前　言

jQuery 是网页设计中一柄锋利的宝剑，使用 jQuery 框架技术，可以快速实现页面上各种强大的效果。所以，要搭建网站和设计网页，就必须要学会 jQuery 技术。本书是一本讲述代码实践的书，为读者全面深入地讲解了针对各种页面元素和页面特效的 jQuery 技术。近 350 段代码给读者带来的不仅仅是网页开发速度的提升，还有如何创建高性能网页、如何处理 jQuery 特效、如何应对跨浏览器兼容、如何进行前端代码优化等时时刻刻困扰网站开发人员的问题解决方案。

jQuery 中的那些事儿

有些网站开发人员写了多年 jQuery，依然在面对新问题时束手无策，这些技术难点具有普适性，读者能有多少了解呢？

- 跨浏览器的兼容
- 选择器的优化
- 事件注册和事件注销
- 动态修改 CSS 样式
- 文字、图片的流行特效
- 基于 jQuery 的插件开发
- Callback 回调机制
- AJAX 无刷新更新页面
- IE 老版本的兼容
- jQuery Mobile 移动端扩展

以上所有内容在本书的代码中都有讲解。除这些常见 jQuery 技术难点外，本书力求将最有用的 jQuery 代码段汇总在一起，提供各种解决实际问题的方案。

jQuery 学习方法

11 个字就能帮助我们更好地学习 jQuery。

- **多看、多练**：观摩成功的网站设计，分析并练习网站开发常用的代码。
- **多想、多问**：思考功能实现的原理，提出自己的问题并通过各种渠道寻找答案。
- **多总结**：记录前人已经探索出来的 jQuery 技巧，总结实战中碰到的问题及解决方案。

第 1 章　jQuery 操作网页

jQuery 框架是一个简洁快速的 JavaScript 脚本库，它可以在网页上简单快捷地操作文档、控件和元素，实现页面特效、动画等功能。使用 jQuery 框架进行设计会改变传统 JavaScript 代码的编写方式，极大地提高编程效率。

本章主要介绍如何使用 jQuery 实现网页操作，包括以下内容：

- jQuery DOM 的操作方法。
- jQuery 动画效果的实现。
- jQuery 控件下 iframe 的使用。
- jQuery 实现各种文本特效。
- jQuery 控制页面布局的方法。
- jQuery 与 Flash 组合应用。

1.1　显示或隐藏网页内容

用户使用在线支付平台选择具体付款银行时，会发现银行数量较多而无法全部显示出来，这时网页会通过一个类似“展开全部”或者“显示更多”的功能按钮，让用户浏览到全部银行，如图 1.1 和图 1.2 所示。此功能使用 jQuery 来操作非常简单，仅仅需要 2~3 个函数就可以完美地实现。

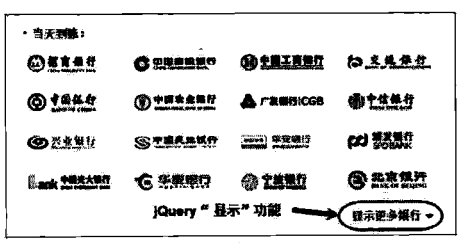

图 1.1　jQuery“显示”网页内容操作

图 1.2　jQuery“隐藏”网页内容操作

本例主要代码如下：

```
/* showhide.html */
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<script src="../code.jquery.com/jquery-1.9.1.min.js"></script>
<link rel="stylesheet" href="css/demos.css">
<script type="text/javascript">
$(document).ready(function(){
    $("#id-button-show").click(function(){
        $("h3").show();
        /* 省略部分代码，此处可以添加用户代码 */
    });
    $("#id-button-hide").click(function(){
        $("h3").hide();
        /* 省略部分代码，此处可以添加用户代码 */
    });
});
</script>
</head>
<body>
    <h2 class="h2-caption">超实用的 jQuery 代码段 - jQuery 实现显示和隐藏网页内容功能
</h2>
    <button type="button" id="id-button-show">显示</button>
    <button type="button" id="id-button-hide">隐藏</button>
    /* 省略部分代码，此处可以添加用户代码 */
</body>
</html>
```

jQuery 代码本质上也是属于 JavaScript 的脚本范畴。一般来讲，使用 JavaScript 代码的方法主要有以下两种：

- 第 1 种方法可以将 JavaScript 代码段直接嵌入到 HTML/JSP/PHP 等页面文件中使用。
- 第 2 种方法可以将 JavaScript 代码写在单独的后缀名为.js 的脚本文件中，然后在 HTML/JSP/PHP 等页面文件中引用.js 文件。

对于第 2 种方法，用户首先将标准的 jQuery 框架类库文件在 HTML/JSP/PHP 等页面文件的<head></head>头部引入，然后就可以在页面内使用 jQuery 框架的“显示”和“隐藏”功能了。

第 10~13 行的“显示”功能通过 jQuery 框架的 show()函数来实现，该函数用于显示 HTML 页面中的元素，其语法如下：

```
$(selector).show(speed,callback);
```

其中 speed 参数用来设置显示的速度，可取值为 slow、fast 或毫秒；callback 参数用来设置动作完成后所执行的函数。

第14~17行的“隐藏”功能通过jQuery框架的hide()函数来实现，它用于隐藏HTML页面的元素，其语法如下：

```
$(selector).hide(speed,callback);
```

其中speed参数与callback参数的用法与show()函数的参数是相同的。

提示：一般开发复杂页面时（例如目前流行的Web 2.0网站），建议采用第2种方法，这样在页面文件增多时，可以引用同一个JavaScript脚本文件，同时也使得代码可读性和文件规范性更好。

1.2 切换页面的显示或隐藏

上一个jQuery代码段向读者演示了通过show()和hide()函数实现“显示”和“隐藏”页面元素的功能。其实，使用jQuery框架的toggle()函数也能实现同样的效果。toggle()函数用于切换页面元素的可见状态，如果被选择元素处于可见状态，则隐藏这些元素，如果被选择元素处于隐藏状态，则显示这些元素。

本例代码如下：

```
01  /* toggle.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $("button").click(function(){
05          $("h3").toggle();
06          /* 省略部分代码，此处可以添加用户代码 */
07      });
08  });
09  ……//页面body代码
10  <body>
11      <h2 class="h2-caption">超实用的jQuery代码段 - jQuery切换显示和隐藏功能的方法
12      </h2>
13      <hr><br>
14      <button type="button">显示/隐藏</button>
15  <h3 class="h3-text">jQuery 效果 - 隐藏和显示</h3>
16      <h3 class="h3-text">jQuery toggle()函数 - 通过jQuery，用户可以使用toggle()函数来切
17  换hide()函数和show()函数</h3>
18  /* 省略部分代码，此处可以添加用户代码 */
19  </body>
```

第05行实现了切换“显示”和“隐藏”页面元素的功能，其中，toggle()函数用于切换hide()函数和show()函数，其语法如下：

```
$(selector).toggle(speed,callback);
```

speed参数用来设置显示的速度，可取值为slow、fast或毫秒；callback参数用来设置动作完成后所执行的函数。

注意：toggle()函数适用于通过 jQuery 框架隐藏的页面元素或在 CSS 中声明为“display:none”的页面元素，但是它不适用于在 CSS 中声明为“visibility:hidden”的页面元素。

1.3 实现幻灯片式的淡入淡出效果

使用 PowerPoint 编写过幻灯片报告的用户，肯定对幻灯片切换时的“淡入淡出”效果印象颇深。其实，在网页开发中使用 jQuery 框架实现整个页面的“淡入淡出”效果也很方便，如图 1.3 所示。

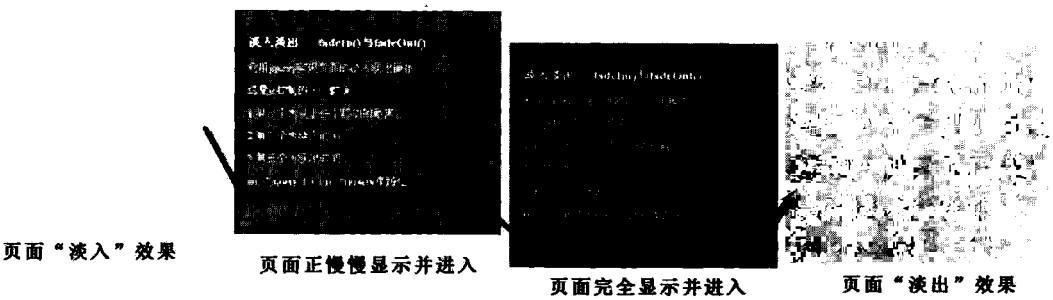

图 1.3 jQuery 实现页面“淡入淡出”操作

本例主要代码如下：

```
01  /* fadeInOut.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $("#id-button-fadeout").click(function(){
05          $("#id-div-fade").fadeOut(3000);              //设定淡出效果时间为 3000 毫秒
06          /* 省略部分代码，此处可以添加用户代码 */
07      });
08      $("#id-button-fadein").click(function(){
09          $("#id-div-fade").fadeIn("fast ");            //设定淡入效果时间 fast 模式
10          /* 省略部分代码，此处可以添加用户代码 */
11      });
12  });
13  </script>
14  <body>
15      <h2 class="h2-caption">超实用的 jQuery 代码段 - jQuery 实现页面淡入淡出操作</h2>
16      /* 省略部分代码，此处可以添加用户代码 */
17      <h3 class="h3-text">页面淡入淡出 --- fadeIn()与 fadeOut()函数</h3>
18      <button id="id-button-fadeout">页面淡出效果</button>
19      <button id="id-button-fadein">页面淡入效果</button>
20      <div id="id-div-fade">
```

```
21              <p>这里 js 控制的两个数值:</p>
22              <p>1.第 1 个为淡入时间;</p>
23              <p>2.第 2 个为淡出时间;</p>
24          </div>
25          /* 省略部分代码，此处可以添加用户代码 */
26      </body>
```

第 04~07 行实现的“淡出”效果通过 fadeOut()函数实现，该函数使用淡出效果来隐藏被选择的页面元素，其语法如下：

```
$(selector).fadeOut(speed,callback);
```

第 08~11 行实现的“淡入”效果通过 fadeIn()函数实现，该函数使用淡入效果来显示被选择的页面元素，其语法如下：

```
$(selector).fadeIn(speed,callback);
```

提示：使用 fadeIn()函数实现页面元素“淡入”效果时，如果该元素已经显示，则该效果不产生任何变化，除非规定了 callback 函数。同理，使用 fadeOut()函数实现页面元素“淡出”效果时，如果元素已经隐藏，则该效果不产生任何变化，除非规定了 callback 函数。

注意：fadeIn()函数适用于通过 jQuery 框架隐藏的页面元素或在 CSS 中声明为“display:none”的页面元素，但该函数不适用于在 CSS 中声明为“visibility:hidden”的页面元素。

1.4 切换页面的淡入淡出

上一个 jQuery 代码段向读者演示了通过 fadeIn()和 fadeOut()函数实现“淡入”和“淡出”页面元素的效果。其实，使用 fadeToggle()函数也能实现同样的效果，下面看具体的代码：

```
01  /* fadeToggle.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $("#id-button-fadetoggle").click(function(){
05          $("#id-div-fade").fadeToggle(3000);
06          $("#id-div-fade").fadeToggle(3000);
07          $("#id-div-fade").fadeTo(5000,0.6);
08      });
09  });
10  </script>
11  <body>
12      <h2 class="h2-caption">超实用的 jQuery 代码段 - jQuery 切换页面淡入淡出操作</h2>
13      <hr><br>
14      <h3 class="h3-text">页面淡入淡出 --- fadeToggle()与 fadeTo()函数</h3>
15      <button id="id-button-fadetoggle">切换页面淡入淡出效果</button>
16      <div id="id-div-fade">
17          <p>这里 js 控制的两个数值:</p>
```

```
18              <p>1.第 1 个为淡入时间;</p>
19              <p>2.第 2 个为淡出时间;</p>
20          </div>
21  /* 省略部分代码，此处可以添加用户代码 */
22  </body>
```

第 05~06 行中的 fadeToggle()函数可以在 fadeIn()函数与 fadeOut()函数之间进行切换。如果元素已淡出，则 fadeToggle()函数会向元素添加淡入效果；如果元素已淡入，则 fadeToggle()函数会向元素添加淡出效果。第 07 行中的 fadeTo()函数允许为渐变指定不透明度（值介于 0 与 1 之间），其语法如下：

语法：$(selector).fadeTo(speed,opacity,callback);

其中 speed 参数用来设置显示的速度，可取值为 slow、fast 或毫秒；opacity 参数用于指定不透明度（值介于 0 与 1 之间）；callback 参数用来设置 fading 效果完成后所执行的函数。

提示：目前，使用 opacity 参数实现的半透明效果在网页开发中应用十分广泛，尤其在层与层相互叠加时美化效果十分明显，其取值在[0，1]闭区间内，越接近数值 0 透明度越高，越接近数值 1 不透明度越高，当取值等于 1 时则完全不透明。

1.5 页面的滑动隐藏

越来越多的人习惯用手机浏览页面，而手机上可以显示的内容很少，如果我们要做响应式页面，就要考虑用户体验，不能堆积内容。在很多移动网页上，用户会发现页面初始时只有标题栏，标题栏上除了窗口名称，还会提供类似“展开窗口”或“收起窗口”的功能按钮。与页面“显示”和“隐藏”效果不同的是，单击功能按钮时，窗口是慢慢向下或向上以滑动的方式展现给用户的，如图 1.4 和图 1.5 所示。

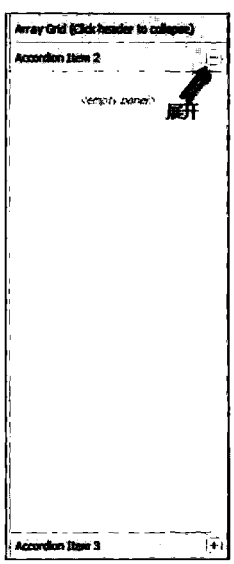

图 1.4 窗口“滑动”关闭状态

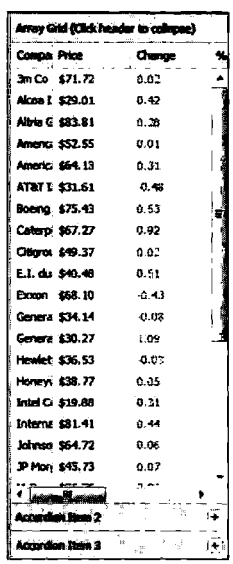

图 1.5 窗口“滑动”打开状态

本例主要代码如下：

```
01  /* slideUpDown.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $("#btn-slideup").click(function(){
05          $("#id-div-slide").slideUp("fast");
06          /* 省略部分代码，此处可以添加用户代码 */
07      });
08      $("#btn-sildedown").click(function(){
09          $("#id-div-slide").slideDown(1000);
10          /* 省略部分代码，此处可以添加用户代码 */
11      });
12  });
13  </script>
14  <body>
15      <h2 class="h2-caption">超实用的 jQuery 代码段 - jQuery 页面滑动操作</h2>
16      <hr><br>
17      <div id="id-div-slide">
18          <h3 class="h3-text">jQuery 效果 - 页面滑动操作</h3>
19          <h3 class="h3-text">jQuery slideUp()函数/slideDown()函数</h3>
20      </div>
21      <button id="btn-slideup">页面滑动隐藏</button>
22      <button id="btn-sildedown">页面滑动显示</button>
23      /* 省略部分代码，此处可以添加用户代码 */
24  </body>
```

第 04~07 行中的“滑动隐藏”效果通过 slideUp()函数实现，该函数使用滑动效果隐藏被选择元素，前提是该元素已处于显示状态。第 08~11 行中的“滑动显示”效果通过 slideDown()函数实现，它使用滑动效果显示隐藏着的被选择元素。

1.6 切换页面的滑动

上一个 jQuery 代码段向读者演示了通过 slideUp()和 slideDown()函数实现“滑动隐藏”和“滑动显示”页面元素的效果。其实，使用 slideToggle()函数也能实现同样的效果，如图 1.6 所示。

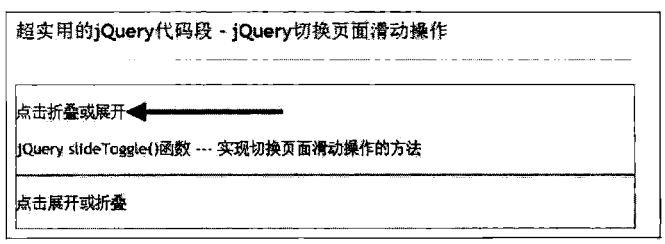

图 1.6 切换页面滑动操作效果

本例的具体代码如下：

```
/* slideToggle.html */
<script type="text/javascript">
$(document).ready(function(){
    $(".help h3").click(function(){
        $(this).next("h3").slideToggle("fast").siblings("p:visible").slideUp("fast");
        /* 省略部分代码，此处可以添加用户代码 */
    });
});
</script>
<body>
    <h2 class="h2-caption">超实用的 jQuery 代码段 - jQuery 切换页面滑动操作</h2>
    <hr><br>
    <div class="help">
        <div class="cssDiv">
            /* 省略部分代码，此处可以添加用户代码 */
        </div>
        <div class="cssDiv">
            /* 省略部分代码，此处可以添加用户代码 */
        </div>
    </div>
    /* 省略部分代码，此处可以添加用户代码 */
</body>
```

第 05 行中的 slideToggle()函数可以在 slideDown()函数与 slideUp()函数之间进行切换。如果元素向下滑动，则 slideToggle()函数可向上滑动它们；如果元素向上滑动，则 slideToggle()函数可向下滑动它们。

提示：对于 slideToggle()函数，如果元素已经是完全可见的，则该效果不产生任何变化，除非规定了 callback 函数。

1.7 图片的动画效果

用户如果在 Web 2.0 风格的网站上浏览过图片的话，经常会看到图片沿水平、垂直或自定义轨迹移动的动画效果。有些动画效果需要用户激活，有些则是自动开启。网站中加入这些动画效果大大提升了页面的交互性能。例如，WOW.js 网站首页加入的动画效果如图 1.7 所示。

本例代码如下：

```
/* animate.html */
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("div").animate({left:'350px'});
```

```
06      });
07  });
08  </script>
09  <body>
10      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现 Animate 动画</h2>
11      <hr><br>
12      <button>开始动画</button>
13      /* 省略部分代码，此处可以添加用户代码 */
14  <div
    style="background:#98bf21;height:60px;width:100px;position:absolute;text-align:center;">
15          /* 省略部分代码，此处可以添加用户代码 */
16      </div>
17      /* 省略部分代码，此处可以添加用户代码 */
18  </body>
```

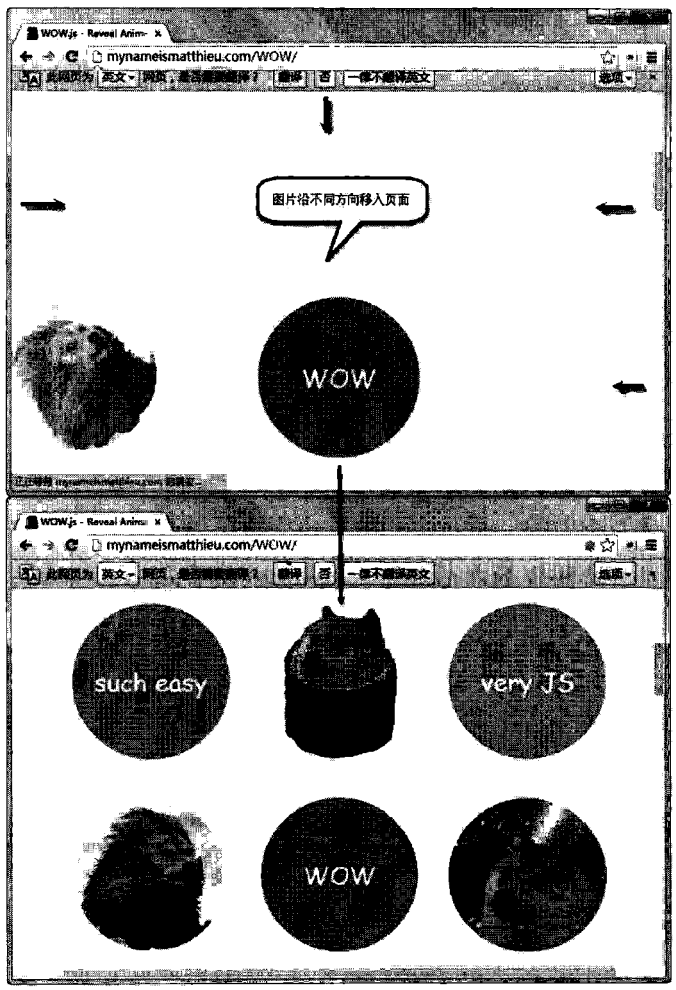

图 1.7 图片“移入”动画效果

“动画”效果通过第 05 行的 animate()函数实现，该函数执行 CSS 属性集的自定义动

画。具体来讲，就是通过 CSS 样式将元素从一个状态改变为另一个状态，且 CSS 属性值是逐渐改变的，这样就可以创建动画效果。animate()函数的写法如下：

```
$(selector).animate({params},speed,callback);
```

其中 params 参数用来定义形成动画的 CSS 属性；speed 参数用来规定效果的时长，可取值为 slow、fast 或毫秒；callback 参数用来设置动作完成后所执行的函数名称。

以上代码实现的“动画”演示效果如图 1.8 所示。

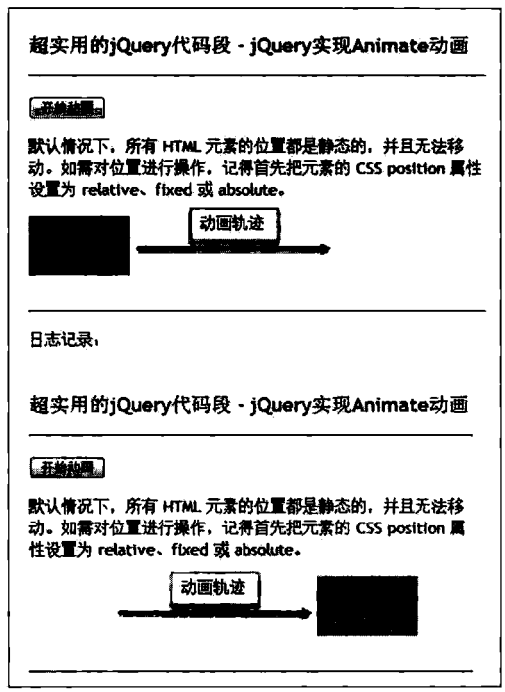

图 1.8 “动画”效果图

除了以上的基本动画功能，animate()函数还支持队列功能，这意味着如果用户需要编写多个 animate()调用，jQuery 框架会创建包含这些方法调用的“内部”队列，然后逐一运行这些 animate()调用。具体实现代码如下：

```
<script type="text/javascript">
$("button").click(function(){
    var div=$("div");
    div.animate({height:'350px',opacity:'0.6'},"slow");
    div.animate({width:'350px',opacity:'0.8'},"slow");
    div.animate({height:'150px',opacity:'0.6'},"slow");
    div.animate({width:'150px',opacity:'0.8'},"slow");
});
</script>
```

注意：“动画”函数需要修改 HTML 元素的位置属性。因为默认情况下，所有 HTML 元素的位置都是静态的，并且无法移动。如果需要对位置进行操作，要在页面 CSS 样式代码中把元素的 position 属性设置为 relative、fixed 或 absolute。

1.8 停止动画效果

动画效果可以反复实现，在动画执行过程中，可以添加停止动画的功能，如图1.9、图1.10和图1.11所示，我们可以在播放动画的过程中单击“停止滑动”按钮来停止动画。

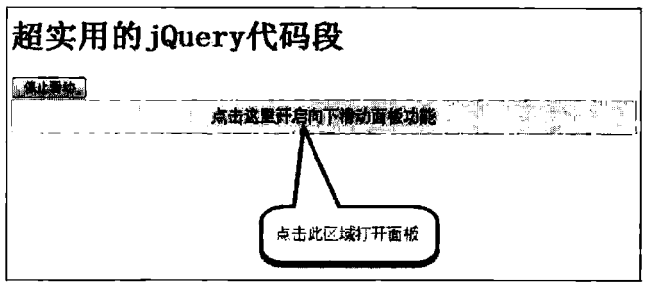

图1.9 “停止动画”操作（一）

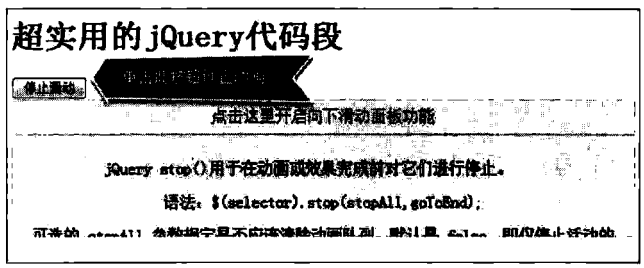

图1.10 “停止动画”效果图（二）

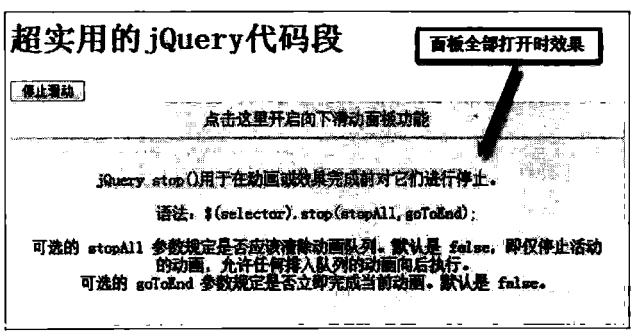

图1.11 “停止动画”效果图（三）

本例代码如下：

```
01  /* stop.html */
02  <script>
03  $(document).ready(function(){
04        $("#flip").click(function(){
05        $("#panel").slideDown(5000);
06  });
07  $("#stop").click(function(){
08        $("#panel").stop();
09        });
```

```
10  });
11  </script>
12  <body>
13      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现停止滑动</h2>
14      <hr><br>
15      <button id="stop">停止滑动</button>
16      <div id="flip">单击这里开启向下滑动面板功能</div>
17      <div id="panel">
18          <p>jQuery stop()用于在动画或效果完成前对它们进行停止。</p>
19      </div>
20      /* 省略部分代码，此处可以添加用户代码 */
21  </body>
```

“停止动画”效果通过第 08 行的 stop()函数来实现，其语法如下：

```
$(selector).stop(stopAll, goToEnd);
```

其中，stopAll 参数用来规定是否应该清除动画队列；goToEnd 参数用来规定是否立即完成当前动画。

注意：“停止动画”函数的 goToEnd 参数只能在设置了 stopAll 参数时才能使用。

1.9 不可不知的 Callback 回调

前面介绍了一些网页动态效果的实现方法，其实在这些动态效果执行完后，还可以激活一个新的函数来完成一些附加功能，此函数一般称其为回调函数（Callback）。本节将详细介绍 jQuery 框架中 Callback 回调函数的使用方法。本例演示效果如图 1.12 和图 1.13 所示。

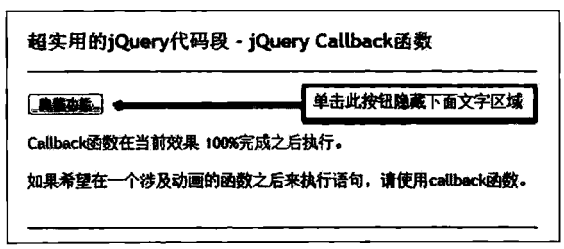

图 1.12 “Callback 回调函数”操作（一）

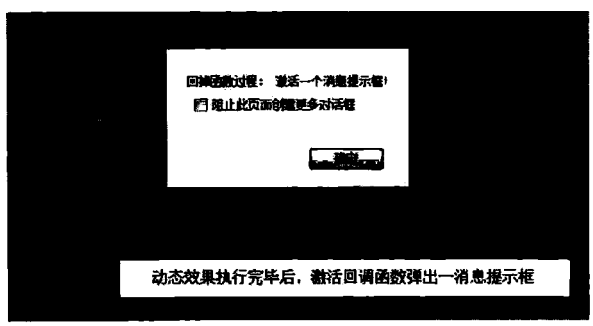

图 1.13 “Callback 回调函数”操作（二）

Callback 回调函数代码如下：

```
01  /* callback.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $("button").click(function(){
05          $("p").hide(1000,function(){                    //回调函数
06              alert("回调函数过程：激活一个消息提示框!");
07              /* 省略部分代码，此处可以添加用户代码 */
08          });
09      });
10  });
11  </script>
12  <body>
13      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery Callback 函数</h2>
14      <button type="button">隐藏功能</button>
15      <p>Callback 函数在当前效果 100%完成之后执行。</p>
16      <p>如果希望在一个涉及动画的函数之后来执行语句，请使用 callback 函数。</p>
17      /* 省略部分代码，此处可以添加用户代码 */
18  </body>
```

代码第 05~08 行实现的回调函数在当前动态效果 100%完成之后执行，本例中的回调函数中激活了一个消息提示框，用于提示用户注意事项。

1.10 提高效率的链式（Chaining）操作

如果用户阅读过 jQuery、Prototype 与 Dojo 等开源框架的话，一定会对其中大量使用的链式（Chaining）操作印象深刻。以前，设计人员都是一次写一条 jQuery 语句（一条接着另一条，使用分号隔开）。现在有一种名为链式（Chaining）操作的技术，允许设计人员在相同的元素上运行多条 jQuery 命令。jQuery 框架对链式操作有十分完整的实现，借助 jQuery 框架，设计人员可以把动作与方法像锁链一样链接起来，实现在相同的元素上使用一条语句操作多个 jQuery 函数的效果。本节将向用户详细介绍 jQuery 框架中链式操作的使用方法。

链式操作的具体代码如下：

```
01  /* chaining.html */
02  <script>
03  $(document).ready(function(){
04      $("button").click(function(){
05          $("#p1").css("color","red").slideUp(2000).slideDown(2000);      //Chaining 操作
06          /* 省略部分代码，此处可以添加用户代码 */
07      });
08  });
09  </script>
```

```
10  <body>
11      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 链式（Chaining）操作</h2>
12      <hr><br>
13      <p id="p1">通过 jQuery 您可以把动作/方法链接起来。<br>Chaining 允许我们在一条语句
14  中运行多个 jQuery 函数（在相同的元素上）。</p><br>
15      <button>请用户单击这里</button>
16      /* 省略部分代码，此处可以添加用户代码 */
17  </body>
```

jQuery 链式操作允许在相同的元素上使用一条语句操作多个 jQuery 函数，其写法见第 05 行。本例就是将 slideUp()与 slideDown()函数写在一条语句中执行，演示效果如图 1.14、图 1.15 和图 1.16 所示。

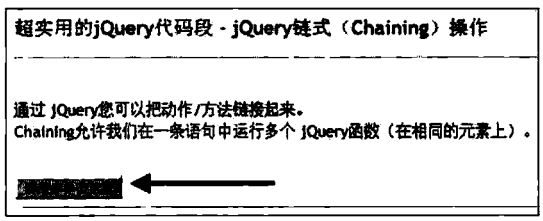

图 1.14　jQuery 的链式操作（一）

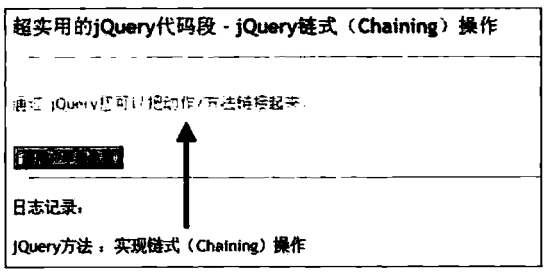

图 1.15　jQuery 的链式操作（二）

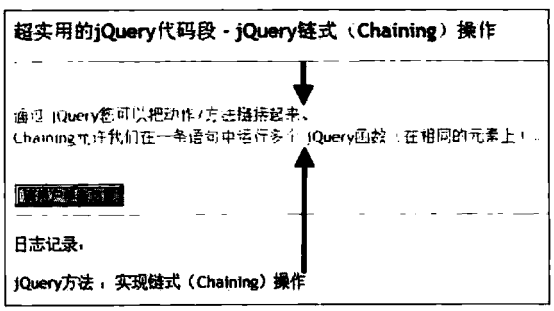

图 1.16　jQuery 的链式操作（三）

注意：过长的链式语法可能导致代码过于冗长，难以理解，因此应该对链式操作保持合理的长度，避免影响代码的可读性。

注意：链式操作会自动抛掉函数后多余的空格，并按照一行长代码来执行语句。使用链式操作，浏览器就不必多次查找相同的元素。如需链接一个动作，设计人员只需简单地把该动作追加到之前的动作上即可。

1.11 在新窗口中打开链接

经常看新闻网页的读者一定知道，很多时候我们单击一个链接打开新闻时，都会在一个新窗口中打开新闻的详细页面。用技术一点的术语来说，就是在新窗口中打开用户操作的链接。使用 HTML 中 a 标签的 target="_blank"属性可以在新窗口打开链接，而在目前流行的 jQuery 框架下，此功能还可以有几种不同的形式来实现。本例效果如图 1.17 所示。

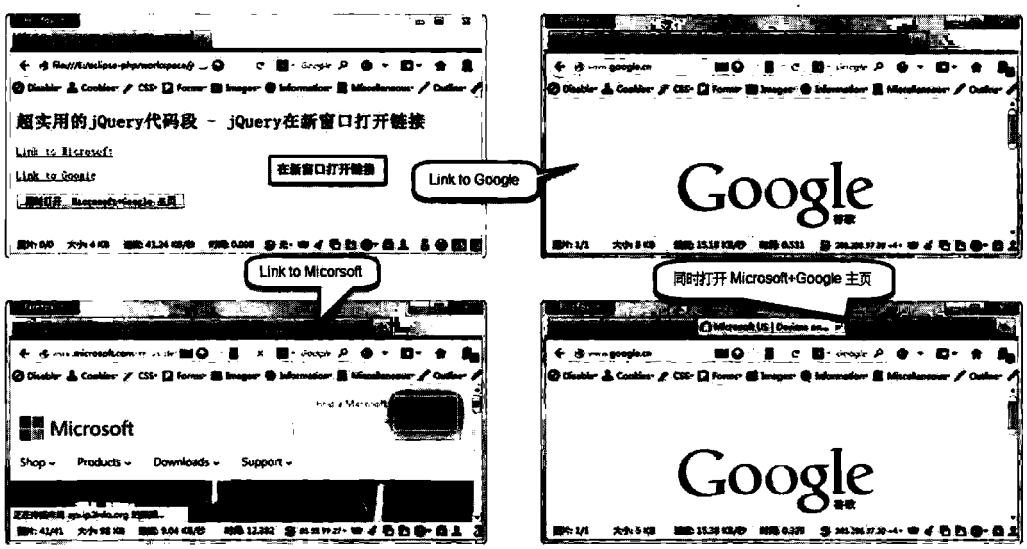

图 1.17 在新窗口中打开链接的演示效果

本例主要代码如下：

```
01  /* openurl.html */
02  <script>
03  $(document).ready(function(){
04      $('.jqopenurl').attr({target:"_blank"}); //使用.attr()函数，设置 target 属性值为"_blank"
05      $('a[rel=ext]').click(function(){        //通过设置 a 的 rel 属性值为 ext 方式激活链接
06          $(this).attr('target','_blank');     //当单击链接时添加 target=_blank 属性值
07      });
08  });
09  function open_url(){
10      window.open("http://www.microsoft.com/");
11      window.open("http://www.google.cn/");
12  }
13  </script>
14  <body>
15      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 在新窗口打开链接</h2>
16      <p><a href="http://www.microsoft.com/" class="jqopenurl" title="title">Link to
17  Microsoft</a></p>
18      <p><a href="http://www.google.cn/" rel="ext" title="title">Link to Google</a></p>
```

```
19        <form>
20            <input type="button" value="同时打开  Microsoft+Google 主页"
21    onclick="open_url()"/>
22        </form>
23        /* 省略部分代码，此处可以添加用户代码 */
24    </body>
```

本例代码分别演示了“在新窗口中打开链接”的几种方法，具体如下。

（1）第1种方法，见代码第04行，使用jQuery框架的.attr()函数。

jQuery框架属性操作attr()函数用于设置或返回被选择元素的属性值，根据该函数不同的参数，其工作方式也有所差异，其语法形式如下：

```
$(selector).attr(attribute);
```

（2）第2种方法，见代码第05~07行，通过给链接指定rel=ext属性，自动在新窗口中打开链接，具体代码分析如下：

- $('a[rel=ext]').click()函数用于获取HTML文档中定义的rel属性值为ext的<a>元素，然后相应click单击事件并完成后续处理。
- $(this).attr()函数通过设定target属性值为'_blank'来实现在新窗口中打开链接，此处与上面第1种方法相同。

（3）第3种方法，见代码第10~11行。window.open()函数用于打开一个新的浏览器窗口或查找一个已命名的窗口，其语法如下：

```
window.open(URL,name,features,replace);
```

其中URL参数声明了要在新窗口中显示的文档链接地址；name参数声明新窗口的名称；features参数声明新窗口要显示的标准浏览器的特征；最后replace参数定义为一个布尔值，用于判断装载到窗口的URL是在窗口的浏览历史中创建一个新条目，还是替换浏览历史中的当前条目。

设计人员需要注意以下两点：

- 在XHTML 1.0 Strict版本中并不支持target="_blank"属性，而使用jQuery框架可以很好地解决这个问题。
- 设计编码时不要混淆函数window.open()与document.open()，这两者的功能完全不同。这里有个小技巧，为了使代码清晰明了，可以使用window.open()函数替代open()函数。

1.12 强制在弹出窗口中打开链接

上一个jQuery代码段介绍了在新窗口中打开链接的方法，本例将讲解如何在弹出窗口中打开链接。本例效果如图1.18与图1.19所示。

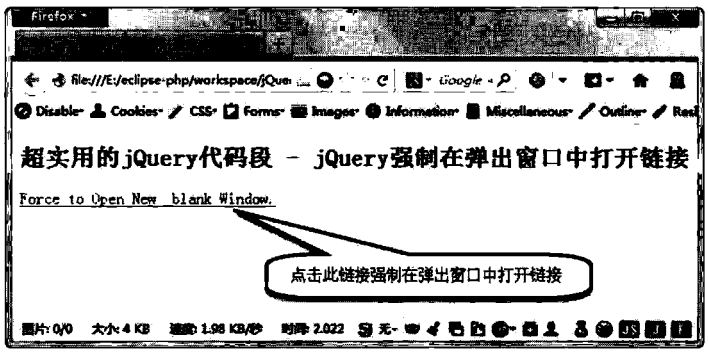

图 1.18 “强制在弹出窗口中打开链接”的演示效果（一）

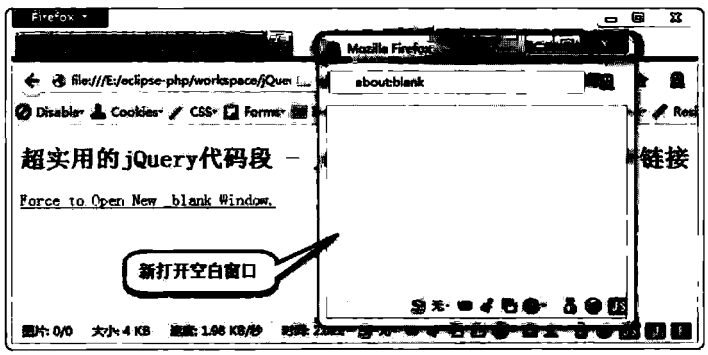

图 1.19 “强制在弹出窗口中打开链接”的演示效果（二）

本例主要代码如下：

```
01  /* forceopenurl.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $("a.popup").live("click",function(){
05          var forcenewwindow=window.open("","width=300,height=200");
06          if(window.focus){
07              forcenewwindow.focus();//让页面成为当前窗体
08          }
09          return false;
10      });
11  });
12  </script>
13  <body>
14      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 强制在弹出窗口中打开链接</h2>
15      <p><a href="#" class="popup" title="title">Force to Open New _blank Window.</a></p>
16      /* 省略部分代码，此处可以添加用户代码 */
17  </body>
```

第 04~10 行中的 live()函数用于为被选择元素附加一个或多个事件处理程序，并规定当这些事件发生时需要运行的函数，其语法如下：

```
$(selector).live(event,data,function);
```

其中 event 参数用来规定附加到元素的一个或多个事件；data 参数用来规定传递到该函数的额外数据；function 参数用来规定当事件发生时运行的函数。

注意： 由 jQuery 框架的 live()函数附加的事件处理程序，适用于匹配选择器的当前及未来的元素（如由脚本创建的新元素）。

1.13 平滑滚动页面到某个锚点

如果用户在使用浏览器时仔细观察的话，会发现 Microsoft 的 IE 浏览器打开时默认是平滑滚动页面的。网页滚动从开始到结束，中途有一个过渡时间，这样使得页面可以显示更多的内容细节。从用户体验上来讲，在网页上下滚动时开启平滑滚动，比禁用时效果更平滑舒服。本例效果如图 1.20、图 1.21 和图 1.22 所示。

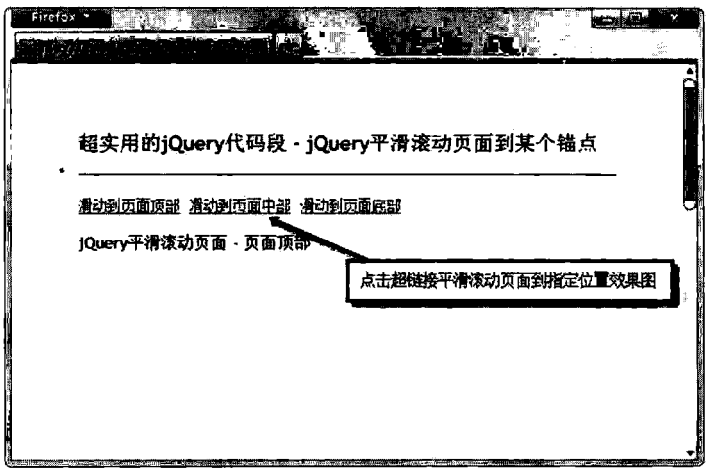

图 1.20 “平滑滚动页面到某个锚点”的演示效果（一）

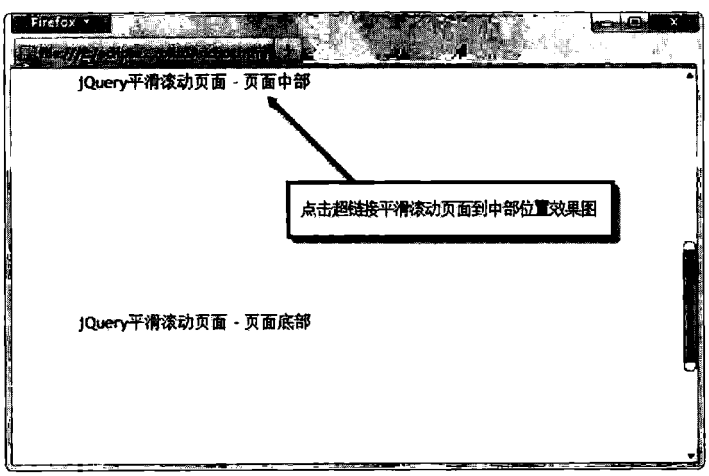

图 1.21 “平滑滚动页面到某个锚点”的演示效果（二）

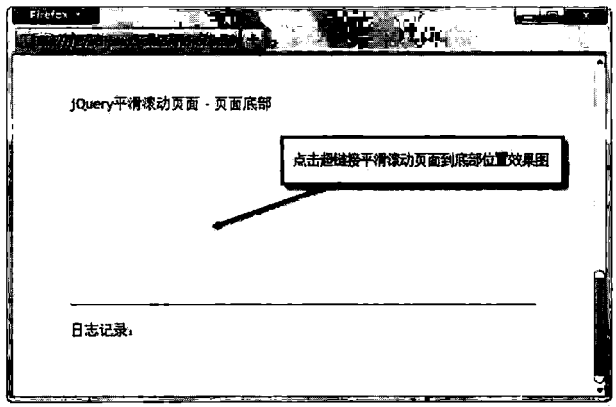

图 1.22 “平滑滚动页面到某个锚点”的演示效果（三）

本例主要代码如下：

```
01  /* scrolltoAnchor.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      jQuery.scrollTo=function(scrolldom,scrolltime){
05          $(scrolldom).click(function(){
06              var scrollToDom = $(this).attr("page-scroll");
07              $(this).addClass("thisscroll").siblings().removeClass("thisscroll");
08              $('html,body').animate(
09                  {
10                      scrollTop:$(scrollToDom).offset().top
11                  },
12                  scrolltime
13              );
14              return false;
15          });
16      };
17      $.scrollTo("#scrollnav a",1000);
18  });
19  </script>
20  <body>
21      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 平滑滚动页面到某个锚点</h2>
22      <hr><br>
23      <div class="scrollnav" id="scrollnav">
24          <a href="#" page-scroll="#scrolltoTop">滑动到页面顶部</a>
25          <a href="#" page-scroll="#scrolltoMid">滑动到页面中部</a>
26          <a href="#" page-scroll="#scrolltoEnd">滑动到页面底部</a>
27      </div>
28      /* 省略部分代码，此处可以添加用户代码 */
29  </body>
```

本例通过自定义函数 scrollTo()实现平滑滚动页面到锚点的功能，第 08~13 行使用 animate()函数实现 HTM 页面元素的“滑动”效果。

在 animate()函数内，利用 scrollTop()函数把内容滚动到指定的坐标，其语法如下：

```
$(selector).scrollTop(offset);
```

其中参数 offset 为可选项，用来规定相对滚动条顶部的偏移，以像素为单位。

提示：本例实现的“平滑滚动页面到某个锚点”效果实际上是通过.animate()函数与.offset()函数组合来实现的。如今，这种方法已经替代传统 HTML 操作 DOM 的 scrollTo()函数了。

1.14 阻止文本行换行

从用户体验上来讲，阻止文本不必要的自动换行可以使得浏览效果更美观、更节省页面空间，其实用价值是显而易见的。本例效果如图 1.23 所示。

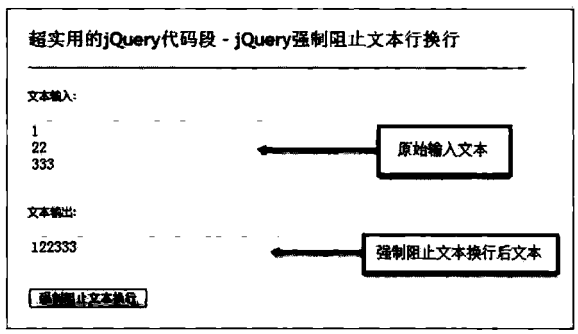

图 1.23 jQuery “强制阻止文本行换行” 演示效果

本例代码如下：

```
/* forbiddenbreak.html */
<script type="text/javascript">
function strReplace(str){
    var t="",len=0;
    t=str.replace(/\r\n/g,"\\n").replace(/\n/g,"");
    document.form.output.value=t;
}
</script>
<body>
    <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 强制阻止文本行换行</h2>
    <hr><br>
    <form id="form" name="form" method="post" action="#">
        <label>文本输入:</label><br>
        <p>
            <textarea name="input" cols="30" rows="3" id="input"></textarea><br>
        </p>
        <label>文本输出:</label><br>
        <p>
            <textarea name="output" cols="30" rows="1" id="output"
readonly="readonly"></textarea><br>
```

```
21              </p>
22              <p>
23                  <label>
24                      <input type="button" name="Submit" value="强制阻止文本换行"
25  onclick="strReplace(input.value)">
26                  </label>
27              </p>
28          </form>
29          /* 省略部分代码，此处可以添加用户代码 */
30  </body>
```

本例使用正则表达式消除文本换行字符，并借助jQuery框架的字符替换函数进行操作，关键代码是第05行，其中的replace()函数表示在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子串，其语法如下：

```
stringObject.replace(regexp/substr,replacement);
```

其中，regexp/substr参数用来规定子字符串或要替换的模式的RegExp对象；replacement参数用来定义一个字符串值，规定了替换文本或生成替换文本的函数。

提示：使用 replace()函数时，如果第一个参数值是一个字符串，则将其作为要检索的直接变量文本模式，而不是首先被转换为RegExp对象。

1.15 实现iframe高度自适应

资深的网页设计人员对iframe控件一定不会陌生，虽然它已渐渐退出历史舞台，但对于某些特定的项目还不得不去使用它。在使用iframe控件过程中会遇到高度的问题，原因是由于被嵌套的页面长度不固定而迫使iframe控件显示出滚动条。这样不仅影响页面美观，还会给用户操作带来不便，于是自动调整iframe的高度就是必须要解决的问题。本例效果如图1.24所示。

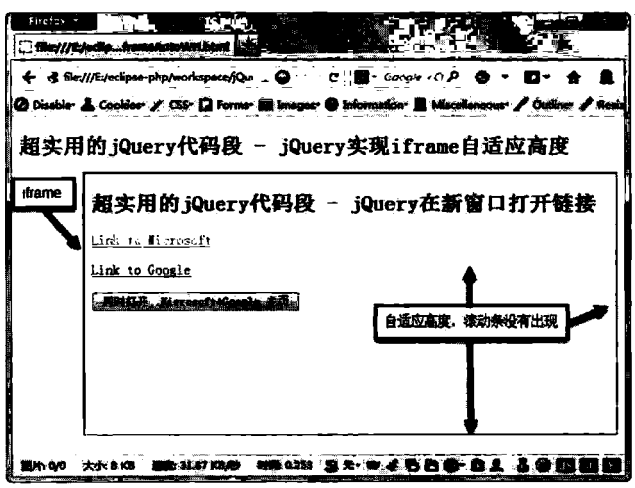

图1.24 实现iframe高度自适应

下面看看如何借助 jQuery 框架实现 iframe 高度自适应的效果：

```
01  /* iframeAutoWH.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $("#iframe-content").load(function(){
05          var vHeight = $(this).contents().find("body").height()+32;
06          $(this).height(vHeight<300?300:vHeight);
07      });
08  });
09  </script>
10  <body>
11      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现 iframe 自适应高度</h2>
12      <hr><br>
13      <div>
14          <iframe id="iframe-content" src="ch01.openurl.html" frameborder="3"
15  scrolling="no"
16  style="margin-left:2px;margin-top:-3px;width:600px;height:350px;float:right;"></iframe>
17      </div>
18  </body>
```

