，中文地址为 http://jquery.org.cn/。jQuery 具有如下特点。

- 语法简练、语义易懂、学习快速、丰富文档。
- jQuery 是一个轻量级的脚本，其代码非常小巧，最新版的 jQuery 框架文件仅有 30KB 左右。
- jQuery 支持 CSS1~CSS3 定义的属性和选择器，以及基本的 xPath 技术。
- jQuery 是跨浏览器的，它支持的浏览器包括 IE 6.0+、FF 1.5+、Safari 2.0+和 Opera 9.0+。
- 可以很容易地为 jQuery 扩展其他功能。
- 能将 JavaScript 脚本与 HTML 源代码完全分离，便于后期编辑和维护。
- 插件丰富，除了 jQuery 自身带有的一些特效外，可以通过插件实现更多功能，如表单验证、Tab 导航、拖放效果、表格排序、DataGrid、树形菜单、图像特效以及 Ajax 上传等。

jQuery 能够改变你编写 JavaScript 脚本的方式，降低学习和使用 Web 前端开发的复杂度，提高网页开发效率，无论对于 JavaScript 初学者，还是 Web 开发资深专家，jQuery 都应该是必备的工具。jQuery 适合于设计师、开发者以及 Web 编程爱好者，同样适合商业开发。可以说 jQuery 适合任何 JavaScript 应用的地方，也可用于不同的 Web 应用程序中。

在使用 jQuery 之前，你需要下载 jQuery 技术框架文件，并引入到页面中。jQuery 框架文件是一个 js 文件，压缩大小约为 30KB，未压缩文件为 97.8KB，可以说是非常小的。导入 jQuery 框架文件方法如下：

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

引入 jQuery 框架文件之后便可在页面脚本中调用 jQuery 对象、方法或属性，并以 jQuery 特色语法规范来编写脚本。

2 jQuery 构造器

构造器是 jQuery 框架的内核（core），它犹如 JavaScript 语言的构造函数（Function）。构造器由 jQuery()函数（可简写为$()）负责实现，该函数是整个 jQuery 框架的核心，jQuery 中的一切操作都构建于这个函数之上。jQuery()函数可以接收四种类型的参数。

- jQuery(expression,[context]): 根据 CSS 选择器字符串在页面中匹配一组元素，或者利用 context 参数指定匹配的范围。
- jQuery(html): 根据 HTML 标记字符串，动态创建由 jQuery 对象包装的 DOM 元素。
- jQuery(elements): 将一个或多个 DOM 对象转化为 jQuery 对象。
- jQuery(callback): $(document).ready()的简写。允许绑定一个在 DOM 文档加载完毕之后执行的函数。

例如，构建一个简单的文档结构，代码如下所示（请不要忘记在文档顶部导入 jquery.js 文件）：

```
    <script           language="javascript"           type="text/javascript"
src="images/jquery.js"></script>
    <div><span>文本块 1</span></div>
    <p><span>文本块 2</span></p>
```

然后定义“文本块 2”字符串显示为红色，则代码如下（该代码段位于控制结构的底部）：

```
    <script language="javascript" type="text/javascript">
    var red = $("p span");      //选择元素中包含的 span 元素，并返回该元素的 jQuery 对
象引用指针
    red.css("color","red");     //调用 jQuery 对象的 css()函数，定义选取对象的 CSS 样
式，其中第一个参数表示 CSS 属性名，第二个参数表示 CSS 属性值。
    </script>
```

也可以采用如下方式定义：

```
    var red = $("span","p");    //在指定范围内（所有段落文本 p 元素内选择 span 元素
```

```
    red.css("color","red");      //定义该选取对象的样式
```

借助 jQuery(html)类型函数，可以为指定元素增加 HTML 结构。例如，下面代码能够在 body 元素内增加“ 文本块 3 ”HTML 结构。

```
    $("<div><span>文本块 3</span></div>").appendTo("body");
```

jQuery(elements)类型函数能够把一个 DOM 对象转换为 jQuery 对象，这样就可以调用 jQuery 对象的方法和属性。例如，下面示例代码把 span 元素转换为 jQuery 对象，然后调用 css()方法来定义字体大小。

```
    $("span").css("font-size","12px");
```

借助 jQuery(callback)类型函数可以在页面加载完毕时执行包含的函数。例如，下面代码将在页面加载完毕之后弹出一个提示对话框，显示“Hello World!”提示信息。

```
    <script language="javascript" type="text/javascript">
    $(function(){
           alert("Hello World!");
    });
    </script>
```

通过上面示例，我们可以看到：**jQuery 对象与 DOM 对象是两个不同的概念**，它们不能够相互调用。jQuery 对象只能够使用 jQuery 定义的方法和属性；而 DOM 也只能够使用 DOM 组件和 JavaScript 定义的方法和属性。因此，在调用对象的方法和属性时，应该清楚它属于什么对象。

当然，也可以通过一定的方法转换 jQuery 与 DOM 这两种对象。对于普通的 DOM 对象来说，如果要转换为 jQuery 对象，则使用 jQuery()函数即可。例如，在下面代码中，document.getElementsByTagName("span")[0]将获取页面中第一个 span 对象，此时所定义的 span 变量就是一个 DOM 对象，可以调用 DOM 定义的节点方法对其进行操作。

而$(span)则表示一个 jQuery 对象，其中包含的 span 变量就表示 DOM 对象（即 document.getElementsByTagName("span")[0]），这时我们就可以为变量 span 调用 jQuery 对象的方法来定义 span 元素的显示样式（即红色字体）。

```
    <div><span>文本块 1</span></div>
    <p><span>文本块 2</span></p>
    <script language="javascript" type="text/javascript">
    var span = document.getElementsByTagName("span")[0];   //获取节点对象，此时返回 DOM 元素对象
    var span = $(span);                    //把 DOM 对象转换为 jQuery 对象
    span.css("color","red");               //调用 jQuery 对象的 css()方法定义字体颜色为红色
    </script>
```

当然也可以把 jQuery 对象转换为 DOM 对象，由于 jQuery 对象实际上就是一个 JavaScript 数据集合，如果要把 jQuery 对象转换为 DOM 对象，则必须从对象中选取其中的某一项元素，即通过索引选取其中一个元素对象即可。例如，针对上面示例中 jQuery 对象$(span)，我们可以

使用如下写法把它转换为 DOM 对象，再调用 DOM 属性来定义样式：

```
    var span = $(span)[0];                    //把 jQuery 对象转换为 DOM 对象
    span.style.color = "blue";                //调用 DOM 对象的 style 属性，设置字体颜
色为蓝色
```

除了使用集合索引值把 jQuery 对象转换为 DOM 对象外 ,还可以使用 jQuery 的 get()获取对象内指定索引的元素：

```
    $("span").get(0);
```

3 jQuery 基本用法

要发挥 jQuery 构造器的强大功能 ,应先熟悉 jQuery 定义的各种方法和属性 ,特别是 jQuery 对象的基本属性和方法。下面将讲解 jQuery 定义的各种基本属性和方法 ,为具体应用奠定基础。

3.1 访问 jQuery 对象

2 节已经介绍过 jQuery 对象是一个集合，要访问这个集合，除了使用索引值以外，还可以使用 jQuery 定义的几个方法和属性。

- each(callback)。

each(callback)方法实际上是 JavaScript 集合遍历的一种功能包装，它以 jQuery 对象内的集合元素为遍历对象，并循环执行指定的函数。在循环体内的函数中，this 关键字都会自动指向当前元素，且会自动向函数体内传递元素的索引值（从 0 开始）。

例如，下面示例将首先获取所有 span 元素，再调用 jQuery 对象的 each()方法，该方法包含了一个简单的函数，定义每个元素的字体大小样式。其中函数体内的 this 表示循环时每个当前元素，此时它应该是一个 DOM 对象，所以可以调用 DOM 的属性来定义字体样式，而参数 n 表示 each()方法自动循环时传递的索引值。

```
    <div><span>文本块 1</span></div>
    <p><span>文本块 2</span></p>
    <script language="javascript" type="text/javascript">
    var span = $("span");                     //获取文档中的 span 元素
    span.each(function(n){                    //遍历 span 元素集合
          this.style.fontSize = (n+1)*12+"px";    //为每个 span 元素定义字体大小
样式
    });
    </script>
```

演示结果：第一个 span 元素包含字体大小为 12 像素，第二个 span 元素包含字体大小为 24 像素。如果想在函数体内调用 jQuery 对象的方法，则需要把 this 转换为 jQuery 对象。例如，修改上面示例为下面形式：

```
    var span = $("span");                          //获取文档中的 span 元素
    span.each(function(n){                                  //遍历 span 元素集合
          $(this).css("font-size",(n+1)*12+"px");      //为每个 span 元素定义 CSS
样式，设置字体加倍递增
    });
```

如果函数中途返回 false，则将停止循环。如果每次执行函数都返回 true，则将自动循环执行函数，直到遍历结束。

- size()和 length。

size()方法能够返回 jQuery 对象中元素的个数，而 length 属性与 size()方法功能相同。例如，下面代码使用 size()方法和 length 属性返回值都为 2。

```
    <span>文本块 1</span><span>文本块 2</span>
    <script language="javascript" type="text/javascript">
    alert($("span").size());                                  //返回值为 2
    alert($("span").length);                                  //返回值为 2
    </script>
```

- get()和 get(index)。

get()方法能够把 jQuery 对象转换为 DOM 中的元素集合。例如，在下面示例中，使用$()函数获取所有 span 元素，然后使用 get()方法把该 jQuery 对象转换为 DOM 集合，再调用 JavaScript 数组方法 reverse()把数组元素的位置颠倒过来。最后为数组中第一个元素定义样式为红色字体，演示效果为“文本块 2”中文本显示为红色。

```
    <span>文本块 1</span><span>文本块 2</span>
    <script language="javascript" type="text/javascript">
    var spans = $("span").get().reverse();          //把当前 jQuery 对象转换为 DOM
对象，并颠倒它们的位置
    spans[0].style.color = "red";                   //把当前 jQuery 对象设置为红色
    </script>
```

get(index)方法与 get()功能相同，但是它能够获取指定索引值的元素对象，请注意它返回的是 DOM 对象。

- index(subject)。

获取 jQuery 对象中指定元素的索引值。如果找到了匹配的元素，从 0 开始返回；如果没有找到匹配的元素，返回 - 1。例如，在下面这个示例中，获取 body 元素下包含的所有元素，同时使用 DOM 方法获取其中的 div 元素的引用指针，然后利用 index()方法获取 div 元素在 body 中的索引值，最后返回为 2，即位于 body 元素内的第三个。

```
    <body>
    <span></span><p></p><div></div><h1></h1>
    <script language="javascript" type="text/javascript">
    var a = $("body *");                                      //获取 body 元素包含
的所有子元素
    var e = document.getElementsByTagName("div")[0];          //获取 div 元素在文档
```

```
中的索引值
    alert(a.index(e));                                    //显示索引值
    </script>
    </body>
```

3.2 访问 DOM 对象的属性

使用 jQuery 函数能够很轻松地操控页面内元素，同时也可以方便地控制元素的属性。这主要依赖下面几个方法。

- attr(name)：根据属性名（name 参数）获取 jQuery 对象中第一个元素的对应属性值。例如，将返回超链接的 href 属性值。

```
    <a href="images/1.html" accesskey="k" id="img" target="_blank">超链接</a>
    <script language="javascript" type="text/javascript">
    var href = $("a").attr("href");
    alert(href);
    </script>
```

- attr(key,value)：为 jQuery 对象定义属性并赋值。例如，下面示例将为图像动态增加一个 width 属性，并设置属性值为 100 像素。注意，由于 HTML 标签属性中不能够设置像素单位，否则无效，这与 CSS 中必须设置单位不同。

```
    <img src="images/1.jpg" />
    <script language="javascript" type="text/javascript">
    $("img").attr("width","100");
    </script>
```

- attr(key,fn)：我们还可以为传递的属性值设置为一个函数，通过函数计算所得的值来为属性赋值。例如，在下面示例中把匿名函数作为属性值赋予给 title 属性，该函数返回值为当前对象的 src 属性值。当把光标移到图像上时会显示该图像的 URL 提示信息。

```
    <img src="images/1.jpg" />
    <script language="javascript" type="text/javascript">
    $("img").attr("title",function(){
           return this.src;
    });
    </script>
```

- attr(properties)：为 jQuery 对象同时定义多个属性。多个属性以名/值对的形式组成对象进行参数传递。例如，下面示例同时为图像增加了宽、高、提示文本和替换图像属性，并进行赋值。对象内的这些属性都是以名/值对的形式组成一个对象成员集合。

```
    <img src="images/1.jpg" />
    <script language="javascript" type="text/javascript">
    $("img").attr({width:"100",height:"100",title:"这是一个示例",alt:"示例图像
"});
    </script>
```

- removeAttr(name)：该方法能够移出 jQuery 对象内指定属性。例如，在下面示例中，

removeAttr()方法将删除图像中的 width 属性及其属性值。

```
<img src="images/1.jpg" width="100" height="200" />
<script language="javascript" type="text/javascript">
$("img").removeAttr("width");
</script>
```

3.3 访问 DOM 样式类

样式类实际上是一种特殊的属性（class）。不过 jQuery 为此定义了三个方法专门用来控制样式类，其中包括增加类、删除类和开关类，具体说明如下。

如果要为元素增加样式类，可以使用 addClass(class)方法。例如，下面脚本将为页面内所有的段落文本增加一个红色字体样式类。

```
<p>段落文本 1</p><p>段落文本 2</p>
<script language="javascript" type="text/javascript">
$("p").addClass("red");
</script>
```

当然，也可以使用 attr()方法定义样式类，此时是把它看作一个普通的属性进行增加，代码如下：

```
$("p").attr("class","red");
```

如果要删除样式类，则可以使用 removeClass(class)方法，具体用法就不再举例，同样也可以使用 removeAttr(name)方法来删除样式类。

另外，jQuery 还自定义了一个 toggleClass(class)方法，它如同一个开关，如果元素已经定义了指定样式类，则会删除该样式类，否则增加该样式类。巧用这个方法，可以很轻松地设计鼠标经过或单击时动态改变元素样式的效果。

例如，在下面这个示例中，建立了一个简单的列表结构，并定义了两个样式类，bg 和 bg1。在脚本中，首先使用 jQuery 构造器函数获取所有列表项元素，然后调用 each()方法遍历对象内所有元素，并在 each()方法内为每个元素注册三个事件，分别设计当鼠标经过时、移开时以及单击时开/关样式类，演示效果如图 1 和图 2 所示。

```
<style type="text/css">
.bg { background:#FF99FF; }                    /* 样式类 1 */
.bg1 { background:#00CCFF; }                   /* 样式类 2 */
</style>
<ul>
        <li>列表项 1</li>
    <li>列表项 2</li>
        <li>列表项 3</li>
</ul>
<script language="javascript" type="text/javascript">
$("li").each(function(){                       //遍历 jQuery 对象内所有对象
        this.onmouseover = function(){         //为当前元素注册鼠标经过时的事件
```

```
            $(this).toggleClass("bg");          //开关背景样式类（bg）
        }
        this.onmouseout = function(){           //为当前元素注册鼠标移开时的事件
            $(this).toggleClass("bg");          //开关背景样式类（bg）
        }
        this.onclick = function(){              //为当前元素注册鼠标单击时的事件
            $(this).toggleClass("bg1");         //开关背景样式类（bg1）
        }
});
</script>
```

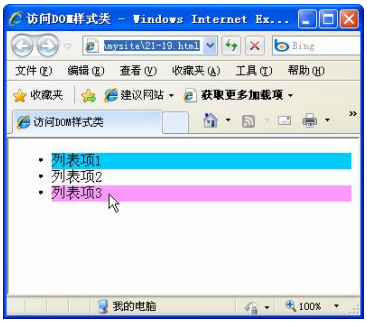

图 1　IE 8 下预览效果

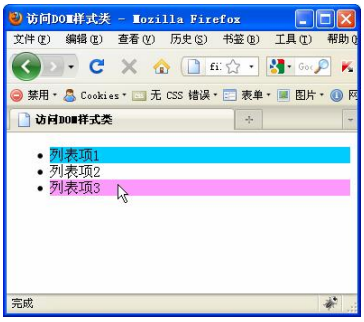

图 2　FF 3.5 下预览效果

3.4　访问 DOM 元素包含信息

访问元素包含的信息可以使用 text()方法获取。例如，在下面示例中获取段落标签中包含的所有内容，即“段落文本 1“。如果 p 包含了其他元素，则省略不计，因此其中包含的 span 元素将被忽略。

```
<p>段落文本<span>1</span></p>
<script language="javascript" type="text/javascript">
alert($("p").text());
</script>
```

反过来，也可以为 text()传递文本字符串，则将自动为元素添加指定文本字符串，同时会自动删除该元素包含的已有文本。例如，下面代码将会自动覆盖段落文本中原有信息，而增加“”字符串，该字符串虽然是 HTML 源代码，但是在页面中仅显示字符串信息，而不是图像。

```
<p>段落文本<span>1</span></p>
<script language="javascript" type="text/javascript">
$("p").text("<img src='images/1.jpg' width='100' />");
</script
```

请注意，text()和 text(val)方法能够读写 jQuery 对象所有匹配元素的内容，结果是由所有匹配元素包含的文本内容组合起来的文本，这个方法对 HTML 和 XML 文档都有效。

如果希望获取段落中包含的所有内容（包括标签结构），则可以使用 html()方法。例如，针对上面示例如果使用下面代码，则将弹出“段落文本1”提示文本信息，而不是“段落文本 1“。

```
alert($("p").html());
```

同理，如果希望在段落中插入图像，而不是“”字符串，则可以使用如下方法，这样所插入的字符串就不被编码，而是直接插入到文档结构中。

```
<p>段落文本<span>1</span></p>
<script language="javascript" type="text/javascript">
$("p").html("<img src='images/1.jpg' width='100' />");
</script>
```

请注意，html()和 html(val)方法只作用于 jQuery 对象中第一个元素，也就是说只能够读写第一个元素内的 HTML 源代码，且不能用于 XML 文档，仅适用于 XHTML 文档。

text()和 html()方法能够读写元素包含的信息和 HTML 源代码，但是如何读写表单域中的值却存在一定的问题。例如，下面是一个下拉列表框。

```
<select>
    <option value="1">选项 1</option>
    <option value="2" selected="selected">选项 2</option>
    <option value="3">选项 3</option>
</select>
```

如果使用 text()方法读取 select 下拉列表框的值，则弹出提示信息为“选项 1 选项 2 选项 3”。

```
alert($("select").text());
```

如果使用 html()方法读取 select 下拉列表框的值，则弹出提示信息为“<option value="1">选项 1</option> <option value="2" selected="selected">选项 2</option> <option value="3">选项 3</option>”长字符串。

```
alert($("select").html());
```

那么该如何读取下拉菜单的当前值呢？为此 jQuery 定义了 val()方法，该方法专门用来读写表单值。例如，针对上面的下拉列表框表单域结构，可以使用如下方法读取下拉列表框当前选择值（提示为 2）。

```
alert($("select").val());
```

再如，下面是多个不同类型的输入域：

```
<input type="text" value="文本框" />
<input type="radio" value="单选按钮" />
<input type="checkbox" value="复选框" />
<input type="hidden" value="隐藏域" />
<input type="submit" value="提交按钮"/>
```

要获取第一个输入域（文本框）的值，则可以使用如下语句，返回提示信息为“文本框”。

```
alert($("input").val());
```

如果要获取第二个以及后面的输入域值，就不能使用上面语句了。因为 val()方法仅能够获

取第一个匹配元素的当前值。在 jQuery 1.2 版本中曾经支持该方法可以返回所有匹配表单域的值，包括 select。现在（1.26 版本）如果要获取它们的值，可以使用如下方法：

```
alert($($("input")[1]).val());                    //显示单选按钮的值
alert($($("input")[2]).val());                    //显示复选框的值
alert($($("input")[3]).val());                    //显示隐藏域的值
alert($($("input")[4]).val());                    //显示提交按钮的值
```

先使用$("input")[1]获取对应表单域的 DOM 对象指针，然后再使用 jQuery 构造器转换为 jQuery 对象，再调用 val()即可。

对于多选列表框，则可以获取多个值。例如，在下面这个示例中，列表框是一个多选框，则 val()方法返回值为“2, 3”。

```
<select multiple="multiple">
    <option value="1">选项 1</option>
    <option value="2" selected="selected">选项 2</option>
    <option value="3" selected="selected">选项 3</option>
</select>
<script language="javascript" type="text/javascript">
alert($("select").val());
</script>
```

如果在 val()附带参数，则可以为每一个匹配的元素设置值。在 jQuery 1.2 中也可以为 select 元素赋值。例如，在下面示例中，脚本将自动为文本框赋值为“文本框的值”。

```
<input type="text" />
<script language="javascript" type="text/javascript">
$("input").val("文本框的值");
</script>
```

当然也可以为单选按钮、复选框和列表框设置选定值。方法是在 val()中以数组形式传递多项参数值。例如，在下面的表单域中，选中第二个单选按钮和第二个复选框，以及选中下拉列表框中第一项和第三项，可以使用如下脚本来实现，显示效果如图 3 所示。

```
<input type="radio" value="radio1" />单选按钮 1
<input type="radio" value="radio2" />单选按钮 2
<input type="checkbox" value="check1" />复选框 1
<input type="checkbox" value="check2" />复选框 2
<select multiple="multiple">
    <option value="1">选项 1</option>
    <option value="2">选项 2</option>
    <option value="3">选项 3</option>
</select>
<script language="javascript" type="text/javascript">
$("input").val(["radio2","check2"]);
$("select").val(["1","3"]);
</script>
```

图 3　使用 jQuery 选中表单域

对于下拉菜单，也可以不用数组来传递值，但对于单选按钮和复选框必须使用数组来传递参数值。例如，下面语句都是有效的。

```
$("select").val(["3"]);
$("select").val("3");
$("input").val(["radio2"]);
```