本例的关键首先是获取页面 body 元素的高度，然后判断该高度是否小于 300，最终设定 iframe 的自适应高度值，具体代码参考第 05~06 行。

第 06 行的 height()函数用于返回或设置匹配元素的高度，其语法如下：

```
$(selector).height(length);
```

提示：使用 height()函数时，如果没有规定长度单位，则使用默认的 px 单位。

1.16 实现左右 div 自适应相同高度

前端网页设计中经常会出现这样一个问题：页面左侧导航只有几个链接，而且很短，页面右侧的正文部分却可能会很长，这导致页面布局很难看、不协调。怎么才能让页面左侧导航高度能随着页面右侧正文高度的增加而自动调整呢？解决方案就是让页面左右两侧的 div 高度能自动调整且保持一致。本例演示效果如图 1.25 所示。

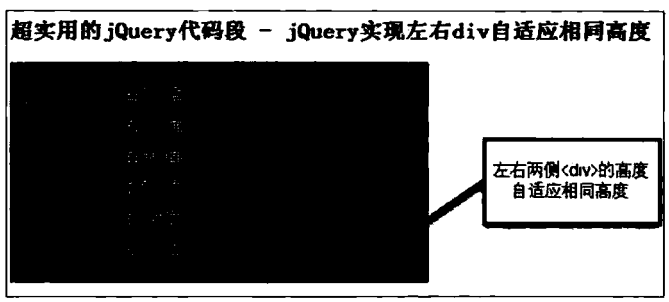

图 1.25 实现左右 div 自适应

本例主要代码如下：

```
/* divAutoLR.html */
</head>
<body>
    <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现左右 div 自适应相同高度
</h2>
    <hr><br>
    <div id="left">
        <div style="padding:10px">
            <p>左侧页面</p>
        </div>
    </div>
    <div id="right">
        <div style="padding:10px">
            <p>右侧页面</p>
            <p>右侧页面</p>
            <p>右侧页面</p>
            <p>右侧页面</p>
            <p>右侧页面</p>
            <p>右侧页面</p>
        </div>
    </div>
<script type="text/javascript">
function $(id){
    return document.getElementById(id);
}
function autoHeight(){
    if($("left").offsetHeight>=$("right").offsetHeight){        //判断左侧 div 高度是否大于右侧 div
        $("right").style.height=$("left").offsetHeight + "px"; //比较结果为真，调整右侧 div
    }else{
        $("left").style.height=$("right").offsetHeight + "px"; //否则，调整左侧 div
    }
}
window.onload = function(){
    autoHeight();
};
</script>
    <div style="clear:both">
    </div>
    <br><hr>
    <div id="div-log">
        <p>日志记录：</p>
    </div>
</body>
```

本例关键代码是第 26~30 行。offsetHeight 属性用于获取对象相对于版面的高度或由父坐标属性指定的高度，针对不同的浏览器可以理解如下：

- IE、Opera 浏览器定义的 offsetHeight 属性是网页内容高度+滚动条+边框的总和。
- Firefox 浏览器定义的 offsetHeight 属性是网页内容实际高度。

提示：以上代码是基于 DTD HTML 4.01 Transitional 标准实现的。如果是 DTD XHTML 1.0 Transitional 标准则意义又会不同，在 XHTML 中 offsetHeight 属性表示内容的实际高度。另外，新版本的浏览器大多支持根据页面指定的 DOCTYPE 来启用不同的解释器。

1.17 获取鼠标在屏幕中的坐标

相信绝大多数 Windows 用户都使用过一款名为“按键精灵”的桌面软件，该软件有一个“抓图抓色”的功能，是在用户抓图过程中将当前屏幕鼠标指针所在的像素点的颜色和 *X/Y* 轴的坐标显示出来，效果既实用又有趣。在前端网页设计中，也经常要获取当前鼠标的位置来完成一些特定效果，借助 jQuery 能方便地获取鼠标在屏幕中的坐标。本例效果如图 1.26 所示。

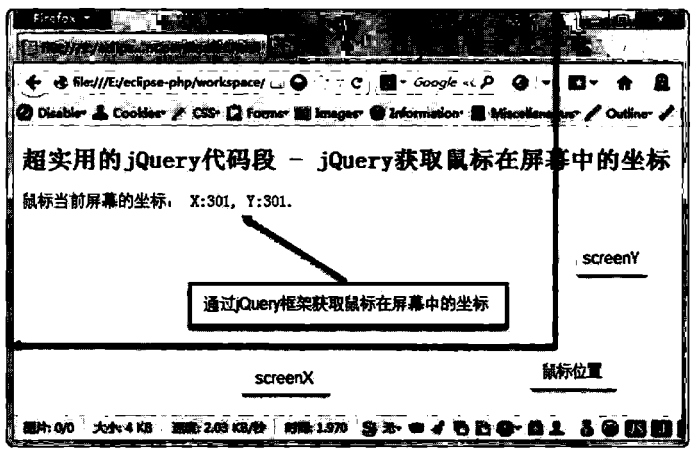

图 1.26 获取鼠标在屏幕中的坐标

本例主要代码如下：

```
01  /* getScreenCoordinates.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $(document).mousemove(function(e){
05          getScreenCoordinates(e);          //调用 getScreenCoordinates()函数
06      });
07  });
08  function getScreenCoordinates(e){
09      x=e.screenX;                          //获取屏幕 X 坐标
10      y=e.screenY;                          //获取屏幕 Y 坐标
```

```
11        $("span").text("X:"+x+",Y:"+y);              //将坐标值输出到屏幕中
12    }
13    </script>
14    <body>
15        <h2>超实用的jQuery代码段 - jQuery获取鼠标在屏幕中的坐标</h2>
16        <p>鼠标当前屏幕的坐标： <span></span>.</p>
17    </body>
```

本例关键代码是第08~12行。其中，screenX与screenY事件属性可返回事件发生时鼠标指针相对于屏幕的水平与垂直坐标位置。

提示： screenX事件属性是鼠标位置相对于用户屏幕的水平偏移量，而screenY事件属性是垂直方向的偏移量。此时的参照点（原点）是屏幕的左上角，而屏幕偏移量与窗口客户区偏移量概念是不同的，后面的实例会做进一步讲解。

1.18 获取鼠标在窗口客户区中的坐标

屏幕中的坐标与窗口客户区中的坐标概念是不同的，本例借助jQuery框架来获取鼠标在窗口客户区中的坐标，效果如图1.27所示。

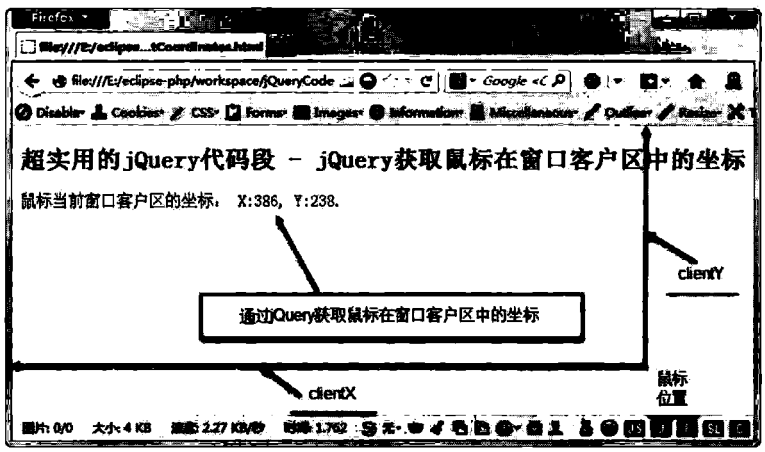

图1.27 获取鼠标在窗口客户区中的坐标

本例主要代码如下：

```
01  /* getClientCoordinates.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $(document).mousemove(function(e){
05          getClientCoordinates(e);              //调用 getClientCoordinates()函数
06      });
07  });
08  function getClientCoordinates(e){
09      x=e.clientX;                  //获取窗口客户区中的X坐标
```

```
10        y=e.clientY;                    //获取窗口客户区中的 Y 坐标
11        $("span").text("X:"+x+", Y:"+y);          //将坐标值输出到屏幕中
12    }
13    </script>
14    <body>
15        <h2>超实用的 jQuery 代码段 - jQuery 获取鼠标在窗口客户区中的坐标</h2>
16        <p>鼠标当前窗口客户区的坐标： <span></span>.</p>
17    </body>
```

关键代码是第 08~12 行。其中，clientX 与 clientY 事件属性返回当事件被触发时鼠标指针相对于浏览器页面（或客户区）的水平坐标与垂直坐标位置，客户区指的是当前窗口。

提示：clientX 事件属性是鼠标位置相对于用户窗口客户区的水平偏移量，而 clientY 是垂直方向的偏移量，此时的参照点（原点）是窗口客户区的左上角，因此该参照点会随滚动条的移动而移动。窗口客户区与页面区域偏移量概念不同，后面将会做进一步讲解。

1.19 获取鼠标在窗口页面中的坐标

窗口客户区中的坐标与窗口页面中的坐标，两个概念也是有区别的。本例借助 jQuery 框架获取鼠标在窗口页面中的坐标，效果如图 1.28 所示。

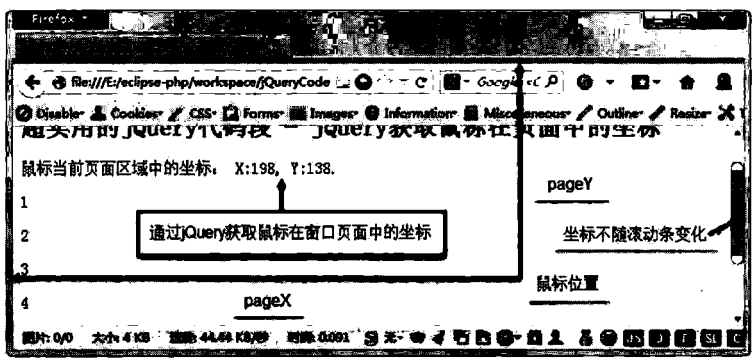

图 1.28 获取鼠标在窗口页面中的坐标

本例主要代码如下：

```
01  /* getPageCoordinates.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $(document).mousemove(function(e){
05          getPageCoordinates(e);          //调用 getPageCoordinates()函数
06      });
07  });
08  function getPageCoordinates(e){
09      x=e.pageX;                  //获取页面区域中的 X 坐标
10      y=e.pageY;                  //获取页面区域中的 Y 坐标
```

```
11        $("span").text("X:"+x+", Y:"+y);          //将坐标值输出到屏幕中
12    }
13    </script>
14    <body>
15        <h2>超实用的jQuery代码段 - jQuery获取鼠标在页面中的坐标</h2>
16        <p>鼠标当前页面区域中的坐标： <span></span>.</p>
17    </body>
```

本例关键代码是第08~12行。其中，pageX与pageY事件属性返回当事件被触发时鼠标指针相对于文档左边缘的位置。

提示：pageX事件属性是鼠标位置相对于用户页面区域的水平偏移量，而pageY事件属性是垂直方向的偏移量，此时的参照点（原点）是窗口客户区的左上角，因此该参照点不会随滚动条的移动而移动。

1.20 设置Flash对象的WMode窗口模式

熟悉Flash技术的用户一定对WMode模式有所了解，它用来在Flash对象上设置不同的窗体模式。那么如何在Flash对象运行时改变WMode呢？默认情况下，所有嵌入式Flash对象在得到WMode窗口时将会阻止所有其他的HTML元素，所以是不可能在服务器端改变其属性的，而在客户端通过JavaScript技术改变却是可行的。本例是借助jQuery框架实现改变WMode窗口模式的功能。

本例代码如下：

```
01  /* setVMode.html */
02  <script type="text/javascript">
03  //FireFox浏览器兼容模式
04  $j("embed").attr("wmode","opaque");
05  $j(document).ready(function(){
06  //IE浏览器兼容模式
07  var embedTag;
08  $j("embed").each(function(i){
09      embedTag=$j(this).attr("outerHTML");
10      if((embedTag!=null)&&(embedTag.length>0)){
11          embedTag=embedTag.replace(/embed /gi, "embed wmode=\"opaque\" ");
12          $j(this).attr("outerHTML",embedTag);
13      }else{
14          $j(this).wrap("<div></div>");
15      }
16  });
17  });
18  </script>
19  <body>
20      <h2>超实用的jQuery代码段 - jQuery设置Flash对象WMode窗口模式</h2>
```

```
21        <embed src="*.swf" wmode="transparent" quality="high"></embed>
22    </body>
```

本例关键代码见第 08~16 行，关于 WMode 窗口模式的参数说明见表 1.1。

表 1.1　WMode窗口模式参数说明

参数名称	参数说明
window模式	是默认情况下的显示模式。在该模式下Flash Player有自己的窗口句柄，这就意味着Flash影片是Windows系统中的一个运行实例，并且是在浏览器核心窗口之上显示的。所以，该模式也是Flash最快最有效率的渲染模式
opaque模式	这是一种无窗口模式。在该模式下Flash Player没有自己的窗口句柄，这就需要浏览器告诉Flash Player在浏览器的渲染表面绘制的时间和位置。这时Flash影片就不会在高于浏览器HTML渲染表面，而是与其他元素一样在同一个页面上。因此，在该模式下就可以使用z-index参数值来控制HTML元素是遮盖Flash影片或者是被遮盖
transparent模式	这是一种透明模式。在该模式下Flash Player会将stage的背景色alpha值设定为0，并且只会绘制stage上真实可见的对象。同样，在该模式下用户也可以使用z-index来控制Flash影片的深度值。但是，使用Transparent模式会降低Flash影片的回放效果

注意：在 Web 开发中可能会遇到 Flash 影片遮挡页面元素的情况，无论怎么设置 Flash 容器和层的深度（z-index）也无济于事，现有的解决方案是在插入 Flash 影片的 embed 或 object 标签中，加入 wmode 属性并设置 wmode 为 transparent 或 opaque。

1.21　实现类 Twitter 的字数限制效果

最早在 Twitter 网站上实现的 TextArea 字数限制效果让用户大为惊艳，原来土里土气的 TextArea 控件还可以变得这么有趣！从 Twitter 之后，几乎所有的论坛、社交、微博网站都是按照此模式实现的 TextArea 控件。类 Twitter 风格的文本字数限制效果如图 1.29 所示。

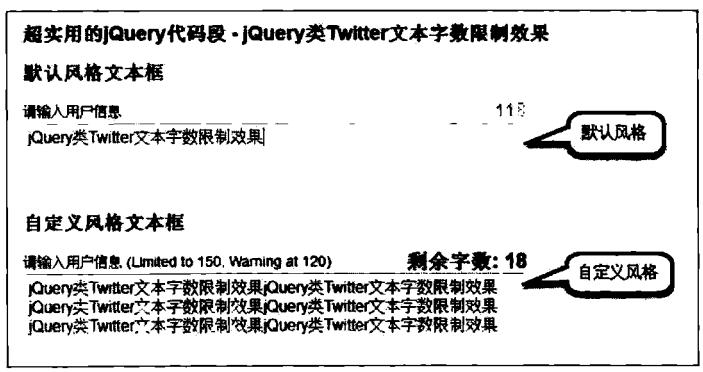

图 1.29　类 Twitter 风格的文本字数限制效果

本例主要代码如下：

```
01  /* likeTwitterTextArea.html */
02  <script src="../plugin.js.com/charCount.js"></script>
03  <script type="text/javascript">
```

```
04  $(document).ready(function(){
05      $("#id-textarea-default").charCount();              //默认使用方法
06      $("#id-textarea-custom").charCount({                //自定义使用方法
07          allowed:150,
08          warning:120,
09          counterText:'剩余字数:'
10      });
11  });
12  </script>
13  <body>
14      <form id="form" method="post" action="">
15          <h2>默认风格文本框</h2>
16          <div>
17              <label for="message">请输入用户信息</label>
18              <textarea id="id-textarea-default" name="id-textarea-default"></textarea>
19          </div>
20          <h2>自定义风格文本框</h2>
21          <div>
22          <label for="message">请输入用户信息 (Limited to 150, Warning at 120)</label>
23              <textarea id="id-textarea-custom" name="id-textarea-custom"></textarea>
24          </div>
25      </form>
26  </body>
```

关键代码是第 05~10 行，这里通过 charCount 插件实现了文本字数的限制效果。关 charCount 的属性参数见表 1.2。

表 1.2 charCount插件属性参数说明

参数名称	参数说明
allowed	该参数用于设定文本字数限制值，默认值为140
warning	该参数用于设定文本字数限制警告值
counterText	该参数用于设定文本字数限制提示文本

提示：charCount 插件的 warning 属性描述的是，当用户可输入字数剩余到该参数指定的值时提示警告，而不是已经输入的字数值。

1.22 提示文本的隐藏与显示

网页搜索功能已经成了所有网站的标配，几乎每一件不确定、待确认的事情都离不开搜索引擎的帮助。不知道用户有没有特别留意过搜索框的风格样式，大多数的搜索框内会留有一个默认的灰色提示文本，当搜索框获得焦点时，提示文本会自动隐藏。而当鼠标焦点离开时又会恢复默认提示文本。本例效果如图 1.30 与图 1.31 所示。

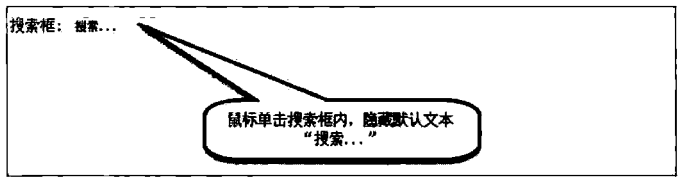

图 1.30　搜索文本框未获得焦点

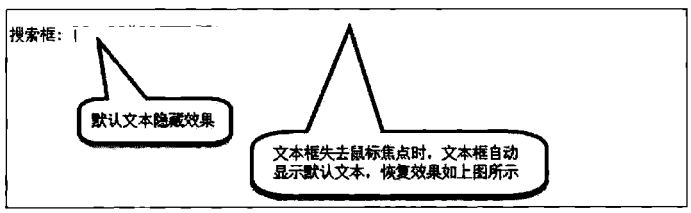

图 1.31　搜索文本框获得焦点后

本例主要代码如下：

```
01  /* textShowHide.html */
02  <script>
03  $(document).ready(function(){
04      var vSearchInput=$("#id-txtSearch");          //获取控件 id
05      vSearchInput.focus(function(){                //focus 事件处理程序
06          if(vSearchInput.val()==this.title){       //判断搜索框内文本是否与其 title 属性值相等
07              vSearchInput.val("");                 //实现隐藏搜索框内文本效果
08          }
09      });
10      vSearchInput.blur(function(){                 //blur 事件处理程序
11          if(vSearchInput.val()==""){               //判断搜索框内文本是否为空
12              vSearchInput.val(this.title);         //实现显示搜索框内文本效果
13          }
14      });
15      vSearchInput.blur();
16  });
17  </script>
18  <body>
19  <h2>超实用的 jQuery 代码段 - jQuery 鼠标单击实现隐藏与显示文本</h2>
20  <form>
21      <label>搜索框:</label>
22      <input id="id-txtSearch" type="text" value="搜索..." title="搜索..." />
23  </form>
24  </body>
```

本例的关键代码是第 05~15 行。其中，focus()函数与 blur()函数分别实现了文本的显示与隐藏。

第 05~09 行中的 focus()函数描述当元素获得焦点时发生的事件，其语法如下：

```
$(selector).focus(function);
```

其中，参数 function 为可选项，规定当发生 focus 事件时运行的函数。

第 10~15 行中的 blur()函数描述当元素失去焦点时发生的事件，其语法如下：

```
$(selector).blur(function);
```

其中，参数 function 为可选项，规定当发生 blur 事件时运行的函数。

需要用户特别注意的有以下 3 点：

- 当鼠标单击选中元素或通过 Tab 键定位到元素时，该元素就会获得焦点。
- blur()无参函数同样可以触发 blur 事件；如果设置了 function 参数，该函数也可规定当发生 blur 事件时执行的代码。
- 早期浏览器中的 blur 事件仅发生于表单元素上，而当今流行的浏览器中，该事件可用于任何 HTML 元素。

1.23 实现文字闪烁效果

相信大多数人都做过 PPT 演示文稿，无论是毕业论文答辩、企业产品介绍、学术报告还是其他演讲，基本都离不开它。PPT 中的文本有各种各样的特效，例如文本闪烁就十分实用，展示出来的效果绚丽多彩，十分吸引观众眼球。网页上也有文字闪烁功能，起初设计人员都是通过 GIF 图片来实现，但随着 JavaScript 技术的兴起，使用 JS 脚本来实现文字闪烁效果逐渐成为主流。本例效果如图 1.32、图 1.33 和图 1.34 所示。

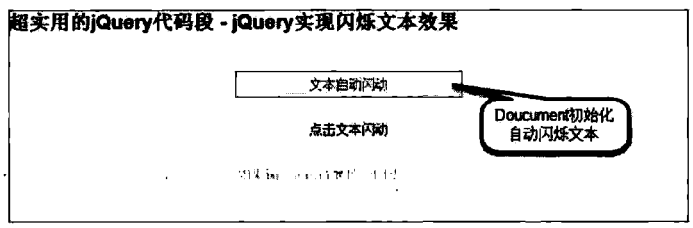

图 1.32 闪烁效果（一）

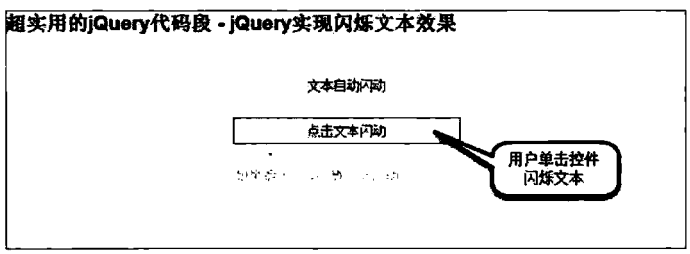

图 1.33 闪烁效果（二）

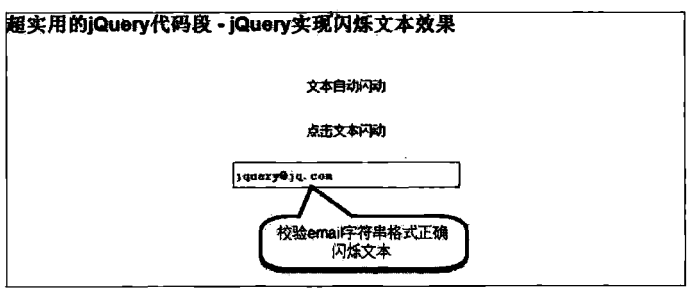

图 1.34 闪烁效果（三）

本例代码如下：

```
/* textFlash.html */
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<script src="../jquery/jquery-1.9.1.min.js"></script>
<link rel="stylesheet" href="css/demos.css">
<script>
$(document).ready(function(){
    function textFlash(obj/*jQuery Object*/,cssName/*CSS 样式名称*/,times/*闪烁次数*/){
        var i=0,t=false,o=obj.attr("class")+" ",c="",times=times||2;
        if(t) return;
        t=setInterval(function(){                    //实现闪烁功能
            i++;
            c=i%2?o+cssName:o;
            obj.attr("class",c);
            if(i==2*times){
                clearInterval(t);                    //清除闪烁功能
                obj.removeClass(cssName);
            }
        },200);
    };
    $(function(){
        //自动闪烁文本
        textFlash($("#id-div-autoFlash"),"red",3);
        //单击闪动文本
        $("#id-div-clickFlash").bind({
            click:function(){
                textFlash($(this),"red",3);
                return false;
                }
            });
        //通过 E-mail 格式校验闪动文本
        $("#id-input-emailFlash").blur(
            function(){
                //使用正则表达式校验 E-mail 格式字符串
                if(/^\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*/.test($(this).val())){
                    textFlash($(this),"red",3);
                }
            }
        );
    });
});
</script>
</head>
```

```
<body>
    <h2>超实用的 jQuery 代码段 - jQuery 实现闪烁文本效果</h2>
    <div class="main">
        <div id="id-div-autoFlash" class="cssFlash">文本自动闪动</div>
        <div id="id-div-clickFlash" class="cssFlash">单击文本闪动</div>
        <input class="cssFlash" type="email" id="id-input-emailFlash" placeholder="如果输
入 email 地址会闪动" />
    </div>
</body>
</html>
```

本例关键代码如下：

```
setInterval(function(){
  //省略部分代码
  clearInterval(time),
  1000,
);
```

以上代码中的 setInterval()函数可按照指定的周期（以毫秒计）来调用函数或计算表达式，其语法如下：

```
setInterval(code,millisec[,"lang"]);
```

其中 code 参数定义调用的函数或要执行的代码段；millisec 参数定义周期性执行的调用函数或要执行的代码段之间的以毫秒计的时间间隔。

而 clearInterval()函数可取消由 setInterval()函数设置的周期性调用的时间间隔，其语法如下：

```
clearInterval(id_of_setinterval);
```

其中参数 id_of_setinterval 为必选项，表示由 setInterval()函数返回的 id 值。

设计人员要特别注意以下两点：

- setInterval()函数会不停地调用函数，直到 clearInterval()函数被调用或窗口被关闭。
- clearInterval()函数的参数必须是由 setInterval()函数返回的 id 值。

1.24 实现文字动画效果

上一个例子是使用脚本技术来实现文本闪烁效果，本例通过 jQuery 框架与 quoteRotator 插件的组合来实现文字动画。本例效果如图 1.35 和图 1.36 所示。

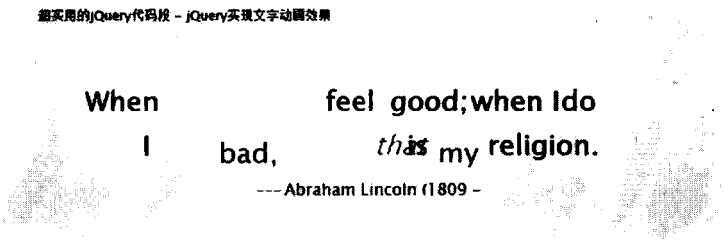

图 1.35 文字动画效果（一）

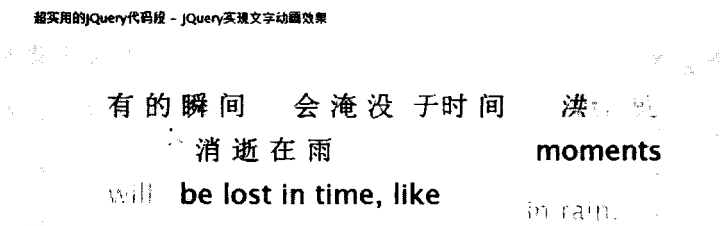

图 1.36 文字动画效果（二）

本例代码如下：

```
01  /* textAni.html */
02  <script src="js/js.textAni.com/jquery.quoterotator.min.js"></script>
03  <script>
04  $(document).ready(function(){
05      $('#words').quoteRotator();                    //调用 jQuery quoteRotator 插件函数
06  });
07  </script>
08  <body>
09      <div id="wrapper">
10      <h2>超实用的 jQuery 代码段 - jQuery 实现文字动画效果</h2>
11      <div id="words">
12          <ul class="word-container">
13              <li data-author="---  Abraham Lincoln (1809 - 1865), (attributed)">
14              When I do good, I feel good; when I do bad, I feel bad, and that is
15              my religion. </li>
16              <li data-author="">所有的瞬间都会淹没于时间的洪流，就像
17  泪水消逝在雨中。
18              All those moments will be lost in time, like tears in rain.</li>
19              <li data-author="" data-easeout="fadeOutDown">The animation can be
20              in random or pre-defined in the HTML. Next quote animation will be
21              all in fadeInDown. Optional click to next quote and hover to pause
22              the slideshow.</li>
23              //省略部分代码
24          </ul>
25          <div class="quote">
26              <blockquote>
27                <p class="quote-content"></p>
28              </blockquote>
29              <cite class="quote-author"></cite>
30          </div>
31      </div>
32  </div>
33  </body>
```

上述代码省略了部分样式风格的文字，读者可以参考 quoteRotator 插件的 CSS 样式文件源代码进行了解。本例关键代码是第 05 行，其中 quoteRotator()插件函数可以实现多种文

字动画效果，其语法如下：

```
$('selector').quoteRotator();
```

其中参数说明见表 1.3。

表 1.3 quoteRotator插件函数参数说明

参数名称	默认值	参数说明
data-author	可选	动画文本作者名称
data-easeout	可选	动画文本效果，具体可选属性值： lightSpeedOut lightSpeedIn fadeOutDown fadeInDown bounceOut bounceIn

提示：读者可以仔细阅读 quoteRotator 插件的源代码，其中名称为 animate.css 的样式文件中定义了多种文字动画风格，具体使用方法可以参考本例代码。

1.25 实现文字跟随鼠标移动变化的动画效果

假设页面中有一段文本，当用户操作鼠标在文字上移动时，文字像跳舞一样随鼠标变化，岂不是非常有趣！本例结合 jQuery 框架和 jQuery.moatext 插件，实现了文字跟随鼠标移动变化的动画效果，效果如图 1.37 和图 1.38 所示。

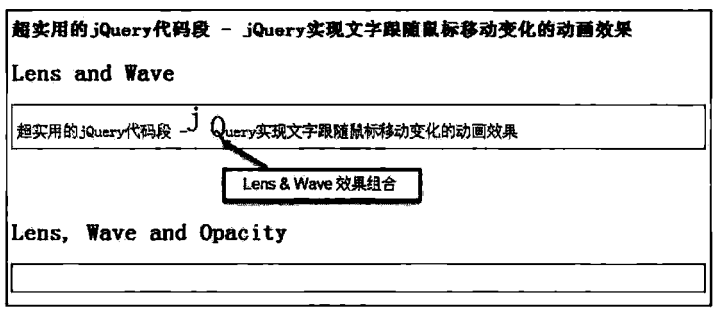

图 1.37 文字跟随鼠标移动效果（一）

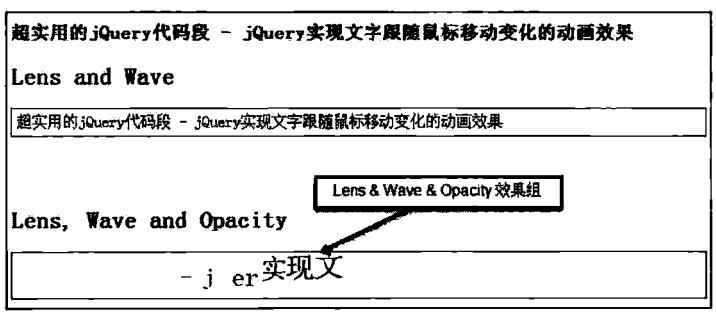

图 1.38 文字跟随鼠标移动效果（二）

本例主要代码如下：

```
01  /* textMouseMove.html */
02  <script type="text/javascript" src="js/js.textMouseMove.com/jquery.moatext-min.js"></script>
03  <script>
04  $(function(){
05      //初始化 jQuery.moatext 插件
06      //实现 lens、wave 与 opacity 混合风格
07      $(".moamix2").moatext({effects:["lens","wave","opacity"]});
08  });
09  </script>
10  <body>
11  <h3>超实用的 jQuery 代码段 - jQuery 实现文字跟随鼠标移动变化的动画效果</h3>
12  <h2><a name="lensandwave">Lens and Wave</a></h2>
13  <div style="border: 1px solid #000; padding: 5px 5px 5px 5px" class="moamix1">
14  超实用的 jQuery 代码段 - jQuery 实现文字跟随鼠标移动变化的动画效果
15  </div>
16  <br><br><br>
17  <h2><a name="lenswaveandopacity">Lens, Wave and Opacity</a></h2>
18  <div style="border: 1px solid #000; padding: 5px 5px 5px 5px" class="moamix2">
19  超实用的 jQuery 代码段 - jQuery 实现文字跟随鼠标移动变化的动画效果
20  </div>
21  </body>
```

本例的关键代码是第 04~08 行，其中.moatext()初始化函数可以实现多种文字跟随鼠标移动变化的动画效果，其语法如下：

```
$('selector').moatext();
```

其中参数说明见表 1.4。

表 1.4　moatext插件函数参数说明

参数名称	默认值	参数说明
moatext_lens	可选	文字透镜效果
moatext_wave	可选	文字波浪效果
moatext_opacity	可选	文字透明效果

1.26　文本域中光标的定位

目前有很多学习代码的在线网站，可以实现代码编写所见即所得的效果，但如果你输入的代码有错，网站会给出提示并将光标定位在错误处，如第几个单词处有错，鼠标会显示在这个单词后面。本例就来实现光标定位的功能，其效果如图 1.39 所示。

本例主要代码如下：

```
01  /* getCursorPos.html */
02  <!doctype html>
```

```
<html>
<head>
<script>
function getCursorPos(obj){
    var rngSel=document.selection.createRange();          //建立选择域
    var rngTxt=obj.createTextRange();                     //建立文本域
    var flag=rngSel.getBookmark();                        //用选择域建立书签
    rngTxt.collapse();
    rngTxt.moveToBookmark(flag);                          //使文本域移动到书签位
    rngTxt.moveStart('character',-obj.value.length);      //获得文本域左侧文本
    str=rngTxt.text.replace(/\r\n/g,'');                  //替换回车换行符
    //return(str.length);                                 //返回文本域文本长度
    $("#div-log").html($("#div-log").html()+"<p>文本域中光标定位："+str.length+"
        characters"+"</p>");
}
</script>
<title>获取文本域中光标的定位</title>
</head>
</html>
```

本例的关键代码是第 06~17 行所定义的 getCursorPos()函数，其中 TextRange 对象是 DHTML 语言的高级特性。使用该对象可以实现很多和文本有关的任务，例如搜索和选择文本。TextRange 对象是在 HTML 文档将要显示的文本流上建立开始和结束位置的抽象对象。

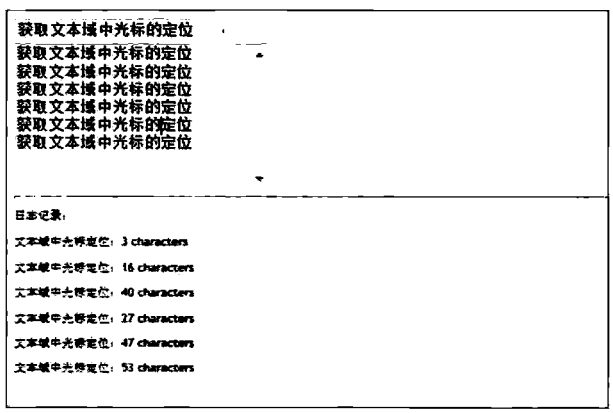

图 1.39 文本域中光标的定位

TextRange 对象的属性说明见表 1.5。

表 1.5 TextRange对象属性说明

参数名称	默认值	参数说明
boundingHeight	可选	获取绑定TextRange对象的矩形的高度
boundingLeft	可选	获取绑定TextRange对象的矩形左边缘和包含TextRange对象的左侧之间的距离

续表

参数名称	默认值	参数说明
offsetLeft	可选	获取对象相对于版面或由offsetParent属性指定的父坐标的计算左侧位置
offsetTop	可选	获取对象相对于版面或由offsetParent属性指定的父坐标的计算顶端位置
htmlText	可选	获取绑定TextRange对象的矩形的宽度
text	可选	设置或获取范围内包含的文本

TextRange 对象的函数说明见表 1.6。

表 1.6 TextRange对象函数说明

参数名称	返回值	参数说明
moveStart	void	更改范围的开始位置
moveEnd	void	更改范围的结束位置
collapse	void	将插入点移动到当前范围的开始或结尾
move	void	折叠给定文本范围并将空范围移动给定单元数
execCommand	void	在当前文档、当前选中区或给定范围上执行命令
select	void	将当前选择区置为当前对象
findText	string	在文本中搜索文本并将范围的开始和结束点设置为包围搜索字符串

提示：如果用户想将光标位置移动到指定位置，可以使用该 TextRange 对象的 move() 函数来实现。另外，借助于 TextRange 对象还可以实现文本区域选择或关键字搜索等许多功能，设计人员可以以此类推，举一反三。

1.27 实现可折叠效果

同一个页面有多组不同的内容，每组包含数量不定的 div 层，这时可以按照组别来单独折叠或展开内容。这类效果的实现原理其实很简单，折叠时隐藏该组中所有元素内容，展开时再将其隐藏属性设置为显示。针对该效果比较成功的实现是 jQuery UI 的 Accordion 控件，其效果如图 1.40、图 1.41 和图 1.42 所示。

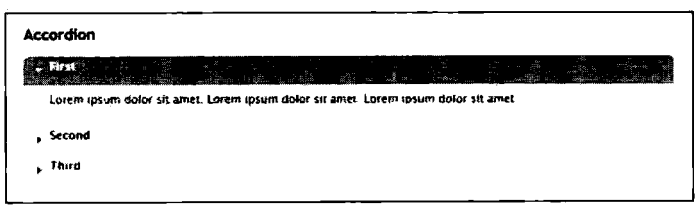

图 1.40 展开“first”层时的效果

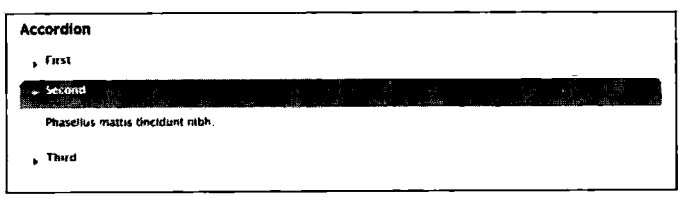

图 1.41 展开“second”层时的效果

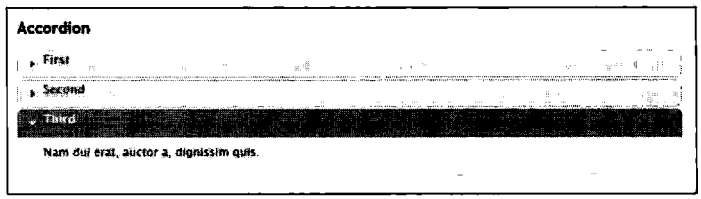

图 1.42 展开“third”层时的效果

本例主要代码如下：

```
01  /* jqToggle.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $(".help h3").click(function(){
05          $(this).next("p").slideToggle("fast").siblings("p:visible").slideUp("fast");
06      });
07  });
08  </script>
09  <body>
10  <h2>超实用的 jQuery 代码段 - jQuery 实现可折叠效果的方法</h2>
11  <div class="help">
12      <div class="cssDiv">
13          <h3>单击折叠或展开</h3>
14          <p>jQuery 实现可折叠效果的方法</p>
15      </div>
16      <div class="cssDiv">
17          <h3>单击展开或折叠</h3>
18          <p style="display:none;">jQuery 实现可折叠效果的方法</p>
19      </div>
20  </div>
21  </body>
```

本例的关键代码是第 05 行，其中 siblings()函数获得匹配集合中每个元素的同胞，通过选择器进行筛选是可选的，其语法如下：

```
.siblings(selector);
```

其中参数 selector 为可选项、字符串值，表示包含用于匹配元素的选择器表达式。

设计人员使用 siblings()函数时需要注意以下两点：

- 如果给定一个表示 DOM 元素集合的 jQuery 对象，siblings()函数允许用户在 DOM 树中搜索这些元素的同胞元素，并用匹配元素构造一个新的 jQuery 对象。
- siblings()函数接受可选的选择器表达式，与 jQuery 框架的$()函数中传递的参数类型相同。如果应用这个选择器，则将通过检测元素是否匹配该选择器来筛选元素。

1.28 文本框内容自动缩进

在很多种情况下，页面需要将文本框里的内容进行美化工作，这样可以保证页面的简

洁美观。例如 Mirosoft Office Word 办公软件就提供了很多种文本格式以供用户选择，像分段格式、字体格式、标题格式等等。本例结合 jQuery 框架与 elasticTextArea 插件来实现文本框内容的自动缩进，效果如图 1.43 所示。

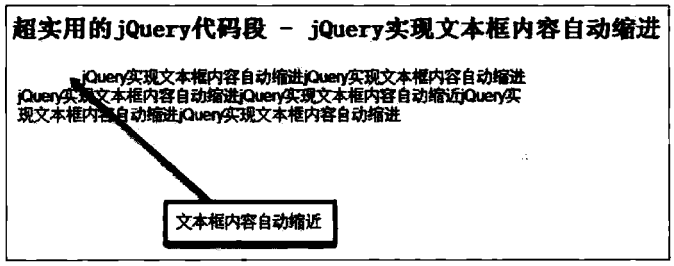

图 1.43　文本框内容自动缩进

本例主要代码如下：

```
01  /* textAutoFormat.html */
02  <script type="text/javascript" src="js/js.textAutoFormat.com/ext-core/ext-core.js"></script>
03  <script type="text/javascript" src="js/js.textAutoFormat.com/elastic-textarea.js"></script>
04  <script>
05  $(document).ready(function(){
06     elasticTextArea("id-textFormat");
07  });
08  </script>
09  <body>
10        <h2>超实用的 jQuery 代码段 - jQuery 实现文本框内容自动缩近</h2>
11        <p>
12              <textarea id="id-textFormat">
13              jQuery 实现文本框内容自动缩进 jQuery 实现文本框内容自动缩进 jQuery 实现文本框
14  内容自动缩进 jQuery 实现文本框内容自动缩进 jQuery 实现文本框内容自动缩进 jQuery 实现文
15  本框内容自动缩进
16        </textarea>
17        </p>
18  </body>
```

本例的关键代码是第 05~07 行，其中 elasticTextArea()插件函数是一个用来根据 Textarea 文本编辑框输入的数据自动实现文本首行缩进的函数，其语法如下：

```
$.elasticTextArea(selector);
```

其中参数 selector 为可选项，用来指定 Textarea 文本编辑框的 id。

1.29　禁止页面滚动的方法

如果想在页面中给用户呈现很关键的内容，例如网站的重要通知、插入的图片广告或视频文件等等，可能就需要强行禁止用户滚动页面，当然要在一个合适的时间段后自动关闭这些插入信息并重新启用页面滚动功能。本例就来实现最基本的强制禁止页面滚动的方

法，其效果如图 1.44 与图 1.45 所示。

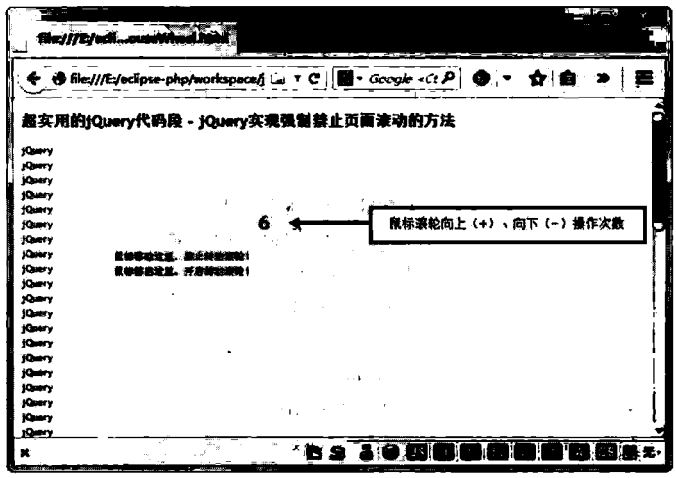

图 1.44 禁止滚轮

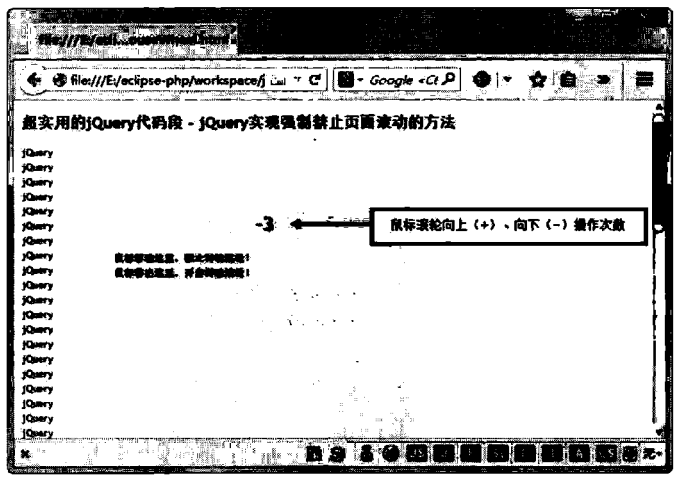

图 1.45 开启滚轮

本例代码如下：

```
01  /* forbiddenMouseWheel.html */
02  <script type="text/javascript">
03  window.onload=function(){
04      $("forbiddenMouseWheel").onmousewheel=function scrollWheel(e){
05          if(navigator.userAgent.toLowerCase().indexOf('firefox')>=0){
06              //firefox 支持 onmousewheel
07              addEventListener('DOMMouseScroll',//添加'DOMMouseScroll'事件监听过程
08              function(e){                          //事件监听回调函数
09                  var obj=e.target;
10                  var onmousewheel;
11                  while(obj){                       //通过循环判断页面响应事件
12              onmousewheel=obj.getAttribute('onmousewheel')||obj.onmousewheel;
```

```
13                          if(onmousewheel) break;
14                          if(obj.tagName=='BODY') break;
15                          obj=obj.parentNode;
16                      };
17                      if(onmousewheel){              //判断鼠标滚轮事件
18                      if(e.preventDefault) e.preventDefault();
19                      e.returnValue=false;           //禁止页面滚动
20                  if(typeof obj.onmousewheel!='function'){//将 onmousewheel 转换成 function
21                          eval('window._tmpFun=function(event){'+onmousewheel+'}');
22                          obj.onmousewheel=window._tmpFun;
24                          window._tmpFun=null;
25                      };
26                      setTimeout(function(){              //设定时间延迟
27                          obj.onmousewheel(e);
28                      },1);
29                  };
30              },
31              false);
32      };
33      };
34  }
35  </script>
36  <body>
37      <h2>超实用的 jQuery 代码段 - jQuery 实现强制禁止页面滚动的方法</h2>
38      <p>
39          <div id="forbiddenMouseWheel" style="">
40              <h1 id="he" style="text-align:center;width:100%;color:#f00;">0</h1>
41              鼠标移动这里，禁止转动滚轮！
42              鼠标移出这里，开启转动滚轮！
43          </div>
44      </p>
45  </body>
```

本例的关键代码是第 18~19 行中的 preventDefault()自定义函数，该函数完成取消事件的默认动作功能，其语法如下：

```
$.event.preventDefault();
```

该函数将通知 Web 浏览器不要执行与事件关联的默认动作（如果存在这样的动作）。例如，如果 type 属性是 submit，在事件传播的任意阶段可以调用任意的事件句柄，通过调用该函数，可以阻止提交表单。

设计人员要注意以下两点：

- 如果 Event 对象的 cancelable 属性是 fasle，那么就表示没有默认动作，或者不能阻止默认动作。无论哪种情况，调用该函数都没有作用。
- 本例代码最后不直接执行 obj.onmousewheel(e)语句，是因为若 onmousewheel(e)运行时间较长的话，会导致锁定滚动失效，而使用 setTimeout()函数可以避免该情况的出现。

1.30 页面加载后消息框居中显示

设计页面时经常会需要使用alert()警示消息框来提示用户需要注意的问题，为了美观，建议在弹出消息框时居中显示。本例演示效果如图1.46与图1.47所示。

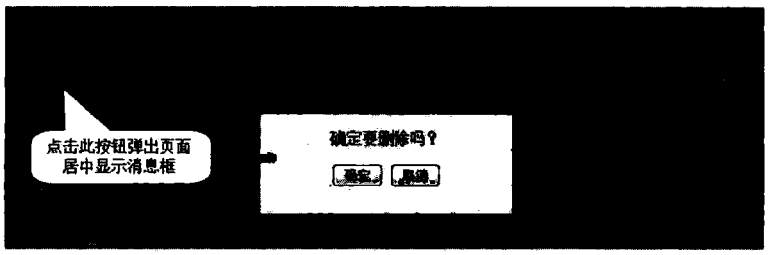

图1.46 居中显示消息框

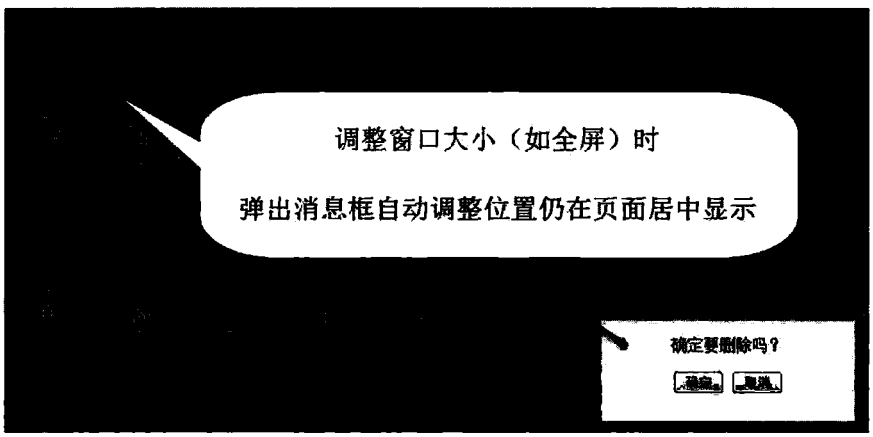

图1.47 调整窗口后再显示消息框

本例主要代码如下：

```
01  /* midAlert.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $('.btn').click(function(){                    //定义 click 事件
05      $('.mask').css({'display':'block'});
06      center($('.mess'));                            //调用 center(居中)函数
07      check($(this).parent(),$('.btn1'),$('.btn2'));
08  });
09  //定义居中函数
10  function center(obj){
11      var screenWidth=$(window).width();             //当前浏览器窗口的宽度
12      var screenHeight=$(window).height();           //当前浏览器窗口的高度
13      var scrolltop=$(document).scrollTop();         //获取当前窗口距离页面顶部高度
14      var objLeft=(screenWidth-obj.width())/2;       //计算弹出消息框对象的 left 定位
```

```
15        var objTop=(screenHeight-obj.height())/2+scrolltop;  //计算弹出消息框对象的 top 定位
16        obj.css({left: objLeft + 'px', top: objTop + 'px','display': 'block'}); //通过 CSS 定位弹出消
17  息框
18        //浏览器窗口大小改变时调整函数
19        $(window).resize(function(){
20        screenWidth=$(window).width();      //当前浏览器窗口的宽度
21        screenHeight=$(window).height();    //当前浏览器窗口的高度
22        scrolltop=$(document).scrollTop();  //获取当前窗口距离页面顶部高度
23        objLeft=(screenWidth-obj.width())/2; //计算弹出消息框对象的 left 定位
24        objTop=(screenHeight-obj.height())/2+scrolltop;      //计算弹出消息框对象的 top 定位
25        obj.css({left:objLeft+'px',top:objTop+'px','display':'block'});//通过 CSS 定位弹出消息框
26  });
27  //浏览器有滚动条时调整函数
28  $(window).scroll(function(){
29        screenWidth=$(window).width();
30        screenHeight=$(widow).height();
31        scrolltop=$(document).scrollTop();
32        objLeft=(screenWidth-obj.width())/2;
33        objTop=(screenHeight-obj.height())/2+scrolltop;
34        obj.css({left:objLeft+'px',top:objTop+'px','display':'block'});
35  });
36  }
37  </script>
38  <body>
39  <h2>超实用的 jQuery 代码段 - jQuery 页面加载后居中显示消息框的方法</h2>
40  <input type="button" class="btn" value="单击弹出消息框"/>
41  <div class="mask"></div>
42  <div class="mess">
43  //省略部分代码
44  </div>
45  </body>
```

本例的关键代码是第 19~26 行中的 resize()自定义函数，具体分析如下：第 20 行中的 $(window).width()函数获取整个浏览器可视窗口的宽度；第 21 行中的$(window).height()函数获取整个浏览器可视窗口的高度；第 22 行中的$(document).scrollTop()函数获取浏览器可视窗口顶端距离网页顶端的高度（垂直偏移）；第 23~24 行负责计算弹出消息框对象的 left 与 top 定位；第 25 行中的 obj.css()函数返回或设置匹配的元素的一个或多个样式属性。

设计人员要注意以下两点：

- 如果用户使用$(window).height()函数无法获取实际窗口高度时，需要在 HTML 页面顶部添加<!DOCTYPE html>关键字标识。
- 当使用$(selector).css(name)函数用于返回一个值时，不支持简写的 CSS 属性（例如，background 属性和 border 属性）。

1.31 创建页面固定浮动栏的方法

曾几何时，页面广告是许多网站的主要收入来源，其最常见的呈现形式就是固定在页面右侧的浮动层，而且随着页面上下滑动该浮动层位置不变。大家都知道CSS定位中有一种用法是“position:fixed”可以让层处于固定位置，但无奈的是很多旧版本浏览器并不支持该特性，看起来CSS的浏览器兼容性是个大问题。本例学习利用jQuery框架来创建页面固定浮动栏的方法，效果如图1.48所示。

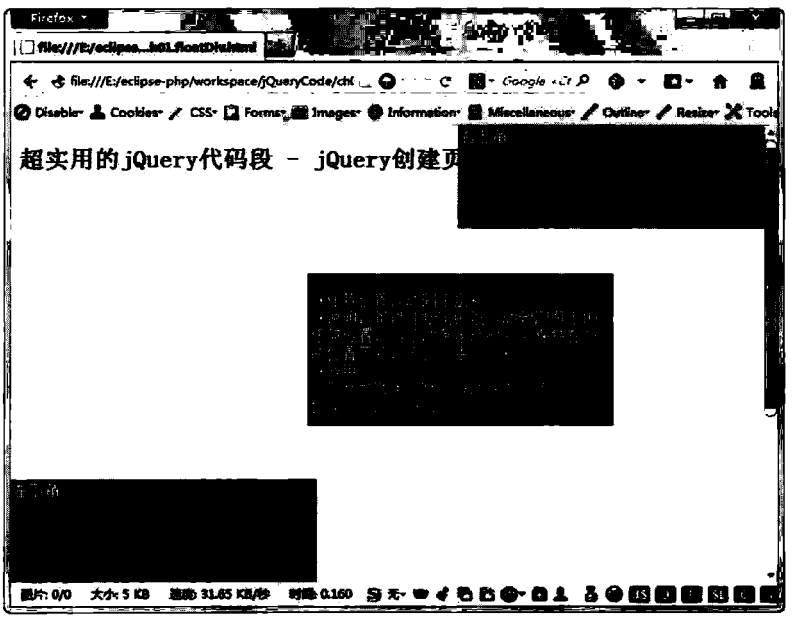

图1.48 创建页面固定浮动栏

本例主要代码如下：

```
01  /* floatDiv.html */
02  <script type="text/javascript" src="js/js.floatDiv.com/jquery.floatDiv.js"></script>
03  <script type="text/javascript">
04  $(function(){
05      $("#randomDiv").floatdiv({top:"150px",right:"150px"});
06      $("#rtop").floatdiv("righttop");
07      $("#lbottom").floatdiv("leftbottom");
08  });
09  </script>
10  </head>
11  <body>
12  <h2>超实用的jQuery代码段 - jQuery创建页面右侧固定浮动栏的方法</h2>
13  <div id="rtop" style="">右上角</div>
14  <div id="lbottom" style="">左下角</div>
15  <div id="randomDiv" style=""></div>
```

```
16  //省略部分代码
17  </body>
18  </html>
```

本例创建页面固定浮动栏方法是基于 floatDiv 插件实现的，关键代码见第 05~07 行，其中$("#rtop").floatdiv()函数用来创建右上角固定浮动栏，写法如下：

```
$(selector).floatdiv(parameters);
```

其中参数说明见表 1.7。

表 1.7 floatdiv()函数参数说明

参数名称	默认值	参数说明
parameters	可选	该参数规定浮动栏的位置与呈现形式。可选属性值： rightbottom，页面右下角 leftbottom，页面左下角 lefttop，页面左上角 righttop，页面右上角 middle，页面居中 自定义位置

第 2 章　jQuery 操作 DOM 元素

DOM 是 Document Object Model 的缩写，其含义是文档对象模型。DOM 是一种跟浏览器、平台以及语言都没有关系的规范，本质上就是一个接口，它提供一种可以访问页面中所有节点的机制。

本章主要介绍如何使用 jQuery 操作 DOM 元素，包括以下内容：

- 创建、插入、删除 DOM 节点。
- 使用 jQuery 操作 DOM 节点的属性。
- 使用 jQuery 操作 DOM 节点的样式。

2.1　如何验证某个元素是否为空

用户在使用页面表单提交用户注册信息时，需要验证表单中的每一个元素是否为空，根据验证结果来决定下一步的操作。本例主要代码如下：

```
01  /* checkElementNull.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      var len=$('#element').length;
05      if(len>0){
06          alert('Extsts');
07      }
08      else{
09          alert('not Extsts');
10      }
11  });
12  </script>
13  <body>
14  /* 略去部分代码 */
15  <label id="element"></label>
16  /* 略去部分代码 */
17  </body>
```

通过 jQuery 选择器获取 element 时总能返回一个对象，该对象永远是存在的，因此并不能通过此方法判断 element 是否为空。如果要验证 element 是否为空，则需要判断该对象的长度是否大于零，如代码第 4~10 行所示。还可以通过转换成 JavaScript 对象的方式来进

行处理，代码如下：

```
if($("#element")[0]){
//do something...
}
```

提示：在jQuery框架中，$符号是jQuery的别名，所有使用$的地方也都可以使用jQuery关键字来替换，如$('#element')等同于jQuery('#element')。

2.2 检查特定的HTML元素是否存在

了解搜索引擎工作原理的读者一般都知道，设计人员需要验证页面中是否包含特定的HTML元素，根据验证结果来决定搜索引擎下一步的工作。本例演示效果如图2.1所示。

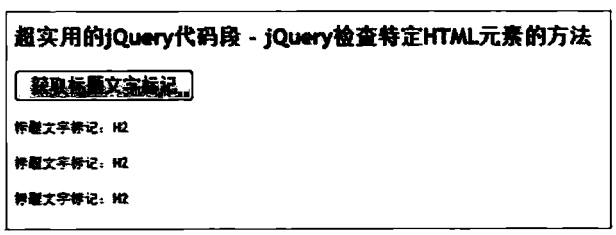

图2.1 检查特定HTML元素

本例主要代码如下：

```
/* checkHTMLElement.html */
<script type="text/javascript">
$(function(){
    $("#id-input-h2").click(function(){
    $("#div-log").html(                          //将用户操作内容显示在该div层中
        $("#div-log").html()+
        "<p>标题文字标记："+
        $("h2#h2-caption")[0].tagName+           //获取<h2>标签 tagname
        "</p>"
        );
    });
});
</script>
<body>
<h2 id="h2-caption">超实用的jQuery代码段 - jQuery检查特定HTML元素的方法</h2>
<form>
<input type="button" id="id-input-h2" value="获取标题文字标记">
</form>
</body>
```

通过jQuery选择器检查特定HTML元素是否存在，其实有几种方法都可以实现。例如有一段HTML代码如下：

```
<body>
/* 略去部分代码 */
<a id="link" name="link" href="http://www.google.com">Google 主页</a>
/* 略去部分代码 */
</body>
```

可以通过以下方法来检查特定的HTML元素是否存在。

（1）使用HTML元素的id：

```
link.href，返回值为 http://www.google.com
```

（2）使用HTML元素的name：

```
document.all.link.href，返回值为 http://www.google.com
```

（3）使用HTML元素的sourseIndex：

```
document.all(4).href        //注意前面还有 HTML、HEAD、TITLE 和 BODY 元素，所以数值是 4
```

（4）使用HTML元素的链接集合：

```
document.anchors(0).href        //方法 3 和方法 4 同样是使用集合的，只是一个是 all，可以包
                                //括页面所有标记，而 anchors 只包括链接
```

（5）使用getElementById()函数：

```
document.getElementById("link").href
```

（6）使用getElementsByName()函数：

```
document.getElementsByName("link")[0].href  //该函数也是使用一个集合，是所有 name 等于
                                            //该函数所带参数的标记的集合
```

（7）使用getElementsByTagName()函数：

```
document.getElementsByTagName("A")[0].href      //该函数也是使用一个集合，是所有标记名称
                                                //等于该函数所带参数的标记的集合
```

（8）使用HTML元素tags集合方法：

```
document.all.tags("A")[0].href          //方法 8 与方法 7 一样是按标记名称取得一个集合
```

在jQuery中需要特别注意all方法与getElementById()、getElementsByName()、getElementsByTagName()这些函数的区别：

- 尽量不要采用all方法来获取元素，这种方法不符合W3C标准，而且只能在IE或IE内核的浏览器中有效。
- 使用getElementById()函数通过id来取得document中的某一特定元素，只能访问设置了id的元素。
- 使用getElementsByName()函数获得元素数组时，如果document中只有一个指定name的标签时，也返回数组。document中每一个元素的id是唯一的，但name却可以重复。
- 使用getElementsByTagName()函数通过tagName来获得元素时，一个document中当然会有相同的标签，所以这个函数也是取得一个数组。

2.3 判断HTML元素是否嵌套

HTML元素嵌套是很常见的情况，设计人员需要验证页面中的HTML元素是否为嵌套

关系，再进行相关的操作。本例主要代码如下：

```
01  /* checkElementNested.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      if($("p.child").parents("#parent").length==1){
05          // “真”，子元素包含在父元素中
06          alert('Yes, the child element is inside the parent.');
07      }else{
08          // “假”，子元素不包含在父元素中
09          alert("No, it is not inside.");
10      }
11  })
12  </script>
13  <body>
14  /* 略去部分代码 */
15  <div id="parent">
16  <p class="child">
17  This paragraph is very important.
18  </p>
19  </div>
20  /* 略去部分代码 */
21  </body>
```

通过 jQuery 选择器判断元素嵌套一般是获取元素对象的父级元素，然后通过判断父级元素的 length 属性长度是否等于 1 来确定是否嵌套，如以上代码第 04~10 行所示。

第 04 行用到了类选择器，它用来描述一组元素的样式。该选择器可以单独使用，也可以与其他元素结合使用，如：

```
举例：$("p.child")        /* 选择标签<p>中 class 样式为 child 的元素 */
```

2.4 获取当前元素的索引值

本例针对 HTML 中的树、列表、组合框等控件，获取其中某个元素的索引值，根据获得的索引值再进行诸如插入、删除或替换等操作。本例效果如图 2.2 所示。

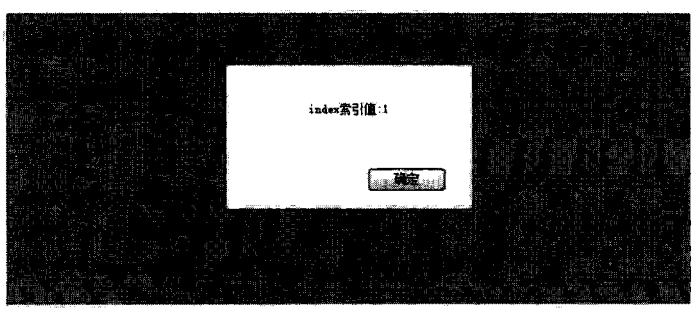

图 2.2 获取当前元素的索引值

本例主要代码如下：

```
/* indexElement.html */
<script type="text/javascript">
$(document).ready(function(){
    $("button").click(function(){
//获取 id 值等于"id-li-sel"的元素，相对于 jQuery 选择器(class="css-li-index")的 index 索引值
        alert("index 索引值:"+$(".css-li-index").index($("#id-li-sel")));
    });
});
</script>
<body>
<h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 获取当前元素的索引值</h2>
<button>获取 index 索引值</button>
<ul>
    <li>苹果</li>                    //该<li>列表选项没有设定 class="css-li-index"属性值
    <li class="css-li-index">香蕉</li>          //该<li>列表选项没有设定 id="id-li-sel"属性值
    <li class="css-li-index" id="id-li-sel">橙子</li>              //该<li>列表选项满足要求
</ul>
<p>请单击"获取 index 索引值"按钮</p>
<p>以获得 id="id-li-sel"的元素</p>
<p>且相对于 jQuery 选择器 (class="css-li-index")的 index 索引值</p>
</body>
```

如果设计人员想知道用户在操作 ul 列表控件时，具体单击的是哪一个 li 列表项，就需要获取该 li 列表项的索引值。通过函数 index()，可以轻松获取当前元素的索引值，具体代码如下：

```
$("li").hover(function(){
  alert($("li").index(this));
});
```

其中 index()函数返回指定元素相对于其他指定元素的 index 位置。这些元素可通过 jQuery 选择器或 DOM 元素来指定。具体说明如下：

- 如果需要获得第一个匹配元素相对于其同胞元素的 index 位置，其语法形式如下：

```
$(selector).index();          //获取第一个匹配元素的 index，相对于同胞元素
```

- 如果需要获得元素相对于选择器的 index 位置，该元素可以通过 DOM 元素或 jQuery 选择器来指定，其语法形式如下：

```
$(selector).index(element);    //获取相对于同胞元素的匹配元素的 index
```

其中 element 参数为可选项，表示要获得 index 位置的元素，可以是 DOM 元素或 jQuery 选择器。

提示：index()函数的含义是搜索与参数相匹配的元素，并返回相应元素的索引值。如果使用 index()函数找到了匹配的元素，返回值从 0 开始；如果没有找到匹配的元素，则返回值为-1。

2.5 插入节点元素

在页面中插入节点元素的操作是实现动态页面的基本条件之一，插入节点有很多种方法，具体代码如下：

```
/* insertElement.html */
<script type="text/javascript">
$(document).ready(function(){
    $("button#button-append").click(function(){
        $("p").append("<b>插入节点元素</b>");          //向<p>元素中追加<b>元素
    });
    $("button#button-appendTo").click(function(){
        $("<b>追加节点元素</b>").appendTo("p");       //将<b>元素追加到<p>元素中
    });
    $("button#button-prepend").click(function(){
        $("p").prepend("<b>前置节点元素</b>");         //在<p>元素之前添加<b>元素
    });
    $("button#button-prependTo").click(function(){
        $("<b>前置到节点元素</b>").prependTo("p");   //将<b>元素前置到<p>元素中
    });
    $("button#button-after").click(function(){
        $("p").after("<b>节点元素后插入节点元素</b>");//向<p>元素元素后插入<b>元素
    });
    $("button#button-insertAfter").click(function(){
        $("<b>插入到节点元素后</b>").insertAfter("p"); //将<b>元素插入到<p>元素后边
    });
    $("button#button-before").click(function(){
        $("p").before("<b>节点元素前插入节点元素</b>");    //在<p>元素之前添加<b>元素
    });
    $("button#button-insertBefore").click(function(){
        $("<b>插入到节点元素前</b>").insertBefore("p");//将<b>元素插入到<p>元素前面
    });
});
</script>
<body>
<h2>超实用的 jQuery 代码段 - jQuery 插入节点元素的方法</h2>
<button id="button-append">插入节点元素 - append()函数</button><br>
<button id="button-appendTo">插入节点元素 - appendTo()函数</button><br>
<button id="button-prepend">插入节点元素 - prepend()函数</button><br>
<button id="button-prependTo">插入节点元素 - prependTo()函数</button><br>
<button id="button-after">插入节点元素 - after()函数</button><br>
<button id="button-insertAfter">插入节点元素 - insertAfter()函数</button><br>
<button id="button-before">插入节点元素 - before()函数</button><br>
<button id="button-insertBefore">插入节点元素 - insertBefore()函数</button><br>
```

```
40  <p>您好！您最喜欢的 IT 公司是：</p>
41  <ul>
42      <li title="Google">Google</li>
43      <li title="Apple">Apple</li>
44      <li title="Microsoft">Microsoft</li>
45  </ul>
46  </body>
```

本例演示效果分别如图 2.3 和图 2.4 所示。

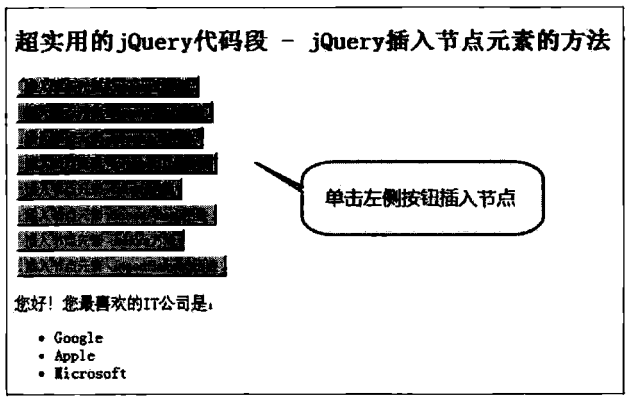

图 2.3　jQuery“插入节点元素的方法”演示效果（一）

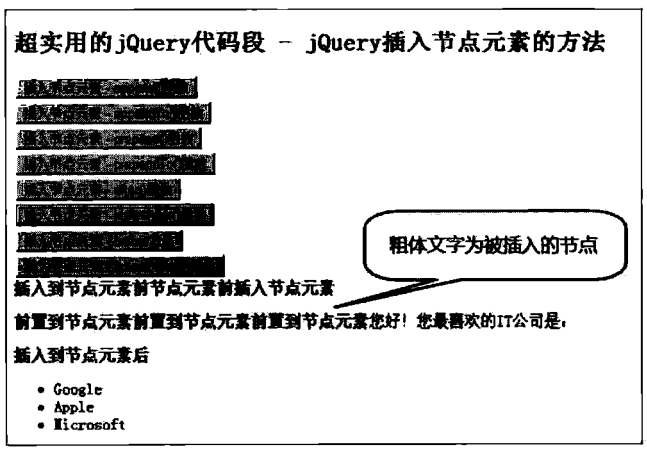

图 2.4　jQuery“插入节点元素的方法”演示效果（二）

将新创建的节点元素插入文档最简单的办法是使用 append()函数让它成为这个文档的某个节点的子节点。当然，将新创建的节点插入某个文档的方法并非只有一种，在 jQuery 框架中还提供了其他几种插入节点的函数，设计人员可以根据项目的实际需求灵活地做出多种选择。

jQuery 代码分析如下。

- append()函数在被选元素的结尾（仍然在内部）插入指定内容。语法如下：

```
$(selector).append(content);
```

其中 content 参数为必选项，表示要插入的内容（可包含 HTML 标签）。

- appendTo()函数在被选元素的结尾（仍然在内部）插入指定内容。语法如下：
$(content).appendTo(selector);

其中 selector 参数是必选项，表示把内容追加到哪个元素上。

- prepend()函数在被选元素的开头（仍位于内部）插入指定内容。语法如下：
$(selector).prepend(content);
- prependTo()函数在被选元素的开头（仍位于内部）插入指定内容。语法如下：
$(content).prependTo(selector);
- after()函数在被选元素后插入指定的内容。语法如下：
$(selector).after(content);
- insertAfter()函数在被选元素之后插入 HTML 标记或已有的元素。语法如下：
$(content).insertAfter(selector);
- before()函数在被选元素之后插入 HTML 标记或已有的元素。语法如下：
$(selector).before(content);
- insertBefore()函数在被选元素之前插入 HTML 标记或已有的元素。语法如下：
$(content).insertBefore(selector);

其中 content 有两种类型：一种是选择器表达式，一种是 HTML 标记。

实现本例的方法有很多，但需要注意以下两点：

（1）append()和 appendTo()函数执行的任务相同，不同之处在于内容和选择器的位置。

（2）prepend()和 prependTo()函数作用相同，不同之处在于内容和选择器的位置。

2.6 复制节点元素

在页面中复制节点元素的操作也是实现动态页面的基本条件之一，复制节点的具体代码如下：

```
/* copyElement.html */
<script type="text/javascript">
01  $(document).ready(function(){
02      $("ul li").click(function(){
03          //复制当前单击的节点，并将它追加到<ul>元素中，当添加参数时复制它的事件
04          $(this).clone(true).appendTo("ul");
05      });
06  });
07  </script>
08  <body>
09  <h2>超实用的 jQuery 代码段 - jQuery 复制节点元素的方法</h2>
10  <p>您好！您最喜欢的 IT 公司是：</p>
11  <ul>
12      <li title="Google">Google</li>
13      <li title="Apple">Apple</li>
14      <li title="Microsoft">Microsoft</li>
15  </ul>
```

```
16  </body>
```

本例演示效果分别如图 2.5 和图 2.6 所示。

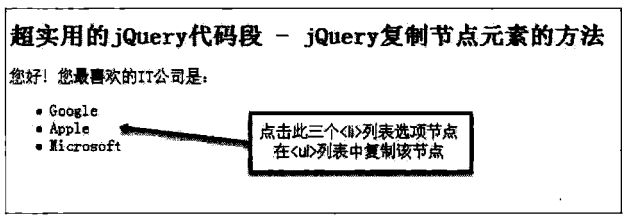

图 2.5　复制节点效果（一）

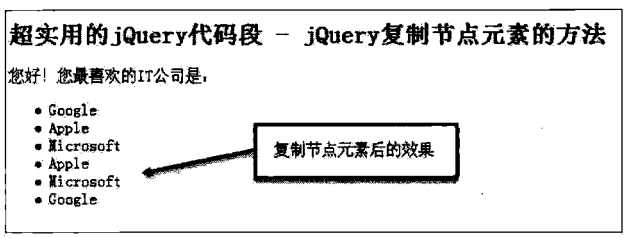

图 2.6　复制节点效果（二）

除了需要动态创建 HTML 元素并将新创建的节点元素插入文档中，有时候还需要复制节点元素的操作，使用 clone()函数可以完成节点元素复制功能。具体代码如下：

```
$(this).clone(true).appendTo("ul");
```

clone()函数生成被选元素的副本，包含子节点、文本和属性。其语法如下：

```
$(selector).clone(includeEvents);
```

其中 includeEvents 参数为可选项，为布尔值，其规定了是否复制元素的所有事件处理。

提示：默认情况下，clone()函数生成的副本中不包含事件处理器。

2.7　替换节点元素

还有一种就是在页面中替换节点元素的操作，该操作同样是实现动态页面的基本条件之一，具体代码如下：

```
01  /* replaceElement.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $("button#button-replaceWith").click(function(){
05      $("p").replaceWith("<strong>您好！您最喜欢的 IT 公司是：</Strong>");
06      $("[name='name-replace']").replaceWith("<tr><td>name-replace</td><td>name-replace</td>
07  <td>name-replace</td><td>name-replace</td></tr><tr><td>name-replace</td><td>name-rep
08      lace</td><td>name-replace</td><td>name-replace</td></tr>");
09      });
10  });
11  </script>
```

```
12  <body>
13  <h2>超实用的 jQuery 代码段 - jQuery 插入节点元素的方法</h2>
14  <p>您好！您最喜欢的 IT 公司是：</p>
15  <ul>
16      <li title="Google">Google</li>
17      <li title="Apple">Apple</li>
18      <li title="Microsoft">Microsoft</li>
19  </ul>
20  <table name="name-replace"></table>
21  <button id="button-replaceWith">替换节点元素 – replaceWith </button>
22  </body>
```

本例效果如图 2.7 和图 2.8 所示。

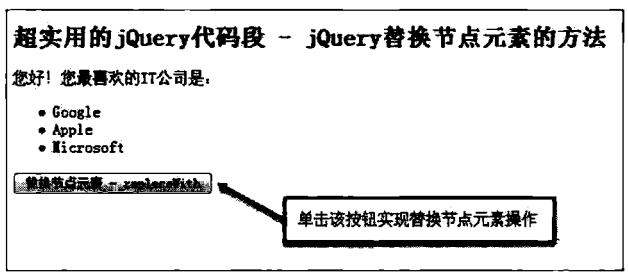

图 2.7　替换节点效果（一）

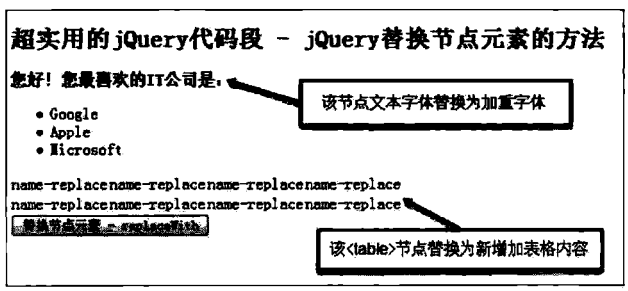

图 2.8　替换节点效果（二）

针对 HTML 节点元素进行替换的操作也是十分常用的，使用 replaceWith()函数可以完成节点元素的替换功能。具体代码如下：

```
$("p").replaceWith("<strong>您好！您最喜欢的 IT 公司是：</Strong>");
```

replaceWith()函数用指定的 HTML 内容或元素替换被选元素，其语法如下：

```
$(selector).replaceWith(content);
```

其中 content 参数有 3 种可选值：

- HTML 代码，比如(" ")
- 新元素，比如(document.createElement("div"))
- 已存在的元素，比如($(".id-div"))

提示：replaceWith()函数与 replaceAll()函数作用相同，两者之间的差异在于内容和选择器的位置不同。replaceAll()函数无法使用函数进行替换。

2.8　删除节点元素

除了节点的插入、复制和替换外，jQuery 还支持在页面中删除节点元素，具体代码如下：

```
/* removeElement.html */
<script type="text/javascript">
$(document).ready(function(){
    $("button#button-delete-1").click(function(){
        var $li=$("ul li:eq(0)").remove(); //删除<ul>节点中第 1 个元素节点
        $("ul").append($li);              //把刚删除的元素节点重新添加到<ul>元素中去
    });
    $("button#button-delete-2").click(function(){
        $("ul li").remove("li[title=IBM]"); //将<ul>元素下 title 属性等于"IBM"的<li>元素删除
    });
    $("button#button-delete-3").click(function(){
        $("ul").empty();                  //清空<ul>节点全部<li>元素的内容
    });
});
</script>
<body>
<h2>超实用的 jQuery 代码段 - jQuery 删除节点元素的方法</h2>
<p>您好！您最喜欢的 IT 公司是：</p>
<ul>
    <li title="Google">Google</li>
    <li title="Apple">Apple</li>
    <li title="Microsoft">Microsoft</li>
    <li title="Intel">Intel</li>
    <li title="IBM">IBM</li>
    <li title="Facebook">Facebook</li>
</ul>
<button id="button-delete-1">删除节点元素 - remove - 删除 ul 中第 1 个 li 元素节点并重
新添加到 ul 元素中去 </button><br>
<button id="button-delete-2">删除节点元素 - remove - 将 li 元素 title 属性等于"IBM"的元
素删除 </button><br>
<button id="button-delete-3">删除节点元素 - empty - 清空 ul 节点下第 1 个 li 元素的内容
</button><br>
</body>
```

本例演示效果分别如图 2.9 和图 2.10 所示。

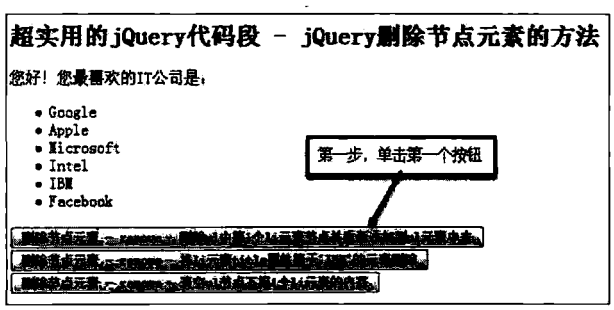

图 2.9　删除节点效果（一）

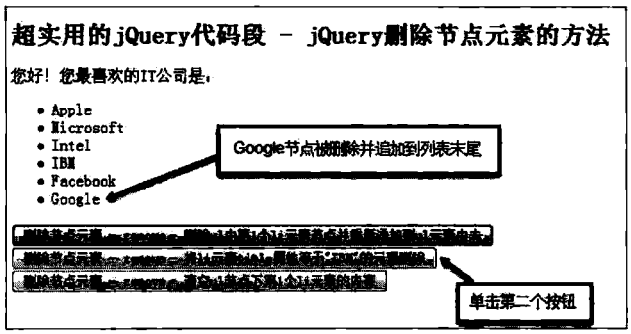

图 2.10 删除节点效果（二）

大多数情况下，删除 HTML 节点元素使用 remove()函数和 empty()函数就可以完成，如上述代码第 9 行和第 12 行所示。remove()函数移除被选元素，包括它所有的文本和子节点。但该函数不会把匹配的元素从 jQuery 对象中删除，因而可以在将来再使用这些匹配的元素。empty()函数从被选元素移除所有内容，包括它所有的文本和子节点。

两个函数都能删除 HTML 节点，但要注意以下几点。

（1）使用 remove()函数除了这个元素本身得以保留之外，不会保留元素的 jQuery 数据。其他的比如绑定的事件、附加的数据等都会被移除。

（2）如果要移除指定的元素，会发现 empty()函数与 remove([expr])函数都可以实现。但如果仔细观察操作效果就可以发现，empty()函数只移除了指定元素中的所有子节点。例如：

- $("p").empty()函数只是把 content 中的文本给移除了，而 节点元素仍保留在 DOM 中并占据原有的位置。
- remove([expr])函数则把节点元素从 DOM 中删除，不会保留其所占据的原有位置。

2.9 为元素绑定事件

我们都知道如何对页面元素调用 HTML 事件，譬如单击事件、双击事件、滚动事件等等。其实还有一种事件使用方法，就是将事件绑定到元素上，那么如何将事件和函数绑定到元素上，绑定后如何执行下一步的操作呢？本例主要代码如下：

```
01  /* bindHTMLElement.html */
02  <script type="text/javascript">
03  $(function(){
04      $('#id-button-bind').bind('click',{d1:'jQuery:',d2:'将事件和函数绑定到元素'},myBindFunc);
05  });
06  function myBindFunc(e){
07      $("#div-log").html($("#div-log").html()+"<p>"+e.data.d1+e.data.d2+"</p>");
08  }
09  </script>
10  <body>
```

```
11  <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 将事件和函数绑定到元素</h2>
12  <input type="button" id="id-button-bind" value="将事件和函数绑定到元素"/>
13  <div id="div-log">
14      <p>日志记录：</p>
15  </div>
16  </body>
```

本例效果分别如图 2.11 和图 2.12 所示。

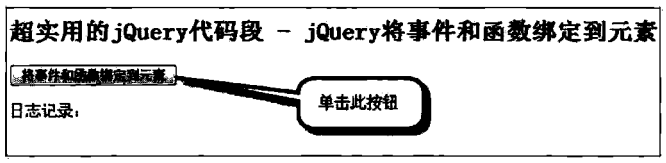

图 2.11　将事件和函数绑定到元素效果（一）

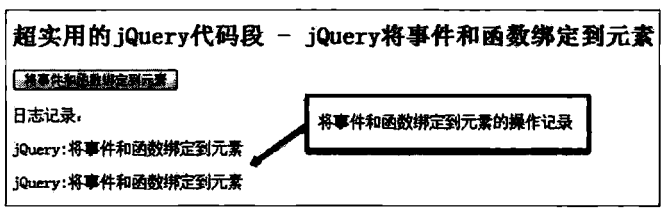

图 2.12　将事件和函数绑定到元素效果（二）

jQuery 框架的优势之一就是可以将事件和函数绑定到 HTML 元素上，既简化了传统 JavaSrcipt 代码冗长的编写方式，又美化了 jQuery 代码的书写效果，可谓一举两得，这也是广大程序员如此喜爱 jQuery 框架的原因之一。

本例实现绑定的关键代码是第 03~05 行，其中 bind()函数为被选元素添加一个或多个事件处理程序，并定义事件发生时运行的函数。具体写法如下：

```
$(selector).bind(event,data,function);
```

其中 event 参数规定添加到元素的一个或多个有效事件，data 参数定义传递到函数的额外数据，该参数是可选择的参数；function 参数为当事件发生时运行的函数。

2.10　如何从元素中除去 HTML 标签

如何从元素中去除 HTML 标签呢？去除 HTML 标签后的页面会是什么效果呢？其实，掌握该方法可以对标记语言与文本文字进行自由转换，可以将具有不同样式风格的页面转化为仅含有文本文字的页面。本例主要代码如下：

```
01  /* stripHTMLTag.html */
02  <script type="text/javascript">
03  (function($){
04      $.fn.stripHtmlTag=function(){
05          var regexp = /<("[^"]*"|'[^']*'|[^'">])*>/gi;
06          this.each(function(){
07              $(this).html($(this).html().replace(regexp,''));
```

```
        });
    return $(this);
};
})(jQuery);
$(function(){
    $("#id-button-striphtmltag").click(function(){
        $("#div-striphtmltag").stripHtmlTag();
    });
});

</script>
<body>
<h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 如何从元素中除去 HTML 标签</h2>
<input type="button" id="id-button-striphtmltag" value="从元素中除去 HTML 标签"/><br>
<div id="div-striphtmltag">
<p>
    <table border="1" style="padding:2px;">
        <tr>
            <td>1</td>
            <td>jQuery</td>
            <td>如何从元素中除去 HTML 标签</td>
        </tr>
        <tr>
            <td>2</td>
            <td>jQuery</td>
            <td>如何从元素中除去 HTML 标签</td>
        </tr>
        <tr>
            <td>3</td>
            <td>jQuery</td>
            <td>如何从元素中除去 HTML 标签</td>
        </tr>
    </table>
</p>
</div>
</body>
```

本例效果分别如图 2.13 和图 2.14 所示。

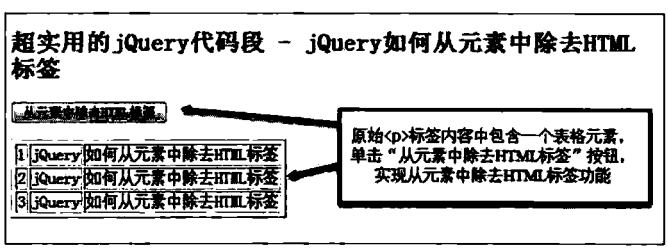

图 2.13　从元素中除去 HTML 标签效果（一）

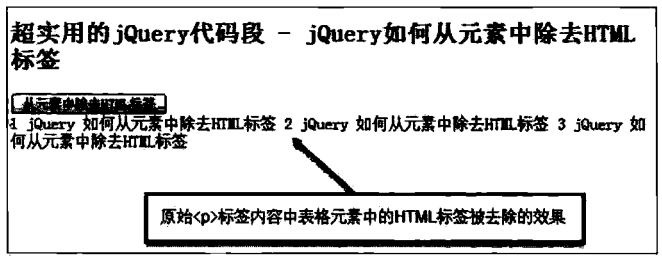

图 2.14　从元素中除去 HTML 标签效果（二）

有些情况下需要将 HTML 元素中包含的 HTML 标签去除，使用 jQuery 框架并借助正则表达式可以很轻松地实现此功能。实现的关键代码是第 04~09 行，其中 each()函数为每个匹配元素指定要运行的函数。

提示：设计人员使用 jQuery 框架的 each()遍历函数时，如果想提前终止遍历操作，可以通过返回 false 值来终止。

2.11　如何限制文本域中字符的个数

使用过网银的用户都会留意到，在输入银行卡或信用卡的时候，输入字符的个数是严格限制好位数的，输入位数不够将无法进行下一步操作，同样想多输入几位键盘也会不听使唤。其实是页面的文本域被严格限制了输入字符的个数，这样可以有效避免一些误操作带来的无法预知的严重后果。本例效果分别如图 2.15 和图 2.16 所示。

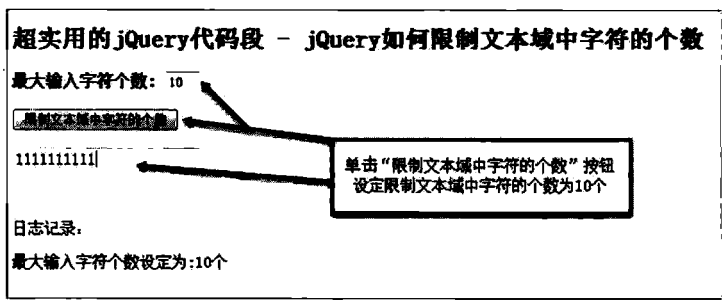

图 2.15　限制文本域中字符的个数（一）

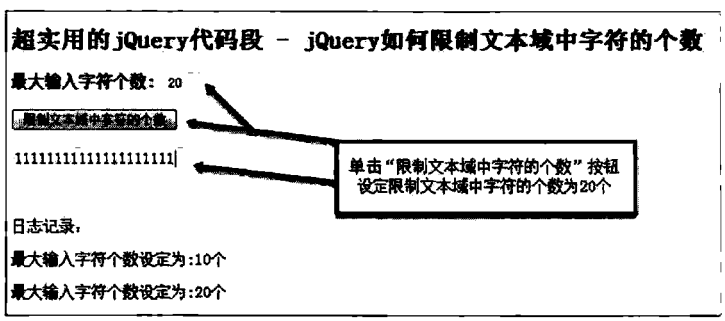

图 2.16　限制文本域中字符的个数（二）

本例主要代码如下：

```
/* maxLenTextArea.html */
<script type="text/javascript">
$(function(){
    jQuery.fn.maxLength=function(max){
        return this.each(function(){
            var type=this.tagName.toLowerCase();    //获取文本域类型
            var inputType=this.typ?this.type.toLowerCase():null;
            //判断 input、text 和 password 文本域类型
            if(type=="input"&&inputType=="text"||inputType=="password"){
                this.maxLength=max;
            }else if(type=="textarea"){                    //判断 textarea 文本域类型
                //onkeypress 事件处理过程
                this.onkeypress=function(e){
                    var ob=e||event;
                    var keyCode=ob.keyCode;        //键盘输入编码
                this.onkeyup=function(){                //onkeyup 事件处理过程
                    if(this.value.length>max){ //判断文本域中字符个数是否大于设定值
                        this.value=this.value.substring(0,max);    //截断多余字符
                    }
                };
            }
        });
    };
    $("#id-button-maxLen").click(function(){
        var vMaxLen=$('input#id-text-maxLen').val();
        var iMaxLen=parseInt(vMaxLen);
        $('#id-maxLenTextarea').maxLength(iMaxLen);
        $("#div-log").html($("#div-log").html()+"<p>"+"最大输入字符个数设定
为:"+vMaxLen+"个"+"</p>");
    });
});
</script>
<body>
<h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 如何限制文本域中字符的个数</h2>
<p>
    <b>最大输入字符个数:</b>
    <input type="text" id="id-text-maxLen" style="width:32px;" value="10"/>
</p>
<p>
    <input type="button" id="id-button-maxLen" value="限制文本域中字符的个数"/>
</p>
<div>
    <textarea id="id-maxLenTextarea" name="maxLenTextarea">
```

```
43          </textarea>
44      </div>
45      <div id="div-log">
46          <p>日志记录：</p>
47      </div>
48  </body>
```

本例向读者介绍如何基于jQuery框架实现限制文本域中字符的个数。用JavaScript脚本语言限制一个文本域（譬如：文本框TextArea控件）输入字符的个数，并适时显示可以继续输入的字符长度，此功能常见于网页留言板或评论上，是一个限制用户不能超过多少字符的页面效果。

本例关键代码有两处，第一处为第13行代码开始的监控键盘按键按下的onkeypress事件，该事件会在键盘按键被按下并释放一个按键时触发，在事件内部获取键盘输入的编码；第二处为第16行代码开始的监控键盘按键被松开的onkeyup事件，该事件会在键盘按键被松开时被触发，在事件内部首先判断文本域中字符个数是否大于设定值，如果大于则截断多余字符。

提示： document.selection对象表示当前激活选中的区域，即高亮文本块。对于document.selection对象的典型用途就是作为用户的输入，以便识别正在对文档的哪一部分进行操作，或者作为某一操作的结果输出给用户。

2.12 如何选中页面上的所有复选框

如何选中页面上的所有复选框？实现这样的页面快速操作功能是十分有用的，譬如在购物网站上选择购物车内的商品时，如果想买下自己选好的全部商品时，会有一个“选择全部”的按钮供用户使用，既方便又快捷。

本例主要代码如下：

```
01  /* checkboxAll.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      //全选
05      $("#allselect").click(function(){
06          $(":checkbox").each(function(){
07              $(this).prop("checked",true);
08              $(this).next().css({"background-color":"blue","color":"White" });
09          });
10      });
11      //反选
12      $("#invert").click(function(){
13          $(":checkbox").each(function(){
14              if($(this).prop("checked")){
15                  $(this).prop("checked",false);
```

```
                $(this).next().css({"background-color": "White","color":"black" });
            }else{
                $(this).prop("checked",true);
                $(this).next().css({"background-color": "blue","color":"White" });
            }
        });
    });
    //取消
    $("#cancel").click(function(){
        $(":checkbox").each(function(){
            $(this).prop("checked",false);
            $(this).next().css({"background-color": "White","color":"black" });
        });
    });
    //所有复选(:checkbox)框单击事件
    $(":checkbox").click(function(){
        if($(this).prop("checked")){
            $(this).next().css({"background-color": "blue","color":"White" });
        }else{
            $(this).next().css({"background-color": "White","color":"black" });
        }
    });
    //输出
    $("#output").click(function(){
        $(":checkbox").each(function(){
            if($(this).prop("checked")){
                alert($(this).val());
            }
        });
    });
});
</script>
<body>
    <h2 id="h2-caption">超实用的jQuery代码段 - jQuery复选框 全选 反选 取消 选中输出
等功能</h2>
    <hr><br>
    <div class="mar20">
        <input name="newslist" id="newslist-1" type="checkbox" value="1" /><label
for="newslist-1">超实用的 jQuery 代码段<a href="#">超实用的 jQuery 代码段</a></label>
    </div>
    /* 省略部分代码，此处可以添加用户代码 */
    <div class="mar20">
        <input name="allselect" id="allselect" type="button" value="全选" />
        <input name="invert" id="invert" type="button" value="反选" />
        <input name="cancel" id="cancel" type="button" value="取消" />
```

```
61              <input name="output" id="output" type="button" value="输出" />
62          </div>
63          <br><hr>
64          <div id="div-log">
65          </div>
66      </body>
```

本例第40~44行的关键代码如下：

```
$(":checkbox").each(function(){
    $(this).prop("checked",true);
});
```

其中，prop(propertyName)函数获取匹配的元素集中第一个元素的属性值。参数propertyName表示要得到的属性的名称。prop(propertyName,value)函数为匹配的元素设置一个或更多的属性。参数value是要为属性设置的值。

提示：如果prop()函数中元素的一个属性都没有设置，或者如果没有匹配的元素，则返回undefined值。为了能为每个元素设置单独的值，可使用循环结构(如.each()函数或.map()函数）来实现。

2.13 禁用表单的回车键提交

在很多种情况下，为了防止用户误操作，需要禁用表单的回车键提交。本例效果如图2.17所示。

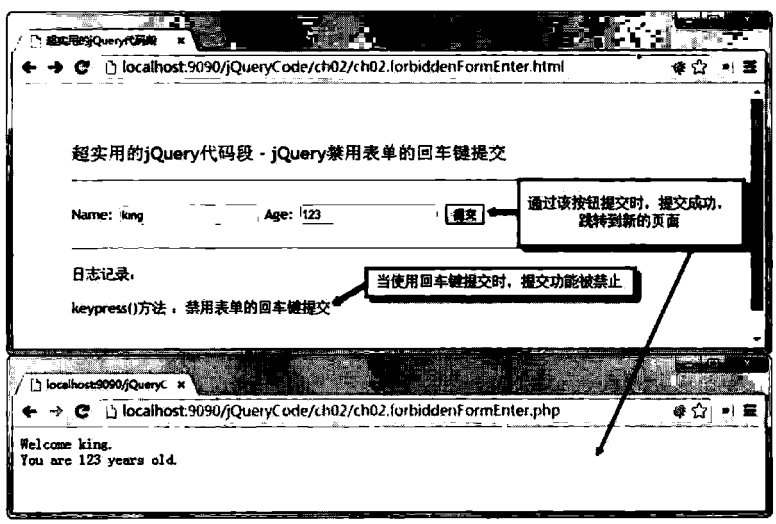

图2.17 禁用表单的回车键提交

本例主要代码如下：

```
01  /* forbiddenFormEnter.html */
02  <script type="text/javascript">
```

```
$(function(){
    $("input").keypress(function(e){
        var keyCode=e.keyCode?e.keyCode:e.which?e.which:e.charCode;
        if(keyCode==13){
            for(var i=0;i<this.form.elements.length;i++){
                if(this==this.form.elements[i]) break;
            }
            i=(i+1)%this.form.elements.length;
            this.form.elements[i].focus();
            return false;
        }else{
            return true;
        }
    });
});
</script>
<body>
    <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 禁用表单的回车键提交</h2>
    <hr><br>
    <form action="ch02.forbiddenFormEnter.php" method="post">
        Name: <input type="text" name="name"/>
        Age: <input type="text" name="age"/>
        <input type="submit" value="提交"/>
    </form>
    /* 省略部分代码，此处可以添加用户代码 */
</body>
```

本例的原理主要是对表单的 input 控件绑定 keypress 按键事件，然后监控用户的输入事件，我们都知道回车键的二进制编码，只要针对该编码进行过滤就可以了。

keypress()的语法如下：

```
$(selector).keypress();                    //触发 keypress 事件
or
$(selector).keypress(function);            //将函数绑定到 keypress 事件
```

其中，keypress 事件表示当按键被按下时触发，其发生在当前获得焦点的元素上。keypress(function)函数也是实现一样的功能，但它还实现了将函数绑定到 keypress 事件的功能。其中，function 参数为可选项，该参数规定当发生 keypress 事件时要运行的函数。

关于过滤回车键的操作，需要判断用户输入按键的编码是否等于 13（注：回车键字节编码为 13），实现代码如下：

```
if(keyCode==13){                    //判断用户是否按下回车键
    //用户代码，实现屏蔽回车键提交功能
    return false;
}else{
    return true;
}
```

2.14 禁用右键单击上下文菜单

设计人员有时为了保护页面内容的版权，会禁用右键单击上下文菜单，这样就不能进行复制操作了。本例效果如图2.18所示。

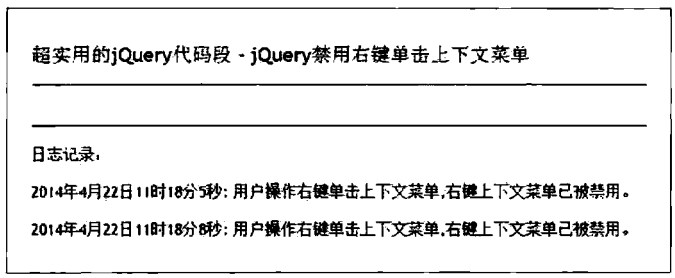

图2.18 禁用右键单击上下文菜单

本例主要代码如下：

```
/* forbiddenContextmenu.html */
<script type="text/javascript">
$(function(){
    $(document).bind("contextmenu",function(e){
        $("#div-log").html($("#div-log").html()+"<p>"+current()+": "+"用户操作右键单击上下
文菜单,右键上下文菜单已被禁用。"+"</p>");
        return false;
    });
    function current(){
        var d=new Date(),str='';
        str+=d.getFullYear()+'年';
        str+=d.getMonth()+1+'月';
        str+=d.getDate()+'日';
        str+=d.getHours()+'时';
        str+=d.getMinutes()+'分';
        str+=d.getSeconds()+'秒';
        return str;
    }
});
</script>
<body>
    <h2 id="h2-caption">超实用的jQuery代码段 - jQuery禁用右键单击上下文菜单</h2>
    <hr><br>
    <br><hr>
    /* 省略部分代码，此处可以添加用户代码 */
</body>
```

本例的原理主要是通过对 HTML 5 的全局 contextmenu 属性进行编程实现的。contextmenu属性为元素规定上下文菜单（默认为浏览器上下文菜单），该菜单会在用户右