4　jQuery 选择器

在学习 CSS 时，曾经接触过各种类型的样式选择器。例如，使用 id 选择器选定页面中特定元素并为其定义样式；使用类选择器为页面相同效果的对象定义公共样式；使用标签选择器可以为某一类型的元素定义样式。在 JavaScript 中虽然没有选择器这个概念，但是可以使用 getElementByIId()和 getElememtsByTagName()方法选择特定对象或一组对象，以便进行控制。jQuery 提供了强大、易用的选择工具，用来选取网页对象或元素，这些选择工具主要包括两种方式。

方式一，模拟 CSS 和 Xpath 选择器，并混合它们的用法。这种方式可以把选择器字符串传递给 jQuery 构造器，从而达到选择复杂元素的目的。

方式二，使用 jQuery 对象定义的函数进行筛选。

这两种方式可以混合使用。

4.1　常用选择器

前面示例中已经使用过各种 jQuery 选择器来选择页面中的元素。实际上，如果没有提供强大的选择器，jQuery 所定义的各种方法和属性也就无用武之地。在第 6 章中我们曾经介绍过很多类型的 CSS 选择器，而 jQiery 选择器严格遵循 CSS1 至 CSS3 选择器的规范和用法，所以精通 CSS 语法将更有利于学习 jQuery。jQuery 常用选择器说明如表 1 所示。

表 1　jQuery 常用选择器及其说明

jQuery 常用选择器	说 明
#id	根据 ID 值匹配特定元素，与 CSS 中的 ID 选择器对应
element	根据给定的元素名匹配所有元素，与 CSS 中的标签选择器对应
.class	根据给定的类匹配元素，与 CSS 中的类选择器对应
*	匹配所有元素，与 CSS 中通配符类似，但更灵活，可以匹配局部所有元素
selector1,selector2,selectorN	将每一个选择器匹配元素合并后一起返回，与 CSS 中的选择器分组对应
ancestor descendant	在指定祖先元素下匹配所有的后代元素，与 CSS 中的包含选择器对应
parent > child	在给定的父元素下匹配所有的子元素，与 CSS 中的子选择器对应
prev + next	匹配所有紧接在 prev 元素后的 next 元素，与 CSS 中的相邻选择器对应
prev ~ siblings	匹配 prev 元素之后的所有 siblings 元素，与 CSS 没有对应选择器

（续表）

jQuery 常用选择器	说 明
[attribute]	匹配包含给定属性的元素，与 CSS 中的属性选择器对应
[attribute=value]	匹配给定属性等于特定值的元素，与 CSS 中的属性选择器对应
[attribute!=value]	匹配给定的属性不包含某个特定值的元素，与 CSS 中的属性选择器对应
[attribute^=value]	匹配给定的属性是以某些值开始的元素，与 CSS 中的属性选择器对应
[attribute$=value]	匹配给定的属性是以某些值结尾的元素，与 CSS 中的属性选择器对应
[attribute*=value]	匹配给定的属性是以包含某些值的元素，与 CSS 中的属性选择器对应
[selector1][selector2][selectorN]	复合属性选择器，需要同时满足多个条件时使用，与 CSS 中的属性选择器对应

例如，在下面这个简单结构中。

```
<div id="box1">
    <p id="p1">段落文本 1</p>
</div>
<div id="box2">
    <p id="p2">段落文本 2</p>
</div>
<p id="p3">段落文本 3</p>
```

要选择“段落文本 1”包含的对象，则可以使用如下语句之一：

```
$("#p1");                              //ID 选择器
$("#box1 p");                          //包含选择器
$("#box1>#p1");                        //子选择器
$("p#p1");                             //指定标签选择器
$("p[id='p1']");                       //匹配属性等于特定属性值
$("[id$='1']");                        //匹配属性值结尾的值
$("#box1 [id^='p']");                  //包含选择器，配合匹配属性值开始的值
$("[id*='1']");                        //匹配属性值包含某个字符串
```

如果要选择“段落文本 2”包含的对象，则可以使用如下语句之一，当然也可以使用属性选择

器。

```
$("#p3");                                    //ID 选择器
$("#box2 + p");                              //相邻选择器
$("#box1~p");                                //后续选择器
```

使用各种选择器所返回的对象为 jQuery 对象（即集合对象），即使返回的元素仅有一个也属于集合，因此不能直接调用 DOM 定义的方法。

4.2 伪选择器

在 CSS 中曾经讲解过伪类和伪对象选择器，这些选择器不是选择页面中可视化的元素，而是选择元素或页面中的某种状态或特性，以实现动态捕获特定元素，因此在页面中无法找到所要选择的对象。jQuery 支持所有 CSS 伪类和伪对象，同时还扩展了丰富的伪选择器，详细说明如表 2 所示。

表 2　jQuery 伪选择器及其说明

jQuery 伪选择器	说 明
:first	匹配找到的第一个元素
:last	匹配找到的最后一个元素
:not(selector)	去除所有与给定选择器匹配的元素
:even	匹配所有索引值为偶数的元素，从 0 开始计数
:odd	匹配所有索引值为奇数的元素，从 0 开始计数
:eq(index)	匹配一个给定索引值的元素
:gt(index)	匹配所有大于给定索引值的元素
:lt(index)	匹配所有小于给定索引值的元素
:header	匹配如 h1、h2、h3 之类的标题元素
:animated	匹配所有没有在执行动画效果中的元素
:first-child	匹配第一个子元素
:last-child	匹配最后一个子元素
:only-child	如果某个元素是父元素中唯一的子元素，那将会被匹配
:nth-child(index/even/odd/equation)	匹配父元素下的第 N 个子或奇偶元素
:contains(text)	匹配包含给定文本的元素
:empty	匹配所有不包含子元素或者文本的空元素
:has(selector)	匹配含有选择器所匹配的元素的元素
:parent	匹配含有子元素或者文本的元素
:hidden	匹配所有不可见元素，input 元素的 type 属性为“hidden”时也可匹配
:visible	匹配所有的可见元素

例如，设计一个简单的列表结构。

```
<ul id="list">
    <li>列表项 1</li>
    <li>列表项 2</li>
    <li class="l3">列表项 3</li>
    <li></li>
</ul>
```

如果选择第一个列表项元素，则可以使用：$("li:first");，或者$("li:first-child");，或者$("ul :first-child")。但是:first-child 能够匹配第一个子元素，并为每个父元素匹配一个子元素。而:first 只匹配一个元素。

如果选择最后一个列表项元素，则可以使用$("li:last");，或者$("li:last-child");，或者$("ul :last-child")。

如果选择列表项中索引值为偶数的元素，则可以使用$("li:even");。

如果选择列表项中索引值为奇数的元素，则可以使用$("li:odd");。

如果选择列表结构中第三个列表项目，则可以使用$("li:eq(2)");。

如果选择列表结构中第二个以及后面所有项目，则可以使用$("li:gt(0)");。

如果选择列表结构中第三个以及前面所有项目，则可以使用$("li:lt(3)");。

如果选择所有列表项，但是不包含 class="l3"的元素，则可以使用$("li:not(.l3)");。

如果选择列表项中包含“列表项 1”文本内容的元素，则可以使用$("li:contains('列表项 1')");。

如果选择列表项中不包含任何内容的元素，则可以使用$("li:empty");。

如果选择 ul 列表结构中包含.l3 类选择器能够匹配的子元素时，则可以使用$("ul:has(.l3)");。

如果选择 li 元素中包含子元素或者文本内容的项，则可以使用$("li:parent");。

再看一个示例，下面是三个 div 元素。

```
<div>盒子 1</div>
<div style="display:none">盒子 2</div>
<div><p>盒子 3</p></div>
```

如果把隐藏元素显示出来，则可以使用：

```
$("div:hidden").css("display","block");
```

如果把所有显示的元素隐藏起来，则可以使用：

```
$("div:visible").css("display","none");
```

4.3 表单专用选择器

由于表单对象比较特殊，很多表单域都共用同一个元素 input，这为快速选择特定表单域带来困难。为此 jQuery 定义了一组表单专用选择器，详细说明如表 3 所示。

表 3　jQuery 表单选择器及其说明

jQuery 表单选择器	说 明
:input	匹配所有 input、textarea、select 和 button 表单元素

:text	匹配所有的单行文本框
:password	匹配所有密码框
:radio	匹配所有单选按钮
:checkbox	匹配所有复选框
:submit	匹配所有提交按钮
:image	匹配所有图像域
:reset	匹配所有重置按钮
:button	匹配所有按钮
:file	匹配所有文件域
:hidden	匹配所有不可见元素，或者 type 为 hidden 的元素
:enabled	匹配所有可用元素
:disabled	匹配所有不可用元素
:checked	匹配所有选中的复选框元素
:selected	匹配所有选中的选项元素

例如，在下面这个表单结构中，分别使用这些表单选择器进行操作。

```
<form>
    <input type="text" value="文本框" />
    <input type="checkbox" value="复选框" />复选框
    <input type="radio" value="单选按钮" />单选按钮
    <input type="image" src="images/menu.gif" value="图像按钮" />
    <input type="file" value="文件域" />
    <input type="submit" value="提交按钮" />
    <input type="reset" value="重置按钮" />
    <input type="password" value="密码域" />
    <input type="button" value="普通按钮" />
    <select>
        <option value="下拉菜单">下拉菜单</option>
    </select>
    <textarea>文本域</textarea>
    <button>按钮</button>
</form>
```

如果获取所有表单域，并设置它们的边框为红色实线，则设置如下：

```
$(":input").css("border","solid 1px red");
```

根据表 3 所列选择符，我们可以为不同的输入域定义边框样式，代码如下：

```
<script language="javascript" type="text/javascript">
$("input:text").css("border","solid 1px red");
$("input:checkbox").css("border","solid 1px red");
$("input:radio").css("border","solid 1px red");
$(":image").css("border","solid 1px red");
```

```
$(":file").css("border","solid 1px red");
$(":submit").css("border","solid 1px red");
$(":reset").css("border","solid 1px red");
$(":password").css("border","solid 1px red");
$(":button").css("border","solid 1px red");
</script>
```

4.4 筛选函数

在 4.2 节中介绍了很多功能实用的伪选择器。jQuery 在此基础上还优化并扩展了很多筛选函数，这些函数作为 jQUery 对象的方法直接使用，这样就能够在选择器的基础上更加精确地控制对象。请注意，筛选函数与选择器在用法上是不同的。例如，在下面这个列表结构中：

```
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
</ul>
```

如果要设置第二个列表项的字体颜色为红色，可以使用伪选择器来进行选择：

```
$("li:eq(2)").css("color","red");
```

而如果使用筛选函数，则可以使用如下方法来实现：

```
$("li").eq(1).css("color","red");
```

选择器是构建 jQuery 对象的基础，而筛选函数则是 jQuery 对象的成员，用法截然不同。jQuery 定义的筛选函数如表 4 所示。

表 4　jQuery 筛选函数及其说明

jQuery 筛选函数	说 明
eq(index)	获取指定索引值位置上的元素，索引值从 0 开始算起
hasClass(class)	检查当前的元素是否含有某个特定的类，如果有，则返回 true
filter(expr)	筛选出与指定表达式匹配的元素集合。这个方法用于缩小匹配的范围，用逗号分隔多个表达式
filter(fn)	筛选出与指定函数返回值匹配的元素集合
is(expr)	用一个表达式来检查当前选择的元素集合，如果其中至少有一个元素符合这个给定的表达式就返回 true
map(callback)	将一组元素转换成其他数组（不论是否是元素数组）
not(expr)	删除与指定表达式匹配的元素
slice(start,[end])	选取一个匹配的子集，与原来的 slice 方法类似
add(expr)	把与表达式匹配的元素添加到 jQuery 对象中。这个函数可以用于连接分别与两个表达式匹配的元素结果集
children([expr])	取得一个包含匹配的元素集合中每一个元素的所有子元素的元素集合
contents()	查找匹配元素内部所有的子节点（包括文本节点）。如果元素是一个 iframe，则查找文档内容

find(expr)	搜索所有与指定表达式匹配的元素。这个函数是找出正在处理的元素的后代元素
next([expr])	取得一个包含匹配的元素集合中每一个元素紧邻的后面同辈元素的元素集合
nextAll([expr])	查找当前元素之后的所有元素
parent([expr])	取得一个包含着所有匹配元素的唯一父元素的元素集合
parents([expr])	取得一个包含着所有匹配元素的祖先元素的元素集合（不包含根元素）
prev([expr])	取得一个包含匹配的元素集合中每一个元素紧邻的前一个同辈元素的元素集合
prevAll([expr])	查找当前元素之前所有的同辈元素，可以用表达式过滤
siblings([expr])	取得一个包含匹配的元素集合中每一个元素的所有唯一同辈元素的元素集合。可以用可选的表达式进行筛选
andSelf()	加入先前所选的当前元素中，对于筛选或查找后的元素，要加入先前所选元素时将会很有用
end()	回到最近的一个“破坏性”操作之前，即将匹配的元素列表变为前一次的状态

jQuery 定义了很多能够从选取对象中过滤部分元素的方法，这些方法是对选择器功能的补充。例如，设计一个简单的结构：

```
<div></div>
<div id="box1"></div>
<div class="red"></div>
```

然后使用 jQuery 构造函数选取三个 div 元素。

```
var divs = $("div");
```

设置第一个 div 元素字体显示为红色。

```
divs.eq(0).css("color","red");
```

使用 filter()方法筛选出 class 等于 red 的元素，以及 id 属性值为 box1 的元素，并定义这些元素的字体颜色为红色。

```
divs.filter(".red,#box1").css("color","red");
```

定义前两个 div 元素的字体显示为红色，slice()方法的第一个参数表示起始位置，第二个参数表示结束位置。

```
divs.slice(0,2).css("color","red");
```

5 文档处理

我们曾经讲解过如何使用 DOM 为元素节点增加子元素或文本节点。DOM 提供的方法比较烦琐，需要先选中对象，再定义子节点，最后才能够使用 appendChild()方法实现插入子元素或文本。jQuery 提供的文档处理方法要比 DOM 简单得多，且功能更为强大和灵活。

5.1 插入内容

jQuery 把插入分为内部插入和外部插入两种操作。所谓内部插入，就是把内容直接插入到

指定的元素内部。

- append()。

append()方法与 DOM 的 appendChild()方法功能类似，都是在元素内部增加子元素或文本。例如，在下面示例中，append()方法将把 HTML 源代码插入到 div 元素内部，但是插入内部不会覆盖该元素已经存在的内容，且它会自动位于原内容的末尾。

```
<div>盒子</div>
<script language="javascript" type="text/javascript">
$("div").append("<span style='color:red'>套子</span>");
</script>
```

与 DOM 中 appendChild()方法不同的是：jQuery 中的 append()方法能够同时为 jQuery 对象中多个元素增加内容，所增加的内容不需要先定义成节点，可以直接把它作为字符串插入到元素内。

另外，jQuery 还定义了一个反操作的方法——appendTo(content)，它可以把所有匹配的元素追加到另一个指定的元素集合中。例如，在下面示例中，appendTo()方法能够把 p 插入到 div 中。

```
<p>段落</p>
<div id="box">盒子</div>
<script language="javascript" type="text/javascript">
$("p").appendTo("#box");
</script>
```

- prepend()。

prepend()方法与 append()方法作用相同，都是把指定内容插入到 jQuery 对象元素中，但是 prepend()方法能够把插入的内容放置在最前面，而不是放置在最末尾。例如，在下面示例中，prepend()能够把 id 为 box 的 div 元素插入到 p 元素内部，且位于段落文本的最前面，即 盒子 段落 。

```
<p>段落</p>
<div id="box">盒子</div>
<script language="javascript" type="text/javascript">
$("p").prepend($("#box")[0]);
</script>
```

如果使用 append()方法，则 div 元素会位于段落文本的后面，即 段落 盒子 。与 appendTo()方法相对应的是 prependTo()，该方法能够把所有匹配的元素前置到另一个指定的元素集合中。例如，在下面这个示例中，将把段落 p 插入到 div 元素内部，且位于最前面，即 段落 盒子 。

```
<p>段落</p>
<div id="box">盒子</div>
<script language="javascript" type="text/javascript">
$("p").prependTo($("#box")[0]);
</script>
```

append()、appendTo()、prepend()和 prependTo()是相互联系，且操作紧密的四个方法，它们都能够实现内部插入，只是操作的方向和位置略有不同。

所谓外部插入 ,就是把内容插入到指定 jQUery 对象相邻元素内。与内部插入操作基本类似，外部插入也包含四种方法。

- after(content): 在每个匹配的元素之后插入内容。
- before(content): 在每个匹配的元素之前插入内容。
- insertAfter(content): 把所有匹配的元素插入到另一个指定的元素或元素集合的后面。
- insertBefore(content): 把所有匹配的元素插入到另一个指定的元素或元素集合的前面。

例如 ,在下面示例中使用 after()方法把 div 元素插入到 p 元素的后面 ,相当于颠倒两个元素的排列顺序。

```
<p>段落</p>
<div id="box">盒子</div>
<script language="javascript" type="text/javascript">
$("div").after($("p"));
</script>
```

也可以使用 before()方法实现相同的颠倒设置：

```
$("p").before($("div"));
```

或者

```
$("p").insertAfter($("div"));
```

或者

```
$("div").insertBefore($("p"));
```

这四种方法可以实现相同的功能 ,但是它们的作用点却各有侧重。请注意 ,除了使用 jQuery 对象作为插入内容的参数外，还可以插入 DOM 元素或元素集合，以及 HTML 结构字符串。

5.2 嵌套结构

嵌套与插入操作有几分相似，虽然它们都可以实现相同的操作目标，但是两者在概念上还是存在一些区别。嵌套重在结构的构建，而插入侧重内容的显示。jQuery 定义了如下六个嵌套结构的方法。

- wrap(html): 把所有匹配的元素分别用指定结构化标签包裹起来。
- wrap(element): 把所有匹配的元素分别用指定元素包裹起来。
- wrapAll(html): 把所有匹配的元素用一个结构化标签包裹起来。
- wrapAll(element): 把所有匹配的元素用一个元素包裹起来。
- wrapInner(html): 把每一个匹配的元素的子内容（包括文本节点）使用一个 HTML 结构包裹起来。
- wrapInner(element): 把每一个匹配的元素的子内容（包括文本节点）使用元素包裹起来。

例如，对于如下三个超链接文本：

```
<a href="#">超链接 1</a><a href="#">超链接 2</a><a href="#">超链接 3</a>
```

如果希望为每个超链接包裹一个 标签，则可以使用如下方法：

```
$("a").wrap("<div></div>");
```

则最终显示效果的代码结构如下：

```
<div><a href="#">超链接 1</a></div>
<div><a href="#">超链接 2</a></div>
<div><a href="#">超链接 3</a></div>
```

如果希望利用页面中现有的元素来包含超链接。例如，如果希望使用超链接底部的列表项元素来包裹超链接文本。

```
<a href="#">超链接 1</a><a href="#">超链接 2</a><a href="#">超链接 3</a>
<ul>
        <li> </li>
</ul>
```

可以使用如下代码：

```
$("a").wrap(document.getElementsByTagName("li")[0]);
```

则所得的结果如下所示：

```
<li><a href="#">超链接 1</a></li>
<li><a href="#">超链接 2</a></li>
<li><a href="#">超链接 3</a></li>
<ul>
        <li> </li>
</ul>
```

如果再为所有列表项外面包一层列表结构 ul 元素，则可以使用如下代码：

```
$("a").wrap(document.getElementsByTagName("li")[0]);
$("li").wrapAll(document.getElementsByTagName("ul")[0]);
```

所得结果如下：

```
<ul>
    <li><a href="#">超链接 1</a></li>
    <li><a href="#">超链接 2</a></li>
    <li><a href="#">超链接 3</a></li>
</ul>
<ul>
        <li> </li>
</ul>
```

如果希望为每个列表项内嵌入一个 span 元素，则可以使用如下代码：

```
$("a").wrap(document.getElementsByTagName("li")[0]);
$("li").wrapAll(document.getElementsByTagName("ul")[0]);
$("li").wrapInner("<span></span>");
```

所得结果如下：

```
<ul>
    <li><span><a href="#">超链接 1</a></span></li>
    <li><span><a href="#">超链接 2</a></span></li>
    <li><span><a href="#">超链接 3</a></span></li>
</ul>
```

```
<ul>
        <li> </li>
</ul>
```

5.3 替换结构

jQuery 提供了 replaceWith(content)和 replaceAll(selector)方法来实现 HTML 结构替换。replaceWith()能够将所有匹配的元素替换成指定的 HTML 或 DOM 元素。例如，对于下面三个 span 元素，使用 replaceWith()把匹配的所有 span 元素及其包含的文本都替换为“ 盒子 ”。

```
<span>包子</span><span>包子</span><span>包子</span>
<script language="javascript" type="text/javascript">
$("span").replaceWith("<div>盒子</div>");
</script>
```

最后，所得到的效果如下：

```
<div>盒子</div><div>盒子</div><div>盒子</div>
```

replaceAll(selector)方法与 replaceWith(content)方法操作正好相反。例如，要实现上面的替换效果，我们可以使用如下代码：

```
$("<div>盒子</div>").replaceAll("span");
```

5.4 删除和克隆结构

删除结构也有两种方法：一是使用 empty()删除匹配元素包含的所有子节点。例如，在下面示例中将删除 div 元素内所有子节点和文本，返回“ ”两个空标签。

```
<div>盒子</div>
<div><p>盒子</p></div>
<script language="javascript" type="text/javascript">
$("div").empty();
</script>
```

二是使用 remove([expr])方法删除匹配的元素，或者符合表达式的匹配元素。例如，在下面示例中将删除 div 元素及其包含的子节点，最后返回的是“ 段落文本 3 ”。

```
<div>盒子 1</div>
<div><p>段落文本 2</p></div>
<p>段落文本 3</p>
<script language="javascript" type="text/javascript">
$("div").remove();
</script>
```