击元素时出现。

提示：contextmenu 属性是 HTMl 5 标准的新功能，需要支持 HTML 5 标准的浏览器来实现页面效果。

2.15 IE 下禁用文本选择功能

同样在 IE 浏览器下，为了保护页面内容的版权，需要禁用文本选择功能。本例效果如图 2.19 所示。

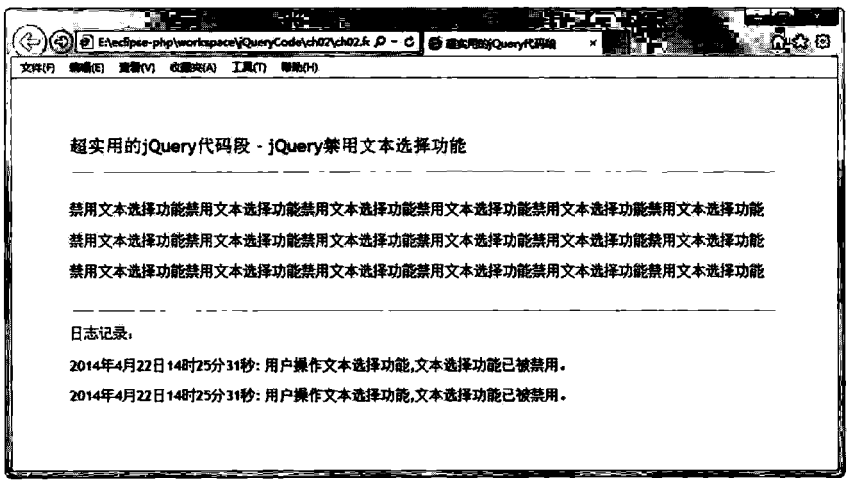

图 2.19 IE 下禁用文本选择功能

本例主要代码如下：

```
/* forbiddenSelectStart.html */
<script type="text/javascript">
$(function(){
    $(document).bind("selectstart",function(){
        $("#div-log").html($("#div-log").html()+"<p>"+current()+": "+"用户操作文本选择功能，
文本选择功能已被禁用。"+"</p>");
        return false;
    });
    /* 省略部分代码，此处可以添加用户代码 */
});
</script>
<body>
    <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 禁用文本选择功能</h2>
    <hr><br>
    <p>禁用文本选择功能禁用文本选择功能禁用文本选择功能禁用文本选择功能禁用文本选
择功能禁用文本选择功能</p>
    /* 省略部分代码，此处可以添加用户代码 */
```

```
18  </body>
```

本例的原理主要是通过对 HTML 5 的全局 selectstart 属性进行编程实现的。selectstart 属性具体的说明如下：

contextmenu 属性几乎可以用于所有页面对象，其触发时间为目标对象被开始选中时（即选中动作刚开始，尚未实质性被选中）。

本例的关键代码是第 04~08 行，主要是通过绑定 selectstart 属性并返回布尔值“否”来实现。

提示： selectstart 属性仅仅支持 IE 浏览器，但其不被<input>和<textarea>标签支持。所以在 BODY 中绑定 selectstart 属性并返回 false 后，虽然文本不让选择，但是页面上的文字输入框可以设置成可选择。

2.16 输入框获取焦点时文本高亮提示

输入框获取焦点时让输入框内的文本高亮提示，具有提醒用户的功能。本例效果如图 2.20 所示。

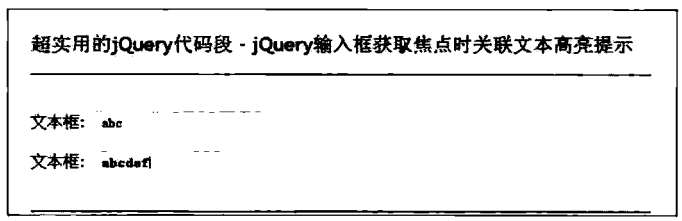

图 2.20 输入框获取焦点时文本高亮提示

本例主要代码如下：

```
01  /* inputFocusHighlight.html */
02  <script type="text/javascript">
03  window.onload=function(){
04      for(var i=0;i<$("input").length;i++){
05          $("input")[i].onfocus=function(){
06              this.className="css-text-high";
07          };
08          $("input")[i].onblur=function(){
09              this.className="css-text";
10          };
11      }
12  };
13  </script>
14  <body>
15      <h2 id="h2-caption">超实用的jQuery代码段 - jQuery输入框获取焦点时关联文本高亮提
16  示</h2>
17      <hr><br>
```

```
18      <form>
19          <h2>文本框高亮提示</h2>
20          <p><label>文本框:  </label><input class="css-text" type="text"/></p>
21          <p><label>文本框:  </label><input class="css-text" type="text"/></p>
22      </form>
23      /* 省略部分代码，此处可以添加用户代码 */
24  </body>
```

本例的原理主要是通过对 onfocus 和 onblur 事件进行编程实现的，onfocus 事件在对象获得焦点时发生，onblur 事件会在对象失去焦点时发生。本例的关键代码是第 05 行和第 08 行，在 onfocus 和 onblur 事件函数内，通过改变文本框内文本的样式表来实现高亮提示功能。

提示：在较早的浏览器版本中，onblur 事件仅发生于表单元素上，而在较新的浏览器中，该事件可用于任何元素。

2.17 实现多个输入框同步操作

在有些情况下为了提高效率，需要实现多个输入框同步操作，以提高人机交互的友好性。本例效果如图 2.21 所示。

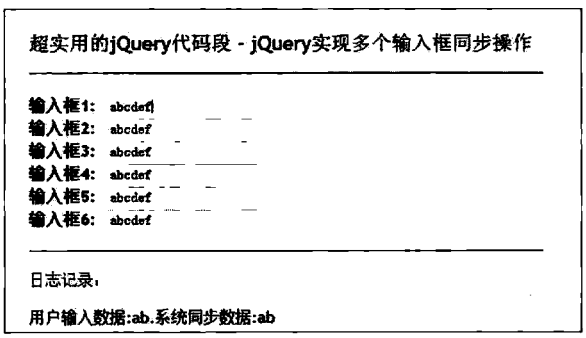

图 2.21 实现多个输入框同步操作

本例主要代码如下：

```
01  /* synInput.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $(":text").bind("keyup",function(){     //绑定多个具有相同样式类的文本框的 keyup 事件
05          $(":text").val($(this).val());      //对多个具有相同样式类的文本框进行同步输入
06          /* 省略部分代码，此处可以添加用户代码 */
07      });
08  });
09  </script>
10  <body>
11      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现多个输入框同步操作</h2>
12      <hr><br>
```

```
13        <form id="id-form-syninput">
14            <b>输入框 1:  </b><input type="text" name="input-syninput-1"/><br/>
15            /* 省略部分代码，此处可以添加用户代码 */
16        </form>
17        /* 省略部分代码，此处可以添加用户代码 */
18    </body>
```

本例的原理主要是通过对 keyup 事件进行编程实现的。一个完整的 keypress 过程分为两个部分：按键被按下；按键被松开并复位。当按钮被松开时，触发 keyup 事件。

本例的关键代码是第 04~07 行，在 keyup()事件函数内，通过对多个具有相同样式类的文本框进行同步文本输入来实现多个输入框同步操作。

2.18　在新窗口中打开外部链接

有时设计人员为了满足页面特殊功能，需要在新窗口中打开外部链接。本例效果如图 2.22 所示。

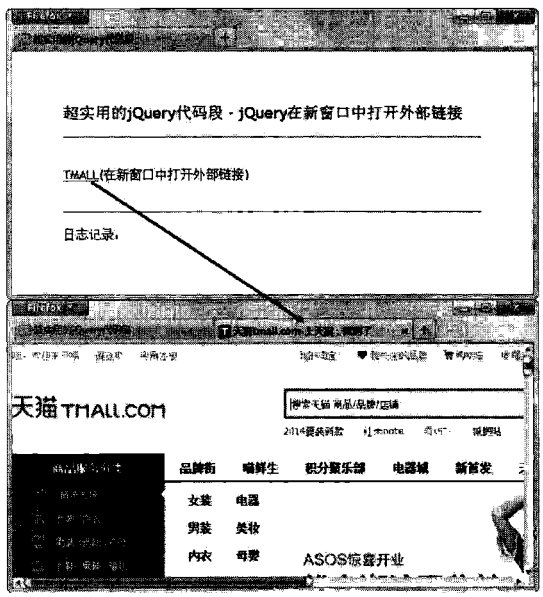

图 2.22　在新窗口中打开外部链接

本例主要代码如下：

```
01  /* openNewLink.html */
02  <script type="text/javascript">
03  jQuery(document).delegate('a','click',function(){        //对超链接<a>的 click 事件进行委托
04      var root=location.href.replace(location.pathname+location.search+location.hash,'');
05      if(!this.href) return;
06      if(this.href.indexOf(root)!=0){
07          window.open(this.href);                          //在新的页面打开链接节点地址
```

```
08              return false;
09          }
10  });
11  </script>
12  <body>
13      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 在新窗口中打开外部链接</h2>
14      <hr><br>
15      <p>
16  <a href="http://www.tmall.com/?spm=1.6659421.754904973.1.e6ux8o">TMALL</a>
17      </p>
18      /* 省略部分代码，此处可以添加用户代码 */
19  </body>
```

本例的原理主要是通过对超链接 a 元素的 click 事件进行委托编程实现的。delegate()函数为指定的元素（属于被选元素的子元素）添加一个或多个事件处理程序，并规定当这些事件发生时运行的函数。语法如下：

```
$(selector).delegate(childSelector,event,data,function);
```

其中 childSelector 参数规定要附加事件处理程序的一个或多个子元素；event 参数规定附加到元素的一个或多个有效事件；data 参数规定传递到函数的额外数据；function 参数规定当事件发生时运行的函数。

本例的关键代码是第 03~09 行，在 delegate()函数内，首先获取当前路径的根节点，然后对该节点进行判断，如果是空节点则不进行任何操作，如果节点有效则通过 window.open()函数在新的页面打开该节点的链接。

提示：使用 delegate()函数的事件处理程序适用于当前或未来的元素（比如由脚本创建的新元素）。

2.19 jQuery 实现 outerHTML 属性

在 Javascript 语言规范中，outerHTML 是一个功能十分强大的属性。outerHTML 包括整个标签（标签就是 这类标记），而不仅限于标签内部的内容。遗憾的是，在 jQuery 框架中却没有 outerHTML 的相关实现。本例通过插件的方式实现了 jQuery 版本的 outerHTML。

本例主要代码如下：

```
/* outerHTML.html */
01  <script type="text/javascript">
02  $(function(){
03      $.fn.outerText=function(){              //模拟实现 outerText 属性
04          return $('<div/>').append(this.clone()).html();
05      };
06      jQuery.fn.outerHTML=function(s){    //模拟实现 outerHTML 属性
07      return (s)?this.before(s).remove():jQuery("&lt;p&gt;").append(this.eq(0).clone()).html();
```

```
08          };
09          $('#getOuterHTML').change(function(){
10               $("#div-log").html($("#div-log").html()+$('#id-outerHTML').outerText());
11          });
12          $('#setOuterHTML').change(function(){
13               $('#id-outerHTML').outerHTML('<select name="id-outerHTML"
14     id="id-outerHTML"><option value="1" selected="selected">outerHTML</option>
15     <option value="2">outerHTML</option><option value="3">outerHTML</option></select>');
16          });
17     });
18     </script>
19     <body>
20          <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现 outerHTML</h2>
21          /* 省略部分代码，此处可以添加用户代码 */
22     </body>
```

本例的原理主要是通过 jQuery 自定义函数实现的。如代码第 03~05 行所示，在自定义函数 outerText()内，通过 clone()函数复制外部文本并显示在后面的 div 元素中，以模拟实现 outerText 属性；根据代码第 06~08 行所示，在自定义函数 outerHTML()内，将由参数 s 传递进来的字符串编码转换为 HTML 页面代码并显示在 p 元素中，以模拟实现 outerHTML 属性。

提示：关于函数扩展需要说明一下。jQuery.extends()是直接扩展，而 jQuery.fn.extends()很明显是扩展的原型，其源自较早的 ProtoType 框架，这就是为什么 jQuery.fn.extends()中的大部分方法来自于 jQuery.extends()的原因。

2.20 实现带固定表头的表格

相信使用过 Microsoft Office Word 和 Excel 办公软件的用户对于固定表头的印象很深刻。有时为了满足页面特殊功能，需要在网页表格中实现固定表头的功能。本例效果如图 2.23 所示。

Check out this header	Look here's another one	Wow, look at me!
Quisque in wisi quis orci tincidunt fermentum	Mauris aliquet mattis metus	Etiam eu ante non leo egestas nonummy
Some content goes in here	Praesent vitae ligula nec orci pretium vestibulum	Maecenas tempus dictum libero
Quisque in wisi quis orci tincidunt fermentum	Mauris aliquet mattis metus	Etiam eu ante non leo egestas nonummy
Some content goes in here	Praesent vitae ligula nec orci pretium vestibulum	Maecenas tempus dictum libero
Quisque in wisi quis orci tincidunt fermentum	Mauris aliquet mattis metus	Etiam eu ante non leo egestas nonummy
Some content goes in here	Praesent vitae ligula nec orci pretium vestibulum	Maecenas tempus dictum libero
Quisque in wisi quis orci tincidunt fermentum	Mauris aliquet mattis metus	Etiam eu ante non leo egestas nonummy
Some content goes in here	Praesent vitae ligula nec orci pretium vestibulum	Maecenas tempus dictum libero
Quisque in wisi quis orci tincidunt fermentum	Mauris aliquet mattis metus	Etiam eu ante non leo egestas nonummy
Some content goes in here	Praesent vitae ligula nec orci pretium vestibulum	Maecenas tempus dictum libero
Quisque in wisi quis orci tincidunt fermentum	Mauris aliquet mattis metus	Etiam eu ante non leo egestas nonummy

图 2.23 在表格中实现固定表头

本例主要代码如下：

```
01  /* fixedTableHeader.html */
02  <script type="text/javascript" src="js/jquery.chromatable.js"></script>
03  <script type="text/javascript">
04  $(document).ready(function(){
05      /* 省略部分代码，此处可以添加用户代码 */
06      $("#yourTableID").chromatable({
07          width: "900px",
08          height: "400px",
09          scrolling: "yes"
10      });
11  });
12  </script>
13  <body>
14  <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现表头固定效果</h2>
15  <hr><br>
16  <table id="yourTableID2" width="100%" border="0" cellspacing="0" cellpadding="0">
17  <thead>
18      /* 省略部分代码，此处可以添加用户代码 */
19  </thead>
20  <tbody>
21      <tr>
22          <td>Some content goes in here</td>
23          <td>Praesent vitae ligula nec orci pretium vestibulum</td>
24          <td>Maecenas tempus dictum libero</td>
25      </tr>
26  /* 省略部分代码，此处可以添加用户代码 */
27      </tbody>
28  </table>
29  </body>
```

当滚动条向下移动时，表头位置固定不变。代码第 06~10 行用到了 jquery.chromatable 插件，使用它对页面中定义好的 table 表格控件进行绑定就可以实现固定表头。jquery.chromatable 插件内置了一些非常实用的配置参数，可以定义表格的外观，具体说明详见表 2.1。

表 2.1　jquery.chromatable插件参数说明

参数名称	默认值	参数说明
width	可选	定义表格宽度，默认值为900px
height	可选	定义表格高度，默认值为300px
scrolling	可选	定义表格是否支持滚动条操作，默认值为“yes” Scrolling参数的属性值为“yes”或“no”，“yes”表示表格支持滚动条操作，“no”表示相反

提示：感兴趣的设计人员可以阅读 jquery.chromatable 插件的源代码文件，根据实际需求可以扩展其参数来实现更为复杂的功能。

2.21 为表单内控件设定缺省数值和文本

网页上需要我们填写的内容越来越多，有时候真是看得人眼花缭乱，经常漏填各种信息，如果系统能为我们漏填的信息设置默认值就太好了。本例效果如图2.24所示，当用户焦点离开控件时，该控件自动显示缺省数值和文本。

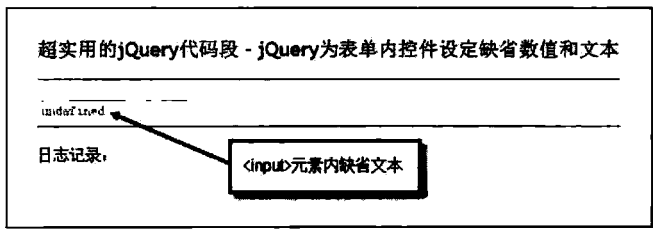

图2.24 为表单内控件设定缺省数值和文本

本例主要代码如下：

```
/* inputDefaultText.html */
<script type="text/javascript">
window.onload=function(){
    var o=$("#input-defaultText");
    o.setAttribute("valueCache",o.value);
    o.onblur=function(){
        if(o.value==""){
            o.valueCache="";
            o.value=o.tips;
        }else
            o.valueCache=o.value;
    };
    o.onfocus=function(){
        o.value=o.valueCache;
        var e=event.srcElement;
        var r=e.createTextRange();
        r.moveStart('character',e.value.length);
        r.collapse(true);
        r.select();
    };
    o.onblur();
};
</script>
<body>
    <h2 id="h2-caption">超实用的jQuery代码段 - jQuery为表单内控件设定缺省数值和文本
</h2>
    <hr><br>
    <input id="input-defaultText" type="text" name="" style="color:#808080;" value=""/>
```

```
29        /* 省略部分代码，此处可以添加用户代码 */
30    </body>
```

本例实现原理是通过 setAttribute()函数为该控件设定一个名称为“valueCache”的缺省值，然后对该控件绑定 onblur()与 onfocus()事件来实现。当触发 onblur()事件时，自动为该控件设定缺省数值和文本。

setAttribute()函数用于把指定的属性设置为指定的值，其语法如下：

```
elementNode.setAttribute(name,value);
```

提示：如果 setAttribute()函数不存在具有指定名称的属性，该函数将创建一个新属性。

2.22 防止单个页面重复提交按钮

有时候由于网络速度不给力，用户往往在提交按钮后得不到反馈信息而反复提交按钮，其实在用户数据已经上传服务器的情况下，这样的操作只会带来更不好的作用。因此，对于用户页面表单而言，防止单个页面重复提交按钮是一个很重要的功能。本例主要代码如下：

```
01  /* forbiddenSubmit.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $("#submit").click(function(){
05          $("#submit").attr("disabled","disabled");          //禁用提交按钮
06          var dataString="name=text";
07          $.ajax({
08              type:"POST",                                   //定义 AJAX 通信方式
09              url:"ch02.forbiddenSubmit.php",                //定义 AJAX 通信地址
10              data:dataString,                               //定义 AJAX 通信数据
11              success:function(){                            //操作成功后返回函数
12                  $("#submit").removeAttr("disabled");       //恢复提交按钮功能
13              }
14          });
15          return false;
16      });
17  });
18  </script>
19  <body>
20      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 防止单个页面重复提交按钮方法
21  </h2>
22      <hr><br>
23      <form>
24          Name:<input type="text" name="name" id="name"/>
25          <input type="submit" name="submit" id="submit"/>
26      </form>
```

```
27        /* 省略部分代码，此处可以添加用户代码 */
28    </body>
29    </html>
```

首先，绑定提交按钮的单击事件，在该事件函数内通过 attr()函数先禁用该提交按钮的单击功能，原因是此时用户已经提交过一次了，禁用该提交按钮防止用户反复操作提交功能。然后，使用 ajax()函数实现异步数据通信功能，详见代码第 07~14 行，这里分别设定了通信方式、通信参数与通信地址。最后，如果 AJAX 通信操作成功，则在返回函数内重新激活提交按钮的功能。

提示：jQuery 框架的 ajax()函数是 jQuery 底层的 AJAX 实现。$.ajax()函数返回其创建的 XMLHttpRequest 对象，大多数情况下设计人员无需直接操作该函数，除非需要操作不常用的选项以获得更多的灵活性。

2.23 取得列表控件选中的 option 对象

在使用 select 列表控件时，需要取得列表控件选中的 option 对象，根据结果来决定下一步的操作。本例效果如图 2.25 所示。

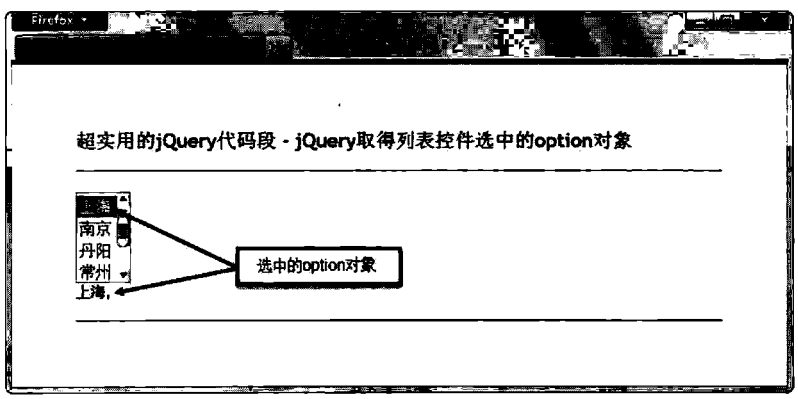

图 2.25 取得列表控件选中的 option 对象

本例主要代码如下：

```
01  /* getDefaultOptions.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04        $("#s1").change(function(){
05              var str="";
06              $("select :selected").each(function(){
07                    str += $(this).text() + ',';
08              });
09              $("#div1").html('<b>' + str + '</b>');
10        });
11  });
```

```
12  </script>
13  </head>
14  <body>
15      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 取得列表控件选中的 option 对象
16  </h2>
17      <hr><br>
18      <form>
19      <select id="s1" multiple="multiple">
20          <option>苏州</option>
21          <option selected="selected">上海</option>
22          <option>南京</option>
23          <option>丹阳</option>
24          <option>常州</option>
25          <option>杭州</option>
26          <option>无锡</option>
27      </select>
28      <div id="div1">option 信息</div>
29      </form>
30      /* 省略部分代码，此处可以添加用户代码 */
31  </body>
32  </html>
```

本例实现原理如下：

（1）通过监听 change 事件来判断用户是否操作选择了列表选项。change 事件是当元素的值发生改变时所触发的事件。change 事件适用于文本域（text field）、textarea 和 select 元素。jQuery 框架通过 change()函数触发 change 事件，或规定当发生 change 事件时运行的函数。具体语法如下：

语法：$(selector).change(function);

其中 function 参数为可选项，该参数定义当 change 事件发生时运行的函数。

（2）在 change 事件定义的触发函数内，通过查询具体是哪个 option 选项被选中来获取该 option 对象属性值，如代码第 06~07 行所示。

（3）将获取的 option 对象属性值显示在页面定义好的 div 元素中，如代码第 09 行所示。

提示：change 事件用于 select 元素时，会在选择某个选项时触发；用于 text field 或 text area 时，会在元素失去焦点时触发。

2.24 限制输入框仅接受特殊字符的输入

在很多种页面表单中，需要限制输入框仅接受特殊字符的输入（如输入银行卡卡号时仅接受数字），来避免一些用户误操作。本例效果如图 2.26 所示。

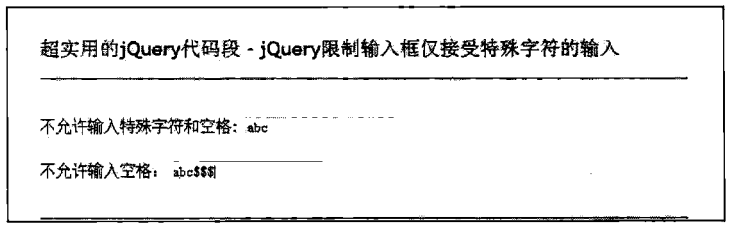

图 2.26　限制输入框仅接受特殊字符的输入

本例主要代码如下：

```
/* forbiddenSpecChar.html */
<script type="text/javascript">
function ValidateSpecialCharacter(){
    var code;
    if(document.all) {                    //判断是否是 IE 浏览器
        code = window.event.keyCode;
    }else{
        code = arguments.callee.caller.arguments[0].which;
    }
    var character = String.fromCharCode(code);
    //特殊字符正则表达式
    var txt=new
RegExp("[ ,\\`,\\~,\\!,\\@,\#,\\$,\\%,\\^,\\+,\\*,\\&,\\\\,\\/,\\?,\\|,\\:,\\.,\\<,\\>,\\{,\\},\\(,\\),\\",\\;,\\=,\"]");
    if(txt.test(character)){
        if(document.all){
            window.event.returnValue=false;
        }else{
            arguments.callee.caller.arguments[0].preventDefault();
        }
    }
}
/* 省略部分功能代码，此处可以添加用户代码 */
</script>
<body>
    <h2 id="h2-caption">超实用的jQuery代码段 - jQuery限制输入框仅接受特殊字符的输入
</h2>
    <hr><br>
    <p>
    不允许输入特殊字符和空格:<input id="code" onkeypress="return
ValidateSpecialCharacter();" onblur="validate(this)"/><br/><br/>
    不允许输入空格：<input id="space" onkeyup="value=value.replace(/\s/g,'')"/>
    </p>
    /* 省略部分代码，此处可以添加用户代码 */
</body>
```

本例实现了“不允许输入特殊字符和空格”与“不允许输入空格”两个子功能。

对于“不允许输入特殊字符和空格”功能，将会屏蔽用户输入的特殊字符与空格，其具体实现原理如下：

（1）创建一个名称为 ValidateSpecialCharacter()的函数，先定义判断特殊字符的正则表达式，定义方法如下：

```
var txt=new RegExp("[ ,\\`,\\~,\\!,\\@,\#,\\$,\\%,\\^,\\+,\\*,\\&,\\\\,\\/,\\?,\\|,\\:,\\.,\\<,\\>,\\{,\\},\\(,\\),\\",\\;,
\\=,\"]");
```

上面的代码使用 JavaScript 的 RegExp 对象来判断特殊字符，RegExp()函数定义正则表达式，它是对字符串执行模式匹配最有效最强大的工具。其语法如下：

```
new RegExp(pattern,attributes);
```

其中参数说明详见表 2.2。

表 2.2 RegExp()函数参数说明

参数名称	默认值	参数说明
pattern	必选	该参数是一个字符串，指定了正则表达式的模式或其他正则表达式
attributes	可选	该参数包含属性g、i和m，分别用于指定全局匹配、区分大小写的匹配和多行匹配
返回值		返回一个新的RegExp对象，具有指定的模式和标志

（2）通过 RegExp 对象的测试方法辨别用户输入的特殊字符与空格，并取消事件的默认动作，如代码第 14~20 行所示。

（3）在 input 元素中定义 onkeypress 事件函数实现“不允许输入特殊字符和空格”功能。具体代码如下：

```
onkeypress="return ValidateSpecialCharacter();"
```

对于“不允许输入空格”功能，其实现方法就简单一些，具体代码如下：

```
onkeyup="value=value.replace(/\s/g,'')"
```

以上代码通过 replace()函数替换用户输入的空格，正则表达式[/\s/g]的含义是匹配全局所有空格字符，然后使用空字符替换掉。

提示：如果 pattern 不是合法的正则表达式，或 attributes 含有 g、i 和 m 之外的字符，则抛出 SyntaxError 异常；如果 pattern 是 RegExp 对象，但没有省略 attributes 参数，抛出 TypeError 异常。

2.25 禁止页面内全部超链接

对于那些仅仅有阅读页面表单权限的用户，禁止页面内全部超链接是有效限制用户操作的方法，本例就来实现禁止页面内全部超链接的方法，具体效果如图 2.27 所示。

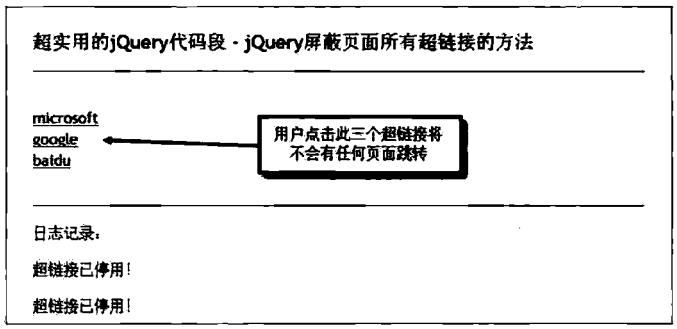

图 2.27　禁止页面内全部超链接

本例主要代码如下：

```
01  /* forbiddenHyberLink.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      $("a").attr('href','#');
05      $("a").click(function(event){
06          event.preventDefault();
07          $("#div-log").html($("#div-log").html()+"<p>"+"超链接已停用！ "+"</p>");
08      });
09  });
10  </script>
11  <body>
12      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 屏蔽页面所有超链接的方法</h2>
13      <hr><br>
14      <a href="http://www.microsoft.com">microsoft</a><br>
15      <a href="http://www.google.com">google</a><br>
16      <a href="http://www.baidu.com">baidu</a><br>
17      /* 省略部分代码，此处可以添加用户代码 */
18  </body>
```

本例的关键代码是第 04~08 行。首先将全部超链接的 href 属性赋值为#（表示跳转地址为本页），然后绑定全部超链接的单击事件，在事件函数内使用 preventDefault()函数阻止元素的默认行为。preventDefault()阻止元素的默认行为（例如，当单击“提交”按钮时阻止对表单的提交），其语法如下：

```
event.preventDefault();
```

event 参数来自事件绑定函数，表示要阻止的是哪个事件的默认动作。

提示：preventDefault()函数并不被 IE 浏览器支持，在 IE 下需要用 window.event.returnValue=false 来实现。

2.26　实现动态组合列表框

在页面表单中实现动态组合列表框，可以提高页面的可操作性与用户的交互体验。本

例效果如图 2.28 所示。

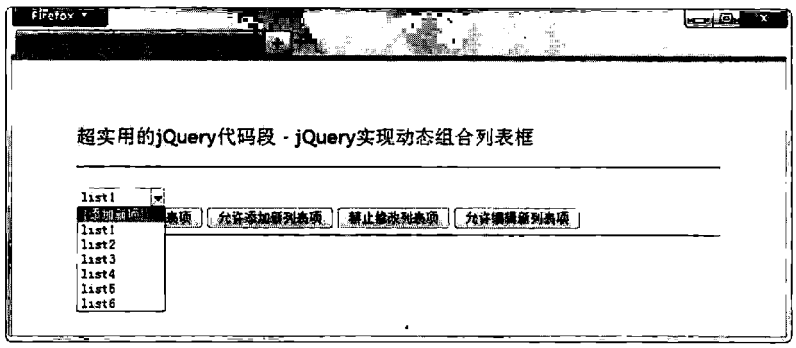

图 2.28　动态组合列表框

本例主要代码如下：

```
01  /* dynComboBox.html */
02  <script src="js/jquery.dyncombobox.js"></script>
03  <script type="text/javascript">
04  $(function(){
05      $("#id-dyncombobox").dynComboBox({
06          'allowNewElements':true,        //default : true
07          'editableElements':true         //default : true
08      });
09      $("#disableNewElements").click(function(){
10          $("#id-dyncombobox").eComboBox("disableAddingNewElements");
11          $("#div-log").html($("#div-log").html()+"<p>"+"禁止添加新列表项"+"</p>");
12      });
13      $("#enableNewElements").click(function(){
14          $("#id-dyncombobox").eComboBox("enableAddingNewElements");
15          $("#div-log").html($("#div-log").html()+"<p>"+"允许添加新列表项"+"</p>");
16      });
17      $("#disableEditingElements").click(function(){
18          $("#id-dyncombobox").eComboBox("disableEditingElements");
19          $("#div-log").html($("#div-log").html()+"<p>"+"禁止修改列表项"+"</p>");
20      });
21      $("#enableEditingElements").click(function(){
22          $("#id-dyncombobox").eComboBox("enableEditingElements");
23          $("#div-log").html($("#div-log").html()+"<p>"+"允许编辑新列表项"+"</p>");
24      });
25  });
26  </script>
27  <body>
28      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现动态组合列表框</h2>
29      <hr><br>
30      <select id="id-dyncombobox">
31      <option>list1</option>
```

```
32        <option>list2</option>
33        <option>list3</option>
34        <option>list4</option>
35        <option>list5</option>
36        <option>list6</option>
37        </select><br/>
38        <input id="disableNewElements" type="button" value="禁止添加新列表项"/>
39        <input id="enableNewElements" type="button" value="允许添加新列表项"/>
40        <input id="disableEditingElements" type="button" value="禁止修改列表项"/>
41        <input id="enableEditingElements" type="button" value="允许编辑新列表项"/>
42        /* 省略部分代码，此处可以添加用户代码 */
43    </body>
```

本例向读者介绍了如何基于 jquery.dyncombobox 插件实现动态组合列表框的方法，该组合列表框支持动态添加列表项功能。使用 jquery.dyncombobox 插件对页面中定义好的 select 列表控件进行绑定就可以实现这个功能，其语法如下：

```
$("#id-dyncombobox").dynComboBox({
    'allowNewElements':true,        //default : true
    'editableElements':true         //default : true
});
```

jquery.dyncombobox 插件内置了一些非常实用的配置参数，可以定义组合列表框的行为属性，其具体说明详见表 2.3。

表 2.3　jquery.dyncombobox插件参数说明

参数名称	默认值	参数说明
allowNewElements	true	定义该组合列表框是否支持动态添加列表项
edittableElements	true	定义该组合列表框是否支持可编辑特性

2.27　如何使用属性过滤器

在页面代码中使用各种属性过滤器来实现页面元素控件的各种高级选择过滤功能，是实现动态页面编程的必要手段之一。本例效果如图 2.29 所示。

本例主要代码如下：

```
01  /* multiProperty.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04        $("#id-input-add-element").click(function(){
05              $("<li><a href='http://www.google.com' rel='google' target='_blank'>谷歌
06  </a></li>").appendTo("div.ulclass>ul");
07              $("div.ulclass>ul").append("<li><a href='http://www.microsoft.com' rel='microsoft'>
08  微软</a></li>");
09              $("#div-log").html($("#div-log").html()+"<p>"+"动态添加元素"+"</p>");
```

```
    });
    $("a[rel='selfViewOpen']").click(function(e){
        //设置元素属性 attr
        $(this).attr("target","_self");
        $("#div-log").html($("#div-log").html()+"<p>"+"通过 rel='selfViewOpen'属性值过滤元
素,设定本页打开连接方式"+"</p>");
    });
    $("#id-input-sel-nohttp").click(function(){
        $("a").not("[href*='http://']").append("<b>---本地链接网站</b>");
        $("#div-log").html($("#div-log").html()+"<p>"+"过滤本地链接"+"</p>");
    });
    $("#id-input-sel-httpstart").click(function(){
        $("a[href^='http://']").attr("target","_blank").css("background-color","gray");
        $("#div-log").html($("#div-log").html()+"<p>"+"过滤连接开头"+"</p>");
    });
    $("#id-input-sel-httpend").click(function(){
        $("a[href$='edu.cn']").css("background-color","red").append("---教育网");
        $("#div-log").html($("#div-log").html()+"<p>"+"过滤连接结尾"+"</p>");
    });
    $("#id-input-sel-multi").click(function(){
        $("a[rel='google'][target='_blank']").css("background-color","green").append("---多
属性过滤器");
        $("#div-log").html($("#div-log").html()+"<p>"+"过滤连接结尾"+"</p>");
    });
});
</script>
<body>
    <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 如何使用属性过滤器</h2>
    <hr><br>
    <input type="button" id="id-input-add-element" value="动态添加元素">
    <input type="button" id="id-input-sel-nohttp" value="过滤本地链接">
    <input type="button" id="id-input-sel-httpstart" value="过滤连接开头">
    <input type="button" id="id-input-sel-httpend" value="过滤连接结尾">
    <input type="button" id="id-input-sel-multi" value="多属性过滤器">
    <div class="ulclass">
        <span>网站导航</span><br/>
        <ul>
            <li><a href="http://www.baidu.com" rel="selfViewOpen">百度</a></li>
            <li><a href="http://www.pku.edu.cn">北京大学</a></li>
            <li><a href="#">No http://</a></li>
        </ul>
    </div>
    /* 省略部分代码，此处可以添加用户代码 */
</body>
```

本例向读者介绍了如何基于jQuery框架使用属性过滤器的方法，其中分别实现了“动态添加元素”、“过滤本地链接”、“过滤链接开头”、“过滤链接结尾”和“多属性过滤器”5种功能。

下面这段代码通过“div.ulclass>ul”过滤器选取了类名为ulclass的div元素中的ul列表元素，并在列表末尾动态添加了两个元素。

```
$("#id-input-add-element").click(function(){
    //动态添加"谷歌"与"微软"两个元素
    $("<li><a href='http://www.google.com' rel='google' target='_blank'>谷歌</a></li>").appendTo
("div.ulclass>ul");
    $("div.ulclass>ul").append("<li><a href='http://www.microsoft.com' rel='microsoft'>微软</a>
</li>");
});
```

下面这段代码通过“rel='selfViewOpen'”全字符匹配属性过滤器选取了属性为selfViewOpen的a元素，并实现了过滤本地链接的功能。

```
//jquery [attribute=value]属性过滤器，全字符匹配
$("a[rel='selfViewOpen']").click(function(e){
    //设置元素属性 attr
    $(this).attr("target","_self");
});
```

下面这段代码通过 not("[href*='http://']")全字符匹配属性过滤器选取了不包括 http://字符串的元素并添加一段字符串，并实现了过滤链接开头的功能。

```
$("#id-input-sel-nohttp").click(function(){
    //JQuery [attribute!=value]同 not([attr="value"])
    //全字符匹配，取出属性不包括指定字符串的元素
    //取得不包含 http://字符串的元素并添加一段字符串
    //[href*=value]属性过滤，模糊匹配
    $("a").not("[href*='http://']").append("<b>---本地链接网站</b>");
    $("#div-log").html($("#div-log").html()+"<p>"+"过滤本地链接"+"</p>");
});
```

下面这段代码通过 href^='http://'属性过滤器取出属性以指定字符串开头的元素，并实现了过滤链接结尾的功能。

```
$("#id-input-sel-httpstart").click(function(){
    //JQuery [attribute^=value]取出属性以指定字符串开头的元素
    $("a[href^='http://']").attr("target","_blank").css("background-color","gray");
    $("#div-log").html($("#div-log").html()+"<p>"+"过滤连接开头"+"</p>");
});
```

下面这段代码通过[rel='google'][target='_blank']多属性过滤器取出以指定字符串结尾的元素，并实现了多属性过滤器的功能。

```
$("#id-input-sel-multi").click(function(){
    //JQuery [attribute$=value]取出属性以指定字符串结尾的元素
    $("a[rel='google'][target='_blank']").css("background-color","green").append("—多属性过滤器");
    $("#div-log").html($("#div-log").html()+"<p>"+"过滤连接结尾"+"</p>");
});
```

提示：在jQuery框架中，除了直接使用id和class属性作为选择器之外，还可以根据各种属性对由选择器查询到的元素进行过滤。属性过滤器大致可以分为8种：包含属性过滤器、属性等于过滤器、属性包含过滤器、属性包含单词过滤器、属性不等于过滤器、属性开始过滤器、属性结尾过滤器与复合属性过滤器，感兴趣的读者可以进一步查阅jQuery官方文档。

2.28 如何测试某个元素是否可见

在页面中实现测试某个元素是否可见的功能，可以完成页面不同区域的显示与隐藏效果。本例效果如图2.29所示。

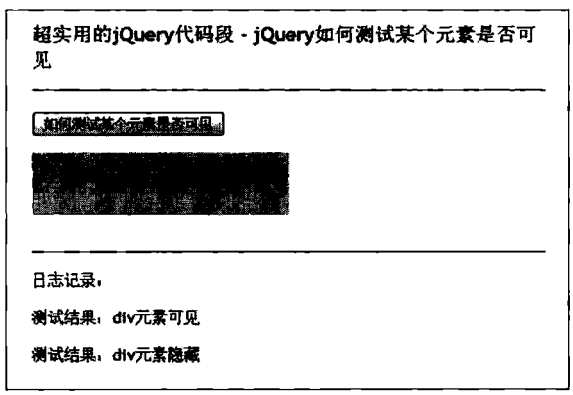

图2.29 测试某个元素是否可见

本例主要代码如下：

```
/* checkElementExist.html */
<script type="text/javascript">
$(function(){
    $("#id-button-check").click(function(){
        if($("#id-div").is(":visible")){
            $("#id-div").hide(1000);
            /* 省略部分代码，此处可以添加用户代码 */
        }else{
            $("#id-div").show(1000);
            /* 省略部分代码，此处可以添加用户代码 */
        }
    });
});
</script>
<body>
    <h2 id="h2-caption">超实用的jQuery代码段 - jQuery如何测试某个元素是否可见</h2>
    <hr><br>
    <input type="button" id="id-button-check" value="如何测试某个元素是否可见"/><br>
```

```
19      <p>
20          <div id="id-div">测试某个元素是否可见</div>
21      </p>
22      /* 省略部分代码，此处可以添加用户代码 */
23  </body>
```

本例的关键代码是第 05~11 行，它通过“:visible”选择器判断该 div 元素是否可见，并根据可见状态进行显示或隐藏。“:visible”选择器的语法如下：

```
$(":visible");                    //:visible 选择器选取每个当前可见的元素
```

关于元素是否可见的判断有一些具体规定，除以下几种情况之外的元素即是可见元素：

- 设置为 display:none。
- type="hidden"的表单元素。
- width 和 height 设置为 0。
- 隐藏的父元素（同时隐藏所有子元素）。

第 3 章　jQuery 操作 HTML 事件

jQuery 框架是非常棒的 JS 类库，不但有丰富的 UI 库和插件，还可以操作各种 HTML 事件，并提供了丰富的函数，包括绑定、一次绑定、触发等等。利用 jQuery 来绑定事件非常方便，有 bind()、live()、one()等等很多函数辅助绑定。

本章主要介绍了使用 jQuery 操作 HTML 事件的代码，主要包括以下内容：

- jQuery 事件触发器。
- jQuery 绑定与反绑定事件监听器。
- jQuery 事件的交互处理。
- jQuery 的内置事件类型。
- jQuery 页面载入完毕响应事件。

3.1　禁止或启用输入框

在某些页面表单中，需要根据用户输入信息来选择性的禁止或启用输入框，来完成一些特定功能需求。例如，对于不够权限级别的用户在注册个人信息时，可以选择性的禁用或启用一些输入框，来达到个性化输入的效果。本例将介绍禁止或启用表单中输入框的方法，其效果如图 3.1 所示。

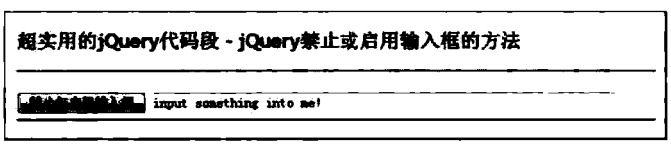

图 3.1　禁止或启用输入框

本例主要代码如下：

```
01  /* inputEnableDisable.html */
02  <script type="text/javascript">
03  $(function(){
04      $("button:eq(0)").click(function(){
05          var input1=$("input:text:eq(0)");
06          if(input1.attr("disabled")=="disabled"){
07              input1.removeAttr("disabled");
08          }else{
```

```
09                  input1.attr("disabled","disabled");
10              }
11          });
12  });
13  </script>
14  <body>
15      <h2 id="h2-caption">超实用的jQuery代码段 - jQuery禁止或启用输入框的方法</h2>
16      <hr><br>
17      <button>禁止或启用输入框</button>
18      <input type="text" value="input something into me!" size="64"/>
19      /* 省略部分代码，此处可以添加用户代码 */
20  </body>
```

本例的关键代码是第04~11行，其中第06行通过attr()函数判断input1元素的disabled属性值是否为禁用，如果条件为true，则通过removeAttr()函数移除disabled属性值来启用输入框输入功能；如果条件为false，则在第09行中使用attr()函数设定input1元素的disabled属性值为disabled以禁止输入框输入功能。关于attr()函数与removeAttr()函数的语法如下：

```
$(selector).attr(attribute);          //返回被选元素的属性值
$(selector).attr(attribute,value);    //设置被选元素的属性和值
$(selector).removeAttr(attribute);    //从被选元素中移除属性
```

3.2 实时监听输入框字符的变化

在某些页面表单中，需要根据用户输入信息来实时监听输入框字符变化的情况，这个功能在一些留言板、博客中是非常常见的。本例就来实现该功能，其页面效果如图3.2所示。

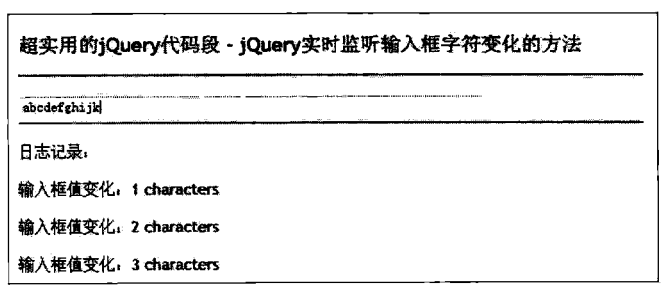

图3.2 实时监听输入框字符的变化

本例主要代码如下：

```
01  /* jqInputCountListen.html */
02  <script type="text/javascript">
03  $(function(){
04      $('#id-input-value').bind('input propertychange',function(){
05      /* 省略部分代码，此处可以添加用户代码 */
06      });
07  });
```

```
08  </script>
09  <body>
10      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实时监听输入框字符变化的方法
11  </h2>
12      <hr><br>
13      <input type="text" id="id-input-value" value="" size="64"/>
14      /* 省略部分代码，此处可以添加用户代码 */
15  </body>
```

本例的关键代码是第 04~06 行，其中用 bind()函数同时绑定了 input 与 propertychange 事件，并在 bind()函数内定义的事件函数中编写用户代码实现监听输入框字符变化的功能。关于 input 与 propertychange 事件的说明如下。

（1）input 事件是 HTML 5 的标准事件，对于监听<textarea>、<input:text>、<input:password>和<input:search>这几个元素非常有用，input 事件在内容修改后立即会被触发，input 事件不像 onchange 事件那样，只有在失去焦点时才被触发。

（2）propertychange 事件是 IE 浏览器所特有的，该事件在用户界面改变或者使用脚本直接修改内容两种情况下都会被触发，具体有以下 3 种情况：

- 修改了 input:checkbox 或者 input:radio 元素的选中状态，checked 属性发生变化。
- 修改了 input:text 或者 textarea 元素的值，value 属性发生变化。
- 修改了 select 元素的选中项，selectedIndex 属性发生变化。

提示：input 事件在 IE 9 以下版本中无法支持，因此需要用 propertychange 事件来替代。

3.3 实时监听输入框值的变化

有了上一个例子的基础，我们引申一下就可以想到实时监听输入框值变化的方法。如果能根据用户输入信息来实时监听输入框值的变化情况，就可以实现一些诸如验证用户输入信息合法性等一些复杂功能了。本例效果如图 3.3 所示。

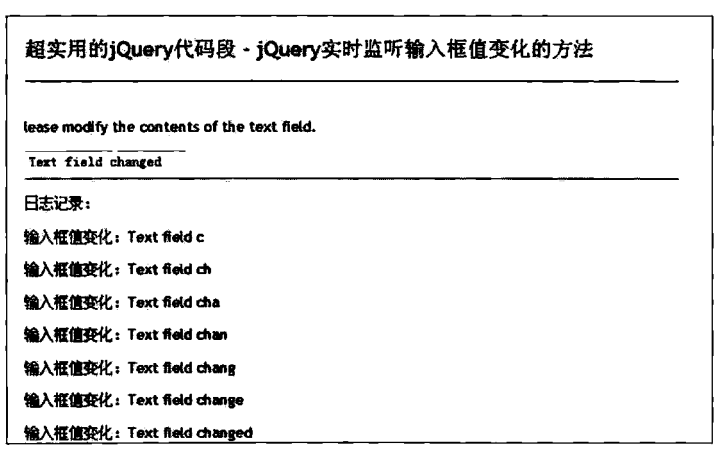

图 3.3 实时监听输入框值的变化

本例主要代码如下：

```
01  /* jqInputValueListen.html */
02  <script type="text/javascript">
03  /针对 Firefox、Chrome、Opera、Safari、IE 9 几款浏览器的方法
04  function OnInput(event){
05      $("#div-log").html($("#div-log").html()+"<p>输入框值变化："+event.target.value+"</p>");
06  }
07  /针对版本低于 Internet Explorer 9 浏览器的方法
08  function OnPropChanged(event){
09      if(event.propertyName.toLowerCase()=="value"){
10          $("#div-log").html($("#div-log").html()+"<p>输入框值变化：
11  "+event.srcElement.value+"</p>");
12      }
13  }
14  </script>
15  <body>
16      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实时监听输入框值变化的方法
17  </h2>
18      <hr><br>
19      <p>lease modify the contents of the text field.</p>
20      <input type="text" oninput="OnInput(event)"
21  onpropertychange="OnPropChanged(event)" value="Text field"/>
22      /* 省略部分代码，此处可以添加用户代码 */
23  </body>
```

本例的关键代码是第 03~13 行，这段代码与上一节的代码段一样，通过绑定 OnInput 与 OnPropChanged 事件实时监听输入框值的变化。

在针对 Firefox、Chrome、Opera、Safari、IE 9 等几款浏览器的 OnInput()函数中，通过使用 event 对象的 target 属性来获取输入框中值的变化，其中 target 属性代表 event 对象中触发事件的元素，所以 event.target.value 就代表输入框中具体的内容。

在针对低于 IE 9 的 OnPropChanged()函数中，通过使用 event 对象的 srcElement 属性来获取输入框中值的变化，其中 srcElement 属性代表触发事件源的元素，所以 event.srcElement.value 就代表输入框中具体的内容。

提示：IE 9 以前的版本中，event 对象有 srcElement 属性，但没有 target 属性。而在 Firefox、Chrome、Opera、Safari、IE 9 等几款浏览器中，event 对象有 target 属性，没有 srcElement 属性，但两者的作用是相当的。

3.4　绑定鼠标右键单击事件

通常在页面中单击鼠标右键，会弹出系统上下文菜单，有没有想过定制一下该功能实现自己想要的效果呢？我们需要通过绑定鼠标右键单击事件，来完成这一特定功能。本例

效果如图 3.4 所示。

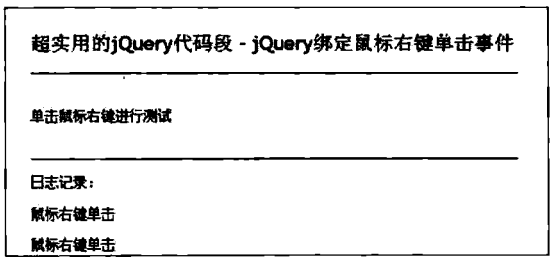

图 3.4　绑定鼠标右键单击事件

本例主要代码如下：

```
01  /* eventRightClick.html */
02  <script type="text/javascript">
03  /*鼠标右键插件*/
04  (function($){
05      $.fn.extend({
06          "rightClick":function(fn){
07              $(document).bind('contextmenu',function(e){
08                  return false;
09              });
10              $(this).mousedown(function(e){
11                  if(e.which==3){
12                      fn();
13                  }
14              });
15          }
16      });
17  })(jQuery);
18  $(document).ready(function(e){
19      /* 省略部分代码，此处可以添加用户代码 */
20  });
21  </script>
22  <body>
23      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 绑定鼠标右键单击事件</h2>
24      <hr><br>
25      <p>单击鼠标右键进行测试</p>
26      /* 省略部分代码，此处可以添加用户代码 */
27  </body>
```