结构复制主要使用 clone()和 clone(true)方法。clone()表示克隆匹配的 DOM 元素并选中克隆的元素。例如，在下面示例中，先使用 clone()方法克隆 div 元素，然后再把它插入到 p 元素内。

```
<div onclick="alert('Hello World')">盒子</div>
```

```
<p>段落</p>
<script language="javascript" type="text/javascript">
$("div").clone().appendTo("p");
</script>
```

最后插入结果为：

```
<div>盒子</div>
<p>段落<div>盒子</div></p>
```

clone(true)方法不仅能够克隆元素，而且还可以克隆元素所定义的事件。例如，在上面示例中如果为 div 元素定义一个 onclick 属性事件，则使用 clone(true)方法将会在克隆元素中也包含属性事件。

```
<div onclick="alert('Hello World')">盒子</div>
<p>段落</p>
<script language="javascript" type="text/javascript">
$("div").clone(true).appendTo("p");
</script>
```

克隆的结果为：

```
<div onclick="alert('Hello World')">盒子</div>
<p>段落<div onclick="alert('Hello World')">盒子</div></p>
```

6 CSS 处理

CSS 与 JavaScript 紧密相联系，在 DHTML 开发中两者缺一不可。jQuery 对于 CSS 的支持是超前的，从 CSS1.0 到 CSS3.0 版本，都能够完全地进行支持，这样当我们为页面设计 CSS 时就不用考虑不同浏览器的支持问题了。jQuery 对于 CSS 技术的支持主要体现在两方面：一是强大完善的选择器支持，使用 jQuery 可以精确选择页面中任意元素和结构；二是定义了几个超强的 CSS 方法，使用这些方法可以很方便地为页面元素定义样式，精确控制元素在页面中的显示。

6.1 css()方法

使用 css(name)可以读写元素的样式。在前面章节中我们也经常使用这个方法来为页面元素定义样式，或者获取指定属性的值。css(name)方法仅能够获取匹配元素中第一个元素的指定属性的属性值。例如，下面示例将弹出提示信息为“blue 1px solid”, 与原属性值排列顺序略有不同，这是因为在返回值时 JavaScript 会重新整理属性值的排列顺序。

```
<p style="border:solid 1px blue;">段落</p>
<script language="javascript" type="text/javascript">
alert($("p").css("border"));
</script>
```

css(name)方法仅能够读取元素的行内样式的属性值，而对于内部或外部样式表中的属性是

无法读取的，这与 DOM 中元素的 style 属性相似。使用 css(name,value)方法还能够为元素定义样式。请注意，这里的属性和属性值都是以字符串的形式存在的。例如，下面示例将为所有匹配的段落 p 设置红色背景。

```
<p>段落 1</p><p>段落 2</p>
<script language="javascript" type="text/javascript">
$("p").css("background","red");
</script>
```

在设置 CSS 属性名和属性值时，不能够使用 DOM 中 style 对象包含的样式属性，只需要保持 CSS 中的名称写法即可。例如，上面的设置样式也可以使用下面方法：

```
$("p").css("background-color","red");
```

也可以利用名/值对对象来为 css()方法传递多个属性和属性值，从而实现在一个方法体内定义多种样式效果。下面示例以格式化方法缩进显示了 css()方法内包含了一个样式对象，该对象包含三个声明，分别定义了匹配元素的字体颜色、字体大小和背景颜色。

```
<script language="javascript" type="text/javascript">
$("p").css({
    color:"blue",
    "font-size":"14px",
    "background-color":"red"
});
</script>
```

在对象体内，CSS 属性名可以不加引号，但是对于复合属性名，则必须增加引号。

6.2 位移

jQuery 定义了一个非常实用的 offset()方法，该方法能够获取匹配元素的第一个元素在当前窗口的坐标。该坐标以窗口左上顶角为圆点进行参考。返回对象包含 top 和 left 属性值，分别表示该元素距离左侧和顶部的记录。

```
<p>段落 1</p><p>段落 2</p>
<script language="javascript" type="text/javascript">
var offset = $("p:last").offset();
alert( "左侧距离: " + offset.left + ", 顶部距离: " + offset.top );
</script>
```

上面示例将返回提示对话框，显示第二个元素距离窗口左上顶角的 X 轴和 Y 轴的坐标值。

6.3 显示大小

除了使用 CSS 样式外，我们还可以使用 jQuery 来操控元素的显示大小。jQuery 定义了四种方法来实现元素的宽和高读写操作。

- height(): 获取第一个匹配元素当前计算的高度值（px）。
- width(): 获取第一个匹配元素当前计算的宽度值（px）。

- height(val): 为每个匹配的元素设置 CSS 高度属性值。如果没有明确指定单位，默认使用 px。
- width(val): 为每个匹配的元素设置 CSS 宽度属性值。如果没有明确指定单位，默认使用 px。

例如，在下面这个示例中，判断图像的大小是否大于 100 像素，如果大于 100 像素，则设置 100 像素。利用这种方法可以有效地控制图像的显示大小。

```
<img src="images/1.jpg" />
<script language="javascript" type="text/javascript">
var img = $("img");                              //获取图像对象
if (img.height() >100)                           //如果高度大于 100 像素
       img.height(100);                          //则设置高度为 100 像素
if (img.width() >100)                            //如果宽度大于 100 像素
       img.width(100);                           //则设置宽度为 100 像素
</script>
```

上述方法只能够设置匹配的第一个元素。因为能够自动控制页面中的所有图像，所以可以使用 each()方法进行遍历控制：

```
var img = $("img");
img.each(function(){
       if ($(this).height() >100)
          $(this).height(100);
       if ($(this).width() >100)
          $(this).width(100);
});
```

7 事件处理

jQuery 继承了 JavaScript 事件机制，并在 JavaScript 事件处理模型基础上进行了部分打包，且提供了很多个性化的事件处理方法，这些方法可以方便用户无须在元素或 DOM 对象上绑定事件，而是直接通过 jQuery 为对象添加事件，这样就加快了开发进度。下面就 jQuery 与 JavaScript 不同的事件处理方法进行讲解。

7.1 页面初始化事件

页面初始化事件是 Web 开发中比较常用的一种事件。在 DOM 一般通过如下两种方法来实现。

```
window.onload = function(){ }
```

或

```
<body onload="function()">
```

但是如果在页面中多处使用该函数，或者外部脚本文件(.js)中也包含 window.onload 函数，就很容发生冲突。jQuery 定义了 ready()方法，能够很好地解决了这个问题，它能够自动将其中

的函数在页面加载完成后运行，并且同一个页面中可以使用多个 ready()方法，而且相互之间不会发生冲突。

```
<script language="javascript" type="text/javascript">
$(document).ready(function(){
        alert("页面加载完毕！");
});
</script>
```

上面脚本将在页面加载完毕之后，自动弹出一个提示对话框，显示提示信息。当然也可以在这个函数中放置任何代码，如在页面初始化之后执行的代码。还可以进一步简写成下面的形式，也就是说在 jQuery()函数中直接包含页面初始化后执行的函数。

```
$(function(){
        alert("页面加载完毕！");
});
```

这是事件模块中最重要的一个函数，因为它可以极大地提高 Web 应用程序的响应速度。不过这种方法仅是 window.load 注册事件的一种替代方法，可以在同一个页面中无限次地使用$(document).ready()事件，其中注册的函数会根据代码中的先后顺序依次执行。

请注意，在使用 ready()方法注册页面初始化事件时，要确保在 body 元素的 onload 事件属性内没有注册函数，否则不会触发$(document).ready()事件。

7.2 绑定事件

jQuery 定义了几个方法用来为 jQuery 对象绑定事件，具体说明如下。

- bind(type,[data],fn)。

bind()方法能够为每一个匹配元素的特定事件绑定一个事件处理器函数。例如，下面示例中为所有 div 元素绑定鼠标单击事件，并注册一个事件处理函数，即单击当前 div 元素时会自动显示该元素包含的文本。

```
<div>盒子</div>
<script language="javascript" type="text/javascript">
$("div").bind("click",function(){
        alert($(this).text());
});
</script>
```

在绑定过程中也可以为事件处理函数绑定多个参数值，参数值以对象的形式进行传递，然后在处理函数中可以把它作为 event.data 属性值传递给处理函数。例如，在下面示例中，bind()方法以对象的形式传递了一个 who 等于“zhu”的参数值，然后在处理函数中利用 event.data 对象的属性把它捕捉并显示出来。

```
<div>盒子</div>
<script language="javascript" type="text/javascript">
$("div").bind("click",{who:"zhu"},function(event){
```

```
        alert(event.data.who);
});
</script>
```

绑定事件之后，也可以使用 unbind([type],[data])方法删除事件绑定，其中第一个参数表示要删除绑定的事件名，第二个参数表示删除的附带参数。例如，下面示例将把刚注册的鼠标单击事件删除掉。

```
<script language="javascript" type="text/javascript">
$("div").bind("click",{who:"zhu"},function(event){
        alert(event.data.who);
});
$("div").unbind("click");
</script>
```

- one(type,[data],fn)。

one()方法是 bind()方法的一个特例，它能够匹配元素绑定一个仅能够执行一次的事件处理函数。其用法与 bind()完全相同。例如，在上面示例的基础上进行简单修改如下。

```
<div>盒子</div>
<script language="javascript" type="text/javascript">
$("div").one("click",{who:"zhu"},function(event){
        alert(event.data.who);
});
</script>
```

这样当单击一次 div 元素之后，该对象的事件处理函数会自动被注销。

- trigger(type,[data])。

trigger 表示开关的意思，jQuery 定义 trigger()方法用来触发默认事件或自定义事件。例如，在下面示例中自定义了一个事件 me，把该事件绑定到 div 元素上，然后定义事件处理函数，弹出提示对话框，显示 div 元素包含的文本信息。

```
<div>盒子</div>
<script language="javascript" type="text/javascript">
$("div").bind("me", function () {
        alert($(this).text());
});
</script>
```

这个自定义事件是无法自动执行的，也不会响应鼠标或键盘行为。我们可以为它定义一个 trigger()方法，代码如下：

```
$("div").bind("me", function () {
        alert($(this).text());
});
$("div").trigger("me");
```

这样当页面加载时会自动执行该自定义事件函数。也可以把这个自定义事件放在另一个事件函数中，这样只有触发其他事件时，才会响应该自定义事件，并执行自定义事件处理函数。

```
$("div").bind("me", function () {
        alert($(this).text());
});
$("div").bind("mouseover",function(){
        $("div").trigger("me");
});
```

上面脚本将在鼠标移过时自动触发自定义的事件处理函数。对于默认事件，如果使用trigger()方法触发该事件处理函数，同时默认事件自身也可以自动触发事件。例如，在下面这个示例中，当鼠标指针移到 p 元素上时将触发绑定的事件 click，而当单击 div 元素时，也能够触发该事件绑定的处理函数。

```
<p>段落文本</p>
<div>盒子</div>
<script language="javascript" type="text/javascript">
$("div").bind("click", function () {
        alert($(this).text());
});
$("p").bind("mouseover",function(){
        $("div").trigger("click");
});
</script>
```

如果希望仅触发指定事件类型上所有绑定的处理函数，但不执行默认事件，则可以使用triggerHandler(type,[data])方法。triggerHandler()方法与 trigger()用法相同，但不会触发默认事件。

例如，在下面这个示例中，每当单击按钮一次，则将会触发 trigger()方法绑定的 focus 事件，同时浏览器默认的 focus 事件也将被触发一次，则“单击一次！”HTML 源代码被增加两次。

```
<input type="text" value=""/>
<button id="ok">ok</button>
<script language="javascript" type="text/javascript">
$("input").focus(function(){
        $("<span>单击一次！</span>").appendTo("body");
});
$("#ok").click(function(){
        $("input").trigger("focus");
});
</script>
```

如果把其中的 trigger()替换为 triggerHandler()，则单击时只执行一次插入操作（代码如下所示），系统默认的 focus 事件没有被自动触发。

```
$("#ok").click(function(){
        $("input").triggerHandler("focus");
```

```
});
```

7.3 交互事件

为了简化用户交互操作，jQuery 自定义了两个事件：hover(over,out)和 toggle(fn,fn)。hover()能够模仿悬停事件，即鼠标移到特定对象上以及移出该对象的方法。它定义当鼠标移到匹配的元素上时，会触发第一个函数。当鼠标移出该元素时，会触发第二个函数。例如，下面示例定义当鼠标移过段落文本时会设置字体显示为红色，而移开时又恢复默认颜色。

```
<p>段落文本</p>
<script language="javascript" type="text/javascript">
$("p").hover(
        function(){
            $(this).css("color","red");
        },
        function(){
            $(this).css("color","transparent");
        }
);
</script>
```

toggle(fn,fn)能够模仿鼠标单击事件，该方法我们已经讲解过，它表示每次单击时切换要调用的函数。如果单击了一个匹配的元素，则触发指定的第一个函数，当再次单击同一元素时，则触发指定的第二个函数，随后的每次单击都重复对这两个函数的轮番调用。例如，在上面示例的基础上，我们修改其中的 hover()方法，替换为 toggle()方法，这样当单击段落文本时，会自动在默认色和红色之间进行切换。对于该方法可以使用 unbind("click")删除。

```
<p>段落文本</p>
<script language="javascript" type="text/javascript">
$("p").toggle(
        function(){
            $(this).css("color","red");
        },
        function(){
            $(this).css("color","transparent");
        }
);
</script>
```

7.4 封装默认事件

除了使用 bind()、one()和 trigger()绑定或触发事件外，jQuery 还把 DOM 默认的事件都封装成了对应的方法，这样就可以在 jQuery 对象中作为一个方法直接调用。例如，click()能够触发元素的单击事件。也可以在 click()方法中设置一个事件处理函数，这样当单击元素时，将触发

执行该参数函数。

例如，下面示例演示了如何使用 click()方法，并为该方法绑定一个事件处理函数。这样当单击段落文本时自动设置文本的字体颜色为红色。

```
<p>段落文本</p>
<script language="javascript" type="text/javascript">
$("p").click(
        function(){
            $(this).css("color","red");
        }
);
</script>
```

我们也可以先为段落文本绑定事件，然后直接使用 click()调用事件处理函数，这样就不再使用鼠标单击即可自动触发该事件，从而设置字体颜色为红色。

```
<p>段落文本</p>
<script language="javascript" type="text/javascript">
$("p").bind("click",
        function(){
            $(this).css("color","red");
        }
);
$("p").click();
</script>
```

jQuery 封装的事件方法如表 5 所示。

表 5　jQuery 事件方法及其说明

jQuery 事件方法	说 明
blur()	触发每一个匹配元素的 blur 事件
blur(fn)	在每一个匹配元素的 blur 事件中绑定一个处理函数
change()	触发每一个匹配元素的 change 事件
change(fn)	在每一个匹配元素的 change 事件中绑定一个处理函数
click()	触发每一个匹配元素的 click 事件
click(fn)	在每一个匹配元素的 click 事件中绑定一个处理函数
dblclick()	触发每一个匹配元素的 dblclick 事件
dblclick(fn)	在每一个匹配元素的 dblclick 事件中绑定一个处理函数
error()	这个函数会调用执行绑定到 error 事件的所有函数

（续表）

jQuery 事件方法	说 明
error(fn)	在每一个匹配元素的 error 事件中绑定一个处理函数

focus()	触发每一个匹配元素的 focus 事件
focus(fn)	在每一个匹配元素的 focus 事件中绑定一个处理函数
keydown()	触发每一个匹配元素的 keydown 事件
keydown(fn)	在每一个匹配元素的 keydown 事件中绑定一个处理函数
keypress()	触发每一个匹配元素的 keypress 事件
keypress(fn)	在每一个匹配元素的 keypress 事件中绑定一个处理函数
keyup()	触发每一个匹配元素的 keyup 事件
keyup(fn)	在每一个匹配元素的 keyup 事件中绑定一个处理函数
load(fn)	在每一个匹配元素的 load 事件中绑定一个处理函数
mousedown(fn)	在每一个匹配元素的 mousedown 事件中绑定一个处理函数
mousemove(fn)	在每一个匹配元素的 mousemove 事件中绑定一个处理函数
mouseout(fn)	在每一个匹配元素的 mouseout 事件中绑定一个处理函数
mouseover(fn)	在每一个匹配元素的 mouseover 事件中绑定一个处理函数
mouseup(fn)	在每一个匹配元素的 mouseup 事件中绑定一个处理函数
resize(fn)	在每一个匹配元素的 resize 事件中绑定一个处理函数
scroll(fn)	在每一个匹配元素的 scroll 事件中绑定一个处理函数
select()	触发每一个匹配元素的 select 事件
select(fn)	在每一个匹配元素的 select 事件中绑定一个处理函数
submit()	触发每一个匹配元素的 submit 事件
submit(fn)	在每一个匹配元素的 submit 事件中绑定一个处理函数
unload(fn)	在每一个匹配元素的 unload 事件中绑定一个处理函数

8 动画处理

JavaScript 没有提供设计动画效果的函数，不过 jQuery 扩展了 JavaScript 这方面的不足。一方面它把平时常用的简单动画封装为直接调用的方法，另一方面还定义了几个比较实用的动画方法，调用这些方法就可以快速实现各种复杂的动画效果。

8.1 基本动画

jQuery 定义了显示、隐藏和切换隐藏/显示的方法，用来实现简单的动画效果。具体说明如下。

- show(): 显示隐藏的匹配元素。
- show(speed,[callback]): 使用动画方式显示所有匹配的元素，并在显示完成后触发回调函数。
- hide(): 隐藏显示的元素。

- hide(speed,[callback]): 使用动画方式隐藏所有匹配的元素，并在隐藏完成后触发回调函数。
- toggle(): 切换元素的可见状态。

例如，在下面示例中先使用 CSS 隐藏段落文本，然后使用 jQuery 定义 show()逐步显示或隐藏段落文本。并在显示完毕之后弹出一个提示对话框，提示文本显示完毕。

```
<p  style="display:none;">段落文本</p>
<script language="javascript" type="text/javascript">
$(function(){
       $("p").show(1000,
          function(){
             alert($(this).text()+"显示完毕");
       })
});
</script>
```

show(speed,[callback])和 hide(speed,[callback])这两种方法的用法相同，这些方法的第一个参数表示动画指定的毫秒数，jQuery 也定义了三个预定义关键字：slow、normal 和 fast。而 show()和 hide()是直接显示和隐藏。

读者在使用这些方法时应注意：只有隐藏元素才可以有效执行 show()或 show(speed,[callback])方法，而只有显示元素才能执行 hide()和 hide(speed,[callback])方法。

8.2 滑动动画

jQuery 还定义了几个滑动效果的显示和隐藏动画。其中 slideDown(speed,callback)可以把隐藏元素以滑动的效果显示出来。例如，下面示例的效果与 show(speed,[callback])方法设计的动画效果相同。

```
<p style="display:none;">段落文本</p>
<script language="javascript" type="text/javascript">
$("p").slideDown("slow");
</script>
```

而下面示例的演示效果正好相反，它把段落文本以滑动形式隐藏起来。

```
<p>段落文本</p>
<script language="javascript" type="text/javascript">
$("p").slideUp("slow");
</script>
```

使用 slideToggle(speed,[callback])方法能够滑动隐藏显示元素或显示隐藏元素。例如，下面示例先向下滑动显示段落文本，然后再向上滑动显示段落文本。

```
<p style="display:none;">段落文本</p>
<script language="javascript" type="text/javascript">
$("p").slideToggle("slow");
$("p").slideToggle("slow");
```

```
</script>
```

8.3 淡入淡出

jQuery 定义了 fadeIn(speed,[callback])和 fadeOut(speed,[callback])方法来设计显示/隐藏元素的淡入淡出效果，其用法与上面示例相同，只不过动画效果不同。例如，下面示例以渐变的方法逐渐显示隐藏的段落文本。

```
<p style="display:none;">段落文本</p>
<script language="javascript" type="text/javascript">
$("p").fadeIn("slow");
</script>
```

当然，混合使用这两种方法可以设计更复杂的动画效果。例如，下面示例中段落文本先是以逐步淡入的效果显示出来，然后当鼠标经过时，又以极快的速度淡出—淡入—淡出—淡入，从而设计出闪动的特效。

```
<p style="display:none;">段落文本</p>
<script language="javascript" type="text/javascript">
$("p").fadeIn(2000);
$("p").bind("mouseover",function(){
        $("p").fadeOut(100);
        $("p").fadeIn(100);
        $("p").fadeOut(100);
        $("p").fadeIn(100);
});
</script>
```

另外，jQuery 还定义了另外一个很实用的 fadeTo(speed,opacity,[callback])，该方法可以把显示的元素逐步淡出为半透明效果，这个半透明可以在参数中进行设置。不透明度值可以是 0 到 1 之间的数字。

例如，在下面示例中，首先利用 fadeTo()方法把显示的段落文本逐步过渡为半透明效果，透明度为 0.2，然后在执行完该淡出效果后，再次在回调函数中调用 fadeTo()方法把半透明的段落文本逐步淡入到不透明状态。整个设计从而产生一种半显半隐的初始化效果。

```
<p>段落文本</p>
<script language="javascript" type="text/javascript">
$("p").fadeTo("slow", 0.2, function(){
        $(this).fadeTo("slow", 1);
});
</script>
```

9 Ajax 技术处理

jQuery 也支持 Ajax 技术，它封装了 XMLHttpRequest 组件并初始化，还封装了 Ajax 请求中各种基本操作，并把这些操作定义为简单的方法。另外，把 Ajax 请求中各种状态封装为事件，这样只要调用对应的事件就可以快速执行绑定的回调函数。

9.1 Ajax 请求

jQuery 封装了多种方法实现与远程进行通信，下面分别进行讲解。

- load(url,[data],[callback])。

load()方法能够载入远程 HTML 文件代码并插入到匹配元素中。默认使用 GET 方式，传递附加参数时自动转换为 POST 方式。例如，新建一个 text.html 文档。

```
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Ajax 请求</title>
    </head>
    <body>
    <li>列表项 1</li>
    <li>列表项 2</li>
    <li>列表项 3</li>
    </body>
    </html>
```