本例的关键代码是第 05~16 行，代码通过函数扩展定义了 rightClick()鼠标右键函数，它包含一个名为 fn 的参数。在该函数内，首先通过绑定 contextmenu 事件并返回布尔值 false 禁用了系统右键菜单；然后为该对象绑定了 jQuery 鼠标按下事件 mousedown，在 mousedown 事件中通过监听键盘事件判断用户是否按下了鼠标右键，判断的代码是“e.which==3”。

提示：mousedown事件与click事件不同，按键被按下还没松开时就触发了mousedown事件。另外，e.which取值为1代表鼠标左键、取值为2代表鼠标中键、取值为3则代表鼠标右键。

3.5 双击不选中文本

实现双击不选中文本的功能，可以有效地避免页面文本被误操作而选中。本例效果如图3.5所示。

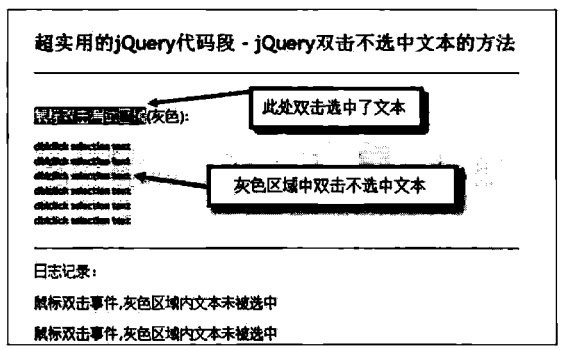

图3.5 双击不选中文本

本例主要代码如下：

```
01  /* dblclicknotext.html */
02  <script type="text/javascript">
03  $(document).ready(function(){
04      var clearSelection=function(){                    //自定义函数 clearSelection()
05          if(document.selection&&document.selection.empty){
06              document.selection.empty();
07          }else if(window.getSelection){
08              var sel=window.getSelection();
09              sel.removeAllRanges();
10          }
11      };
12      $(element).bind("dblclick",function(event){       //绑定鼠标双击事件
13          clearSelection();
14          /* 省略部分代码，此处可以添加用户代码 */
15      });
16  });
17  </script>
18  <body>
19      <h2 id="h2-caption">超实用的jQuery代码段 - jQuery双击不选中文本的方法</h2>
20      <hr><br>
21      <p>鼠标双击测试区域(灰色):</p>
```

```
22        <div id="element" style="background-color:#E8E8E8">dblclick selection text</div>
23    /* 省略部分代码，此处可以添加用户代码 */
24    </body>
```

代码第04~11行首先针对IE浏览器使用document.selection对象选择当前用户激活区域，获取当前用户操作区域后通过document.selection.empty()函数实现双击不选中文本功能；然后代码第07~08行针对非IE浏览器使用window.getSelection对象选择当前用户激活区域，在获取当前用户操作区域后通过第09行的removeAllRanges()函数实现双击不选中文本功能。

提示：一个文档同一时间只能有一个选中区域，选中区域的类型决定了其中为空或者包含文本或元素块。尽管空的选中区域不包含任何内容，编程时仍然可以用空的选中区域作为文档中的位置标志。

3.6 通过单击事件添加或解除绑定

通过单击事件添加或解除绑定，可以快速地实现事件的添加或解除功能。

本例主要代码如下：

```
01  /* clickBindUnbind.html */
02  <script type="text/javascript">
03  $(function(){
04        $("#id-clickbind").click(function(){
05              $('#id-p-bind').bind('click', function() {
06                    $("#div-log").html($("#div-log").html()+"<p>"+"Clicked and
07  Unbounded"+"</p>");
08              });
09        });
10        $("#id-clickunbind").click(function(){
11              $('#id-p-bind').unbind();
12        });
13  });
14  </script>
15  <body>
16        <h2 id="h2-caption">超实用的jQuery代码段 - jQuery通过单击事件添加或解除绑定
17  </h2>
18        <hr><br>
19        /* 省略部分代码，此处可以添加用户代码 */
20        <input type="button" id="id-clickbind" value="通过单击事件添加绑定"/>
21        <input type="button" id="id-clickunbind" value="通过单击事件解除绑定"/>
22        /* 省略部分代码，此处可以添加用户代码 */
23  </body>
```

代码第04~12行通过bind()函数和unbind()函数实现了通过单击事件添加或解除绑定的

功能。关于bind()函数在前面的很多代码段中都有提及，下面介绍unbind()函数的使用。

unbind()函数是移除被选元素的事件处理程序，其能够移除所有的或被选择的事件处理程序，也能在事件发生时终止指定函数的运行。unbind()函数的语法如下：

```
$(selector).unbind(event,function);
```

其中 event 参数规定删除元素的一个或多个事件，由空格分隔多个事件值，如果只规定了该参数，则会删除绑定到指定事件的所有函数；function 参数规定从元素的指定事件取消绑定的函数名。

提示：unbind()函数适用于任何通过 jQuery 附加的事件处理程序。

3.7 激活整个 div 层的单击事件

有时用户需要通过单击事件来激活页面中的整个 div 层，例如切换激活 div 层与隐藏 div 层的状态就需要实现该功能。

本例主要代码如下：

```
01  /* activedivclick.html */
02  <script type="text/javascript">
03  $(function(){
04      $("div").each(function(){
05          $(this).click(function(){
06              $(this).addClass("swap");
07              /* 省略部分代码，此处可以添加用户代码 */
08          });
09      });
10  });
11  </script>
12  <body>
13      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 激活整个 div 层的单击事件</h2>
14      <hr><br>
15      <div id="id-div1">div1 click event</div>
16      <div id="id-div2">div2 click event</div>
17      <div id="id-div3">div3 click event</div>
18      /* 省略部分代码，此处可以添加用户代码 */
19  </body>
```

代码第 04~09 行通过 addClass()函数实现了单击激活整个 div 层的功能。实际上，通过给选定的 div 层增加样式类，就能使得整个 div 层处于激活状态。

3.8 鼠标单击实现 div 的选取

本例与前一个例子类似，用户需要通过单击事件来实现页面中某个 div 层的选取。如

果大家对网页相册有过了解的话，就会知道网页相册其实就是基于该功能来实现的。

本例主要代码如下：

```
01  /* divclicksel.html */
02  <script type="text/javascript">
03  $(function(){
04      $("#id-div").toggle(function(){
05          $(this).removeClass("css-select");
06          $(this).addClass("css-unselect");
07          $("#div-log").html($("#div-log").html()+"<p>"+"单击层被选择"+"</p>");
08      },function(){
09          $(this).removeClass("css-unselect");
10          $(this).addClass("css-select");
11          $("#div-log").html($("#div-log").html()+"<p>"+"再次单击层被反选"+"</p>");
12      });
13  });
14  </script>
15  <body>
16      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 鼠标单击实现 div 的选取</h2>
17      <hr><br>
18      <div id="id-div" class="css-select">click div to select!<br>click div again to
19          unselect!</div>
20      /* 省略部分代码，此处可以添加用户代码 */
21  </body>
```

代码第 04~12 行通过 addClass()与 removeClass()函数实现了鼠标单击选取 div 的功能。通过对 addClass()函数与 removeClass()函数的组合使用，可以很好地切换 div 层的样式类，从而实现选取功能。

3.9 模拟鼠标单击事件

如果用户不操作鼠标，是否也能实现鼠标单击的效果呢？答案是肯定的，我们需要模拟鼠标单击事件来实现这一功能，使用这个方法就可以实现页面自动模拟人工操作的效果了。本例效果如图 3.6 所示。

图 3.6 模拟鼠标单击事件

本例主要代码如下：

```
/* simulatemouseclick.html */
<script type="text/javascript">
$(document).ready(function(){
    $('#dbut').click(function(){
        $("#a").trigger("click");
    });
    $('#a').click(function(){
        alert('模拟鼠标单击事件的方法!');
    });
});
function clickMe(){
    var $aa = $('<a>');
    $aa.html('微软');
    $('#btn').parent().append($aa);
    $aa.attr('href','http://www.microsoft.com');
    $aa.attr('target','_blank');
    if($.browser.msie||$.browser.mozilla){
        $aa.get(0).click();
    }else{
     window.open($aa.attr('href'),$aa.attr('target'));
    }
}
</script>
<body>
    <h2 id="h2-caption">超实用的jQuery代码段 - jQuery模拟鼠标单击事件的方法</h2>
    <hr><br>
    <input type="button" id="dbut" value="点击"/>
    <a id="a" href="http://www.microsoft.com">微软</a><br/>
    <input type="button" id="btn" value="Click Btton" onclick="clickMe()"/>
    /* 省略部分代码，此处可以添加用户代码 */
</body>
```

代码第04~06行通过trigger()函数模拟了鼠标单击事件。trigger()函数触发被选元素的指定事件类型，其语法如下：

```
$(selector).trigger(event,[param1,param2,...]);
```

其中event参数规定了指定元素要触发的事件；param参数数组为可选的，表示传递到事件处理程序的额外参数。

提示：trigger()函数可以使用自定义事件（使用bind()函数来附加）或任何标准事件。

3.10 设定时间间隔的方法

相信大家都对桌面版的按键精灵记忆尤深吧，其设置时间间隔的功能着实强大，可以完成许多复杂的功能效果。其实在网页设计中也可以实现同样的功能效果，本例将主要介绍一下页面中设定时间间隔的方法，效果如图 3.7 所示。

图 3.7 设定时间间隔

本例主要代码如下：

```
01  /* setInterval.html */
02  <script type="text/javascript">
03  var int=self.setInterval("clock()",50);
04  function clock(){
05        var t=new Date();
06        document.getElementById("clock").value=t;
07  }
08  </script>
09  <body>
10        <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 设定时间间隔的方法</h2>
11        <hr><br>
12        <input type="text" id="clock" size="36"/>
13        <button onclick="int=window.clearInterval(int)">Stop interval</button>
14        /* 省略部分代码，此处可以添加用户代码 */
15  </body>
```

本例的关键代码见第 03 行，通过使用 setInterval()函数设定时间间隔。setInterval()函数可以按照指定的周期（以毫秒计）来调用函数或计算表达式，语法如下：

```
setInterval(code,millisec[,"lang"]);
```

其中 code 参数表示要调用的函数或要执行的代码串，本例中调用了一个名称为 clock()的自定义函数，该函数实现了在页面输出日期的功能；millisec 参数表示周期性执行或调用 code 之间以毫秒计的时间间隔，在本例中设定为 50ms。

提示： setInterval()函数会不停地调用函数，直到 clearInterval()被调用或窗口被关闭，由 setInterval()返回的 id 值可用作 clearInterval()函数的参数。

3.11 设定时间延迟的方法

上一个例子中介绍了设定时间间隔的方法，一旦设定了时间间隔，则如果没有外界命令干预的情况下，该函数会一直循环执行下去。当然，有时候我们不希望不停地执行页面效果，如果仅仅希望延迟一段时间执行，并且仅仅执行一次页面效果怎么操作呢？本例将主要介绍一下设定时间延迟方法，效果如图3.8所示。

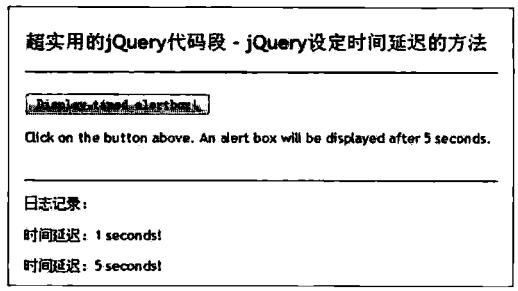

图3.8 设定时间延迟

本例主要代码如下：

```
01  /* setTimeout.html */
02  <script type="text/javascript">
03  var iSecond;
04  function timedMsg(){
05      iSecond=Math.ceil((Math.random()*10));
06      var t=setTimeout("log()",iSecond*1000);
07  }
08  function log(){
09      $("#div-log").html($("#div-log").html()+"<p>时间延迟："+iSecond+" seconds!"+"</p>");
10  }
11  </script>
12  <body>
13      <h2 id="h2-caption">超实用的jQuery代码段 - jQuery设定时间延迟的方法</h2>
14      <hr><br>
15      <form>
16          <input type="button" value="Display timed alertbox!" onclick="timedMsg()"/>
17      </form>
18      <p>Click on the button above. An alert box will be displayed after 5 seconds.</p>
19      /* 省略部分代码，此处可以添加用户代码 */
20  </body>
```

本例的关键代码见第06行，通过使用setTimeout()函数设定时间延迟。setTimeout()函数语法如下：

```
setTimeout(code,millisec[,"lang"]);
```

其中code参数表示要调用的函数或要执行的代码串，本例中调用了一个名称为log的自定义函数，该函数实现了在页面输出日志记录的功能；millisec参数表示表示在执行前需等待的毫秒数，在本例中设定为1000ms的整数倍。

提示：setTimeout()函数只执行code一次，如果要多次调用，必须使用setInterval()或者让code自身再次调用setTimeout()函数。

3.12 延时显示子菜单的方法

对于一些页面菜单，需要实现延时显示子菜单来完成一些特殊的页面效果，以防止菜单弹出速度过快影响页面效果的美观。本例效果如图 3.9 所示。

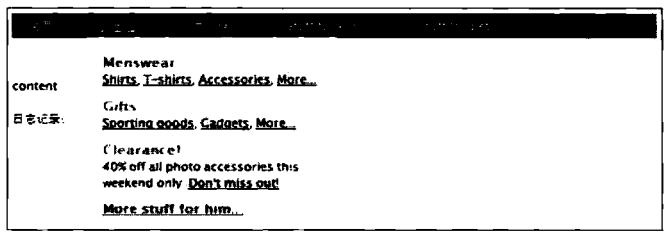

图 3.9 延时显示子菜单

本例主要代码如下：

```
/* delayShowSubMenu.html */
<script type="text/javascript">
/* 省略部分代码，此处可以添加用户代码 */
//<![CDATA[
$(document).ready(function() {
function addMega(){
    $(this).addClass("hovering");
}
function removeMega(){
    $(this).removeClass("hovering");
}
var megaConfig={
     interval:500,
     sensitivity:4,
     over:addMega,
     timeout:500,
     out:removeMega
};
$("li.mega").hoverIntent(megaConfig);
});
//]]>
</script>
<body>
<h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 延时显示子菜单的方法</h2>
<div id="banner"><div>
/* 省略部分代码，此处可以添加用户代码 */
</body>
```

本例向读者介绍了如何 hoverIntent 插件实现延时显示子菜单的方法，具体语法如下：

```
$("li.mega").hoverIntent(megaConfig);
```

上述代码通过对页面中定义好的列表菜单绑定 hoverIntent()插件方法，激活延时显示子菜单功能。hoverIntent()函数包含一个 megaConfig 参数，该参数定义了 hoverIntent 插件的外观行为特性，具体代码如第 12~18 行所示。

hoverIntent 插件重载了鼠标的 mouseover 与 mouseout 事件，设计人员可以自定义这两个事件来实现用户功能，本例中为菜单增添或移除了指定的样式类。

3.13 通过事件获取页面加载时间

获取页面加载时间是很多网站十分看重的功能，通过事件来获取页面加载时间是最基本的方法。本例效果如图 3.10 所示。

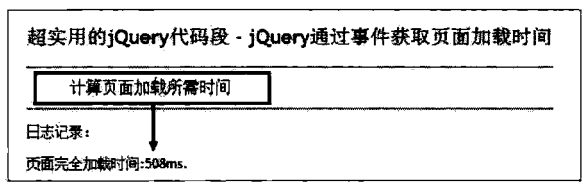

图 3.10 通过事件获取页面加载时间

本例主要代码如下：

```
01  /* pageloadTime.html */
02  <script type="text/javascript">
03  var start_time=new Date();
04  var end_time="";
05  var t=setInterval(function(){
06        if(document.readyState=="complete"){
07              calTotalTime();
08        }
09  },500);
10  function calTotalTime(){
11        end_time=new Date();
12        var vTotalTime=end_time.getTime()-start_time.getTime();
13        $("#div-log").html($("#div-log").html()+"<p>页面完全加载时
14  间:"+vTotalTime+"ms."+"</p>");
15        clearInterval(t);
16  }
17  </script>
18  <body>
19        <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 通过事件获取页面加载时间</h2>
20        /* 省略部分代码，此处可以添加用户代码 */
22  </body>
```

本例的原理是首先判断页面加载完成的状态，通过判断 document.readyState 页面当前

状态位是否等于 complete 来实现，具体代码如第 06~08 行所示。

关于 document.readyState 页面当前状态位的详细说明如下。

- UNINITIALIZED（0）：该状态表示 XML 对象被产生，但没有任何文件被加载。
- LOADING（1）：该状态表示加载程序进行中，但文件尚未开始解析。
- LOADED（2）：该状态表示部分的文件已经加载且进行解析，但对象模型尚未生效。
- INTERACTIVE（3）：该状态表示仅对已加载的部分文件有效，在此情况下，对象模型是有效但只读的。
- COMPLETED（4）：该状态表示文件已完全加载，代表加载成功。

在判断出页面加载完成后，调用自定义函数 calTotalTime()来计算页面加载时间，具体代码如下：

```
function calTotalTime(){
    end_time=new Date();
    var vTotalTime=end_time.getTime()-start_time.getTime();
}
```

提示：readyState 状态位是只读属性，用户无法通过编程来改变其值。

3.14 如何为动态添加的元素绑定事件处理函数

在页面中动态添加元素是非常常见的操作，那么如何为动态添加的元素绑定事件处理函数呢？本例就来实现该功能，效果如图 3.11 所示。

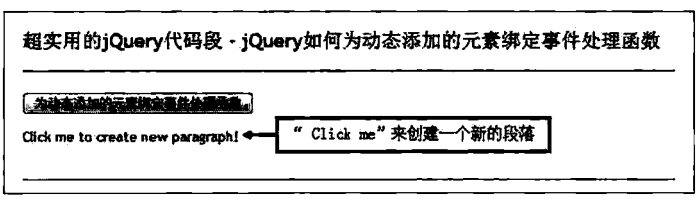

图 3.11 为动态添加的元素绑定事件处理函数

本例主要代码如下：

```
01  /* liveDynCreateHTML.html */
02  <script type="text/javascript">
03  $(function(){
04      $("button:eq(0)").click(function(){
05          $("p").live("click",function(){
06              $(this).after("<p>Another paragraph!</p>");
07          });
08      });
09  });
10  </script>
11  <body>
```

```
12      <h2 id="h2-caption">超实用的jQuery代码段 - jQuery如何为动态添加的元素绑定事件处
13  理函数</h2>
14      <hr><br>
15      <button>为动态添加的元素绑定事件处理函数</button>
16      <p>Click me to create new paragraph!</p>
17      /* 省略部分代码，此处可以添加用户代码 */
18  </body>
```

本例的关键代码是第04~08行，其中live()函数为被选中元素附加一个或多个事件处理程序，并规定当这些事件发生时运行的函数，语法如下：

```
$(selector).live(event,function);
```

其中event参数示规定附加到元素的一个或由空格分隔的多个有效事件，本例中为click单击事件；而function参数定义了当事件发生时运行的函数，本例中该函数实现了一个信息提示框。

提示：通过live()函数附加的事件处理程序，适用于匹配选择器的当前及未来的元素（比如由脚本创建的新元素）。

3.15 为表格行增加单击事件

使用过桌面版Microsoft Excel表格工具的用户一定对简单实用的插入行操作印象深刻，目前该功能在网页版的表格工具中也得到了完美实现，例如Google Doc、Office 365和WPS网络版等工具。基于页面表格行实现复杂操作，就需要为表格行增加相应事件。本例将介绍最基本的为表格行增加单击事件的方法，效果如图3.12所示。

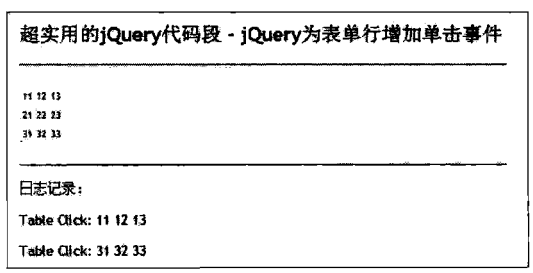

图3.12 为表单行增加单击事件

本例主要代码如下：

```
/* addTableTrClick.html */
01  <script type="text/javascript">
02  $(function(){
03      $("table tr").bind("click",function(){
04          $("#div-log").html($("#div-log").html()+"<p>Table Click:"+$(this).html()+"</p>");
05      });
06  });
07  </script>
```

```
08  <body>
09      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 为表单行增加单击事件</h2>
10      <hr><br>
11      <table>
12      </table>
13      /* 省略部分代码，此处可以添加用户代码 */
14  </body>
```

代码第 03~05 行通过对“table tr”元素绑定 click 单击事件来完成为表单行增加单击事件的功能。

3.16 用回车键模拟 Tab 键

在还没有鼠标的时候，如果我们要填写一张电子表格，通常用 Tab 键来快速定位，也就是从一个输入项移动到下一个输入项。但是越来越多的人习惯于使用回车键来快速定位或输入。本例要模拟的就是当输入回车键时实现的是 Tab 键的操作，效果如图 3.13 所示。

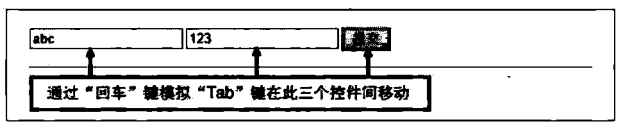

图 3.13 用回车键模拟 Tab 键

本例主要代码如下：

```
01  /* simulateEnterTab.html */
02  <script type="text/javascript" for="document" event="onkeydown"><!-- 将 tab 事件改为回车事
件 -->
03  if(event.keyCode==13&&event.srcElement.type!='button'&&event.srcElement.type!=
'reset'&&04      event.srcElement.type!='textarea'&&event.srcElement.type!='')
05  event.keyCode=9;
06  </script>
07  <script type="text/javascript">
08  $(function(){
09      //加载页面时光标自动聚焦到第一个文本框
10      window.onload=function(){
11          document.getElementById("one").focus();
12      };
13      //触发提交按钮后检验，默认失败，使光标聚焦在第一个文本框
14      function gogo(){
15          alert("hello");
16          document.getElementById("one").focus();
17      }
18  });
19  </script>
```

```
20  <body>
21      <h2 id="h2-caption">超实用的jQuery代码段 - jQuery通过回车事件模拟TAB事件</h2>
22      <hr><br>
23      <form action="" method="post">
24          <input id="one" type="text" onfocus="true" />
25          <input type="text" />
26          <input type="button" onclick="gogo();" value="提交" />
27      </form>
28      /* 省略部分代码，此处可以添加用户代码 */
29  </body>
```

代码第02~06行通过编程方式修改了event.keyCode值，将回车键变更为Tab键。实现原理是，先判断用户输入的键码值，如果监测到event.keyCode的值为13，也就是用户按下了回车键，将其键码值修改为9（event.keyCode值为9代表Tab键），这样在页面显示效果中就表现为当用户按下回车键时，经过键码的替换，实际上是实现了Tab键的功能。

提示：这个代码段使用的event.keyCode属性仅在IE浏览器下使用，在非IE浏览器中可以使用event.which属性替代。为了使代码具有更普遍的兼容性，最好用event.keyCodeevent.which。

第 4 章　jQuery 操作 CSS 样式

平常我们看到的很多渐变效果，都是通过动态修改 CSS 属性来实现的。如果使用原生 JavaScript 实现这类效果，可能需要不少于 3 行的代码，但使用 jQuery 框架，可能一个函数就够了。

本章主要介绍了使用 jQuery 操作 CSS 样式类的方法，包括以下内容：

- 动态添加样式类。
- 动态移除样式类。
- 动态切换样式类。
- 判断样式类。
- 直接修改样式类。

4.1　使用 addClass()函数动态添加样式类

使用 addClass()函数可以向指定的页面元素添加样式类，实现实时增加页面元素动态效果的功能。例如，鼠标移过某段文字时，文字产生字体加重、放大或改变颜色的效果。本例效果如图 4.1 所示。

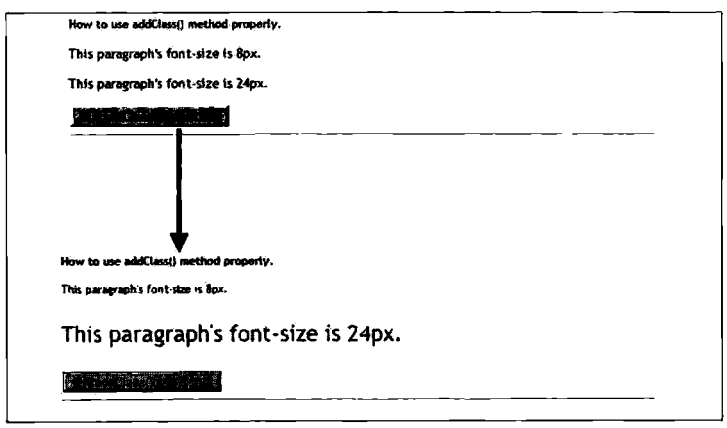

图 4.1　正确地使用 addClass()函数

本例主要代码如下：

```
01  /* addClass.html */
02  <script>
```

```
03  $(document).ready(function(){
04      $("button").click(function(){
05          $('p').addClass(function(n){
06              return 'par_'+n;
07          });
08      });
09  });
10  </script>
11  <body>
12      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 如何正确地使用 addClass()函数
13  </h2>
14      <hr><br>
15      <h3>How to use addClass() method properly.</h3>
16      <p>This paragraph's font-size is 8px.</p>
17      <p>This paragraph's font-size is 24px.</p>
18      <button>使用 addClass()函数添加类</button>
19      /* 省略部分代码，此处可以添加用户代码 */
20  </body>
```

本例的关键代码是第 05~07 行，其中，addClass()函数用于向被选元素添加一个或多个样式类，它通过一个自定义函数返回参数，对名称为 p 的元素添加前缀名为“par_”的样式类。

提示：addClass(class)函数不会移除已存在的 class 属性，仅仅添加一个或多个 class 属性。如需添加多个类，请使用空格分隔类名。

4.2　使用 removeClass()函数动态移除样式类

当然，在应用了 addClass()函数之后，有时我们还要将发生的页面效果还原，相应的也就有了 removeClass()函数。使用 removeClass()函数可以将指定页面元素的样式类移除，实现实时取消页面元素动态效果的功能。本例效果如图 4.2 所示。

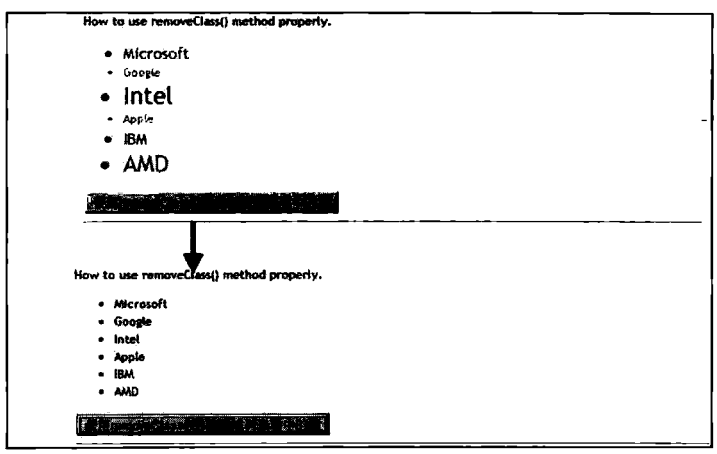

图 4.2　正确地使用 removeClass()函数

本例主要代码如下：

```
01  /* removeClass.html */
02  <script>
03  $(document).ready(function(){
04      $("button").click(function(){
05          $('ul li').removeClass(function(){
06              return 'listitem_'+$(this).index();
07          });
08      });
09  });
10  </script>
11  <body>
12      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 如何正确地使用 removeClass()函
13  数</h2>
14      <hr><br>
15      <h3>How to use removeClass() method properly.</h3>
16      <ul>
17          <li class="listitem_0">Microsoft</li>
18          <li class="listitem_1">Google</li>
19          <li class="listitem_2">Intel</li>
20          <li class="listitem_3">Apple</li>
21          <li class="listitem_4">IBM</li>
22          <li class="listitem_5">AMD</li>
23      </ul>
24      <button>使用 removeClass()函数删除列表项中的类</button>
25      /* 省略部分代码，此处可以添加用户代码 */
26  </body>
```

本例的关键代码是第 05~07 行，其中，removeClass()函数将移除被选元素的一个或多个样式类。它通过一个自定义函数返回参数，对 li 列表元素移除前缀名为“listitem_”的样式类。

提示：removeClass(class)函数中的 class 参数为可选项，表示要移除的 class 名称，如需移除若干类，可以使用空格来分隔类名；如果 removeClass(class)没有参数，则该函数将删除被选元素中的所有类。

4.3 使用 toggleClass()函数切换页面元素的样式类

很多时候，我们需要将 addClass()函数与 removeClass()函数组合在一起使用，用于实现自动切换状态的页面效果。这点 jQuery 框架已经为设计人员考虑好了，使用 toggleClass()函数可以切换指定页面元素的样式类，实现实时变换页面元素动态效果的功能。本例主要代码如下：

```
01  /* toggleClass.html */
```

```
<script>
$(document).ready(function(){
    $("button").click(function(){
        $('ul li').toggleClass(function(){
            return 'listitem_'+$(this).index();
        });
    });
});
</script>
<body>
    <h2 id="h2-caption">超实用的jQuery代码段 - jQuery如何正确地使用toggleClass()函数
</h2>
    <hr><br>
    <h3>How to use toggleClass() method properly.</h3>
    <ul>
        <li class="listitem_0">Microsoft</li>
        <li class="listitem_1">Google</li>
        <li class="listitem_2">Intel</li>
        <li class="listitem_3">Apple</li>
        <li class="listitem_4">IBM</li>
        <li class="listitem_5">AMD</li>
    </ul>
    <button class="btn1">使用toggleClass()函数添加或移除列表项的类</button>
    /* 省略部分代码，此处可以添加用户代码 */
</body>
```

本例的关键代码是第05~07行，其中toggleClass()函数对设置或移除被选元素的一个或多个样式类进行切换。该函数通过一个自定义函数返回参数，对li列表元素切换前缀名为“listitem_”的样式类。

提示：toggleClass(class,switch)函数检查每个元素中指定的类。如果不存在则添加类，如果已存在则删除类，这就是所谓的切换效果。不过，通过使用switch参数，设计人员能够设计只删除或只添加类。

4.4 为body增加class类支持

在很多种情况下，需要为页面body元素添加样式类支持，实现整体页面的风格特效。例如，改变整个页面的颜色、背景或字体风格。本例效果如图4.3所示。

本例主要代码如下：

```
/* addBodyClass.html */
<script>
$(document).ready(function(){
    $("button").click(function(){
```

```
05              $("body").css("background-image","url('images/bgimage.jpg')");
06          });
07  });
08  </script>
09  </head>
10  <body>
11      <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 为 body 增加 class 类支持</h2>
12      <hr><br>
13      <button>为 body 增加 class 类操作</button>
14      /* 省略部分代码，此处可以添加用户代码 */
15  </body>
16  </html>
```

图 4.3　为 body 增加 class 类支持

第 05 行通过 css()函数为 body 元素增加了背景图片。使用 css()函数需要注意两点。

（1）如果是要获取 CSS 属性值，则它返回第一个匹配元素的 CSS 属性值，语法如下：

```
$(selector).css(name);
```

其中 name 参数为可选项，表示要获取的 CSS 属性名称，该参数可包含任何 CSS 属性，比如 color。

（2）如果是要设置 CSS 属性值，则 css()函数设置所有匹配元素的指定 CSS 属性，语法如下：

```
$(selector).css(name,value);
```

提示：当用 css()返回一个属性值时，name 参数不支持简写的 CSS 属性（如 background 或 border）。

4.5 操作div的显示与隐藏

实现页面div层元素的显示与隐藏是实现动态页面效果的最常用功能之一。举例来说，读者都有在网站注册账户的经历，在填写用户信息时有很多不太重要的可选信息是隐藏在一个 div 层面板之中的。当然，用户可以通过单击等方式激活该面板来填写自己感兴趣的信息。本例就要实现这样一个基本功能，效果如图4.4所示。

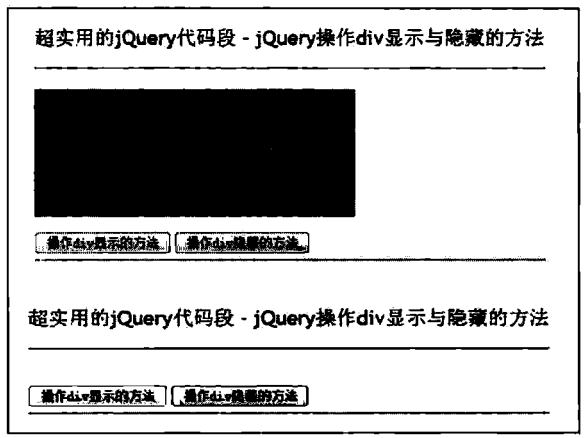

图4.4 操作div的显示与隐藏

本例主要代码如下：

```
/* cssdisplay.html */
01  <script>
02  $(document).ready(function(){
03      $("#id-button-show").click(function(){
04          $("#id-div").css('display','block');
05      });
06      $("#id-button-hide").click(function(){
07          $("#id-div").css('display','none');
08      });
09  });
10  </script>
11  <body>
12      <h2 id="h2-caption">超实用的jQuery代码段 - jQuery操作div显示与隐藏的方法</h2>
13      <hr><br>
14      <div id="id-div">操作div显示与隐藏的方法</div><br>
15      <button id="id-button-show">操作div显示的方法</button>
16      <button id="id-button-hide">操作div隐藏的方法</button>
17      /* 省略部分代码，此处可以添加用户代码 */
18  </body>
```

本例的关键代码是第04行和第07行，这段代码通过css()函数为div元素设置CSS的display属性实现显示/隐藏效果。从代码中可以看出，如果display属性值为block，则div

元素为显示状态；如果 display 属性值为 none，则 div 元素为隐藏状态。

提示：对于 HTML 文档类型，如果不谨慎使用 display 会很危险，因为可能违反 HTML 中已经定义的显示层次结构。对于 XML，由于 XML 没有内置的这种层次结构，所有 display 都是绝对必要的。

4.6 如何设定 div 始终居中显示

在很多种情况下，需要将页面 div 层元素设置为居中显示，比如在实现登录框、消息框置顶居中的风格特效时。本例效果如图 4.5 所示。

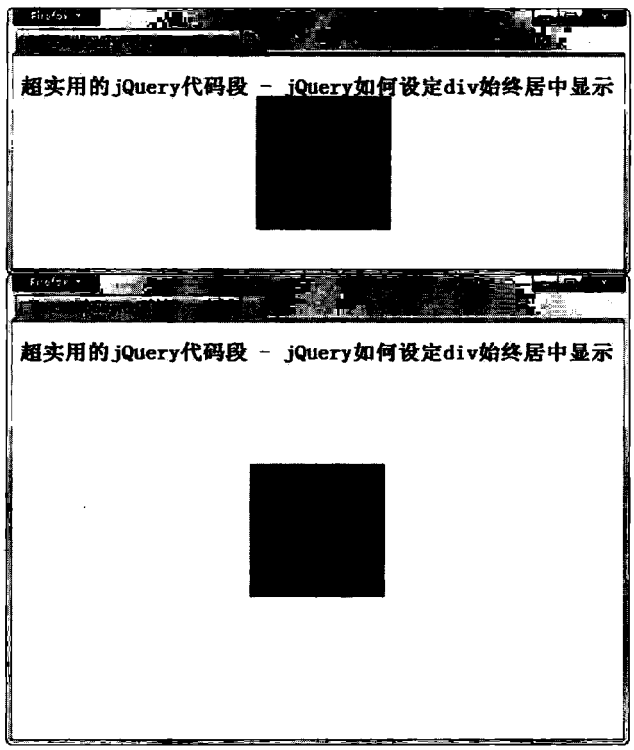

图 4.5 设定 div 始终居中显示

本例主要代码如下：

```
01  /* divCenter.html */
02  <script>
03  $(document).ready(function(){
04      $('#centerDiv').center();
05      $(window).bind('resize',function(){
06          $('#centerDiv').center({transition:500});
07      });
08  });
```

```
09  </script>
10  <body>
11  <h2 id="h2-caption">超实用的jQuery代码段 - jQuery如何设定div始终居中显示</h2>
12  <div id="centerDiv">设定div始终居中显示</div>
13  </body>
```

从图4.5中可以看出，即使浏览器页面发生变化，页面中的div元素始终处于居中显示状态。代码第05~07通过调用center()函数实现本例功能。center()函数是通过扩展函数形式实现的，具体代码如下：

```
01  (function($){
02      $.fn.extend({
03          center:function(options){              //center()插件函数
04              var options=$.extend({             //默认属性值
05                  inside:window,                 //元素居于窗体中心
06                  transition:0,                  //元素居中移动时的时间，单位为millisecond
07                  minX:0,                        //元素最小X方向距边值
08                  minY:0,                        //元素最小Y方向距边值
09                  withScrolling:true,            //是否支持滚动条
10                  vertical:true                  //是否支持垂直居中
11                  horizontal:true                //是否支持水平居中
12              }, options);
13              return this.each(function(){//通过计算窗口尺寸与元素尺寸将元素居中显示
14                  var props={position:'absolute'};
15                  if(options.vertical){
16                      var top=($(options.inside).height()-$(this).outerHeight())/2;
17                      if(options.withScrolling) top+=$(options.inside).scrollTop()||0;
18                      top=(top>options.minY?top:options.minY);
19                      $.extend(props,{top:top+'px'});
20                  }
21                  if(options.horizontal){
22                      var left=($(options.inside).width()-$(this).outerWidth())/2;
23                      if (options.withScrolling) left+=$(options.inside).scrollLeft()||0;
24                      left=(left>options.minX?left:options.minX);
25                      $.extend(props,{left:left+'px'});
26                  }
27                  if(options.transition>0) $(this).animate(props,options.transition);
28                  else $(this).css(props);
29                  return $(this);
30              });
31          }
32      });
33  })(jQuery);
```

提示：感兴趣的读者可以仔细研究center()插件函数关于居中显示的算法，举一反三，可以将其改进成功能更强大的jQuery插件。

4.7 测试浏览器是否支持某些 CSS 3 属性

目前，浏览器的种类繁多，光是主流浏览器就有 IE、Chrome、Firefox 等等，所以有时需要实现测试某款浏览器是否支持 CSS 3 属性的某些功能。

本例主要代码如下：

```
01  /* checkCSS3.html */
02  <script type="text/javascript">
03  var supports = (function(){
04      var div = document.createElement('div'),
05      vendors = 'Webkit Moz O ms'.split(' '),
06      len = vendors.length;
07      return function(prop){
08          var dstyle = div.style;        //获取<div>元素的 style 属性集合
09          if(prop in dstyle) return true;//如果测试的 CSS 3 属性在该属性集合中则返回 true
10          prop = prop.replace(/^[a-z]/, function(val){
11              return val.toUpperCase();
12          });
13          while(len--){
14              if(vendors[len] + prop in dstyle){
15                  return true;
16              }
17          }
18      return false;
19      };
20  })();
21  if(supports('BoxAlign')){
22      document.documentElement.className+=' BoxAlign';alert("Support CSS 3");
23  } else {
24      alert("Not Support CSS 3");
25  }
26  </script>
```

本例关键代码见第 08~09 行，先获取 div 元素的 style 属性集合，然后通过 in 方法来判断某些 CSS 3 属性是否在该属性集合中，如果测试的 CSS 3 属性在该属性集合中，则返回 true 以表明当前浏览器支持该 CSS 3 属性。

4.8 如何添加 hover 类到指定元素

超链接相信读者都非常熟悉了，当用户将鼠标移动到某个超链接上时，它一般都会显示出字体突出、颜色变化的风格，也就是我们都熟悉的“hover”类特征。举一反三，如何为页面指定元素添加 hover 类支持，实现鼠标移过该元素所呈现的特有风格效果呢？本例

就来实现这个功能，效果如图 4.6 所示。

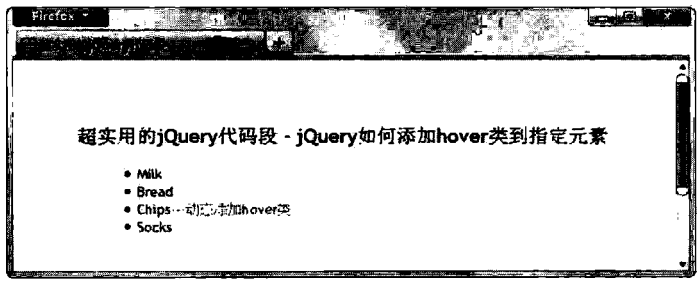

图 4.6 添加 hover 类到指定元素

本例主要代码如下：

```
01  /* addHover.html */
02  <!DOCTYPE html>
03  <html>
04  <head>
05  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
06  <script src="../code.jquery.com/jquery-1.9.1.min.js"></script>
07  <link rel="stylesheet" href="css/demos.css">
08  <script>
09  $(document).ready(function(){
10      $("li").hover(
11          function(){
12              $(this).append($("<span>—动态添加 hover 类</span>"));
13          },
14          function(){
15              $(this).find("span:last").remove();
16          });
17      $("li.fade").hover(function(){
18          $(this).fadeOut(100);
19          $(this).fadeIn(500);
20      });
21  });
22  </script>
23  </head>
24  <body>
25  <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 如何添加 hover 类到指定元素</h2>
26  /* 省略部分代码，此处可以添加用户代码 */
27  </body>
28  </html>
```

本例实现的关键是第 10 行与第 17 行的 hover()函数，该函数实现了事件的切换。所谓切换事件，就是有两个以上的事件绑定于一个元素，在元素的行为动作间进行切换。hover()函数的语法如下：

```
$(selector).hover(handlerIn,handlerOut);
```

其中 handlerIn 参数定义为一个函数，表示当鼠标进入该元素区域时的调用过程。handlerOut 参数同样也定义为一个函数，表示当鼠标离开该元素区域时的调用过程。

说明：hover()函数为设计人员在频繁使用某种任务时，提供了一种“保持在其中”的状态。

4.9 基于 URL 地址为导航链接添加 class 样式

超链接 URL 地址在网页中非常常见，当需要基于不同的 URL 地址为导航链接添加特殊的 class 样式来实现页面导航链接的风格化效果时，我们需要将 addClass()函数与 removeClass()函数组合使用。本例效果如图 4.7 所示。

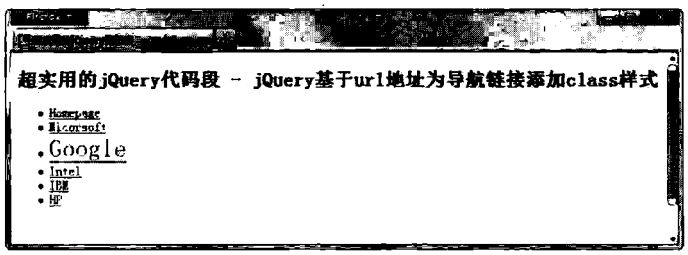

图 4.7 基于 URL 地址为导航链接添加 class 样式

本例主要代码如下：

```
01  /* addNavClass.html */
02  <script>
03  $(document).ready(function(){
04      $(".nav a").each(function(){
05          $(this).click(function(){
06              $(".nav .hover").removeClass("hover");
07              $(this).addClass("hover");
08              return false;//防止页面跳转
09          });
10      });
11  });
12  </script>
13  <body>
14  <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 基于 URL 地址为导航链接添加 class 样
式 15      </h2>
16  <div class="nav">
17      <ul>
18          <li><a href="#.html" class="hover">Homepage</a></li>
19          <li><a href="#.html">Micorsoft</a></li>
20          <li><a href="#.html">Google</a></li>
21          <li><a href="#.html">Intel</a></li>
22          <li><a href="#.html">IBM</a></li>
```

```
23              <li><a href="#.html">HP</a></li>
24          </ul>
25  </div>
26  /* 省略部分代码，此处可以添加用户代码 */
27  </body>
```

本例通过addClass()与removeClass()函数组合应用来实现，如代码第06~07行所示。在这段代码中，先针对具有.nav 与.hover 样式类的超链接元素使用 removeClass()函数去除其.hover样式类，然后再为用户选中的超链接元素使用addClass()函数添加.hover样式类，这样就实现了基于URL地址为导航链接添加class样式的功能。

4.10 如何延迟添加class类

有些情况下，简单添加class样式类已经无法满足页面的特殊需求，我们需要页面有一个动作的延迟来实现特殊的页面效果。这里介绍如何延迟添加class样式类的方法，本例效果如图4.8所示。

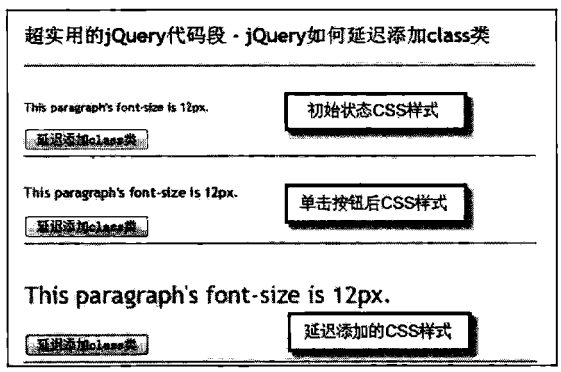

图4.8 延迟添加class类

本例主要代码如下：

```
01  /* delayaddclass.html */
02  <script>
03  $(document).ready(function(){
04          $("button").click(function(){
05                  addClassDelayed($("#id-p"),"pright","perror",1000);
06          });
07  });
08  function addClassDelayed(jqObj,cright,cerror,to){
09          jqObj.removeClass(cright);
10          setTimeout(function(){jqObj.addClass(cerror);},to);
11  }
12  </script>
13  <body>
14          <h2 id="h2-caption">超实用的jQuery代码段 - jQuery如何延迟添加class类</h2>
```

```
15        <hr><br>
16        <p id="id-p" class="pright">This paragraph's font-size is 12px.</p>
17        <button>延迟添加 class 类</button>
18        /* 省略部分代码，此处可以添加用户代码 */
19    </body>
```

本例是通过 setTimeout()函数来实现的。上述代码定义了一个 addClassDelayed()函数，其首先调用 removeClass()函数移除选定元素原有的 class 类，然后通过 setTimeout()函数在指定的时间后调用 addClass()函数为选定元素添加新的 class 类，这样就完成了延迟添加 class 类的功能。

4.11 如何延迟清除 class 类

既然实现了延迟添加 class 类的方法，我们也可以实现延迟清除 class 类的方法。本例介绍如何延迟清除 class 样式类的方法，效果如图 4.9 所示。

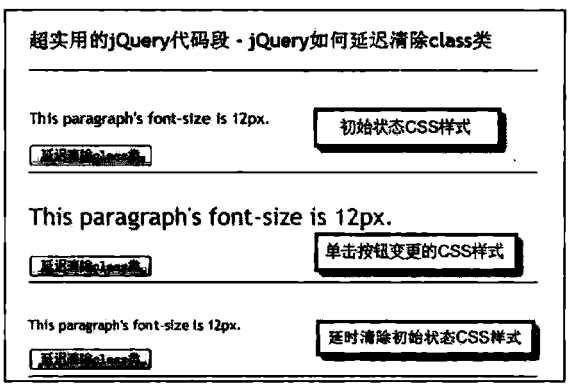

图 4.9 延迟清除 class 类

本例主要代码如下：

```
01  /* delayremoveclass.html */
02  <script>
03  $(document).ready(function(){
04        $("button").click(function(){
05              removeClassDelayed($("#id-p"),"pright","perror",1000);
06        });
07  });
08  function removeClassDelayed(jqObj,cright,cerror,to){
09        setTimeout(function(){jqObj.removeClass(cerror); jqObj.removeClass(cright);},to);
10        jqObj.addClass(cerror);
11  }
12  </script>
13  <body>
14        <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 如何延迟清除 class 类</h2>
15        <hr><br>
```

```
16        <p id="id-p" class="pright">This paragraph's font-size is 12px.</p>
17        <button>延迟清除 class 类</button>
18        /* 省略部分代码，此处可以添加用户代码 */
19    </body>
```

本例也是通过 setTimeout()函数来实现的。代码中定义了一个 removeClassDelayed()函数，其首先通过 setTimeout()函数在指定的时间后调用 removeClass()函数为选定元素清除原有的 class 类，然后通过 addClass()函数为选定元素添加新的 class 类，这样就完成了延迟清除 class 类的功能。

4.12 动态调整页面的字体大小

我们都已经熟悉了网页上标准大小的字体了，但对于一些特殊客户群体，例如老年人或视力较差的人，这个所谓标准字体确实是太小了，也许需要趴到显示屏前才能看得清楚，这时就需要动态调整页面的字体大小，以满足不同客户群的需求。本例效果如图 4.10 所示。

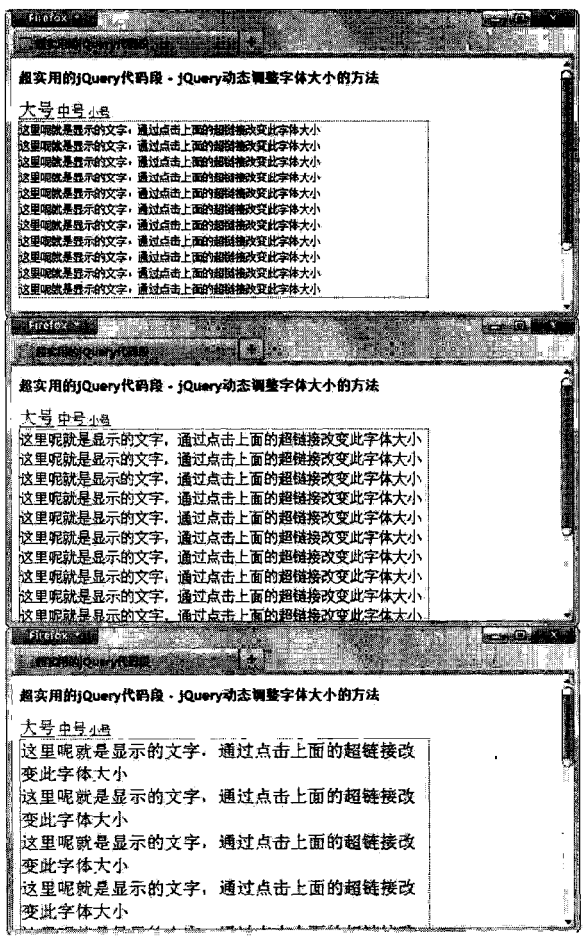

图 4.10 动态调整字体大小

本例主要代码如下：

```
01  /* dynFontSize.html */
02  <script>
03  function changeFontSize(fontSize){              //此种方式降低了 js 和 CSS 的耦合性
04      var divNode=document.getElementsByTagName("div")[0];
05      divNode.style.fontSize=fontSize;
06  }
07  </script>
08  <body>
09  <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 动态调整字体大小的方法</h2>
10  <a href="javascript:void(0)" onclick="changeFontSize('20px')" class="max">大号</a>
11  <a href="javascript:void(0)" onclick="changeFontSize('16px')" class="moren">中号</a>
12  <a href="javascript:void(0)" onclick="changeFontSize('12px')" class="min">小号</a>
13  <div>
14  这里呢就是显示的文字，通过点击上面的超链接改变此字体大小<br/>
15  </div>
16  /* 省略部分代码，此处可以添加用户代码 */
17  </body>
```

本例通过动态调整 fontSize 属性来实现，具体代码如第 03~06 行所示。设置 fontSize 的语法如下：

```
Object.style.fontSize=value;                    //使用 fontSize 属性设置元素的字体大小
```

其取值可以是“smaller”或“larger”单词表示的字体大小，也可以是“12px”或“20px”的具体像素值表示的字体大小，最后其还可以使用百分比“%”表示字体大小。设计人员可以自行选择。

提示：从本例的代码可以看出，fontSize 属性隶属于 HTML DOM style 对象，其实 style 对象包含很多有用的属性值，读者可以举一反三，通过操作 style 对象来实现更多页面动态功能效果。

第 5 章　jQuery 实现用户输入自动完成功能

在页面上为输入框添加用户输入自动完成（Auto-Complete）功能，可以大大增强用户界面的交互性，是当今网站设计的主流方向。本章主要介绍了使用 jQuery 实现用户输入自动完成的方法，主要包括以下内容：

- 实现基本用户输入自动完成的功能。
- 实现远程数据源用户输入自动完成的功能。
- 实现 Scrollable、Combobox 风格用户输入自动完成的功能。
- 实现 XML 数据源用户输入自动完成的功能。
- 实现多维数据用户输入自动完成的功能。

5.1　最简单的用户输入自动完成

大家在使用诸如淘宝、京东、亚马逊等电商平台的商品搜索功能时，往往在仅仅输入商品名称的头一个或几个字符时，就能显示出一长串包含这一个或几个字符商品名称的列表菜单，用户使用鼠标或上下方向键就可以进行快速选择，这样的效率是不是事半功倍呢？其实，该功能最初在 Google、Baidu 或 Yahoo 等这类门户搜索引擎网站中就已经实现了，电商平台将其借鉴过来并进行优化提升，实现了如今很好的用户体验。

在本章的第一个例子中，我们讲解使用 jQuery 框架下 AutoComplete 插件的最基本方法。简单来说，AutoComplete 插件就是指用户在文本框中输入前几个字母或汉字时，该控件就能从存放数据的字典里将所有以这些字母或汉字开头的数据提示给用户，供用户选择，效果如图 5.1 所示。

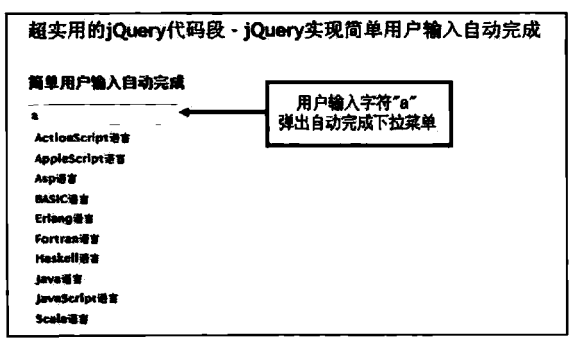

图 5.1　最简单的用户输入自动完成

本例具体代码如下：

```
/* simpleAutoComplete.html */
<script src="../jquery /ui/jquery.ui.autocomplete.js"></script>
<script type="text/javascript">
$(function(){
    var availableTags=[                          //定义并初始化字典库数据源集合
            "ActionScript 语言",
            "AppleScript 语言",
            "Asp 语言",
            "BASIC 语言",
            "C 语言",
            "C++语言",
            ……//省略部分字典列表
            "Ruby 语言",
            "Scala 语言",
            "Scheme 语言"
        ];
    $( "#autocomplete").autocomplete({           //自动完成插件函数
        source:availableTags                     //自动完成字典库数据源
    });
});
</script>
<body>
<h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现简单用户输入自动完成</h2>
<!-- Autocomplete -->
<h2 class="demoHeaders">简单用户输入自动完成</h2>
<div>
    <input id="autocomplete" title="type &quot;a&quot;">
</div>
/* 省略部分代码，此处可以添加用户代码 */
</body>
```

代码第 17~19 行通过 Autocomplete 插件函数绑定页面元素来完成用户输入自动完成的功能，在 autocomplete()函数中，通过代码第 18 行定义的属性 source 来引用字典库数据源，该数据源以一个数组的集合形式提供。具体代码如下：

```
var availableTags=[
    "ActionScript 语言",
    "AppleScript 语言",
    "Asp 语言",
    "BASIC 语言",
    "C 语言",
    "C++语言",
    ……//省略部分列表
    "Ruby 语言",
    "Scala 语言",
```

```
    "Scheme 语言"
];
```

其中，availableTags 变量定义了一个字符串数组作为数据源，列举了目前绝大多数流行的编程语言。当然，该数据源还可以定义成对象数组的形式，例如[{label:"Choice1",value:"value1"},...]。

提示：对于本地数据源来说，通过参数对象的 source 属性设置数据源是最基本、最简单的方法。

5.2　使用远程数据源的自动完成

Autocomplete 插件不光可以实现本地数据源的自动完成，也可以读取远程数据源。这样在 B/S 模式中，就可以轻松地实现从服务器端读取数据源信息的功能。本例效果如图 5.2 所示。

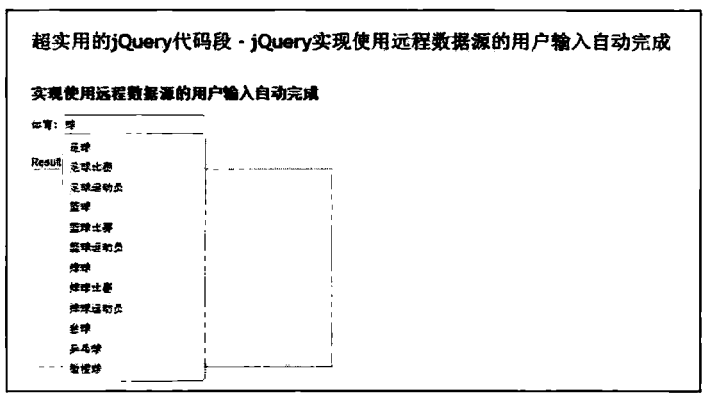

图 5.2　使用远程数据源的用户输入自动完成

本例代码如下：

```
/* remoteAutoComplete.html */
<script src="../ jquery /ui/jquery.ui.autocomplete.js"></script>
$(function(){
    function log(message){
        $("<div>").text(message).prependTo("#log");
        $("#log").scrollTop(0);
    }
    $("#birds").autocomplete({
        source:"search.php",                    //定义远程数据源文件
        minLength:2,                            //定义用户输入最少字符数
        select:function(event,ui){              //定义选择事件回调函数
            log(ui.item ? "Selected: " + ui.item.value + " aka " + ui.item.id : "Nothing
selected, input was " + this.value);            //日志记录函数
        }
```

```
14          });
15  });
16  </script>
17  <body>
18  <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现使用远程数据源的用户输入自动完
19  成</h2>
20  <!-- Autocomplete -->
21  <h2 class="demoHeaders">实现使用远程数据源的用户输入自动完成</h2>
22  <div class="ui-widget">
23          <label for="birds">体育: </label>
24          <input id="birds">
25  </div>
26  <div class="ui-widget" style="margin-top:2em; font-family:Arial">
27          Result:
28          <div id="log" style="height: 200px; width: 300px; overflow: auto; class="ui-widget-
content"> </div>
29  </div>
30  /* 省略部分代码，此处可以添加用户代码 */
31  </body>
```

代码第 08~14 行通过 Autocomplete 插件来完成远程数据源的自动完成功能。在 autocomplete()函数中，首先第 9 行通过属性 source 来引用远程数据源，该数据源是一个 PHP 文件；然后第 10 行通过属性 minLength 定义用户输入的最少字符数，只有当用户输入的字符不少于该值时才会出现提示；最后，第 11 行通过属性 select 来定义当用户选择提示信息时触发的函数，该函数是一个名为 log()的日志函数，完成一些日志记录操作，具体见第 13 行。

说明：如果使用远程数据源，通过参数对象的 select 属性可以完成一些诸如下拉菜单事件触发、显示定制与结果重写等相对复杂且功能强大的效果。

5.3 带缓存的自动完成

Web 缓存实现了对服务器端对象放在客户端存储的方法，其将需要频繁访问的对象保存在离用户最近的系统之中，当再次访问这些对象的时候极大地提高了读取速度。本例讲解如何使用 jQuery 框架下 Autocomplete 插件实现带缓存远程数据源的应用方法，本例效果如图 5.3 所示。

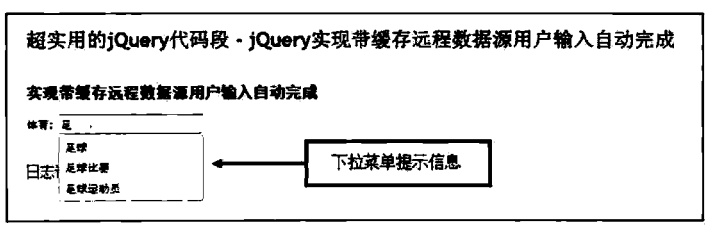

图 5.3 带缓存远程数据源用户输入自动完成

本例代码如下：

```
01  /* cacheAutoComplete.html */
02  <script src="../ jquery /ui/jquery.ui.autocomplete.js"></script>
03  <script type="text/javascript">
04  $(function(){
05      var cache = {};                          //定义缓存变量
06      $("#birds").autocomplete({
07          minLength:2,                         //定义用户输入最少字符数
08          source:function(request,response){   //定义远程数据源的函数
09              var term=request.term;           //定义用户请求信息变量
10              if(term in cache){               //判断用户请求信息是否在缓存中
11                  response(cache[term]);       //如果条件为“真”，直接从缓存数据中响应
12                  return;
13              }
14              $.getJSON("search.php",request,function(data,status,xhr){     //AJAX 请求
15                  cache[term]=data;            //将获取的远程数据源进行缓存
16                  response(data);
17              });
18          }
19      });
20  });
21  </script>
22  <body>
23  <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现带缓存远程数据源用户输入自动完
24  成</h2>
25  <!-- Autocomplete -->
26  <h2 class="demoHeaders">实现带缓存远程数据源用户输入自动完成</h2>
27  <div class="ui-widget">
28      <label for="birds">体育: </label>
29      <input id="birds">
30  </div>
31  /* 省略部分代码，此处可以添加用户代码 */
32  </div>
33  </body>
```

代码第 05 行定义了一个缓存变量 cache；代码第 06~19 行通过 Autocomplete 插件来完成用户输入的自动完成。

在 autocomplete()函数中，代码第 08~18 行通过属性 source 定义了一个引用远程数据源的函数。在该函数内先判断用户请求信息是否在缓存中，如果判断条件为“真”，则直接从缓存中响应用户请求信息。代码第 14~17 行通过 AJAX 请求的.getJSON()函数获取远程的数据源，该数据源同样为一个 PHP 文件，获取的数据信息先会被保存在之前定义好的 cache 变量中，之后再响应用户请求信息。

提示：autocomplete()函数的 source 属性既可以直接引用本地数据源文件，也可以引用远程数据源，还可以通过函数的形式实现 AJAX 异步通信方式等较为复杂的操作。

5.4 带滚动条的自动完成

如果下拉菜单中的提示信息太多太长超出了页面高度怎么办呢？可以想象如果那样的话既影响页面美观，又影响用户操作的便捷性。其实可以通过为下拉菜单增加滚动条来解决这个问题。本例实现带滚动条的用户输入自动完成功能，效果如图 5.4 所示。

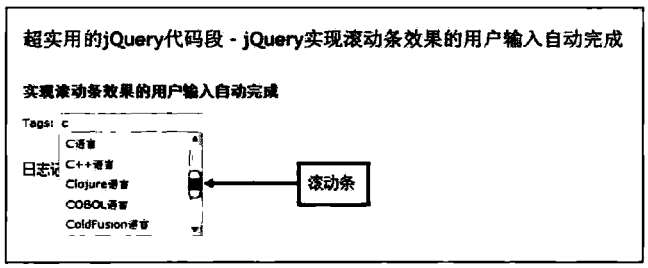

图 5.4 滚动条效果的用户输入自动完成

本例代码如下：

```
/* scrollableAutoComplete.html */
<script src="../ jquery /ui/jquery.ui.autocomplete.js"></script>
<script type="text/javascript">
$(function() {
    var availableTags=[
        "ActionScript 语言",
        "AppleScript 语言",
        "Asp 语言",
        "BASIC 语言",
        "C 语言",
        "C++语言",
        ……//省略部分列表
        "Ruby 语言",
        "Scala 语言",
        "Scheme 语言"
    ];
    $("#tags").autocomplete({
        source:availableTags,
        scroll:true              //设置滚动条选项值为 true
    });
});
</script>
<body>
```

```
24  <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现滚动条效果的用户输入自动完成
25  </h2>
26  <!-- Autocomplete -->
27  <h2 class="demoHeaders">实现滚动条效果的用户输入自动完成</h2>
28  /* 省略部分代码，此处可以添加用户代码 */
29  </body>
```

本例关键代码是第 19 行，在 autocomplete()函数中，通过定义 scroll 选项值为“真”来激活滚动条。

5.5　Combobox 风格的自动完成

Combobox 风格在页面中很常见，它外表看上去像下拉列表框，但实现的功能是输入框。本例实现了 Combobox 风格的用户输入自动完成功能，效果如图 5.5 所示。

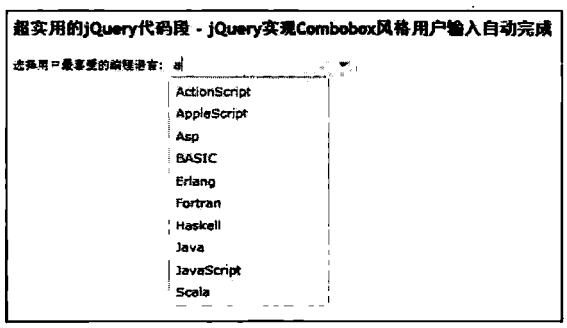

图 5.5　Combobox 风格用户输入自动完成

本例代码如下：

```
01  /* comboboxAutoComplete.html */
02  <script src="../ jquery /ui/jquery.ui.autocomplete.js"></script>
03  <script>
04  $(function(){
05      $("#combobox").combobox();
06      $("#toggle").click(function(){
07          $("#combobox").toggle();
08      });
09  });
10  </script>
11  <body>
12  <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现 Combobox 风格用户输入自动完成
13  </h2>
14  <div class="ui-widget">
15      <label>选择用户最喜爱的编程语言:</label>
16      <select id="combobox">
17          <option value="">Select one...</option>
```

```
18              <option value="ActionScript">ActionScript</option>
19              <option value="AppleScript">AppleScript</option>
20              <option value="Asp">Asp</option>
21              <option value="BASIC">BASIC</option>
22              <option value="C">C</option>
23              <option value="C++">C++</option>
24              <option value="Clojure">Clojure</option>
25              <option value="COBOL">COBOL</option>
26              <option value="ColdFusion">ColdFusion</option>
27              <option value="Erlang">Erlang</option>
28              <option value="Fortran">Fortran</option>
29              <option value="Groovy">Groovy</option>
30              <option value="Haskell">Haskell</option>
31              <option value="Java">Java</option>
32              <option value="JavaScript">JavaScript</option>
33              <option value="Lisp">Lisp</option>
34              <option value="Perl">Perl</option>
35              <option value="PHP">PHP</option>
36              <option value="Python">Python</option>
37              <option value="Ruby">Ruby</option>
38              <option value="Scala">Scala</option>
39              <option value="Scheme">Scheme</option>
40          </select>
41  </div>
42  /* 省略部分代码，此处可以添加用户代码 */
43  </body>
```

本例关键代码是第 05 行，它通过 combobox()函数绑定页面 select 元素来实现自动完成功能。在 Autocomplete 插件中，combobox()函数是通过小部件（widget）的形式实现的，具体代码如下：

```
01  $.widget("custom.combobox",{
02          _create:function(){
03
    this.wrapper=$("<span>").addClass("custom-combobox").insertAfter(this.element);
04              this.element.hide();
05              this._createAutocomplete();     //调用_createAutocomplete()函数创建自动完成
06          },
07          _createAutocomplete:function(){
08              var selected=this.element.children(":selected"),
09              value=selected.val()?selected.text():"";
10              this.input=$("<input>")
11                  .appendTo(this.wrapper)
12                  .val(value)
13                  .attr("title","")
14                  .addClass("custom-combobox-input ui-widget ui-widget-content
```

```
15  ui-state-default ui-corner-left" ).autocomplete({
16                          delay:0,
17                          minLength:0,
18                          source:$.proxy(this,"_source")
19                  }).tooltip({tooltipClass:"ui-state-highlight"});
20              this._on(this.input,{
21                  autocompleteselect:function(event,ui){
22                          ui.item.option.selected=true;
23                          this._trigger("select",event,{
24                                  item: ui.item.option
25                          });
26                  },
27                  autocompletechange:"_removeIfInvalid"
28              });
29      },
30  /* 省略部分源代码*/
31  });
```

5.6　读取 XML 数据的自动完成

随着 Web 应用的大行其道，绝大多数设计人员都会将在 C/S 模式下非常流行的 XML 数据格式引入到 Web 开发之中，并且 XML 数据格式也着实火了好一阵。本例讲解如何使用 Autocomplete 插件读取 XML 数据，效果如图 5.6 所示。

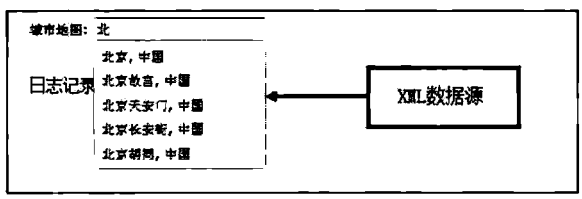

图 5.6　实现 XML 数据解析用户输入自动完成

本例代码如下：

```
01  /* xmlAutoComplete.html */
02  <script src="../ jquery /ui/jquery.ui.autocomplete.js"></script>
03  <script type="text/javascript">
04  $(function(){
05  /* 省略部分代码，此处可以添加用户代码 */
06  $.ajax({                    //选用 AJAX 异步通信方式
07      url:"map.xml",          //定义 XML 数据源文件
08      dataType:"xml",         //定义数据源文件格式
09      success:function(xmlResponse){
10          var data=$("geoname",xmlResponse).map(function(){
11              return{
```

```
12                          value:$("name",this).text()+",
13  "+($.trim($("countryName",this).text())||"(unknown country)"),id:$("geonameId",this).text()
14                      };
15              }).get();
16              $("#birds").autocomplete({              //自动完成插件函数
17                      source:data,
18                      minLength:0,
19                      select:function(event,ui){
20                              log(ui.item?"Selected: "+ui.item.value+", geonameId:
21  "+ui.item.id:"Nothing selected, input was "+this.value );
22                      }
23              });
24      }
25  });
26  });
27  </script>
28  <body>
29  <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现 XML 数据解析用户输入自动完成成
30  </h2>
31  <!-- Autocomplete -->
32  <h2 class="demoHeaders">实现 XML 数据解析用户输入自动完成</h2>
33  <div class="ui-widget">
34      <label for="birds">城市地图: </label>
35      <input id="birds" />
36  </div>
37  /* 省略部分代码，此处可以添加用户代码 */
38  </body>
```

代码第 06~24 行选用 AJAX 异步通信方式获取 XML 数据源。在 ajax()函数中，第 07 行的 URL 数据源定义为本地名称为 map.xml 的 XML 文档；第 08 行的数据源格式 dataType 定义为“xml”；如果通信成功，在第 09~24 行定义的回调函数 success 中，对 XML 文档中的数据进行解析并保存在名称为 data 的变量中。在成功取得数据后，通过 autocomplete() 函数绑定页面 id 值为 birds 的元素来实现自动完成功能。

提示：关于 AJAX 异步通信方式的使用，在本书第 8 章会有更详细地说明。

5.7 多维数据的自动完成

以上几个例子都是针对一维数据进行操作的，现实生活中往往需要从多个角度查询信息。例如挑选衣服时既想找到合适的款式、又想满足颜色的需要，又或者选择家用电器时既要考虑功能的全面实用、又要满足空间的要求，等等类似的问题都需针对多维数据进行操作。本例讲解如何使用 Autocomplete 插件实现多维数据用户输入自动完成的功能，效果如图 5.7 所示。

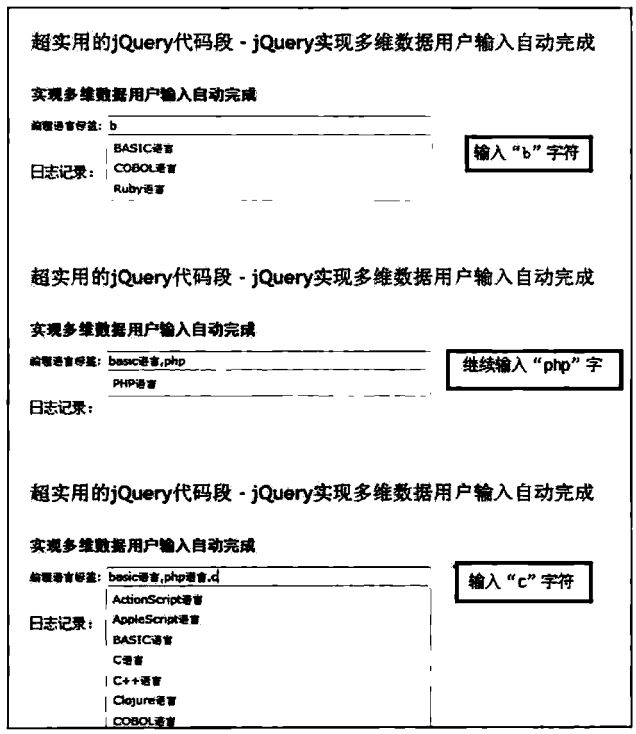

图 5.7　实现多维数据用户输入自动完成

本例代码如下：

```
01  /* multipleAutoComplete.html */
02  <script src="../ jquery /ui/jquery.ui.autocomplete.js"></script>
03  <script type="text/javascript">
04  $(function(){
05      function split(val){
06          return val.split(/,\s*/);
07      }
08      function extractLast(term){
09          return split(term).pop();
10      }
11      $("#tags").autocomplete({              //实现无须用户输入字符即可弹出自动完成
12          minLength:0,
13          source:function(request,response){
14              response($.ui.autocomplete.filter(availableTags,extractLast(request.term)));
15          },
16          focus:function(){
17              return false;
18          },
19          select:function(event,ui){
20              var terms=split(this.value);
21              terms.pop();
```

```
22                  terms.push(ui.item.value);
23                  terms.push("");
24                  this.value=terms.join(",");
25                  return false;
26              }
27          });
28  });
29  </script>
30  <body>
31  <h2 id="h2-caption">超实用的 jQuery 代码段 - jQuery 实现多维数据用户输入自动完成</h2>
32  <h2 class="demoHeaders">实现多维数据用户输入自动完成</h2>
33  <div class="ui-widget">
34      <label for="tags">编程语言标签: </label>
35      <input id="tags" size="50">
36  </div>
37  /* 省略部分代码，此处可以添加用户代码 */
38  </body>
```

代码第 11~27 行通过 autocomplete 插件函数绑定页面 id 值为 tags 的元素来完成用户输入的自动完成。在 autocomplete()函数中，第 12 行通过属性 minLength 定义无须用户输入字符即可弹出自动完成提示信息；代码第 13~15 行通过属性 source 引用名称为 availableTags 的数据源，该数据源是一个数组变量；第 16~18 行通过让 focus()事件函数返回 false 来屏蔽用户输入焦点功能；最后，第 19~26 行通过事件 select 来实现多维数据用户输入自动完成的功能。

本例的关键代码是第 19~26 行。这段代码先使用自定义函数 split()（其功能是将字符串按照指定格式进行分割处理）对用户输入进行格式化操作，然后使用类似堆栈操作的 pop()与 push()函数将格式化好的用户输入保存为数组格式，最后使用 join()函数将用户的输入按照一定格式显示在页面输入框控件之中，从而实现多维数据用户输入自动完成的功能。

说明：多维数据的用户输入自动完成功能在实际应用中十分有用，我们看到的诸如 Google、Baidu、360 搜索等门户网站上都实现了此功能。

第 6 章　jQuery 实现拖放功能

网页上的拖曳操作能够为网页带来生动的展示效果，比如拖动的树状列表、可拖曳的照片等等。无论如何，手动编写拖曳操作的代码稍稍有些复杂，因此很多拖曳效果都被封装成了 jQuery 插件，以简化执行拖曳操作的复杂性。

本章主要涉及的知识点有：

- 使用基本的拖放功能。
- 控制拖放的范围。
- 自动滚动的拖放效果。
- 可拖放排序的列表框。
- 在多个列表框之间拖放。
- 可拖动改变列宽的表格。
- 拖动选择表格的多行数据。

6.1　基本拖放功能

最基本的拖放操作，是依靠网页的 mousedown、mousemove 和 mouseup 等 API 函数来实现的，需要编写较多的 JavaScript 代码。下面通过一个可拖放的 div 来演示如何通过基本的编程 API 来实现拖曳，代码如下：

```
01  /* simpleDrag.html */
02  <script type="text/javascript">
03  // 模块拖曳
04  $(function(){
05  var _move=false;                          //判断目标对象是否处于移动状态
06  var _x,_y;                                //鼠标离控件左上角的相对 x、y 的位置
07      $(".drag").click(function(){
08          }).mousedown(function(e){         //当按下鼠标左键时
09          _move=true;                       //将移动标记设置为 true，表示开始移动
10          _x=e.pageX-parseInt($(".drag").css("left"));      //得到左上角的 x 的位置
11          _y=e.pageY-parseInt($(".drag").css("top"));       //得到左上角的 y 的值置
12          $(".drag").fadeTo(20, 0.5);                       //单击后开始拖动并透明显示
13      });
14      $(document).mousemove(function(e){
```

```
15          if(_move){                          //如果开始移动鼠标且_move 标记设置为 true
16              var x=e.pageX-_x;               //移动时根据鼠标位置计算控件左上角的绝对位置
17              var y=e.pageY-_y;
18              $(".drag").css({top:y,left:x}); //使用 CSS 设置控件的新位置
19          }
20      }).mouseup(function(){
21      _move=false;
22      $(".drag").fadeTo("fast", 1);           //松开鼠标后停止移动并恢复成不透明
23    });
24  });
25  </script>
26  </head>
27  <body>
28    <!--可以被拖动的 div 元素-->
29    <div class="drag">请单击这里进行拖曳</div>
30  </body>
```

第 05 行的全局变量_move 用来标志当前的 div 是否处于拖动状态。第 08~13 行当鼠标按下时，关联了 mousedown 事件，将_move 设置为 true，并且设置_x 和_y 这两个全局变量的值为当前 div 在左上角的位置。第 12 行调用了 fadeTo 来半透明显示 div 元素。第 14~19 行的 mousemove 事件中，如果_move 处于移动状态，则获取当前鼠标所在的 x、y 的位置并减去鼠标按下时 x、y 的位置，然后用 CSS 将左上角的位置设置为新的 x、y 的位置，从而实现移动操作。第 20~23 行当鼠标松开时，即 moseup 事件被触发时，会将_move 设置为 false，并且调用 fadeTo 恢复 div 为不透明状态。

6.2 基于事件的拖放

自行实现拖曳方法比较复杂，不利于模块化的功能实现，jQuery UI 提供了可拖曳的事件，允许用户非常简单地为一个 div 添加拖曳效果。

注意：jQuery UI 是 jQuery 官方支持的 Web UI 代码库，包含底层交互、动画、特效等 API，并且封装了一些 Web 小部件。同时，jQuery UI 继承了 jQuery 的插件支持，有大量的第三方插件可以丰富 jQuery UI 的功能。

下面的实例将使用 jQuery UI 的 draggable 事件来启用拖曳功能，可以看到通过 jQuery UI 的功能，可以很轻松地实现启用拖曳、禁用拖曳和删除拖曳功能，代码如下：

```
01  /* eventDrag.html */
02  <!--添加 jQuery UI 的样式引用-->
03  <link rel="stylesheet"
04  href="http://code.jquery.com/ui/1.10.4/themes/smoothness/jquery-ui.css">
05  <!--添加 jQuery 的引用-->
06  <script src="http://code.jquery.com/jquery-1.10.2.js"></script>
07  <!--添加对 jQuery UI 的引用-->
```

```
08  <script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
09  <style>
10    #draggable { width: 150px; height: 150px; background-color:#990}
11  </style>
12  <script type="text/javascript">
13    $(document).ready(function(e){
14          //使用 draggable 初始化拖曳功能
15          $("#draggable").draggable({cursor:"move",cursorAt:{top:56,left:56}});
16          $("#btnenable").click(function(e){
17              $("#draggable").draggable("enable");    //对禁用的拖曳功能进行启用
18          });
19          $("#btndisable").click(function(e){
20              $("#draggable").draggable("disable");   //对启用的拖曳进行禁用
21          });
22          $("#btndestroy").click(function(e){
23              $("#draggable").draggable("destroy");   //删除拖曳的功能
24          });
25    });
26  </script>
27  </head>
28  <body>
29  <!--拖曳的功能按钮-->
30  <button id="btnenable">允许拖曳</button>
31  <button id="btndisable">禁止拖曳</button>
32  <button id="btndestroy">删除拖曳功能</button>
33  <!--可拖曳的 div 元素-->
34  <div id="draggable" class="ui-widget-content">
35    <p>点击这里进行拖动</p>
36  </div>
37  </body>
```

在第 13~25 行中，当页面就绪时，先调用了 jQuery 选择器的 draggable()函数，用来进行可拖曳操作的初始化，第 15 行使用 JSON 对象配置了可拖曳的相关参数，比如 cursor 为 move 表示鼠标光标为 move 类型，cursorAt 指示光标所在的位置。第 17、20 和 23 行分别表示当 div 处于拖曳状态时，可以使用 enable、disable 或 destroy 这些参数来启用、禁用或删除拖曳功能，当删除了拖曳功能后，必须要再次初始化 draggable()才能继续使用拖曳。

6.3　限制移动范围的拖放

jQuery UI 的 draggable()函数具有很多的可选配置参数，其中一项是可以限制拖动的范围，这个功能在实现一些小游戏时非常有用，比如限制俄罗斯方块只能在一定的范围内移动，或者是在某些条件下只能水平和垂直移动。本例演示了如何限制 div 只能向水平、垂直方向移动，以及只能在一定的范围内进行移动。

```
01  /* constraintRange.html*/
02          <script type="text/javascript">
03              $(function() {
04                  $( "#draggable" ).draggable({ axis: "y" });          //仅可以垂直拖动的盒子
05                  $( "#draggable2" ).draggable({ axis: "x" });         //仅可以水平拖动的盒子
06                  //只能在容器#containment-wrapper 内部拖动的盒子
07                  $( "#draggable3" ).draggable(
08                                  { containment: "#containment-wrapper", scroll: false });
09                  //在当前盒子的父控件中可以拖动的盒子
10                  $( "#draggable5" ).draggable({ containment: "parent" });
11              });
12          </script>
```

第 04 行为拖动对象指定 axis 为 y 表示仅可以垂直移动 div 元素，第 05 行指定 axis 为 x 表示仅可以水平拖动盒子。第 07~08 行中的 draggable3 指定了 containment 为一个容器 div，表示该 div 仅能在容器内进行拖动，无法拖动到容器外面。第 10 行是最后一个 div，它指定了 containment 为 parent，表示只能在当前元素的父容器内部拖动这个 div 元素。

6.4 延迟进行的拖放

使用 draggable()函数，还可以设置延时拖放功能。所谓的延时拖放，是指当鼠标按下时并不立即进行拖曳，而是按指定的条件延迟一段时间以后再拖曳。比如可以在拖动到一定距离之后，或者是等待特定的时间之后再开始进行，下面的代码演示了这个功能：

```
01  /* deferredDrag.html*/
02  <script type="text/javascript">
03      $(function() {
04      $( "#draggable" ).draggable({ distance: 20 });   //指定仅当距离为 20 像素后才开始
05      $( "#draggable2" ).draggable({ delay: 1000 });   //仅当延时 1000 毫秒以后才开始
06      $( ".ui-draggable" ).disableSelection();         //在拖动时使其文字不能被选择
07    });
08  </script>
```

第 04 行中 draggable 的 distance 为 20，表示当距离移动超过 20 像素后，才开始进行拖动，这样就不至于当鼠标一放在上面时就马上开始拖动。第 05 行的 delay 表示当鼠标拉动了 1000 毫秒以后才开始进行拖放。第 06 行的 disableSelection()表示在拖放时，其中的文字不能被选择，否则拉动时选择文字内容会造成异常。

6.5 具有对齐功能的拖放

在一些工具软件中，拖动具有自动边缘对齐的功能，比如一些软件开发 IDE 中，拖动控件时，这些控件可以自动与容器边界进行对齐，这就大大减少了开发人员设计控件对齐

所耗费的工作量。无所不能的 jQuery 当然也提供了这个功能，下面的实例演示了如何创建具有捕捉效果的拖动：

```
01  /* snapDrag.html*/
02      <script type="text/javascript">
03          $(function() {
04              $( "#draggable" ).draggable({ snap: true });          //在拖曳时，捕捉所有的元素
05              $( "#draggable2" ).draggable({ snap: ".ui-widget-header" });//仅捕到特定容器边缘
06              //捕捉特定容器边缘，snapMode 指定为 outer 表示捕捉到外部边缘
07              $( "#draggable3" ).draggable({ snap: ".ui-widget-header", snapMode: "outer" });
08              $( "#draggable4" ).draggable({ grid: [ 20, 20 ] }); //以 20x20 单元格进行捕捉
09              $( "#draggable5" ).draggable({ grid: [ 80, 80 ] }); //以 80x80 单元格进行捕捉
10          });
11      </script>
```

可以看到，第 04 行的 snap 可以让拖曳时的控件具有对齐效果，snap 默认参数 true，表示在拖曳时对所有可能的目标都提供对齐功能。第 05 行的 snap 指定了一个容器类名称，拖动时会将目标控件向容器边缘进行对齐。第 07 行的 snapMode 指定 outer 表示仅对齐外边缘。第 08~09 行的 grid 表示按单元格进行对齐定位，比如按照 20x20 像素单位或 80x80 的像素单位进行捕捉。

6.6　自动滚动的拖放

当控件具有拖曳功能后，有时用户可能会将控件拖动到当前容器的显示范围之外。使用 jQuery UI 的 draggable 插件，可以让 div 容器自动进行滚动，以便显示滚动区域的内容。

本例效果如图 6.1 所示。可以看到，当 div 滚动到屏幕右下方时，相应的容器 div 自动滚动到了拖曳的位置，从而避免用户拖动到不可见的位置而导致控件出现丢失的现象。

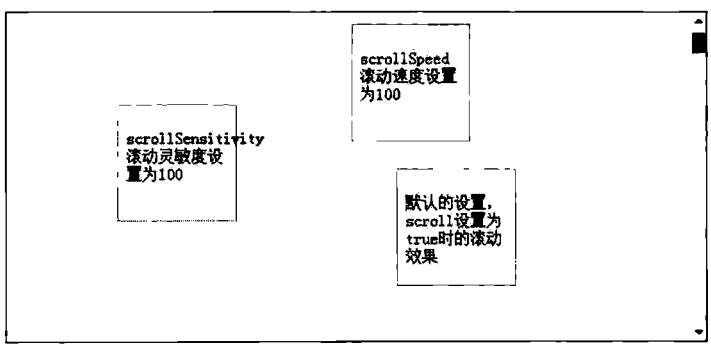

图 6.1　自动滚动的效果（注意右侧滚动条）

本例代码如下：

```
01  /* autoScroll.html*/
02      <script type="text/javascript">
03      $(function() {
```

```
04      $( "#draggable" ).draggable({ scroll: true });     //设置控件可滚动
05      //设置控件可滚动，滚动灵敏度设置为 100
06      $( "#draggable2" ).draggable({ scroll: true, scrollSensitivity: 100 });
07      //设置控件可滚动，滚动速度设置为 100
08      $( "#draggable3" ).draggable({ scroll: true, scrollSpeed: 100 });
09    });
10    </script>
```

第 04 行通过设置 scroll 为 true，让控件具有了在容器中进行自动滚动的效果，因而当拖动到容器之外时，容器可以自动滚动到控件所在的位置，第 06 行通过 scrollSensitivity 来设置滚动的灵敏度，第 08 行通过 scrollSpeed 可以设置滚动的速度。

6.7 反转位置的拖放

反转位置是指当将元素拖曳到某一位置松开鼠标时，并不会将元素放到目标位置，元素会跳回到原来的拖曳位置，这种方式在构建购物车时特别有用，如果没有拖放到购物车上，那么商品将恢复到原始位置。下面的实例演示了如何使用拖曳反转功能，代码如下：

```
01  /* revertDrag.html*/
02    <script type="text/javascript">
03        $(function() {
04            $( "#draggable" ).draggable({ revert: true });     //进行拖曳反转
05            //反转拖曳，以复制的方式拖曳
06            $( "#draggable2" ).draggable({ revert: true, helper: "clone" });
07        });
08    </script>
```

第 04 行通过设置 revert 为 true，可以让拖曳恢复到原来的位置，第 06 行 helper 参数指定为 clone，表示以复制的方式进行拖曳，因此可以看到在拖曳时，元素依然保持在原始位置，当松开鼠标时，被复制的元素会飞回到原来的位置。

6.8 使用事件监控拖曳次数

draggable 插件会在拖动过程中触发几个事件，以便于记录拖动的过程，比如当按下鼠标准备拖动时，触发 start 事件，拖动的过程中触发 drag 事件，拖动完成时触发 stop 事件。可以在 draggable()函数中为这些事件关联事件代码。本例将显示这些事件被触发的次数，以便了解拖动过程实现的原理。本例效果如图 6.2 所示。

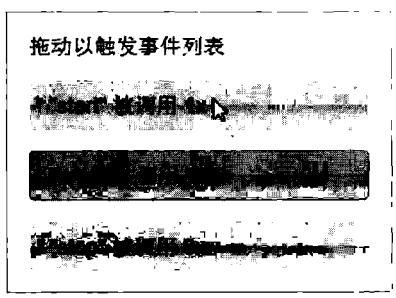

图 6.2 拖动事件列表

本例代码如下：

```
/* dragCount.html*/
  <script type="text/javascript">
      $(function() {
        var $start_counter = $( "#event-start" ),          //开始拖动时的计数器li
          $drag_counter = $( "#event-drag" ),              //拖动时的计数器li
          $stop_counter = $( "#event-stop" ),              //停止计数器li
          counts = [ 0, 0, 0 ];                            //定义起始的计数器数组
        $( "#draggable" ).draggable({                      //关联拖曳函数
          start: function() {                              //关联拖曳起始事件
            counts[ 0 ]++;                                 //计数器加1
            //更新计数器的状态
            updateCounterStatus( $start_counter, counts[ 0 ] );
          },
          drag: function() {                               //关联拖曳进行事件
            counts[ 1 ]++;                                 //添加计数器
            //更新计数器的状态
            updateCounterStatus( $drag_counter, counts[ 1 ] );
          },
          stop: function() {                               //关联拖曳停止事件
            counts[ 2 ]++;                                 //添加计数器
            //更新计数器的状态
            updateCounterStatus( $stop_counter, counts[ 2 ] );
          }
        });
        //更新计数器状态的公共函数
        function updateCounterStatus( $event_counter, new_count ) {
          if ( !$event_counter.hasClass( "ui-state-hover" ) ) { //如果不具有ui-state-hover类
            $event_counter.addClass( "ui-state-hover" )    //添加该CSS类
              .siblings().removeClass( "ui-state-hover" );  //移除相同层次的其他CSS类
          }
          $( "span.count", $event_counter ).text( new_count ); //更新计数器的值
        }
      });
  </script>
```

在draggable()函数内部，关联了start、drag和stop事件，在事件内部，会为全局的计数器数组递增其中的数组元素，然后调用updateCounterStatus()公共函数来更新计数器的显示，从而实现事件触发次数的跟踪效果。

6.9 拖动时动态更改鼠标的光标类型和位置

拖动时光标跟着变化是最理想的效果，draggable插件当然不能缺少这个功能，除此之

外，还可以在拖动时定义光标位于哪个位置（容器的上下左右等位置）。通过为 draggable 指定 cursor 和 cursorAt，可以定义拖动时显示的光标类型和位置。代码如下：

```
01  /* positionDrag.html*/
02      <script type="text/javascript">
03          $(function() {
04              //将光标变为移动光标,指定鼠标位于左上角 56 像素位置
05              $( "#draggable" ).draggable({ cursor: "move", cursorAt: { top: 56, left: 56 } });
06              //将光标更改为十字型光标，指定鼠标位置在图片上左上角的 5 像素处
07              $( "#draggable2" ).draggable
08                      ({ cursor: "crosshair", cursorAt: { top: -5, left: -5 } });
09              //光标不变，光标位置更改为底部
10              $( "#draggable3" ).draggable({ cursorAt: { bottom: 0 } });
11          });
12      </script>
```

第 05 行设置 cursor 为 move，表示在移动时将鼠标光标变成移动图标，cursorAt 指定光标所在的位置，top 和 left 用来指定鼠标光标位于屏幕的左上角。第 07~09 行的第 2 个实例设置 cursor 为 crosshair，将显示十字形光标，并且光标位于拖动元素的左上角 5 像素位置。第 10 行在拖动时将光标置于底部。

6.10 拖曳并放置到目标容器

在编写拖曳实例时，可以使用 jQuery UI 的 draggable 来拖曳一个元素，如果要对放置的目标容器进行控制，可以使用 jQuery UI 提供的 droppable()函数。当为一个元素关联了 droppable 之后，就可以对目标容器进行详细控制（如是否允许放到该位置等）了。本例效果如图 6.3 所示。

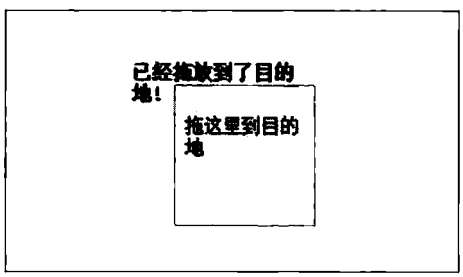

图 6.3 拖曳到目的地的效果

下面使用 droppable 来构造一个拖放到目标容器的实例：

```
01  /* dragDrop.html*/
02      <script type="text/javascript">
03          $(function() {
04              $( "#draggable" ).draggable();          //对可拖曳的元素开启拖放功能
05              $( "#droppable" ).droppable({           //启用拖放功能的支持
```

```
06              drop: function( event, ui ) {          //当拖放到目标容器时，触发 drop 事件
07                $( this )                            //为元素添加新样式
08                  .addClass( "ui-state-highlight" )
09                  .find( "p" )                       //设置元素内部的 p 的 html 代码
10                    .html( "已经拖放到了目的地!" );  } });  });
11    </script>
12  </head>
13  <body>
14   <!--可拖放的元素-->
15  <div id="draggable" class="ui-widget-content">
16    <p>拖这里到目的地</p>
17  </div>
18   <!—目标容器-->
19  <div id="droppable" class="ui-widget-header">
20    <p>请放在这里</p>
21  </div>
22  </body>
```

第 05~10 行为目标容器设置 droppable()函数，并为该方法关联 drop 事件，drop 事件会在被拖曳的元素放到目标容器时触发。在这个实例中，第 08 行更改了目标容器的样式，第 10 行设置其 html 为新的文本内容，表示已经放置到了目的地。

6.11　限制可放置的元素

对于可放置拖曳元素的目标容器，有时候需要限制一些元素，即一些元素可以放置到目的地，而另一些元素不可以放置到目的地。使用 droppable 的功能，可以很容易地实现这个功能。本例效果如图 6.4 所示。

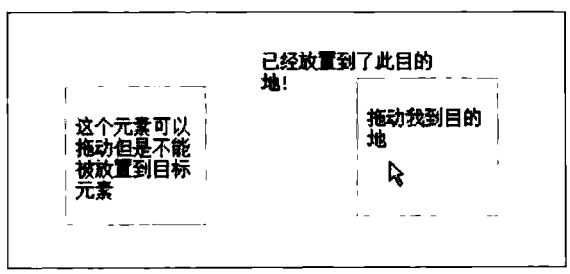

图 6.4　限制可放置的元素

本例代码如下：

```
01  /* dropElement.html*/
02    <script type="text/javascript">
03          $(function() {
04              $( "#draggable, #draggable-nonvalid" ).draggable(); //开启拖放
05              $( "#droppable" ).droppable({                       //设置拖放放置参数
```

```
06              accept: "#draggable",                          //设置允许拖放的 ID
07              activeClass: "ui-state-hover",                 //当鼠标经过时的样式
08              hoverClass: "ui-state-active",                 //当鼠标在上面移动时的样式
09              drop: function( event, ui ) {                  //当放置时的事件处理代码
10                  $( this )
11                      .addClass( "ui-state-highlight" )      //添加 CSS 样式
12                      .find( "p" )
13                          .html( "已经放置到了此目的地!" );    //设置此目的地的文本内容
14              }
15          });
16      });
17  </script>
```

第 05~14 行的 droppable 具有一个参数 accept，可以指定一个可接收元素的选择器或是一个经过计算返回 true 的函数。这样就可以控制哪些元素可以放到容器 div 上面。

6.12 实现可拖放的购物车

购物车是电子商务网站中必须有的一个功能，有了 jQuery UI，可以为购物车添加一些生动的特效。比如从商品列表中拖曳一个产品到购物车中，将购物车当成一个放置的容器。本例实现一个非常简单的可拖放的购物车，稍加完善，就可以实现一套非常有用的购物车了。本例效果如图 6.5 所示。

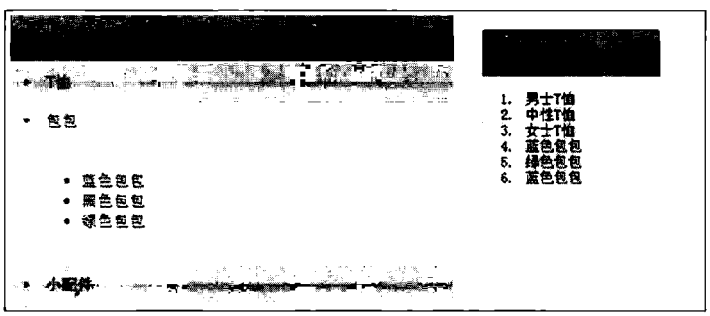

图 6.5 可拖放的购物车效果

本例代码如下：

```
01  /* dragShoppingCart.html*/
02      <script type="text/javascript">
03          $(function() {
04              $( "#catalog" ).accordion();                   //让 div 可以折叠显示
05              $( "#catalog li" ).draggable({                 //让每一个 li 元素都可以拖曳
06                  appendTo: "body",                          //在拖动过程中将元素添加到 body 中
07                  helper: "clone"                            //使用复制的方式进行拖动
08              });
09              $( "#cart ol" ).droppable({                    //将产品放置到购物车中
```

```
10              activeClass: "ui-state-default",          //激活时的样式
11              hoverClass: "ui-state-hover",             //经过时的样式
12              accept: ":not(.ui-sortable-helper)",      //可接受的元素
13              drop: function( event, ui ) {             //放置到目标位置时的事件
14                  $( this ).find( ".placeholder" ).remove();//移除容器
15                  //将拖曳的 li 添加到目标位置
16                  $( "<li></li>" ).text( ui.draggable.text() ).appendTo( this );
17              }
18          }).sortable({                                 //对容器进行排序
19              items: "li:not(.placeholder)",            //获取所有的 li 元素列表
20              sort: function() {                        //对列表进行排序
21                  $( this ).removeClass( "ui-state-default" );
22              }
23          });
24      });
25  </script>
```

第 04 行的 accordion 表示折叠显示 div，第 05 行的 draggable 开启拖曳效果，第 06 行的 appendTo 是指拖动时将 li 添加到 body 部分。第 09 行的 droppable()用来指定目标容器，第 12 行的 accept 指定要接收的是所有商品 li，而非其他的容器 li（比如“添加到购物车”按钮的 li）。第 16 行将拖曳的 li 添加到容器中。第 18 行调用 sortable()对购物车中的商品进行排序显示。

6.13　可排序的拖放

所谓的排序功能是指当拖动一个元素到列表中的另一个位置时，列表中的其他元素进行适当的排列以适应新的位置。jQuery UI 的 shortable()可以很轻松地实现这个功能。本例创建一个简单的排序列表，列表由 7 个 li 元素组成。运行本例可以看到，拖动列表项中的内容，其他列表项中的内容会自动进行排列，如图 6.6 所示。