然后在当前示例文档中输入下面代码：

```
    <ul></ul>
    <script language="javascript" type="text/javascript">
    $("ul").load("test.html");
    </script>
```

这样 jQuery 会自动从 text.html 文档中提取 body 元素包含的代码，并把这些代码插入到匹配的 ul 元素中。最后显示为：

```
    <ul>
    <li>列表项 1</li>
    <li>列表项 2</li>
    <li>列表项 3</li>
    </ul>
```

请注意，在使用 load()方法时，所有页面的字符编码应该设置为 utf-8，否则 jQuery 在加载文档时会显示乱码。另外，匹配的元素应该只有一个，否则系统会出现异常。例如，下面代码将会导致加载出现异常。

```
<ul></ul>
<ul></ul>
<script language="javascript" type="text/javascript">
$("ul").load("test.html");
</script>
```

load(url,[data],[callback])方法还可以附带传递参数，这些参数将以 POST 方式传递给服务器，并可以定义加载成功时执行的回调函数。例如，下面示例将向服务器端的 test.asp 发送一个请求，并以 POST 的方式传递一个参数 name，参数值为“zhu”。加载成功之后会弹出一个提示信息对话框。

```
<ul></ul>
<script language="javascript" type="text/javascript">
$("ul").load("20-88_test.asp", {name:"zhu"}, function(){
    alert("加载成功！");
});
</script>
```

- jQuery.get(url,[data],[callback])。

jQuery.get()方法能够通过远程 HTTP GET 方式请求载入信息。该方法包含三个参数，参数含义与 load()方法相同。例如，新建一个 test.asp 服务器文件，输入下面代码，用来获取客户端传递过来的查询字符串信息，并把这些信息反馈给客户端。请注意，本示例需要服务器环境的支持，否则无法执行。

```
<%
dim str
str = Request.QueryString("name")          '从客户端接收查询字符串参数值
Response.Write str                         '把接收的查询字符串再响应给客户端
%>
```

然后，在示例文档中输入下面代码：

```
<script language="javascript" type="text/javascript">
$.get("test.asp", {name:"zhu"}, function(data){
       alert(data);
});
</script>
```

在上面代码中使用 jQuery 对象的 get()方法向服务器端 test.asp 文件发送一个请求，并以 GET 的方式向服务器传递一个参数，服务器响应之后会把返回值存储在回调函数参数中，所以，最后弹出的提示对话框显示为“zhu”字符信息。

- jQuery.post(url,[data],[callback])。

jQuery.post()与 jQuery.get()的操作方法相同，不同点是它们传递参数的方式不同。

jQuery.post()是以 POST 方式来传递参数，所传递的信息可以不受限制，且可以传递二进制信息，具体用法就不再举例。

9.2 jQuery.ajax()请求

jQuery.ajax(options)方法与 9.1 节讲解的几个方法功能相同，都是向服务器端发送请求，并传递参数，最后调用回调函数获取响应信息。下面看一个示例，该示例是在 9.1 节示例基础上进行修改的。

首先，建立一个服务器端处理文件（test.asp），代码如下：

```
<%
dim user,where
user = Request.Form("user")
where = Request.Form("where")
Response.AddHeader "Content-Type","text/html;charset=utf-8"
Response.Write user&"在"&where
%>
```

在上面代码中，首先获取客户端传递过来的参数值，使用 Request.Form()数据集合进行读取，因为客户端采用 POST 方式进行传递。然后再把这些信息传递给客户端，考虑到传递字符中可能包含中文字符，所以这里需要设置传递的文件字符，编码统一为 utf-8。

在本地文件中输入下面代码：

```
<script language="javascript" type="text/javascript">
$.ajax({
       type: "POST",                          //设置请求方式
       url: "20-91_test.asp",                 //设置请求 URL
       data: "user=朱印宏&where=家里",        //设置传递的参数值
       success: function(msg){                //设置响应成功之后执行的回调函数
          alert(msg );                        //显示服务器响应的信息
       }
});
</script>
```

最后执行文件，会弹出如图 4 所示的提示信息。

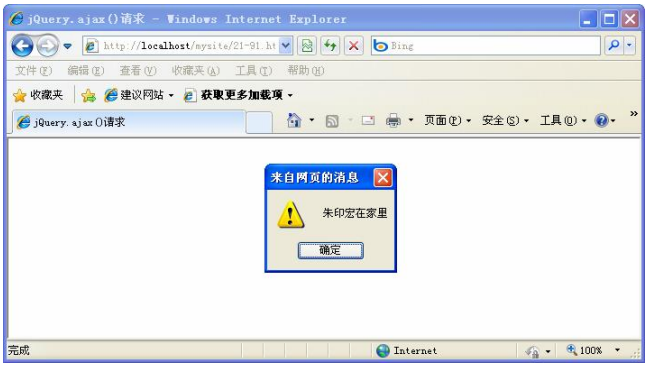

图 4　jQuery.ajax()方法执行效果

jQuery.ajax()方法的参数是一个对象包含的参数名/值对组合，其中主要参数名如表 6 所示。

表 6　jQuery.ajax()方法的主要参数名及其说明

参数名	说 明
async	逻辑值，默认为 true，即请求为异步请求。如果需要发送同步请求，该选项设置为 false。同步请求将锁住浏览器，用户的其他操作必须等待请求完成才可以执行
beforeSend	发送请求前可修改 XMLHttpRequest 对象的函数
cache	是否从浏览器缓存中加载请求信息，默认为 false
complete	请求完成后回调函数，不管请求是成功还是失败均调用
contentType	发送信息到服务器时内容编码类型，默认为适合大多数应用场合
data	发送到服务器的数据将自动转换为请求字符串格式。GET 请求中将附加在 URL 后。如果 processData 选项禁止此自动转换，必须为名/值对格式。如果为数组，jQuery 将自动为不同值对应同一个名称，如{foo:["bar1","bar2"]}转换为'&foo=bar1&foo=bar2'
dataType	预期服务器返回的数据类型。取值包括 xml、html、script、json 和 jsonp
error	请求失败时调用函数
global	是否触发全局 Ajax 事件，默认为 true
ifModified	是否仅在服务器数据改变时获取新数据，默认为 false
processData	发送的数据是否可以被转换为对象，默认为 true
success	请求成功后回调函数，参数为服务器返回数据
timeout	设置请求超时时间（毫秒）
type	发送请求的方式，默认为 GET，取值包含 POST、GET、PUT 和 DELETE
url	发送请求的地址

9.3 Ajax 事件

除了 9.2 节讲解的几个 Ajax 请求所使用的方法外，jQuery 为了方便用户灵活跟踪 Ajax 请求和响应整个完整的过程，还定义了几个事件函数，说明如表 7 所示。

表 7 Ajax 事件函数及其说明

Ajax 事件	说 明
ajaxStart(callback)	Ajax 请求开始时执行函数
ajaxSend(callback)	Ajax 请求发送前执行函数
ajaxComplete(callback)	Ajax 请求完成时执行函数
ajaxSuccess(callback)	Ajax 请求成功时执行函数
ajaxError(callback)	Ajax 请求发生错误时执行函数
ajaxStop(callback)	Ajax 请求结束时执行函数

例如，下面示例连续跟踪 Ajax 请求的全部过程。该过程实际上还是利用 readyState 属性进行跟踪的，把该属性封装到一个方法中，这样使用更加方便。

```
<p></p>
<script language="javascript" type="text/javascript">
$.ajax({
        type: "POST",
        url: "20-91_test.asp",
        data: "user=朱印宏&where=家里"
});
$("p").ajaxStart(function(){
        $(this).text("Ajax 请求开始");
});
$("p").ajaxSend(function(){
        $(this).text("Ajax 请求开始发送");
});
$("p").ajaxComplete(function(){
        $(this).text("Ajax 请求完成");
});
$("p").ajaxSuccess(function(){
        $(this).text("Ajax 请求成功");
});
</script>
```

10 jQuery 技术应用实战

在初步掌握了 jQuery 框架使用的基础上，下面结合三个示例展示 jQuery 在实际开发中的应用。如果用惯了 jQuery 之后，就会发现 Web 开发会非常顺手，与直接使用 JavaScript 来实现某

些功能，简直是天壤之别。当然，jQuery 框架与 JavaScript 脚本是融为一体的，且可以无缝混合使用，不用担心它们的语法冲突。从本质上讲，jQuery 仅是对 JavaScript 逻辑块的封装。

10.1 图片画廊

这是个简单但很实用的示例，如果直接使用 JavaScript 来设计这个效果，可能会需要很多代码，而使用 jQuery 就会感觉很轻松。当用户移动鼠标指针到缩微图上时，该图会自动被放大显示在上面画框内，如图 5 所示。

图 5　图片画廊

该示例主要利用了 jQurey 选择页面元素，然后再利用 attr()方法读写元素属性实现该效果，主要步骤如下。

第一步，设计 HTML 结构。图片画廊的结构包含在一个 div 盒子内，通过两个 p 元素分别来组织大图画框和缩微图列表框。代码如下：

```
<div id="box">
    <h1>图片画廊</h1>
    <p><img id="largeImg" src="images/1.jpg" title="images/1.jpg" /></p>
    <p class="thumbs">
       <a href="images/1.jpg" title="images/1.jpg"><img src="images/1.jpg" /></a>
      <a href="images/2.jpg" title="images/2.jpg"><img src="images/2.jpg" /></a>
      <a href="images/3.jpg" title="images/3.jpg"><img src="images/3.jpg" /></a>
      <a href="images/4.jpg" title="images/4.jpg"><img src="images/4.jpg" /></a>
      <a href="images/5.jpg" title="images/5.jpg"><img src="images/5.jpg" /></a>
    </p>
</div>
```

第二步，设计结构的显示样式。代码如下：

```
<style type="text/css">
body { text-align:center; }                    /* 居中显示 */
#box {/* 盒子样式 */
    width:500px;                               /* 固定宽度 */
```

```
        margin:12px auto;                          /* 居中显示 */
    }
    #largeImg {/* 大图画框样式 */
        border: solid 1px #aaa;                    /* 定义边框 */
        width: 330px;                              /* 定义大图画框的宽度 */
        height: 200px;                             /* 固定高度 */
        padding: 2px;                              /* 定义补白，增加一点内边距 */
    }
    .thumbs img {/* 缩微图样式 */
        border: solid 1px #aaa;                    /* 边框样式 */
        width: 50px;                               /* 宽度 */
        height: 50px;                              /* 高度 */
        padding: 4px;                              /* 增加补白 */
    }
    p {/* 段落样式 */
        padding:0;                                 /* 清除段落补白 */
        margin:6px;                                /* 清除段落边界 */
    }
    </style>
```

第三步，导入 jQuey 框架，然后定义两个函数，设置在页面初始化后（即网页加载完毕）触发。第一个函数定义当鼠标指针移过缩微图时将获取缩微图的地址和提示属性信息，并保存在变量中。然后把这些信息赋值给大图画框内包含的图像，同时设置缩微图的边框颜色，以设计动态效果。最后，在鼠标指针移开缩微图时，再恢复缩微图的边框颜色。

```
    <script type="text/javascript" src="../images/jquery.js"></script>
    <script type="text/javascript">
    $(function(){// 页面初始化激活函数
        $(".thumbs a").mouseover(function(){            // 鼠标经过缩微图时的处理
函数
                var largePath = $(this).attr("href");    //获取缩微图的地址信息
                var largeTitle = $(this).attr("title");  //获取缩微图的提示信息
                $(this).children().css("border-color","#F90");  //设置缩微图内超
链接样式
                //把缩微图的地址和提示信息赋予给大图
            $("#largeImg").attr({ src: largePath, title: largeTitle });
             });
             $(".thumbs a").mouseout(function(){ // 鼠标移开缩微图时的处理函数
                $(this).children().css("border-color","#aaa");  // 恢复缩微图内
超链接的默认边框颜色
             });
         });
    </script>
```

10.2 收缩置顶条

你可能在网上看到过这种可以控制的工具条，不用时收缩，使用时打开，不占网页空间，使用起来很方便，如图 6 所示。其实使用 jQuery 来设计这样的效果会很简单，这里主要使用了 jQuery 的 slideDown()和 slideUp()的动画方法来实现，具体方法如下。

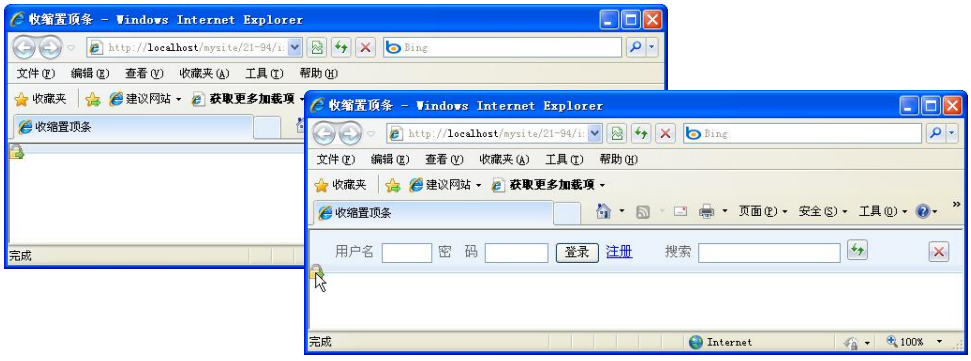

图 6 收缩置顶条

第一步，搭建收缩工具条的 HTML 结构。本示例也是使用 div 元素构建一个包含框，然后插入表单结构框架，同时在这个包含框底部插入一个 div 元素，该元素相当于一个控制按钮，用来打开表单结。

```
<div id="top">
    <form  id="panel" action="#" method="get">
        <label class="left2em" for="user">用户名 </label>
        <input id="user" size="6"  name="user" type="text" />
        <label for="pass">密码 </label>
        <input id="pass" size="8" name="pass" type="text" />
        <input name="" type="submit" value="登录" />
        <a href="#">注册</a>
        <label class="left2em" for="serch">搜索</label>
        <input id="serch" name="serch" type="text" />
        <input class="middle" name="" type="image" src="images/submit.gif" />
        <img id="close" src="images/close.gif" />
    </form>
    <div id="slide"><img src="images/lock.gif" /></div>
</div>
```

第二步，定义样式表。这里重点设计了工具条面板的显示样式以及控制按钮的样式。设置控制面板相对定位，这样可以为内部包含的图像按钮进行精确定位，同时也能够使内部图像对象绝对定位时能够被包含在该工具面板框内，可以实现协调一致的动画效果。在默认显示时，隐藏工具条面板。设置工具条控制元素 div 显示为一条线效果，详细代码如下：

```
<style type="text/css">
body { margin: 0 auto; padding: 0;}     /* 清除页边距 */
```

```
    #panel {/* 工具条面板样式 */
        position:relative;                    /* 相对定位，以便内部包含元素进行绝对定
位 */
        background:#EBF4FD;                   /* 背景色 */
        display: none;                        /* 隐藏显示 */
        width:100%;                           /* 定义表单框架的宽度，因为部分浏览器会
出现兼容问题 */
        margin:0;                             /* 清除表单在动画解析中可能存在的空白区
域 */
        padding-top:6px;                      /* 顶部补白 */
        font-size:14px;                       /* 字体大小 */
        color:#666;                           /* 字体颜色 */
    }
    #slide {/* 滑动控制元素样式 */
        background:url(images/btn-slide.gif) repeat-x center bottom;
        text-align:left;
        height:10px;
        position:relative;                    /* 相对定位,以便内部图像能够绝对定位 */
    }
    #slide img {/* 控制图像样式 */
        position:absolute;                    /* 绝对定位 */
        left:0;                               /* 左侧距离 */
        top:0;                                /* 顶部距离 */
    }
    #close {/* 关闭工具条图像样式 */
        position:absolute;                    /* 绝对定位 */
        right:1em;                            /* 右侧距离 */
        bottom:2px;                           /* 底部距离 */
    }
    .middle {/* 垂直居中样式类 */
        position:relative;
        top:4px;
    }
    .left2em { margin-left:2em; }             /* 左侧间距样式类 */
    </style>
```

第三步，设计脚本用来控制工具条的收缩和展开。在页面初始化事件函数中绑定两个子事件函数，当鼠标移过控制图像上时，将调用 jQuery 的 slideDown()动画方法滑动打开工具条面板。而当单击关闭按钮图像时，即触发单击事件处理函数，该函数调用图像的 slideUp()方法向上关闭工具条面板。详细代码如下：

```
    <script type="text/javascript" src="../images/jquery.js"></script>
    <script type="text/javascript">
    $(function(){                             // 页面初始化事件处理函数
```

```
            $("#slide").children().mouseover(function(){    // 鼠标移过滑动控制
图像时的事件处理函数
                $("#panel").slideDown("normal");          // 滑动展开工具条面板
        });
            $("#close").click(function(){                 // 单击关闭图像时的事件处
理函数
                    $("#panel").slideUp("normal");        // 滑动收缩工具条面板
            });
        });
    </script>
```

10.3 超级链接类型标识图标

CSS 2.0 版本支持属性选择器，目前 IE 7 及其他标准浏览器都支持这些功能。利用属性选择符可以为超链接设计类型标识图标，这样当看到不同类型的图标时就可以直观地了解超链接所链接的文件类型。但是 IE 6 及其以下版本浏览器不支持该功能，而 jQuery 支持 CSS 1.0 至 CSS3.0 所有功能，所以从安全角度考虑，使用 jQuery 定义选择器会更加安全和灵活。如图 7 所示是本示例设计的超级链接类型标识图标，当然还可以进行扩展 jQuery 选择器，以满足个性化设计需求。

图.7 超级链接类型标识图标

本示例主要利用 jQuery 强大选择器功能，通过匹配超链接的类型来选择不同的 a 元素，然后再分别为它们绑定不同的类样式。在类样式中定义文件类型图标，从而实现在超链接中显示为不同类型的图标标识效果。主要设计思路如下。

第一步，在页面中设计多个不同类型的超链接。读者也可以根据需要进行扩展和改写。

```
    <h1>超级链接类型标识图标</h1>
    <p><a href="http://www.css8.cn/name.pdf">PDF 文件</a> </p>
    <p><a href="http://www.css8.cn/name.ppt">PPT 文件</a> </p>
    <p><a href="http://www.css8.cn/name.xls">XLS 文件</a> </p>
    <p><a href="http://www.css8.cn/name.rar">RAR 文件</a> </p>
    <p><a href="http://www.css8.cn/name.gif">GIF 文件</a> </p>
    <p><a href="http://www.css8.cn/name.jpg">JPG 文件</a> </p>
    <p><a href="http://www.css8.cn/name.png">PNG 文件</a> </p>
    <p><a href="http://www.css8.cn/name.txt">TXT 文件</a> </p>
    <p><a href="http://www.css8.cn/#anchor">#锚点超链接</a></p>
```

```
<p><a href="http://www.css8.cn/">http://www.css8.cn/</a></p>
<p><a href="css8.cn">http://www.css8.cn</a></p>
```

第二步，定义不同类型的超链接样式类。类名以文件扩展名进行命名，在每个样式类中定义背景图像为文件类型图标，并定位到超链接的左侧居中位置。

```
<style type="text/css">
p{ float:left; width:200px; margin:4px;}
a.pdf { /* PDF 文件类型链接 */
    background: url(images/icon_pdf.gif) no-repeat left center;
    padding-left: 18px;
}
a.xls { /* XML 样式表文件类型链接 */
    background: url(images/icon_xls.gif) no-repeat left center;
    padding-left: 18px;
}
a.ppt { /* 演示文稿文件类型链接 */
    background: url(images/icon_ppt.gif) no-repeat left center;
    padding-left: 18px;
}
a.rar { /* 压缩文件类型链接 */
    background: url(images/icon_rar.gif) no-repeat left center;
    padding-left: 18px;
}
a.img { /* 图像文件类型链接 */
    background: url(images/icon_img.gif) no-repeat left center;
    padding-left: 18px;
}
a.txt { /* 文本文件类型链接 */
    background: url(images/icon_txt.gif) no-repeat left center;
    padding-left: 18px;
}
a.external { /* 其他不明文件类型链接 */
    background: url(images/window.gif) no-repeat left center;
    padding-left: 18px;
}
</style>
```

第三步，使用 jQuery 属性选择器分别选择不同类型文件超链接。使用 href 属性分别来匹配该属性值最末尾的扩展名。然后为匹配元素增加对应类样式。最后使用 not()方法排除 href 属性值不包含 http://www.字符串或者不包含“#”符号的超链接，为它们定义一种特殊的图标，表示此类型超链级未知。

```
<script type="text/javascript" src="../images/jquery.js"></script>
<script type="text/javascript">
$(function(){
```

```
            $("a[href$=pdf]").addClass("pdf");          // 为 pdf 文件类型超链接增
加.pdf 样式类
            $("a[href$=xls]").addClass("xls");          // 为 xls 文件类型超链接增
加.xls 样式类
            $("a[href$=ppt]").addClass("ppt");          // 为 ppt 文件类型超链接增
加.ppt 样式类
            $("a[href$=rar]").addClass("rar");          // 为 rar 文件类型超链接增
加.rar 样式类
            $("a[href$=gif]").addClass("img");          // 为 gif 文件类型超链接增
加.gif 样式类
            $("a[href$=jpg]").addClass("img");          // 为 img 文件类型超链接增
加.img 样式类
            $("a[href$=png]").addClass("img");          // 为 png 文件类型超链接增
加.png 样式类
            $("a[href$=txt]").addClass("txt");          // 为 txt 文件类型超链接增
加.txt 样式类
            // 为不符合类型的超链接增加特殊样式类，并增加 target 属性
        $("a:not([href*=http://www.])").not("[href^=#]")
                .addClass("external")
                .attr({ target: "_blank" });
    });
    </script>
```