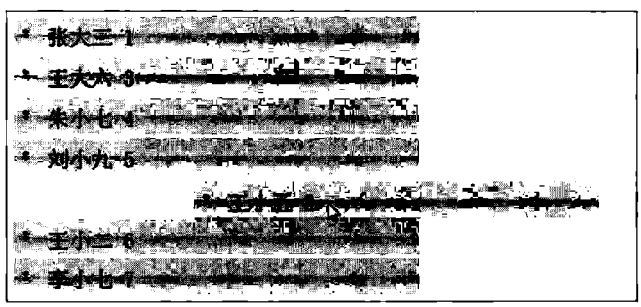

图 6.6　可排序的拖放

本例代码如下：

```
01  /* dragSort.html*/
02      <script type="text/javascript">
```

```
03          $(function() {
04              $( "#sortable" ).sortable();          //使得 ul 中的 li 元素具有可排序的拖动效果
05              $( "#sortable" ).disableSelection();  //禁止选择文本
06          });
07      </script>
08  </head>
09  <body>
10
11  <ul id="sortable">
12      <li class="ui-state-default"><span class="ui-icon ui-icon-arrowthick-2-n-s">
13      </span>张大三 1</li>
14      .....
15      <li class="ui-state-default"><span class="ui-icon ui-icon-arrowthick-2-n-s">
16      </span>李小七 7</li>
17  </ul>
```

第 04~05 行通过使用 sortable()就可以让一个 ul 中的项目元素具有可拖动的排序效果。

注意：sortable()默认共享 draggable 所有属性，并且其内置了 draggable 的功能，因此当使用 sortable()时，便自动具有了 draggable 的功能。

6.14 在多个列表之间进行拖放

在开发一些配置项的网页时，可能经常需要在两个不同的列表框之间进行拖动配置，这个功能可以利用 jQuery UI 提供的 sortable 插件来实现。它是通过 sortable 的 connectWith 选项关联到目标列表框来实现的。本例效果如图 6.7 所示。

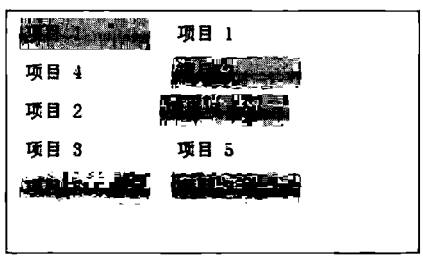

图 6.7 在两个列表框之间进行拖放

本例代码如下：

```
01  /* multiListDrag.html*/
02      <script type="text/javascript">
03          $(function() {
04              $( "#sortable1, #sortable2" ).sortable({   //开启拖动
05                  connectWith: ".connectedSortable"      //使用 class 类作为 connectWith 的拖动分组
06              }).disableSelection();                     //禁用文本选择
07          });
```

```
08    </script>
09  </head>
10  <body>
11
12  <ul id="sortable1" class="connectedSortable">
13    <li class="ui-state-default">项目 1</li>
14    ....
15    <li class="ui-state-default">项目 5</li>
16  </ul>
17   <ul id="sortable2" class="connectedSortable">
18    <li class="ui-state-highlight">项目 1</li>
19    ....
20    <li class="ui-state-highlight">项目 5</li>
21  </ul>
```

第 12~21 行构建了两个 ui 元素，这两个元素的 class 都是 connectedSortable，通过用一个 CSS 样式对其进行分组。在第 04 行调用 sortable()时，设置 connectWith 参数，传入样式类 connectedSortable，这样就实现了在两个 ul 列表之间进行拖动。

6.15　使用拖动方式选择多个元素

在 Windows 的资源管理器中，可以用拖动的方式选中多个文件或文件夹，这种拖动的方式非常方便。jQuery UI 中的 selectable 插件，让用户也可以轻松地使用拖动的方式完成选择。本例效果如图 6.8 所示。

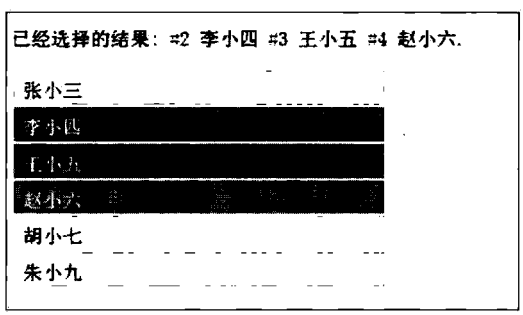

图 6.8　使用拖动方式进行多选

本例代码如下：

```
01  /* dragMultiSelect.html*/
02    <script type="text/javascript">
03    $(function() {
04      $( "#selectable" ).selectable({                          //进行拖动多选
05        stop: function() {                                     //触发 stop 事件
06          var result = $( "#select-result" ).empty();        //清空结果列表
07          $( ".ui-selected", this ).each(function() {        //循环选中的结果
```

```
08              var index = $( "#selectable li" ).index( this );          //获取选中的索引
09              result.append( " #" + ( index + 1 )+" "+$(this).html());  //显示选中的索引和值
10                alert(this);
11            });
12          }
13        });
14    });
15    </script>
16  </head>
17  <body>
18
19  <p id="feedback">
20  <span>已经选择的结果:</span> <span id="select-result">none</span>.
21  </p>
22   <ol id="selectable">
23    <li class="ui-widget-content">张小三</li>
24    ...
25    <li class="ui-widget-content">朱小九</li>
26  </ol>
```

第 04~13 行为 ul 元素调用了 selectable，第 05 行关联了 stop 事件，即当拖动选择结束时，该事件触发。第 06~11 行遍历选中的列表项，即选中的列表项的 CSS 样式类为 ui-selected，通过循环该列表项，得到选中的列表项的索引和值，显示在结果框中。

6.16 在两个 Tab 标签之间进行元素拖动

在实际的网页应用中，经常会使用 Tab 页来分类组织多个不同的项。jQuery UI 的 sortable 和 droppable 可以提供排序拖动的效果，特别是当拖动一个 Tab 页中的元素到 Tab 标签上时，可以动态地切换到该 Tab 页，并且以排序拖动的方式将一个 Tab 页中的元素拖动另一个 Tab 页。本例效果如图 6.9 所示。

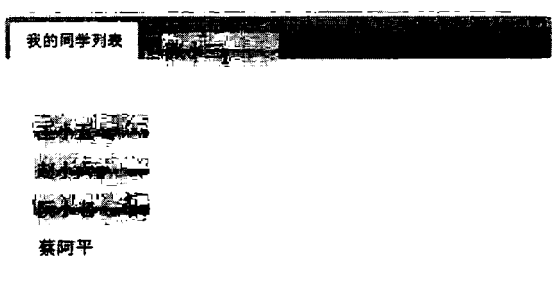

图 6.9 拖动到 Tab 标签页中

本例代码如下：

```
01  /* 2tabDrag.html*/
```

```
02    <script type="text/javascript">
03        $(function() {
04            $( "#sortable1, #sortable2" ).sortable().disableSelection(); //使列表元素可拖动
05             var $tabs = $( "#tabs" ).tabs();              //使之显示 Tab 标签页
06             var $tab_items = $( "ul:first li", $tabs ).droppable({//使 Tab 的标题元素可以放置
07              accept: ".connectedSortable li",             //接收来自其他 Tab 页中的 li 元素
08              hoverClass: "ui-state-hover",                //变更样式
09              drop: function( event, ui ) {                //在目标 Tab 上放置的事件
10                 var $item = $( this );                    //获取当前的 item
11                 var $list = $( $item.find( "a" ).attr( "href" ) ) //获取当前 item 中 a 的 herf 属性
12                   .find( ".connectedSortable" );
13                      ui.draggable.hide( "slow", function() { //隐藏原始元素的显示
14                      //激活当前 tab 的显示
15                   $tabs.tabs( "option", "active", $tab_items.index( $item ) );
16                   $( this ).appendTo( $list ).show( "slow" ); //将当前拖曳的元素添加到目标
17                 });
18              }
19            });
20        });
21    </script>
22  </head>
23  <body>
24   <div id="tabs">
25    <ul>
26      <!--tab 标签页-->
27      <li><a href="#tabs-1">我的同学列表</a></li>
28      <li><a href="#tabs-2">我的好友列表</a></li>
29    </ul>
30      <!--tab1 内容页-->
31    <div id="tabs-1">
32      <ul id="sortable1" class="connectedSortable ui-helper-reset">
33        <li class="ui-state-default">张小三</li>
34        …
35        <li class="ui-state-default">阮小七</li>
36      </ul>
37    </div>
38      <!--tab2 内容页-->
39    <div id="tabs-2">
40      <ul id="sortable2" class="connectedSortable ui-helper-reset">
41        <li class="ui-state-highlight">李阿福</li>
42        …
43        <li class="ui-state-highlight">六阿三</li>
44      </ul>
45    </div>
46  </div>
```

HTML 页面上第 31~45 行定义了两个 Tab 标签，它们使用了 jQuery UI 中的 tabs 插件来为两个 li 元素添加 Tab 页。第 04 行为每一个标签页中的 div 元素设置 sortable 插件，使其可以拖动排序。第 06~19 行在 Tab 标签页上关联 droppable，表示 Tab 标签页上可以放置 li 元素。第 09~17 行表示当 drop 事件触发时，通过代码将 Tab 中放置的元素动态添加到新的 Tab 页中，并且隐藏原来 Tab 标签页中的元素。

6.17 拖动表格选择多行数据

在桌面型的应用程序上，拖动是可以选择多行数据的，这个功能通过 jQuery UI 的 selectable 插件也可以很轻松地实现。本例创建了一个 HTML 表格，表格上放置了水果数据，通过拖动就可以选中多行水果，并在最终结果上显示选中的水果名称。本例效果如图 6.10 所示。

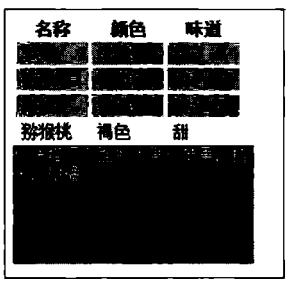

图 6.10 拖动选择多行表格

本例代码如下：

```
01  /* dragTableSelect.html*/
02  <script type="text/javascript">
03    $(function() {
04      $('#fruits').selectable({          //为表格添加 selectable 插件
05        filter:'tbody tr',               //过滤掉 tbody 和 tr，只选择 td
06        stop: function(event, ui){       //当停止选择表格时，在结果区中添加选中的数据
07          var result = $( "#plate" ).empty().html('你选择了下面的水果:'
08              + $( this ).find( ".ui-selected" ).map(function() {
09            return this.id;
10          }).get().join(", "));          //循环添加选中的多行数据
11        }
12      });
13    });
14  </script>
15  </head>
16  <body>
17  <!--构建一个用来进行多选的 HTML 表格-->
18  <table id="fruits">
```

```
19      <thead>
20        <tr><th>名称</th><th>颜色</th><th>味道</th></tr>
21      </thead>
22      <tbody>
23        <tr id="苹果"><td>苹果</td><td>绿色</td><td>甜</td></tr>
24        ……
25      </tbody>
26    </table>
27    <!--显示表格的结果数据-->
28    <div id="plate">多选的结果：</div>
```

从代码可以看出，第 18~26 行构建了一个 HTML 表格，第 04~11 行为这个表格应用了 selectable 插件事件，第 05 行的 filter 过滤掉了 tbody 和 tr，只选中 td。第 06~10 行在 stop 事件中，调用 map()函数循环选中的结果，将选中结果添加到结果 div 中。

6.18　拖动表格时自动选中复选框

很多应用都会在 HTML 的表格上放置一个复选框，为了允许用户更灵活地进行多项选择，可以通过 jQuery UI 的 selectable 让用户在拖动时选定复选框。本例效果如图 6.11 所示。

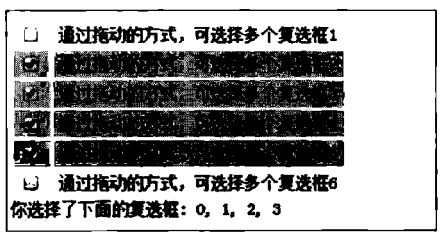

图 6.11　使用拖动进行多选

本例代码如下：

```
01  /* dragCheckbox.html*/
02  <script type="text/javascript">
03      $(document).ready(function(){
04          $(".tablo").selectable({                    //为表格设置拖动选择效果
05          filter: "tr",                               //过滤掉 tr 表格行
06          selected: function(evennt,ui) {             //当选中一行时，将该行的复选框设置为 true
07          $( ".ui-selected", this ).each(function(a,b) {
08                  $('.checkbox',b).attr('checked',true);
09          });
10          },
11        stop: function(event, ui){                    //当停止选择表格时，在结果区中添加选中的数据
12          var result = $( "#plate" ).empty().html('你选择了下面的复选框: '
13              + $( this ).find( ".ui-selected" ).map(function(i) {
14            return i;
```

```
15                }).get().join(", "));          //循环添加选中的多行数据
16            }
17        });
18    });
19    </script>
20    </head>
21    <body>
22    <table class="tablo">
23    <tr class="ui-widget-content">
24    <td><input type="checkbox" class="checkbox" /></td>
25    <td>通过拖动的方式，可选择多个复选框 1</td>
26    </tr>
27          ……
28    </table>
29    <!--显示表格的结果数据-->
30    <div id="plate">多选的结果：</div>
```

第 04~10 行通过设置 selectable 的 selected 事件，在选中某一行时，将该行的复选框设置 true。第 11~16 行在 stop 事件中，向结果 div 中添加值表示已经正确选择了数据。

6.19 拖动表格行并放置到目标位置

表格行可以拖动到一个目标位置，然后将其放置到目标位置，这个功能有了 draggable 的支持变得非常简单。本例将搭建一个寻找发明家的游戏，例子中有一个 HTML 表格，表格中有发明家的名字，要求用户通过鼠标，将发明家的名字拖动到其相应的发明 input 输入框中。本例效果如图 6.12 所示。

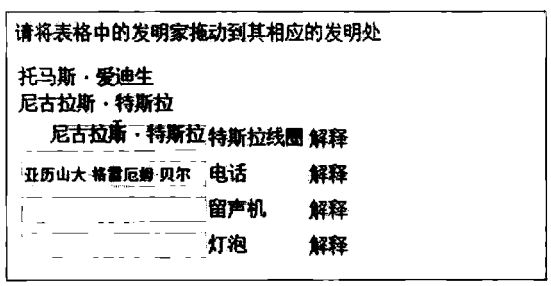

图 6.12 表格行的拖动效果

本例代码如下：

```
01    /* dragTableDrop.html*/
02    <script type="text/javascript">
03        $(document).ready(function(){
04            var c = {};                          //保存已经拖动的 tr 的引用
05            $("#inventor tr").draggable({        //对表格进行拖动
06                    helper: "clone",             //拖动时复制
```

```
07                      start: function(event, ui) {          //在开始拖动时，保存拖动的行
08                          c.tr = this;
09                          c.helper = ui.helper;
10                      }
11              });
12              $("#invention tr").droppable({                //放置元素插件
13                  drop: function(event, ui) {               //放置时，触发该事件
14                      var inventor = ui.draggable.text();   //获取拖动的元素文本
15                      $(this).find("input").val(inventor);  //向文本框中写入文本
16                      $(c.tr).remove();                     //移除表格行和 helper
17                      $(c.helper).remove();
18                  }
19              });
20      });
21  </script>
22  </head>
23
24  <body>
25  <p>请将表格中的发明家拖动到其相应的发明处</p>
26  <div id="inventor">
27  <table>
28      <tbody>
29          <tr id="1"><td>亚历山大·格雷厄姆·贝尔</td></tr>
30          <tr id="2"><td>托马斯·爱迪生</td></tr>
31          <tr id="3"><td>尼古拉斯·特斯拉</td></tr>
32      </tbody>
33  </table>
34  </div>
35  <form>
36    <div id="invention">
37          <table>
38          <tbody>
39          <tr><td><input name="answer1" />特斯拉线圈</td><td>解释</td></tr>
40          <tr><td><input name="answer2" />电话</td><td>解释</td></tr>
41          <tr><td><input name="answer3" />留声机</td><td>解释</td></tr>
42          <tr><td><input name="answer4" />灯泡</td><td>解释</td></tr>
43          </tbody>
44          </table>
45      </div>
46  </form>
```

第 05~19 行可以看到为 inventor 和 invention 这两个 div 的 tr 分别关联了 draggable()和 droppable()函数，第 05~11 行 draggable()中使用 help 参数表示进行复制拖曳，第 12~19 行的 droppable()是指当放置表格行时，为相应的 input 写入文字，实现了拖动表格中的发明家到相应发明中的功能。

6.20 表格的拖动排序

表格的拖动排序可以为网站提供很灵活的互动效果，使用 sortable 插件，这个功能也可以轻松地实现。本例演示了如何实现可滚动的表格的拖放实例，其中也调整了在 Firefox 中的兼容性问题。本例效果如图 6.13 所示，在表格行进行拖动时，会显示滚动条，并且拖动时的行宽度保持与其他表格行一致。

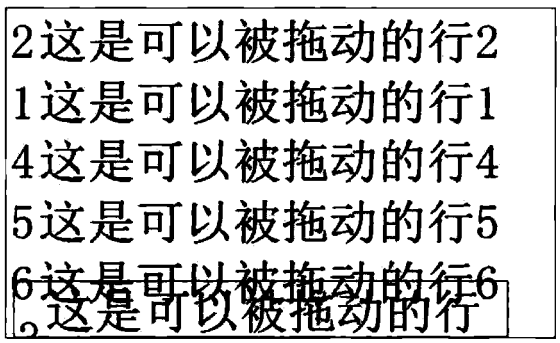

图 6.13 表格行拖动排序

本例代码如下：

```
/* dragTableSort.html*/
<script type="text/javascript">
  $(document).ready(function () {
    $('#myTable tbody').sortable({          //使表格行可以排序拖动
        start: function (event, ui) {
      //如果浏览器为 Firefox 则调整其位置，使其绝对定位显示并设置 margin-top 为当前顶层
        if (navigator.userAgent.toLowerCase().match(/firefox/) && ui.helper !== undefined) {
            ui.helper.css('position', 'absolute').css('margin-top', $(window).scrollTop());
             //当滚动条滚动时关联事件
             $(window).bind('scroll.sortableplaylist', function () {
                ui.helper.css('position', 'absolute').css('margin-top', $(window).scrollTop());
             });
         }
        },
        beforeStop: function (event, ui) {
            //取消对 Firefox 滚动的调整
       if (navigator.userAgent.toLowerCase().match(/firefox/) && ui.offset !== undefined) {
          $(window).unbind('scroll.sortableplaylist');
            ui.helper.css('margin-top', 0);
            }
        },
        helper: function (e, ui) {                    //设置拖动时的辅助显示帮助
            ui.children().each(function () {
                $(this).width($(this).width());       //调整 ui 的宽度
```

```
25                  });
26                  return ui;                          //返回调整后的 ui 的宽度
27              },
28              scroll: true,                           //允许滚动显示
29              stop: function (event, ui) {
30                  //在拖动排序停止时可以保存排序
31              }
32          }).disableSelection();
33      });
34  </script>
```

第 05~14 行的 start 事件和 beforeStop 事件用来设置 Firefox 拖放时的边界显示问题，第 22~27 行的 helper 事件在拖动时，可以设置拖动时元素的等宽显示，第 28~32 行的 scroll 属性设置为 true，表示自动进行滚动显示，在 stop 事件中可以通过循环当前的元素来保存已经设置好的排序。

6.21　拖动调整控件的大小

很多控件都具有拖动时自动调整大小的功能。比如一个文本框，拖动时可以将其放大进而显示更多的内容；比如一张照片，可以通过拖动某个角的方式来进行缩放。要实现这样的效果，可以使用 jQuery UI 的 resizable 插件，这个插件可以自动地为控件添加调整大小的锚点，允许用户通过拖动来调整其大小。本例效果如图 6.14 所示。

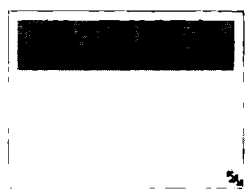

图 6.14　拖动时调整大小

本例代码如下：

```
01  /* dragChangeSize.html*/
02      <script type="text/javascript">
03       $(function() {
04          $( "#resizable" ).resizable();
05       });
06      </script>
07  </head>
08  <body>
09   <!--这是一个可以拖动的 div 元素-->
10  <div id="resizable" class="ui-widget-content">
11      <h3 class="ui-widget-header">拖动右下角的锚点可以调整控件的大小。</h3>
12  </div>
```

第 04 行只需要 div 元素调用 resizable()函数，就可以调整控件的大小了。当鼠标放置在控件上时，鼠标光标变成了调整光标，并且会看到一个可以调整大小的锚点。

6.22 调整控件大小时设置调整约束

通过拖动的方法调整控件大小时，可以设置容器约束，或者是通过指定其 maxHeight、maxWidth、minHeight、minWidth 来设置控件可以调整到的大小。本例演示了如何为调整大小的控件设置约束，代码如下：

```
01  /* dragConstraintWidthHeight.html*/
02    <script type="text/javascript">
03      $(function() {
04        $( "#resizable" ).resizable({
05            //containment: "#container"      //或者通过容器来约束大小
06            animate: true,                   //设置调整大小的动画
07            helper: "ui-resizable-helper",   //指定调整大小时的辅助显示样式
08            maxHeight: 250,                  //指定最大高度
09            maxWidth: 350,                   //指定最大宽度
10            minHeight: 150,                  //指定最小高度
11            minWidth: 200                    //指定最小宽度
12          });
13      });
14    </script>
```

第 06 行的 animate 用来设置调整大小的动画，第 07 行的 helper 设置一个 CSS 样式，用来指定当调整大小时辅助的显示样式。第 08~11 行通过指定最大高度和宽度以及最小高度和宽度来设置控件的调整约束。

注意：可以通过 containment 来设置容器约束，即可调整的大小限制在其最外层的容器内部。

6.23 使用拖动的方式调整表格的宽度

使用 jQuery UI 的 resizable 插件也可以为表格单元格添加拖动调整大小的效果。本例创建一个 HTML 表格，然后为表格单元格应用 resizable 来实现宽度的调整。本例效果如图 6.15 所示。

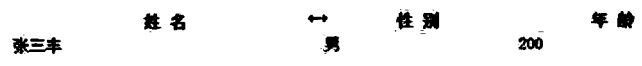

图 6.15 调整表格大小

本例代码如下：

```
01  /* dragTableWidth.html*/
02  <script type="text/javascript">
```

```
03        $(function () {
04            $('table th').resizable({          //为表格表头添加可拖动的效果
05                handles: 'e',                  //指定拖动锚点为 e
06                minWidth: 18                   //指定最小宽度为 18px
07            });  });
08    </script>
09    </head>
10    <body>
11    <!--将要调整单元格宽度的表格-->
12    <table style="width: 80%">
13        <tr>
14            <th>姓 名</th>
15            <th>性 别</th>
16            <th>年 龄</th>
17        </tr><tr>
18            <td>张三丰</td>
19            <td>男</td>
20            <td>200</td>
21        </tr>
22    </table>
23    </body>
```

第 12~22 行定义了一个 HTML 表格，第 04~07 行为这个表格的 th 单元格应用 resizable，指定 handles 为 e，也可以指定为 n、e、s、w、ne、se、sw、nw、all，或者是同时通过逗号分隔来指定多个。第 06 行的 minWidth 指定值为 18，表示最小的调整大小不能低于 18 像素。

6.24　设计可改变单元格宽度并可以多选的表格

通过 selectable 和 resizable 这两个 jQuery UI 的插件，可以让一个表格既能多选，也能调整表格的宽度大小。本例定义了两个 HTML 表格：一个表格用来设置显示表格的表头，另一个表格显示表格的内容，最后通过 selectable 和 resizable 来设置其多选和可调整大小的功能。本例效果如图 6.16 所示。

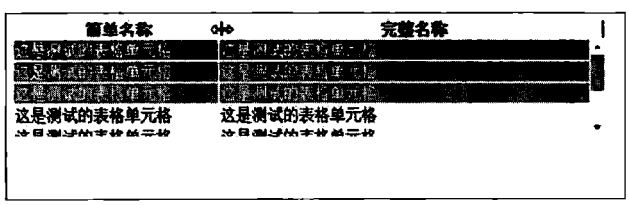

图 6.16　可拖曳和多选的表格

本例代码如下：

```
01    /* changeWidthMultiSelect.html*/
02    <script type="text/javascript">
```

```
03      $(function() {
04       var element = $('#MyParentDiv');                    //获取容器的 div 元素
05       $(".selectable", element).selectable({filter: 'tr'});   //设置表体的 tbody 区可以多选
06       $(".th0", element).resizable({                      //设置表头的单元格可以调整大小
07            alsoResize: '#MyParentDiv .header-container', //当调整该大小时级联调整表体大小
08
09            stop: function(event, ui) {                    //当调整大小结束后
10                 var width1 = $(".th0", element).width();  //得到表头列的宽度
11                 $('.col0', element).width(width1);        //设置表体列的宽度
12                 width1 = $(".header-container", element).width(); //得到容器宽度
13                 $('.y-scroll', element).width(width1);    //设置滚动区域的宽度
14            },
15            handles: 'e'});                                 //指定拖曳手柄
16       $(".th1", element).resizable({                      //调整表体的宽度
17            alsoResize: '#MyParentDiv .header-container', //同时调整表头和容器的宽度
18            stop: function(event, ui) {                    //当调整宽度停止时
19                 var width1 = $(".th1", element).width();  //获取表体列的宽度
20                 $('.col1', element).width(width1);        //调整表体列的宽度
21                 width1 = $(".header-container", element).width();//设置表头的宽度
22                 $('.y-scroll', element).width(width1);    //设置容器滚动条的宽度
23            },
24            handles: 'e'});                                 //指定拖曳手柄
25      });
26  </script>
27  </head>
28  <body>
29       <!--表格容器 div，包含头部表格和内容表格-->
30       <div id="MyParentDiv" class="NCList">
31            <!--头部表格的容器 div-->
32        <div class="header-container" style="width:215px;">
33            <table><thead><tr>
34                 <th><div class="nc-cell th0" style="width:100px;">
35                      简单名称
36                 </div></th>
37                    <th><div class="nc-cell th1" style="width:100px;">
38                      完整名称
39                 </div></th>
40            </tr></thead></table>
41       </div>
42            <!--表体表格的容器 div-->
43       <div class="y-scroll" style="max-height:100px;width:215px;">
44            <table class="valuefield">
45                 <tbody class="selectable">
46  <tr><td><div class="nc-cell col0" style="width: 100px">这是测试的表格单元格</div></td>
47  <td><div class="nc-cell col1" style="width: 100px">这是测试的表格单元格</div></td>
```

```
48                    </tr>
49                    ......
50                </tbody>
51            </table>
52        </div>
53    </div>
```

第 32~41 行将表头放在一个单独的 div 中，用一个 table 元素表示。第 43~52 将表体也放在一个单独的 div 中，用一个 table 元素表示。第 05 行为表体的 table 应用了 selectable，使其支持多选。第 06~24 行分别为表头中的两个单元格 th0 和 th1 应用 resizable 插件。这两个插件都关联了 alsoResize 属性，用来指定当列移动时，相应的表头或表体的 Tab 页也执行 resizable()进行调整大小，形成级联调整的效果。第 09 和 18 行的 stop 事件中，分别调整其宽度以及容器的宽度，以便于容器能够显示或隐藏滚动条。

第 7 章　jQuery 操作图形图像

图片显示与处理是网站开发必不可少的功能之一，jQuery 强大的代码库提供了很多非常有用的图片处理功能，比如图片的淡入淡出特效、图片放大特效、鼠标悬停时的图片特效等等。本章将以 jQuery 图片显示与处理为要点，介绍一些图片操作的 jQuery 实用代码。

本章主要涉及的知识点有：

- 正确处理图片无法显示以及图片 URL 问题。
- 解析 JOSN 文件，显示 Flickr 与 Gloogle Picasaweb 图片。
- 动态调整图片的大小。
- 自适应背景图像的实现。
- 随机与按顺序的淡入淡出图片显示。

7.1　如何更好地处理图片无法显示问题

网页上的图片能够为网页带来生动直观的感观效果，图片的显示一般使用 HTML 的 img 元素，通过其 src 属性指向图片的地址，代码如下：

```
01  /* picNotLoad.html*/
02  <!--使用 HTML 标记 img 显示一幅图片-->
03  <img src="images/googlelogo.jpg" width="486" height="134" alt="Google 公司"
04   longdesc="http://www.google.com" />
```

如果图片缺失或因为图像格式本身的原因，导致图片无法显示，在浏览器中将会显示空白占位符，这会导致极不友好的网页体验，如图 7.1 就是在 Chrome 中显示一幅不存在的图片时的外观。

图 7.1　不存在的图片在 Chrome 中的显示效果

如果图片无法显示，开发人员会使用图像占位符，或通过一段 JavaScript 脚本动态地显示一幅预先设定好的图片。这种方法是在 jQuery 之前一直流行的处理图片无法显示的方式，而 jQuery 提供了更好的办法，使得开发人员可以极其安全地处理图片无法显示的问题。

jQuery 提供了一个名为 error 的事件，如果显示一幅不存在的图片时，error 事件会被触发，此时可以在这个事件中使用 jQuery 代码来安全地替代一幅其他的图片或是显示一段预先设计好的文本。

举个例子，如果希望在图片不存在时，只显示一行文本，而不是让浏览器出现一个占位符，可以使用如下代码：

```
01  /* picNotLoad.html*/
02  <script type="text/javascript">
03  $(document).ready(function(){
04    //当图片无法显示时，替换为文本
05    $("img").error(function(){
06      $("img").replaceWith("<p><b>图片未加载！</b></p>");
07    });
08  });
09  </script>
10  </head>
11  <body>
12  <!--使用 HTML 标记 img 显示一幅图片-->
13  <img src="images/googlelogo_1.jpg" width="486" height="134"
14  alt="Google 公司" longdesc="http://www.google.com" />
15  <br/>
16  <p>当图片不存在时，显示一行文本</p>
17  </body>
18  </html>
```

第 03~08 的页加载就绪事件中，使用$工厂函数选中页面上所有的 img 元素，为其 error 事件添加事件处理代码。第 06 行的事件处理代码中，调用 replaceWith()函数将 img 元素替代成括号中的 p 元素。也就是说，img 元素将从页面上移除，变为了 p 元素。在页面上，添加了一个 img 元素，指向某个不存在的文件。

可以看到 error 事件提供了更多的灵活性，本例效果如图 7.2 所示。

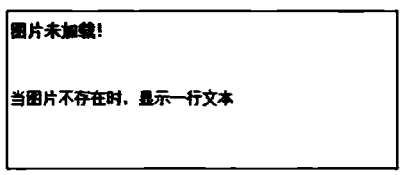

图 7.2 当图片不存在时显示文本内容

也可以将 error 事件处理函数稍加更改，使之显示另一幅图片，比如一幅友好的提示图片不存在的占位符图片，代码如下：

```
01  /* picNotLoad.html*/
02  <script type="text/javascript">
03      $(document).ready(function(){
04        //当图片无法显示时，使用一个占位符图片进行替换
05        $("img").error(function(){
06            $(this).attr("src", "images/noexistspic.jpg");
```

```
07          });
08      });
09  </script>
```

第 05~07 行在 error 事件处理函数内部，调用了 jQuery 的 attr()函数，设置选中的 img 元素的 src 属性为一幅占位符图片，以便在图片无法显示时显示占位符，效果如图 7.3 所示。

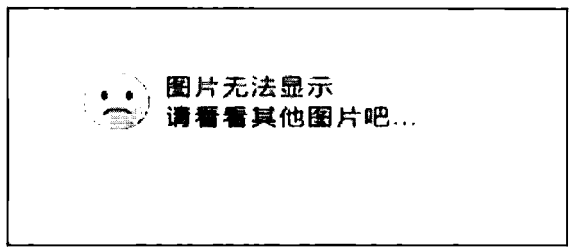

图 7.3　使用占位符图片

由此可见，当不确定图片文件是否存在或是元素没有正确载入时，通过 error 事件，可以给网页一种更加安全的处理办法。

7.2　如何显示图片直到页面加载完成

很多网站都有这样的一种特效，在页面加载时，先显示一幅图片，当页面加载完成后，图片隐藏，显示页面内容。比如一些网站在页面加载时，会显示一张宣传性的图片（或广告），以吸引用户的注意，当页面内容加载完成后，再隐藏这张图片，显示网页的内容。

注意：浏览器在解析 HTML 时，是从上向下依次解析执行的，因此为了实现页面加载时显示图片，只需要在页面的<body>开头部分放一个图片，然后在<body>结束或 jQuery 的页面就绪事件中隐藏这个图片，就可以达到这样的效果。

本例将在网页加载时显示一个加载中的图片，当网页加载完成后，隐藏加载的图片。本例效果如图 7.4 所示。

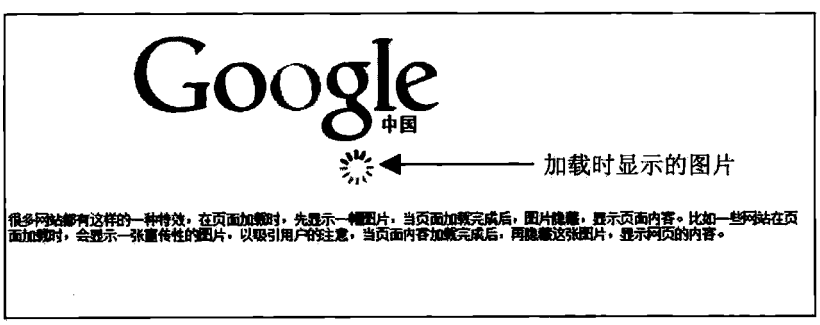

图 7.4　显示图片直到页面加载完成

本例代码如下：

```
01  /* loadPicCompleted.html */
```

```
<title>在页加载完成之前显示图片</title>
<!--添加对 jQuery 库的引用-->
<script type="text/javascript" src="jquery/jquery-1.9.1.js"></script>
<style type="text/css">
body{
    font-size:12px;
}
#loading{
    position:fixed;              /*固定显示*/
    _position:absolute;
    top:50%;                     /*上和左的位置均为页面的 50%，表示居中*/
    left:50%;
    width:124px;                 /*宽度和高度均为 124px*/
    height:124px;
    overflow:hidden;             /*对于溢出内容进行隐藏*/
    z-index:7;                   /*为 z 轴指定一个较大的值，使其显示在顶端*/
    /*背景图片*/
    background-image: url(images/loaderc.gif);
    /*背景不重复*/
    background-repeat: no-repeat;
    margin-top: -62px;           /*外部边距设置*/
    margin-right: 0;
    margin-bottom: 0;
    margin-left: -62px;
}
</style>
</head>

<body>
<!--页面加载时预先显示的图片容器--->
<div id="loading"></div>
<img src="images/googlelogo.jpg" width="486" height="134"
alt="Google 公司" longdesc="http://www.google.com" />
<br/>
<p>很多网站都有这样的一种特效，在页面加载时，先显示一幅图片，当页面加载完成后，图片
隐藏，显示页面内容。比如一些网站在页面加载时，会显示一张宣传性的图片，以吸引用户的
注意，当页面内容加载完成后，再隐藏这张图片，显示网页的内容。</p>
<script type="text/javascript">
  //隐藏 div 元素的显示
  $("#loading").fadeOut();
</script>
</body>
</html>
```

在HTML的body标签下面，紧随<body>起始标签添加一个div元素，用来放置加载时要显示的图片。第09~26行为id等于loading的这个div元素应用样式，使其居中显示一幅加载中的图片，通过CSS代码让其固定在页面中央，并且显示一幅名为loaderc.gif的图片。

注意：图片的CSS属性z-index设置为7，使得这幅图片在页面的最顶端进行显示，这样在页面加载时，首先显示出这幅图片。

在页面的所有内容加载完成后，也就是在</body>之前，第41行使用jQuery脚本隐藏加载图片。当浏览器加载完了body中的所有内容后，通过jQuery的fadeOut()函数来淡出加载时的图像，实现了图片在加载期间显示，加载完成后隐藏的效果。

7.3 预加载显示图片的方法

当浏览器遇到网页上的img元素时，首先会从缓存中查找src属性指定的图片，如果图片不存在，则会向户联网发送请求，获取图片。这样的图片获取方式在一些特殊要求的场合，比如属标悬停时切换图像、幻灯片图片展示等等，会出现等待图片加载的现象，使得整个网页显示极不流畅，给用户留下不好的浏览体验。

注意：预加载图片会先在缓存中加载所需要的图片，在显示网站内容时，在后台悄悄地预先加载图片。这样当网站加载完成时，所需要的图片也已经存在于缓存中，当用户操作（比如鼠标切换图片）时，就会带来更好的用户体验。

预加载图像的原理其实非常简单，它会在页面上创建隐藏的img元素，这些img元素会从互联网上加载图像，然后存储到缓存中，为了让预加载图像的功能更容易使用，通常将其编写为一个jQuery插件函数，代码如下：

```
01  /* preLoadingImage.html */
02  <script type="text/javascript">
03  (function($) {
04    var cache = [];
05    //编写一个预加载图片的jQuery函数
06    $.preLoadImages = function() {
07      //获取函数体的参数个数
08      var args_len = arguments.length;
09      //反向循环参数个数，创建img元素
10      for (var i = args_len; i--;) {
11        var cacheImage = document.createElement('img');
12        //指定img元素的src属性为数组元素的值
13        cacheImage.src = arguments[i];
14        //将HTML元素加入到数组中
15        cache.push(cacheImage);
16      }
```

```
17    }
18  })
19  </script>
```

第 03~18 行为 jQuery 的工厂函数$创建一个新的函数，这个函数接收一系列的图片路径作为参数。第 08~13 行通过反向循环获取传入的图片路径数组参数，然后调用 createElement()函数创建 img 元素，将其 src 属性指向数组元素值。第 15 行将 img 对象实例保存到 cacheImage 变量中。将创建的 cacheImage 变量添加到 cache 数组中，当调用 preLoadImages()函数时，就会自动在页面上下载图片，并且缓存到浏览器缓存中。

接下来通过调用 jQuery 库的 preLoadImages()函数，就可以轻松地完成图片的预加载工作，代码如下：

```
//预加载图片
$.preLoadImages('images/sample1.jpg','images/sample2.jpg','images/sample3.jpg');
```

当网页加载时，preLoadImages()函数被调用，3 张图片便被加载到了缓存中，这样在页面上的任何位置引用图片时，将从缓存中获取所需要的图片，从而提升了用户的体验，使网站获得了更好的流畅度。

7.4 Facebook 风格的图片预加载

Facebook 风格的预加载与前一节中的图片预加载在原理上相同，但是实现方法不同。Facebook 风格的预加载也是先创建一个 img 元素，该元素并未加载到 HTML 页面。它会调用 jQuery 的 load()函数，使用 AJAX 方式异步地从服务器端加载图片，当图片加载完成后，再将图片显示在页面上。这种加载方式在实现相册之类的应用时，可以让用户获得比较流畅的浏览体验。

Facebook 风格的图片预加载代码如下：

```
01  /* faceBookLoading.html */
02  <script type="text/javascript">
03  //定义所要显示的图片的 URL
04  var nextimage = "http://estimation.mycollect.net/201007/201007281543318064.jpg";
05  //在页面加载时，定时预加载
06  $(document).ready(function(){
07    //使用 setTimeout()函数，定时的加载下一张图片
08    window.setTimeout(function(){
09      //调用 load()函数，异步地加载服务器端的图片
10      var img = $("<img>").attr("src", nextimage).load(function(){
11       //图片加载完成，追加到 div 元素中
12        $('div').append(img);
13      });
14    }, 100);
15  });
16  </script>
```

第04行定义了一个变量nextimage，用来指向下一幅图片的网址。第06~15行在jQuery的页就绪事件中，调用setTimeout()函数，在100毫秒后执行图片的预加载。在setTimeout()内部，第10行使用jQuery创建了一个img元素，并指定其src属性为nextimage，然后调用jQuery的load()函数，异步加载图片，在加载完成之后执行内嵌函数，调用append()函数将图片加载到HTML页面上的div元素内部。

注意：使用setTimeout()的目的是为了使得加载不会阻塞网页的执行流，如果load()要执行较长的时间加载时，不会阻塞后续HTML页面内容的产生。

当页面上具有较多的图片时，使用AJAX方式预加载，就可以连续的、无阻塞的来实现图片的依次预加载，从而大大提升网站的用户体验。

7.5 检查图片src是否有效

很多图片上传系统或相册系统都需要在图片显示前，先行验证指定图片的src是否有效，对于无效的src，用安全的提示进行图片替换。在jQuery中实现这个功能非常简单，利用jQuery的load事件，可以轻松地实现图片src的检查工作。

本例在网页上放置了几个img元素，其src属性指向网站的图片URL，然后通过一些jQuery脚本，验证图片src的有效性，HTML页面的定义代码如下：

```
01  /* checkImageUrl.html */
02  <!--设置 CSS-->
03  <style type="text/css">
04    body{
05          font-size:12px;
06    }
07  </style>
08  </head>
09
10  <body>
11    <div>
12     <!--添加实例图片-->
13    <img src="images/sample1.jpg" />
14    <img src="images/sample2.jpg" />
15    <img src="images/sample3.jpg" />
16    <img src="images/sample4.jpg" />
17    </div>
18    <!--显示图片 src 检测消息-->
19    <div id="samplemsg"></div>
20   </body>
21  </html>
```

第04~06行应用了简单的CSS来改变页面的字体。在body区，第11~17行添加了几

个 img 元素，用来指向网站上的图片，其中 sample4.jpg 是一个错误的图片位置，它引用了不存在的图片。最后第 19 行添加了一个 div 元素，用来显示图片 src 检测的结果消息。

接下来定义一段 jQuery 代码，这段代码使用 jQuery 的遍历方法，遍历选中的所有元素，对其应用图片检测，将这段代码添加在 samplemsg 这个 div 元素的下面，代码如下：

```
01  /* checkImageUrl.html */
02    <script language="javascript" type="text/javascript">
03      //定义一个匿名函数，用来执行图片 src 检测
04      $(function (){
05          //循环选中的 img 元素
06          $("img").each(function (i,e){
07              //获取单一 img 元素的 src 属性值
08              var imgsrc = $(e).attr("src");
09              //编写 load 事件处理代码
10              $(e).load(function(){
11                  //显示图片成功加载的消息
12                  $("<p> 加载图片路径"+imgsrc+" </p>").appendTo("#samplemsg");
13              }).error(function() {
14                  //编写 error 事件处理代码，当图片 src 错误时，显示错误消息
15                  $("<p> 错误的图片 src "+ imgsrc  +"</p>").appendTo("#samplemsg");
16                  //同时更改错误的 src 链接为不存在的占位符图片链接
17                  $(e).attr("src","images/noexistspic.jpg");
18              });
19          });
20      });
21    </script>
```

第 04~20 行在 body 中定义了一个匿名函数，这样页面解析到该位置时便会立即执行。在这个匿名函数中，第 06 行使用工厂函数$选中了页面上的所有 img 元素，然后对其应用 each()函数遍历。第 08~17 行 each()函数会对每个 img 元素应用匿名函数，其中参数 i 表示循环的下标，e 表示当前循环的 e 元素本身。第 08 行首先得到当前 img 元素的 src 属性，第 10~17 行编写 load 事件处理函数，如果成功加载，表示图片链接有效，将检测成功消息追加到 samplemsg 元素中，然后追加调用 error 事件处理代码，在 img 元素的 src 属性错误或不存在时，除了显示错误消息外，还将 src 图片替换为一幅占位符图片。

经过上述jQuery脚本的设置，现在浏览网页时，就会以 AJAX 方式进行图片的 src 检测，并给出检测的结果消息，如图 7.5 所示。

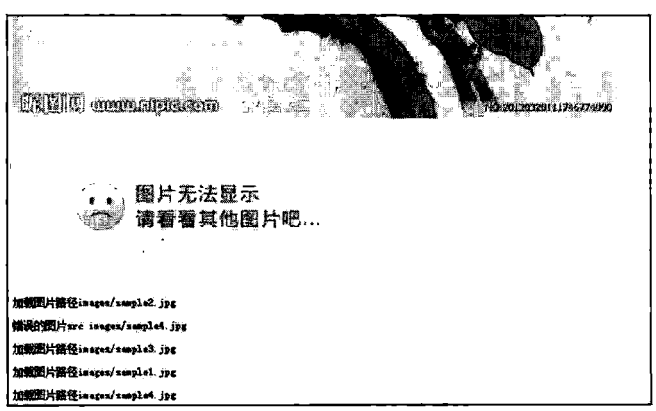

图 7.5 图片 src 检测结果

可以看到，sample4.jpg 由于指向了一个不存在的位置，因此 div 中提示“错误的图片”，并且原来的 src 被一个占位符图片所替代。

7.6 上下滑动的图片

上下滑动的图片效果在很多网站中经常可见，默认情况下，网页整齐地显示一系列无任何文字介绍的图片。当鼠标悬停到图片上时，图片缓慢上滑，显示图片的描述性介绍，当鼠标离开图片时，图片下滑恢复原来的显示样式。这种效果可以节省空间，提高网站的整洁和清爽度，其效果如图 7.6 所示。

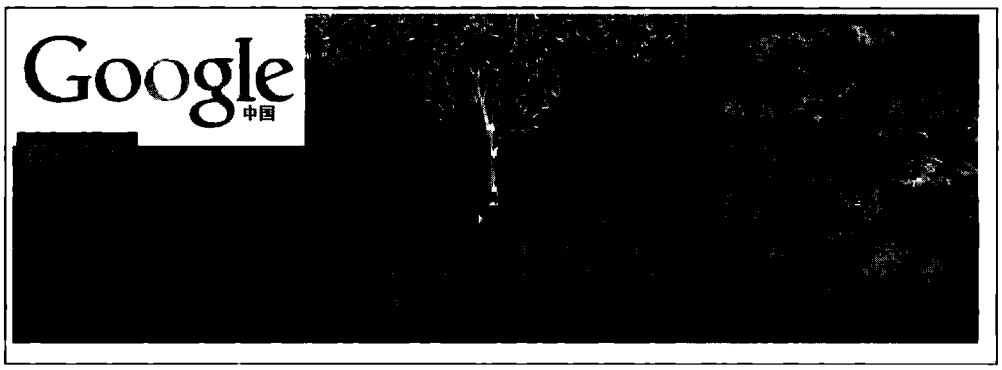

图 7.6 上下滑动的图片效果

可以看到，中间的图片，鼠标悬停其上时，图片上滑并显示文字注释内容。要实现这样的效果，首先新建一个 HTML 页面，添加对 jQuery 库的引用，然后在 HTML 的 body 区中添加如下代码，以便对图文布局进行排列：

```
/* upDownSlip.html */
<body>
<!--定义一个用来放置图片的容器-->
<div class="main">
  <!--图片和文字排列的容器-->
  <div class="holder notactive">
  <!--放置图片-->
  <img class="image" src="images/googlelogo.jpg" height=""/>
    <!--图片标题-->
    <span>谷哥公司</span>
    <!--图片文字-->
    <div class="text">Google 公司（中文官方译名谷歌，NASDAQ：GOOG、FWB：GGQ1），
    是一家美国的跨国科技企业，致力于互联网搜索、云计算、广告技术等领域，
    开发并提供大量基于互联网的产品与服务，其主要利润来自于 AdWords 等广告服务。
     </div>
  </div>
  <div class="holder notactive">
```

```
18    <img class="image" src="images/sample2.jpg"/>
19     <span>水墨画</span>
20      <div class="text">水墨画：由水和墨经过调配水和墨的浓度所画出的画，
21       是绘画的一种形式，更多时候，水墨画被视为中国传统绘画，
22       也就是国画的代表。也称国画，中国画。</div>
23    </div>
24    <div class="holder notactive">
25     <img class="image" src="images/sample5.jpg"/> <span>姹紫嫣红</span>
26      <div class="text">姹紫嫣红，形容花的品种繁多，五颜六色，景色艳美。
27      有名联：姹紫嫣红三春晖，赏心悦目百事兴。
28      出自 明· 汤显祖 《牡丹亭·惊梦》："原来姹紫嫣红开遍，似这般都付与断井颓垣。
29      " 清· 黄景仁《飞雪满群山·冰花》词："空花先现处，是姹紫嫣红后身。"</div>
30    </div>
31    <div class="clear"></div>
32  </div>
33  </body>
```

在body区中的第04~32行代码中可以看到，HTML中定义了一个class为main的div元素，其中包含了3个div，分别用来放置不同的图片和描述文本。无论是div还是img都应用了CSS类以便控制其呈现的效果。特别是对于显示描述信息的类text，它应用了display为none的样式属性，在初始时不显示。所有的样式代码如下：

```
01  /* upDownSlip.html */
02  <style>
03  /*单一图片与文字的容器样式*/
04  .holder {
05        overflow:hidden;      /*隐藏溢出*/
06        width:320px;          /*宽度和高度*/
07        height:333px;
08        float:left;           /*左浮动*/
09        position:relative;
10        background-color:#000;
11        margin-right:1px;
12  }
13  /*容器内图片的样式*/
14  .holder .image {
15        position:absolute;
16        left:-100px;          /*这里左边的位置为缩进100*/
17  }
18  /*容器内标题的样式*/
19  .holder span {
20        background-color:#333;
21        font-size:27px;
22        font-family:Arial, Helvetica, sans-serif;
23        color:#dedede;
24        font-weight:700;
```

```
25          padding:4px;
26          position:absolute;
27          top:120px;
28          left:4px;
29      }
30      /*容器内描述性文本的样式*/
31      .holder .text {
32          padding:20px;
33          display:none;          /*这里将文本设置为 none，表示默认不显示*/
34          font-family:Arial, Helvetica, sans-serif;
35          line-height:26px;
36          position:absolute;     /*使用绝对定位*/
37          top:180px;
38          font-size:16px;
39          color:#fff;
40          width:340px;
41      }
42      /*浮动清除*/
43      .clear {
44          clear:both;
45      }
46      /*主容器的样式*/
47      .main {
48          width:1000px;              /*主容器宽度*/
49          height:333px;              /*高度*/
50          margin:0 auto;             /*居中显示*/
51          overflow:hidden;
52      }
53      .credit {
54          font-size:12px;
55          margin-top:25px;
56      }
57      </style>
```

holder 样式类是单一幅图片的容器类，holder 容器内部包含了图片、标题和描述，其 overflow 应该为 hidden，这样当图片或文字内容溢出容器的大小时，使其隐藏而不会显得混乱。

注意：holder 样式的子样式 image，其 left 属性为-100，表示图片初始时将会隐藏显示部分内容。

第 19~29 行的 holder 样式的子样式类 span 用于放置图片标题，它以绝对定位的方式定位于容器的中间位置。第 31~41 行的 holder 样式的子样式 text 用于放置图片的描述性文本，它以绝对定位方式显示，但是其 display 为 none，表示初始时是隐藏的，当鼠标悬停时，会通过 jQuery 代码控制 CSS 来进行显示。

接下来编写 jQuery 代码，为 holder 样式类所在的容器添加鼠标悬停事件处理代码，jQuery 的 hover()函数可以轻松地实现这个效果。在 hover 事件处理代码中，需要编写两个函数：一个是当鼠标移入容器时的事件处理代码；另一个是当鼠标移出容器时的事件处理代码。最终代码如下：

```
01  /* upDownSlip.html */
02  <script type="text/javascript">
03      //在页面加载事件中，为容器关联 hover 事件
04      $(document).ready(function(){
05          $('.holder').hover(
06              function () {                          //鼠标悬停时的事件处理代码
07              $(this).removeClass('.notactive');     //移除 CSS 类 notactive
08              $('.notactive').stop().animate({'width':'290px'},400);  //设置宽度为 290
09              $(this).find('img').stop().animate({'top':'-165px'},400); //设置 img 的 top 到-165
10              $(this).stop().animate({'width':'380px'},400);          //设置宽度为 380
11              $(this).find('span').css({'background-color':'#000'});  //设置标题背景
12              $(this).find('.text').fadeIn(300);                      //淡入显示文本内容
13          },
14              function () {                          //鼠标移出时的事件处理代码
15                  $('.notactive').stop().animate({'width':'320px'},400); //动画设置宽度
16                  $(this).addClass('.notactive');    //添加 noactive 这个 CSS 类
17                  $(this).find('.text').hide();      //使文本隐藏
18                  $(this).find('img').stop().animate({'top':'0px'},500); //调整 img 的 top 为 0px
19                  $(this).stop().animate({'width':'320px'},400);      //调整宽度为 320px
20                  //设置标题的背景颜色
21                  $(this).find('span').css({'background-color':'#333', 'color':'#dedede'});
22
23          });
24      });
25  </script>
```

在页加载事件中，第 05~13 行代码为容器 div（也就是应用了 CSS 样式类 holder 的 div 元素）应用了 hover()函数，为其关联了鼠标移入与移出的事件处理代码，在 hover()函数内部需要定义鼠标进入和移出的两个函数。第 07~12 行代码第一个函数体内，首先移除了 CSS 类 notactive，表示当前鼠标所在位置的容器 div 处于 active 的状态。接下来，调用了其他具有 CSS 类为 notactive 的 div 元素，也就是其他 div 的宽度为 290px，使左右两边的 div 动画变窄，以为鼠标所在的 div 让出空间。使用 find()函数查找 img 元素，设置图片的 top 属性为-165，由于在 CSS 类中设置了 overflow 为 hidden，因此图片顶端将被隐藏。使 div 元素动画放大宽度为 380，让容器变宽，然后设置标题的背景色，最后对描述文字的 div 调用 fadeIn()函数，使其淡入到 div 中，显示图像的描述。

第 14~23 行的鼠标移出部分使用了与移入部分相反的处理方式，上下滑动的图片实际上是通过 animate()以及 CSS 来控制图片的位置，并且通过 fadeIn()或 fadeOut()来淡入淡出文本内容的显示，以达到当鼠标经过或移出时的图片滑动效果。简单的几行代码，不仅让网站变得具有创意，而且使得网站不至于因为图文混排而带来零乱的效果。

7.7 淡入淡出一幅图片，进入另一幅图片

有时候希望在用户浏览某幅图片时，淡入该图片，而页面上的其他图片淡出，以营造一种突出重点的效果。本例效果如图 7.7 所示。

图 7.7 淡入淡出图片效果

默认情况下，所有的图片都以标准模式显示在页面上，当鼠标悬停在某张图片上时，该图片高亮显示，其他的图片渐渐地淡出，变成了半透明的状态，以达到突出当前浏览图片的效果。

接下来将演示如何实现这样的效果。在 HTML 页面的 body 区域添加如下代码，用来向 HTML 页面中添加一些用来显示的图片：

```
/* fadeInfadeOutImages.html */
<body>
<!--添加图片，以实现淡入淡出的效果-->
<div id="img">
<img src="images/sample1.jpg" width="300">
<img src="images/sample3.jpg" width="200">
<img src="images/sample5.jpg" width="300">
<img src="images/googlelogo.jpg" width="300">
</div>
</body>
```

接下来在 jQuery 的页面加载就绪事件中，为图片应用淡入淡出的动画特效，代码如下：

```
/* fadeInfadeOutImages.html */
<script language="javascript" type="text/javascript">
$(document).ready(function(){                          //定义页面加载事件
    $("#img img").mouseover(function(){                //当鼠标移过时
   //对于非当前的图片元素，动画使其透明度变为 0.1
   $("#img img").not(this).animate({opacity:0.1},400);
  }).mouseout(function(){     //鼠标移出时的效果
   $("#img img").animate({opacity:1},400);             //动画使其透明度变为 100%
  });
```

```
10    });
11    </script>
```

第04行代码为id等于#img的div元素中的所有img元素关联了mouseover和mouseout事件，这里没有使用更简单易用的hover()函数。第04~06行代码在mouseover()函数内部，使用了not()函数，取排除当前img元素的其他img元素。使用animate()函数，使其动画淡出，也就是通过更改其透明度为0.1使其变为大部分透明状态，而第07~08行的mouseout则会使所有的img元素的透明度变为100%，即不透明状态。

通过本例可以看出，通过jQuery的animate()函数以及选择器的灵活应用，可以让网站建设人员完成一些非常实用，但是实现起来又相当简单的特效。

7.8 获取图片原生尺寸的方法

对于单个图片，使用jQuery的width()和height()函数，可以轻松地获取图片的宽度和高度，但是这个大小并不一定就是图片的原生大小，因为HTML代码有可能会修改图片的大小，比如下面的实例：

```
01  /* getOriginalSize.html */
02  <script type="text/javascript">
03        $(document).ready(function(){
04        //使用 width()和 height()获取当前显示的图片的高度
05        $("#picinfo").append("宽度:"+($("img:first").width()));
06        $("#picinfo").append("高度:"+($("img:first").height()));
07        });
08  </script>
09  </head>
10  <body>
11    <!--该 HTML 中的图片已经被调整了大小-->
12    <img src="images/sample2.jpg" width="50" height="50"/>
13    <div id="picinfo"></div>
14  </body>
15  </html>
```

第05~06行调用了图片的width()函数和height()函数，获取当前显示的图片的高度和宽度，由于在HTML中设置了width和height属性，因此显示的结果宽度和高度也是50，输出结果如下：

```
宽度:50 高度:50
```

要得到图片的原生大小，可以在内存中加载这幅图片，然后调用width()和height()函数，得到图片的真正尺寸，代码如下：

```
01  /* getOriginalSize.html */
02  <script type="text/javascript">
03        $(document).ready(function(){
```

```
04        //使用 width()和 height()获取当前显示的图片的高度
05        $("#picinfo").append("宽度:"+($("img:first").width()));
06        $("#picinfo").append("高度:"+($("img:first").height()));
07            $("#picinfo").append("<br/>");
08            var imgWidth, imgHeight;                    //保存图片原生尺寸的变量
09            //这里创建一个图像保存到内存，并没有添加到 HTML 中，只是个参考
10            $("<img/>").attr("src", "images/sample2.jpg").load(function() {
11            imgWidth = this.width;
12            imgHeight = this.height;
13            $("#picinfo").append("图片原生宽度:"+imgWidth); //显示图片的原生宽度和高度
14            $("#picinfo").append("图片原生高度:"+imgHeight);
15            });
16
17      });
18  </script>
```

代码第 10 行使用 jQuery 动态创建元素的语法创建了一个 img 元素，指定其 src 属性为要获取原生大小的图片，然后调用 load()函数，加载图片到内存中。第 11~12 行使用 width 和 height 来获取原生图片的宽度和高度，最后输出宽度和高度，输出如下：

图片原生宽度:550 图片原生高度:360

可以看到，显示出来的图片果然是图片原本的大小，不受 HTML 代码的设置影响。

7.9 检查图像是否已经被完全加载

很多网页要求在所有的图片加载完成后，才能开始更进一步的内容显示。比如当图片加载完成后，开始播放图片动画，或者是图片加载完成后，获取图片的相关信息。这时可以使用 jQuery 的事件 load。

注意：与 jQuery 中 Ajax 的 load()函数进行异步加载不同，事件 load 是当指定的元素（及子元素）已加载时，会触发 load 事件。

下面的代码为网页上的图片关联了 load 事件，当图片加载完成后，将显示图片的路径和图片的大小。

```
01  /* allImageLoaded.html */
02  <script type="text/javascript">
03      //在页就绪事件中，检测当图片加载完成时，图片的大小和路径
04      $(document).ready(function(){
05      var imgWidth, imgHeight;
06          $("img").each(function (i,e){              //循环选中的 img 元素
07              $(e).load(function(){                  //在图片加载完成后，显示图片信息
08                  var imgsrc = $(e).attr("src");     //获取单一 img 元素的 src 属性值
```

```
09                    imgWidth = this.width;
10                    imgHeight = this.height;
11                    //显示图片的路径和大小
12                    $("<p> 加载图片路径"+imgsrc+" </p>").appendTo("#samplemsg");
13                    $("<p> 图片的大小:宽度 "+imgWidth+" 高度:"+imgHeight+"
14                        </p>").appendTo("#samplemsg");
15                }); }); });
16  </script>
17  </head>
18  <body>
19    <div>
20      <!--添加实例图片-->
21    <img src="images/sample1.jpg" />
22    <img src="images/sample2.jpg" />
23    <img src="images/sample3.jpg" />
24    <img src="images/sample5.jpg" />
25    </div>
26    <!--显示图片 src 检测消息-->
27    <div id="samplemsg"></div>
28  </body>
29  </html>
```

在 HTML 页面上，代码第 19~25 行放了 4 幅图片，在第 04~15 行的 jQuery 就绪事件中，调用 each()函数遍历选中的所有 img 元素，为每个图片关联 load 事件，这个事件在图片加载完成后触发。这样在图片加载完成后，通过第 08~14 行获取图片的路径和图片的大小，并显示在 div 中。本例运行结果如下：

```
加载图片路径 images/sample2.jpg
图片的大小:宽度 550 高度:360
加载图片路径 images/sample3.jpg
图片的大小:宽度 668 高度:1024
加载图片路径 images/sample5.jpg
图片的大小:宽度 500 高度:334
加载图片路径 images/sample1.jpg
图片的大小:宽度 1024 高度:718
```

7.10 单击改变背景图案

网站经常需要提供给用户一些特殊的效果，比如添加一个改变背景的链接，当用户单击这个链接时，动态切换网站的背景图片。这类似于一些博客或微博的背景图切换功能，可以为网站的访问用户带来不一样的感观体验。本例效果如图 7.8 所示。

图 7.8　单击改变背景图片

本例代码如下：

```
01  /* clickBackground.html */
02  <script type="text/javascript">
03  var image=new Array(4);                    //定义保存 4 幅图片的数组
04  var n=0;                                   //计数器变量，用于切换图片计数
05  image[0]="images/sample1.jpg";             //定义 4 幅图片的图片地址
06  image[1]="images/sample2.jpg";
07  image[2]="images/sample3.jpg";
08  image[3]="images/sample5.jpg";
09  $(document).ready(function(){              //为页面关联页加载事件
10    $("#btnlink").click(function(){          //为链接关联按钮单击事件
11          $("#imagebk").fadeOut(1000,        //淡出当前图片，淡出结束后，淡出下一幅图片
12              function(){$("#imagebk").css("background-image",
13              "url("+image[n]+")");$("#imagebk").fadeIn();
14              if(n>3){ n=0; }else{   n=n+1; }  //调整计数器计数
15          });
16      });
17   });
18  </script>
19  </head>
20
21  <body>
22    <!--一个按钮，允许用户单击改变背景图-->
23    <a href="#" id="btnlink">单击这里改变背景</a>
24    <!--用来显示背景图的 div-->
25    <div id="imagebk"></div>
26  </body>
27  </html>
```

第 03 行定义了一个名为 image 的数组，第 05~08 行为该数组赋了 4 个数组元素，分别指向不同的图片。第 04 行定义的 n 变量是一个全局变量，用来记录当前播放图片的数组

下标。第09~17行的页面就绪事件中，第10行为id等于btnlink的链接添加了click事件处理代码，第11行在选中的div元素上调用fadeOut()函数，淡出当前显示的背景，第12~15行定义了一个回调函数，在fadeOut()淡出完成后，为div元素添加CSS背景，并且调用fadeIn()来淡入背景图片，最后修改数组下标变量n，以变更不同的图片。

7.11 如何显示Flickr网站的图片

Flickr是世界流行的电子相册网站，它提供了全面、一流、高效的图片服务，用户可以上传图片、设置分类、添加标签、进行图片搜索等等，是在世界范围内很受欢迎的图片分享网站。除了在Flickr上存储和分享照片外，Flickr还提供了一系列的API函数，允许网站开发人员在自己的网站内部整合Flickr，比如上传照片到Flickr等等，Flickr提供的服务API网址如下：

https://www.flickr.com/services/api/

要显示Flickr网站中的图片，除了使用API之外，还可以通过jQuery解析Flickr网站提供的RSS种子来显示Flickr图片的缩略图，如图7.9所示。

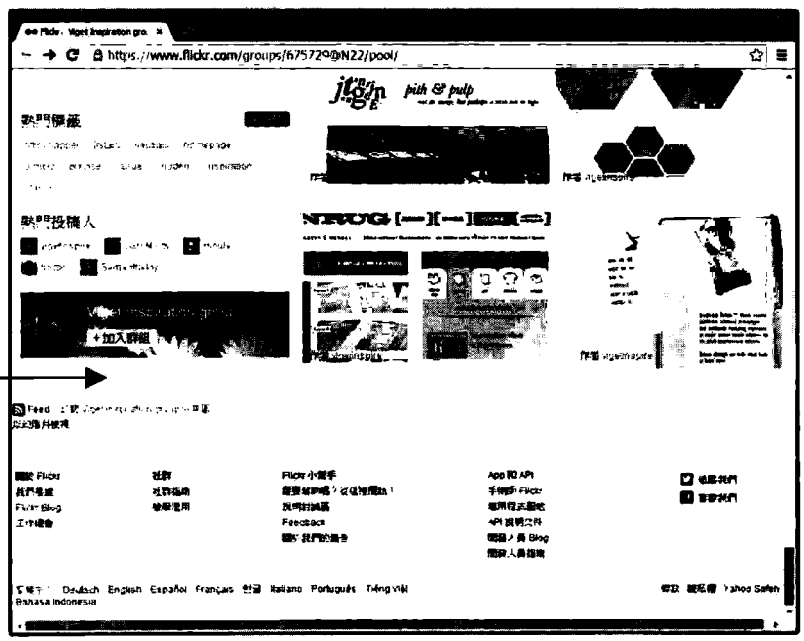

图7.9 Flickr网站的RSS种子

Flickr的RSS种子是JSON格式，通过使用jQuery提供的getJSON()函数，可以用AJAX方式异步地获取到RSS种子内容，然后解析其中的照片路径，就可以显示Flickr网站的图片了。

注意：JSON英文全称是JavaScript Object Notation，是一种轻量级的数据交换格式。它基于JavaScript的一个子集，常用来在网络上传递数据。

本例演示了如何通过 jQuery 的 getJSON()函数获取 Flickr 种子，然后显示最新的 Flickr 照片，代码如下：

```
/* showFlickr.html */
<script type="text/javascript">
  //在页加载就绪事件中，获取并显示图片
  $(document).ready(function(){
    //使用 jQuery 的 getJSON()函数，调用 Flickr 中最近的 20 张照片
    $.getJSON("http://api.flickr.com/services/feeds/groups_pool.gne?
        id=998875@N22&lang=en-us&format=json&jsoncallback=?", displayImages);
    //在获取到 JSON 内容后，解析并显示图片
    function displayImages(data) {
        //随机的选择开始位置，随机数在 0~11 之间，因为正显示 9 张图片
        var iStart = Math.floor(Math.random()*(11));
        var iCount = 0;                                    //计算器置为 0
        var htmlString = "<ul>";                           //将图片写到一个 ul 中
        $.each(data.items, function(i,item){               //循环遍历返回的 JSON 数据图片
        if (iCount > iStart && iCount < (iStart + 10)) {   //随机显示 9 张图片
        var sourceSquare = (item.media.m).replace("_m.jpg", "_s.jpg");     //仅显示小缩略图
        //将显示的图片拼合成 HTML 进行显示
        htmlString += '<li><a href="' + item.link + '" target="_blank">';
                htmlString += '<img src="' + sourceSquare + '"
                    alt="' + item.title + '" title="' + item.title + '"/>';
        htmlString += '</a></li>';
        }
                iCount++;                                  //增加计数器的值
        });
        $('#images').html(htmlString + "</ul>");           //将图片放到 div 中去
    }
});
</script>
</head>
<body>
<!--显示 Flickr 的缩略图片-->
<div id="images"></div>
</body>
</html>
```

当页面加载就绪后，第 06~07 行使用 getJSON()获取了特定用户的 Flickr 相册的种子，其中 id 可以更改为其他用户。在 getJSON()函数异步加载相片完成后，第 09~26 行调用 displayImages()解析并显示 Flickr 网站上的图片。displayImages()将随机显示 JSON 中的数据，然后追加到页面上的 div 元素中。本例效果如图 7.10 所示。

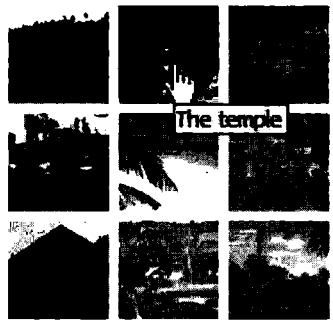

图 7.10 显示 Flickr 网站上的照片

7.12 如何显示 Google Picasaweb 的图片

Google Picasaweb 是另一款世界流行的网络相册，通过其提供的 API 也可以进行编程。本例演示如何通过访问 Google Picasaweb 网站上的种子文件，解析其中包含的相册和相片，以在自己的网站上显示来自 Google Picasaweb 的网站图片。

Google Picasaweb 网站的图片显示方式与 Flickr 类似，需要知道所要访问的 RSS 种子的地址和基本的参数，然后调用 jQuery 的 getJSON()来获取 JSON 内容，通过解析 JSON 内容，得到相册地址。但是同时也会发送一个 AJAX 请求，这个请求用来获取指定相册 ID 下面的图片列表，以便一次性获取多个相册下的多幅图片。本例代码如下：

```
01  /* showPicasaWeb.html */
02  <script type="text/javascript">
03      $(document).ready(function(){
04          //所要访问的种子的 URL 信息
05          var json_Album_URI = "https://picasaweb.google.com/data/feed/base/"
06              + "user/"          +   "fivemu"        //用户名
07              + "?alt="          +   "json"          //返回格式
08              + "&kind="         +    "album"        //相册
09              + "&hl="           +    "en_US"        //语言
10              + "&fields="       +   "entry(media:group,id)"
11              + "&thumbsize=" +    104
12              + "&max-results="+    2               //最大返回结果
13      //调用 getJSON()发送请求，获取 JSON
14      $.getJSON(json_Album_URI,
15      function(data){
16          $.each(data.feed.entry, function(i,item){
17              //得到相册的 URL
18              $.each(item.media$group.media$thumbnail, function(i,item){
19                  var album_thumb_URL = item.url;
20                  $('#images').append("相册缩略图:
21                          <img src=" + album_thumb_URL + '></img><br />');
```

```
22                  });
23                  //获取相册的标题
24                  var album_Title = item.media$group.media$title.$t;
25                  $('#images').append("相册标题: " + album_Title + '<br />');
26                  //获取相册的描述
27                  var album_Description = item.media$group.media$description.$t;
28                  $('#images').append("相册描述: " + album_Description + '<br />');
29                  //相册编号
30                  var album_ID = item.id.$t;
31                  //从相册的 URL 中获取到相册的数字编号
32                  album_ID = album_ID.split('/')[9].split('?')[0];
33                  $('#images').append("相册编号: " + album_ID + '<br /><br />');
34                  //根据指定的相册 ID 向服务器发送请求，获取相册下面的所有图片
35                  var json_Photo_URI = "https://picasaweb.google.com/data/feed/base/"
36                      + "user/"          +    "fivemu"
37                      + "/albumid/"      +    album_ID
38                      + "?alt="          +    "json"
39                      + "&kind="         +    "photo"
40                      + "&hl="           +    "en_US"
41                      + "&fields="       +    "entry(media:group)"
42                      + "&thumbsize="    +    104;
43                  $.ajax({                                    //发送异步 Ajax 请求
44                      type: 'GET',
45                      url: json_Photo_URI,                    //请求的地址
46                      success : function(data){               //请求成功时，执行的回调函数
47                          $.each(data.feed.entry, function(i,item){
48                              var photo_URL;
49                              $('#images').append("下面是相册图片: <br />");
50                              $.each(item.media$group.media$content, function(i,item){
51                                  photo_URL = item.url;                   //图片完整 URL
52                              });
53                              $.each(item.media$group.media$thumbnail, function(i,item){
54                                  var photo_Thumb_URL = item.url;  //缩略图 URL
55                                  $('#images').append
56          ("<a href="+photo_URL+"><img src=" + photo_Thumb_URL + '></img></a><br
57                  />'); });
58                  var photo_Title = item.media$group.media$title.$t;          //图片标题
59                  $('#images').append("图片标题: " + photo_Title + '<br />');
60                  var photo_Description = item.media$group.media$description.$t; //图片描述
61                  $('#images').append("图片描述: " + photo_Description + '<br /><br />');
62              });
63          },
64                      dataType: 'json',                       //数据类型
65                      async: false                            //是否异步调用，false 表示同步
66                  });
```

```
67              });
68          });
69  });
70  </script>
71  </head>
72  <body>
73      <!--显示图片的 div 容器-->
74      <div id="images"></div>
75  </body>
```

在页加载事件中，第05~12行定义的json_Album_URL用来获取user参数指定的用户下面的相册，kind指定为album，表示相册类型。在构造了获取Picasaweb相册的网址后，第14行调用$.getJSON()获取特定网址返回的JSON数据。第18~22行成功获取数据后，在回调函数体内，循环遍历JSON数组。每一行表示一个相册，每一个相册又具有相册缩略图、相册标题以及相册的描述信息。相册的ID包含了字母数字值，实例中通过提取相册编号的整型值，构造了一个获取指定相册下相片的URL，然后第43~68行调用jQuery的$.ajax()函数，以同步的方式，向该网址发送GET请求，获取返回的图片数据，然后循环遍历其中的图片数据，构造出显示的相册结果。

7.13 按比例调整图片的大小

上传相片到相册时，经常需要按比例调整图片的大小，比如让其最高和最宽不能超过一个固定的大小，以避免用户上传了过大的图片，而导致网页的外观变得难看。使用jQuery实现按比例缩放比较简单，效果如图7.11所示。

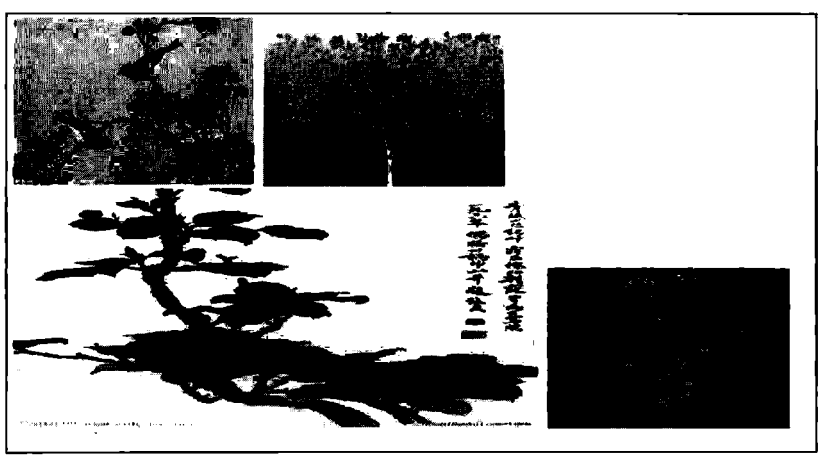

图7.11 按比例缩放图片效果

本例代码如下：

```
01  /* scaledPictureSize.html */
02  <script type="text/javascript">
```

```
03      //要绑定 load 事件，以便在所有内容加载完成，包含图片加载完成后应用新大小
04      //否则在页就绪事件中再应用样式，就不会有效果
05      $(window).bind("load", function() {
06          //循环页面上的所有的图片，应用样式设置 CSS
07          $('#imglist img').each(function() {
08              var maxWidth = 200;                    //设置固定的图片宽度
09              var maxHeight = 200;                   //设置固定的图片高度
10              var ratio = 0;                         //宽度或高度的调整比率
11              var width = $(this).width();           //得到当前图片的宽度
12              var height = $(this).height();         //得到当前图片的高度
13              //如果当前的宽度大于所要求的最大宽度，则需要进行宽度的调整
14              if(width > maxWidth){
15                  ratio = maxWidth / width;
16                  $(this).css("width", maxWidth);    //调整最大宽度
17                  $(this).css("height", height * ratio);  //按调整宽度的比率来调整高度
18                  height = height * ratio;           // height 变量为调整后的高度
19              }
20              //如果当前的高度大于所要求的最大高度，则需要进行高度的调整
21              if(height > maxHeight){
22                  ratio = maxHeight / height;
23                  $(this).css("height", maxHeight);  //调整 CSS 为最大高度
24                  $(this).css("width", width * ratio);  //按比例调整最大宽度
25                  width = width * ratio;  //设置当前的 width 变量为调整后的宽度
26              }
27          });
28      });
29  </script>
30  </head>
31
32  <body>
33  <!--HTML 上的图片列表-->
34  <div id="imglist">
35    <img src="images/sample1.jpg" />
36    <img src="images/sample2.jpg" />
37    <img src="images/sample3.jpg" />
38    <img src="images/sample5.jpg" />
39  </div>
40  </body>
41  </html>
```

在 HTML 页面上（即 body 部分），代码第 34~39 行放置了一个 id 等于 imglist 的 div 元素，在内部放了 4 幅图片。第 05~28 行代码使用 jQuery 绑定到 window.load 事件，这个事件在页面上所有的元素都加载完成后触发，这样可以确保图片加载完成后，再为其应用 CSS 来调整大小，以避免在图片未加载时调整大小出现异常。

注意：在事件处理内部，通过 jQuery 的 each()函数循环选择器集合，maxWidth 和 maxHeight 是要调整的最大宽度和高度，ratio 变量用来保存将会调整的比例值。

第 11~12 行代码获取图片本身的宽度和高度，保存在 width 和 height 变量中。如果宽度大过了最大宽度，则调整当前的宽度为最大的宽度，并且计算调整的比例。调整高度使用了相同的原理。

7.14 滑动效果的背景图片

在一些网站上，经常可以看到缓缓滑动的背景图片，比如漂浮的云彩、缓缓升起的太阳等等，为网站带来了不少令人赏心悦目的体验。在 jQuery 中实现这样的效果也非常简单，下面的代码演示了如何在 div 中创建滑动的背景图，以实现动感的背景效果。

```
01  /* slipBackground.html */
02  <title>滑动的背景图</title>
03  <style type="text/css">
04   #flower {
05            height:500px;
06            /*指定 div 的背景图片，水平方向滚动 0px，宽度为 100%*/
07            background:url('images/sample1.jpg') repeat-x scroll 0 0 transparent; width:100%; }
08  </style>
09  <!--加载 jQuery 库-->
10  <script type="text/javascript" src="jquery/jquery-1.9.1.js"></script>
11  <script type="text/javascript">
12    var scrollSpeed = 70;              //指定滑动速度，也就是定时器的间隔时间，单位为毫秒
13    var step = 1;                      //指定步进值，值越大滑动幅度越大
14    var current = 0;                   //保存当前位置的中转变量
15    var imageWidth = 2247;             //图片滚动的最大宽度
16    var headerWidth = 800;             //边框宽度
17    //图片宽度和边框宽度的差，就是重新开始滑动的宽度
18    var restartPosition = -(imageWidth - headerWidth);
19    function scrollBg(){               //滚动背景函数
20      current -= step;                 //调整步进值
21      if (current == restartPosition){ //恢复初始值
22        current = 0;
23      }
24      //设置背景的位置
25      $('#flower').css("background-position",current+"px 0");
26    }
27    //在页面加载后，使用 setInterval()反复调用 scrollBg 来滚动图片
28    var init = setInterval("scrollBg()", scrollSpeed);
29  </script>
30  </head>
```

```
31  <body>
32    <!--用来滑动背景的 div 元素-->
33   <div id="flower"></div>
34  </body>
35  </html>
```

第 33 行在 HTML 页面上放置了一个 id 等于 flower 的 div 元素，第 04~07 行通过 CSS 设置了其高度和背景图片，其中背景图片以横向滚动。JavaScript 代码部分的重点是第 28 行，使用 setInterval()来反复地调用一个调整背景图片位置的 scrollBg()函数（第 19~23 行），在这个函数体内部，判断变量 restartPosition 与全局变量的值 current 是否相等，如果相等表示需要重置背景的初始位置。在 scrollBg()内部也使用了 css()函数来设置 div 的 background-position 属性，以调整背景图的位置，从而实现了背景图的平滑移动。

7.15 动态表单生成图片预览

在网站开发过程中，经常需要让用户自己配置网站内容的显示，并给出一个预览的结果。比如在设计一个产品展示页面时，如果能让用户在表单中选中图片，输入图片的描述性信息，并能给出具体的预览效果，将会让网站增色不少。本例效果如图 7.12 所示。

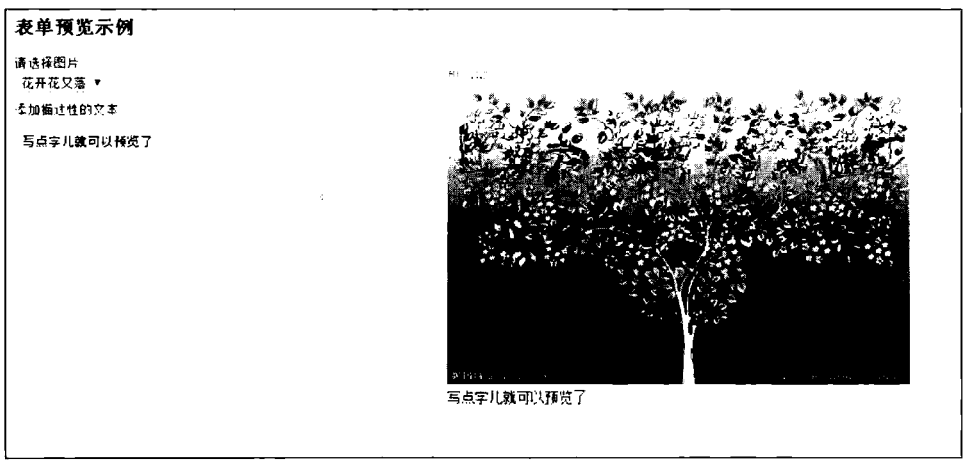

图 7.12 表单图片预览效果

本例代码如下：

```
01  /* dynamicFormPictures.html */
02  <script type="text/javascript">
03      $(document).ready(function(){          //页面就绪事件
04          $('#preview').hide();              //隐藏预览窗口的显示
05          $("#photo").click(update);         //为下拉列表框关联 click 事件
06          $("#title").keypress(update);      //为按钮放开关联 keypress 事件
07      });
08      function update(){                     //通用的回调函数
```

```
09              $('#preview').slideDown('slow');        //重新滑动显示预览区域
10              var title = $("#title").val();          //获取标题内容
11              var photo = $("#photo").val();          //获取照片路径
12              $('#Displaytitle').html(title);         //更新要显示的标题
13              //更新要显示的图片
14              $('#image').html('<img src="'+photo+'" width=460 />');
15          }
16  </script>
17  </head>
18
19  <body>
20  <div id="container">
21    <h3>表单预览实例</h3>
22    <!--表单定义区域-->
23    <div class="left">
24      <form>
25        请选择图片<br/>
26        <select name="pic" class="click" id="photo">
27          <option value="images/sample1.jpg">鸟鸣山更幽</option>
28          <option value="images/sample2.jpg">花开花又落</option>
29          <option value="images/sample3.jpg">春意又盎然</option>
30        </select>
31        <br/>
32        添加描述性的文本
33        <textarea id="title" class="click" name="title" cols="40" rows="4">
34         请在这里输入描述性文本
35         </textarea>
36      </form>
37    </div>
38    <!--图片预览区域-->
39    <div class="right">
40      <div id="preview"> 预览图片
41        <div id="image"></div>
42        <div id="Displaytitle"></div>
43      </div>
44    </div>
45    <div class="clear"></div>
46  </div>
```

在HTML部分，代码第24~36行定义了一个表单，表单上有一个select控件用来保存图片的路径，第33行的textarea允许用户输入图片描述信息。第39~44行定义表单的右侧，就是图片和图片描述的预览区域。在jQuery的页就绪事件中，第04行首先调用hide()函数来隐藏div的显示。第05~06行为下拉列表框关联click事件，并且为textarea关联keypress事件，以便在选择了不同的图片或输入了不同的内容后，可以更新预览区域的显示，这两个事件都调用了一个公共的update()函数（第08~15行）。该函数调用slideDown显示预览

区域的内容，并获取图片描述和图片路径，分别为预览区域中的元素进行赋值，以便更新预览区域的显示，从而实现图片预览的效果。

7.16 平滑滚动的导航菜单

大部分网站在页面顶部都有一个导航菜单，为了节省空间，默认的菜单将半遮半掩地呈现在页面的顶部。当鼠标悬停在菜单条上时，将会显示菜单内容，这样不仅节省了网站的空间，也让整个网站更加整洁。使用 jQuery 实现这样的效果非常简单，只需要在定义好菜单后，为其关联 hover 事件，并使用 animate()函数动态显示/隐藏即可。本例效果如图 7.13 所示。

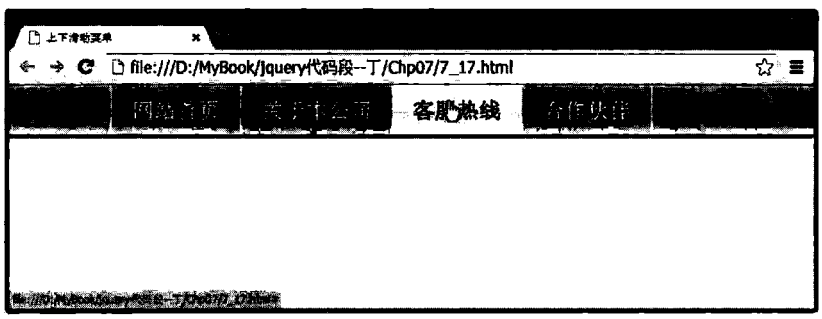

图 7.13 平滑滚动的导航菜单

实例代码如下：

```
/* slipNavigator.html */
<script type="text/javascript">
    $(document).ready( function () {
        if($.browser.msie){                    //如是为 IE，则设置 IE 的背景色以及定位方式
            $('.menu_wrap').css({'background-color' : '#b8e1fc','position' : 'absolute','width' :
              '100%'});
        }
        $('.menu_wrap').css({'opacity' : '0.3'});      //指定菜单的透明度
        $('.menu_wrap').hover ( function () {          //当鼠标经过时为菜单关联 hover 事件
            //使用 animate 动画调整菜单当前的顶部距离，使之显示在页面上
            $(this).animate({'margin-top' : '0px','opacity' : '1'});
        }, function () {
            //鼠标移出时，使用 animate 动画隐藏菜单的一部分显示
            $(this).animate({'margin-top' : '-30px','opacity' : '0.3'});

        });
    });
</script>
</head>
<body>
```

```
<!--定义一个显示的菜单-->
<div class="menu_wrap">
    <div class="menu_links">
      <ul>
        <!--菜单项-->
        <li><a class="first" href="#">网站首页</a></li>
        <li><a href="#">关于本公司</a></li>
        <li><a href="#">客服热线</a></li>
        <li><a href="#">合作伙伴</a></li>
      </ul>
    </div>
</div>
</body>
</html>
```

在HTML页面上，第23~31行构造了一个使用CSS控制的菜单，这个菜单放了几个链接，并没有特别的功能，本例重点在JavaScript代码部分。当页面加载就绪时，第04~07行首先判断是否为IE，如果是IE则应用背景色，否则应用的是CSS 3效果的菜单样式。第08行设置菜单主体的opacity属性为0.3，指定其为半透明状态。第09~16行为菜单主体关联hover事件，该事件会调用animate动画来调整菜单的顶部间距以及透明度，使得菜单具有滑动显示以及滑出的效果。

7.17 图片的放大预览

在实现相册效果时，通常会给出一张缩略图，然后让用户单击缩略图，弹出一幅原始的大图来进行预览。本例演示一种新的预览照片的方法，它会在用户单击某个网页时，将整个网页进行局部放大预览，这就类似一个放大镜的效果。

为了实现这个效果，需要一个辅助插件 zoomooz.js，可以从如下网址了解这个插件的详细信息：

http://jaukia.github.io/zoomooz/

接下来创建一个网页，在网页上新建一个相册，然后使用如下代码为相册添加单击图片放大、单击相册其他区域恢复的效果。代码如下：

```
/* zoomPictures.html */
<!--使用 zoomooz 所需要库文件，以便实现单击图片放大效果-->
<script type="text/javascript" src="jquery/zoomz/sylvester.js"></script>
<script type="text/javascript" src="jquery/zoomz/purecssmatrix.js"></script>
<script type="text/javascript" src="jquery/zoomz/jquery.animtrans.js"></script>
<script type="text/javascript" src="jquery/zoomz/jquery.zoomooz.js"></script>
<script type="text/javascript">
    $(document).ready(function() {
        $(".zoom").click(function(evt) {          //为相册关联单击事件
            evt.stopPropagation();                //停止事件传递
```

```
11                  $(this).zoomTo();                    //调用 zoomTo()函数，放大图片
12              });
13              $(window).click(function(evt) {          //为 window 关联单击事件
14                  evt.stopPropagation();               //停止事件的传递
15                  $("body").zoomTo();                  //单击 body 时，恢复放大的图片
16              });
17              $("body").zoomTo();                      //初始放大 body 区域
18          });
19      </script>
20
21      </head>
22
23      <body>
24      <!--实现简单的相册效果-->
25      <div id="gallery" class="zoom">
26      <ul>
27          <!--相册所需要的图片集-->
28          <li class="zoom"><img src="images/sample1.jpg"/></li>
29          <li class="zoom"><img src="images/sample2.jpg"/></li>
30          <li class="zoom"><img src="images/sample3.jpg"/></li>
31          <li class="zoom"><img src="images/sample5.jpg"/></li>
32      </ul>
33      </div>
34      </body>
35      </html>
```

在 HTML 页面上，代码第 26~32 行放了几幅图片组成了一个简单的相册，代码第 03~06 行在页面的 head 区添加了 jQuery 以及 zoomooz 所需要的脚本文件。在页面就绪事件中，第 09~12 行为图库 div 关联 click 事件，当用户单击图库时，进行放大或缩小，当用户在窗口上单击时，调用 body 标签的 zoomTo 进行放大或缩小。本例效果如图 7.14 所示。

图 7.14　网页放大效果预览

可以看到，当单击某张照片时，除了该张照片放大外，与之相邻的照片也放大了，单

击相册背景，即灰色区域，则会显示所有的相片缩略图，再次单击网页空白区域，则会恢复到初始大小显示，这种方式模拟了一种真实的照片放大效果。

7.18 实现平滑的图片动态缩放效果

对于电子相册来说，人们还是习惯于通过单击图片来放大预览的方式。本节将介绍一种平滑地放大或缩小图片的方法。这种方法跟网上大多数电子相册一样，都是通过单击放大图片，只是这种方式提供了多种放大的特效。本例效果如图 7.15 所示。

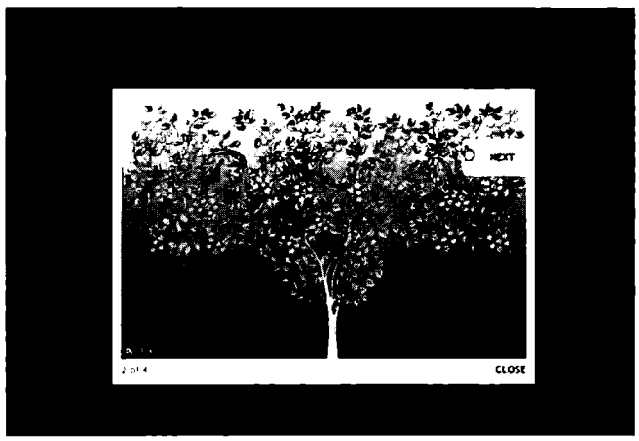

图 7.15 平滑的动态缩放效果

本例使用 imgZoom 插件，它可以实现一种平滑的缩放效果，可以从如下网址获取该插件的详细信息：

http://odyniec.net/projects/imgzoom/

接下来创建一个网页，该网页将演示如何使用 imgZoom 插件来实现图片平滑的缩放效果，代码如下：

```
/* imageZoom.html */
<link rel="stylesheet" href="jquery/imgzoom/css/imgzoom.css" type="text/css"
media="screen" />
<!--添加对 jQuery 库和 imgzoom 库的引用-->
<script src="jquery/imgzoom/scripts/jquery.min.js" type="text/javascript"></script>
<script src="jquery/imgzoom/scripts/jquery.imgzoom.js" type="text/javascript"></script>
<script type="text/javascript">
 $(function () {
     $('#img1').imgZoom({ showOverlay: true });                    //设置第 1 幅图遮罩显示
     $('#img2').imgZoom({ showOverlay: true, opacity: 0 });        //透明第 2 幅图显示
   //旋转显示
     $('#img3').imgZoom({ showOverlay: true, rotate: true, duration: 1000 });
     $('#img4').imgZoom({ showOverlay: true, rotate: false, duration: 300 }); //不旋转显示
 });
```

```
15  </script>
16  </head>
17  <body>
18  <!--实现简单的相册效果-->
19  <div id="gallery">
20  <ul>
21      <!--相册所需要的图片集-->
22      <li><a href="images/sample1.jpg"><img id="img1" src="images/sample1.jpg"/></a></li>
23      <li><a href="images/sample2.jpg"><img id="img2" src="images/sample2.jpg"/></a></li>
24      <li><a href="images/sample3.jpg"><img id="img3" src="images/sample3.jpg"/></a></li>
25      <li><a href="images/sample5.jpg"><img id="img4" src="images/sample5.jpg"/></a></li>
26  </ul>
27  </div>
28  </body>
29  </html>
```

代码第 19~27 行在网页上放了一个 div 元素，内部放了 4 幅图片，只是为每幅图片添加了一个指向自身的链接。第 02~06 行在 head 区中添加了 imgZoom 所需要的 CSS 和 js 文件，并同时添加了对 jQuery 库的引用。第 09~13 行为每一幅图片都应用了 imgZoom()函数，通过其参数来配置图片的放大与缩小方式，比如旋转还是透明等等。

7.19 自动适应的窗口背景

自适应背景图是网页设计中一个非常有用的功能，实现起来也不复杂，主要是关联 window.resize 事件，然后相应地调整背景图的大小。为了更加灵活地使用自适应背景图，网上有人将这个功能封装为 jQuery 插件，插件名为 jquery.fullbg.js，使用这个插件，只用几行代码就可以实现自适应的背景图效果。

本例效果如图 7.16 所示，无论如何调整窗口大小，背景图片总会自动调整来适应窗口大小。

图 7.16 自适应窗口背景效果

下面的代码演示了如何使用这个插件，来创建一个具有自适应背景的网页。

```
01  /* autofitWindow.html */
```

```
02  <title>自适应图片背景</title>
03  <!--添加对 jQuery 库的引用-->
04  <script type="text/javascript" src="jquery/fullbg/jquery.min.js"></script>
05  <!--添加对 jquery.fullbg.min.js 插件的引用-->
06  <script type="text/javascript" src="jquery/fullbg/jquery.fullbg.min.js"></script>
07  <style type="text/css">
08  .fullBg {
09      position: fixed;            /*设置固定背景*/
10      top: 0;                     /*指定背景的起始位置*/
11      left: 0;
12      overflow: hidden;           /*背景的溢出方式为超出背景部分进行隐藏*/
13  }
14  </style>
15  </head>
16  <body>
17  <!--放置一个用作背景图片的 img 元素-->
18  <img src="images/sample1.jpg" alt="" id="background" />
19  <script type="text/javascript">
20  $(window).load(function() {
21      $("#background").fullBg();   //在窗体加载完成后，设置背景图片
22  });
23  </script>
24  </body>
25  </html>
```

有了这个插件，实现自适应的背景就简单很多了，只需要第 06 行添加对 jquery.fullbg.min.js 的引用。第 18 行在 HTML 页面上添加一幅图片，这幅图片就能用作网页的背景图片。在第 20~22 行的 window.load 事件处理代码中，在选定的图片上调用 fullBg() 函数，这个函数会根据当前窗口的大小来调整图片的大小，然后关联 window.resize 事件，以便在窗口大小发生变化时重新调整。

注意：实例中的 CSS 样式类是必须的，在 jquery.fullbg.js 内部初始化时会将这个 CSS 类分配给要用作背景图的图片。

7.20 如何判断加载多张图片的完成状态

有时 Web 程序需要等页面上所有的图片都加载完成后，再执行下一步的动作。如果为多个 img 元素关联 load 事件，会发现每张图片加载完成时，都会执行一次 load 事件处理器中的代码，无法判断所有的图片是否都加载完成，代码如下：

```
01  /* multiImageLoaded.html */
02  <script type="text/javascript">
03    $(document).ready(function(){
04      $('img').load(function(){
```

```
05              alert('img loaded')    //该代码在每一张图片加载完成时都会执行一次
06          })
07      });
08  </script>
```

为了判断多张图片加载完成，可以使用一个变通的方法。使用一个全局变量来得到要加载的图片数量，然后在 load 事件中每执行一次，就将计数器加 1（或者进行一个递减的动作，直到计数器值为 0），当执行次数与要加载的图片数量相等时，表示所有的图片加载完成，代码如下：

```
01  /* multiImageLoaded.html */
02  <script type="text/javascript">
03      $(document).ready(function(){
04          var imgNum=$('img').length;              //要加载的图片张数
05          $('img').load(function(){
06              if(!--imgNum)
07                  {
08                      alert('所有的图片已经加载完成')
09                      }                            //当计数器递减到 0 时，表示图片加载完成
10          })
11      });
12  </script>
```

第 04 行在页就绪事件中，定义了一个全局变量 imgNum，该变量的初始值是所有的 img 元素的个数。第 05~10 行在 img 元素的 load 事件处理代码中，递减该计数器，每加载完成一幅图片，就递减一个计数器，当所有的图片都加载完成时，计数器为 0，表示已经全部加载完成。

7.21 鼠标悬停时的图片放大

鼠标悬停提示是一项非常有用的功能，在一些购物网站或是产品展示网站中，都可以看到鼠标悬停在图片上时，图片局部放大的效果。通过这样的方式可以为用户带来更加清晰的图片展示，比如我们在淘宝上就可以清晰地看到某件衣服的局部细节。本例效果如图 7.17 所示。

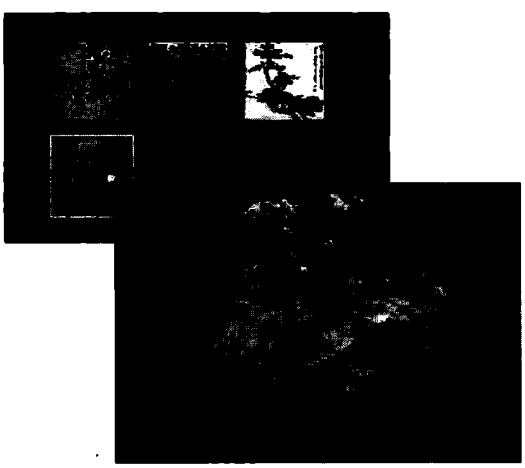

图 7.17 鼠标悬停时图片放大显示

本例代码如下：

```
/* mouseHovertip.html */
<script type="text/javascript" src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript">
    $(function(){
        var x = 10;
        var y = 20;
        $("a.tooltip").mouseover(function(e){
            this.myTitle = this.title;                    //得到当前链接的 titl，即标题
            this.title = "";
            var imgTitle = this.myTitle? "<br/>" + this.myTitle : "";
            var tooltip = "<div id='tooltip'> "+
                          "<img src='"+ this.href +"' alt='放大提示'/>"+
                            imgTitle+"</div>"; //新建一个 id 为 tooltip 的 div 元素
            //元素内部包含一个 img，用来显示图片并且显示标题，追加到 body 区
            $("body").append(tooltip);
            $("#tooltip")                                 //更改 tooptip 的顶部和左侧位置
                .css({
                    "top": (e.pageY+y) + "px",
                    "left":  (e.pageX+x)  + "px"
                }).show("fast");                          //使用 show()函数快速显示
        }).mouseout(function(){
            this.title = this.myTitle;
            $("#tooltip").remove();          //当鼠标移出时，从 body 区删除该 div 元素
        }).mousemove(function(e){    //当鼠标移动时，调整 div 的位置位于鼠标箭头的下方
            $("#tooltip")
                .css({
                    "top": (e.pageY+y) + "px",
                    "left":  (e.pageX+x)  + "px"
                });
        });
    })
</script>
</head>

<body>
<!--实现简单的相册效果-->
<div id="gallery">
<ul>
    <!--相册所需要的图片集-->
    <li><a href="images/sample1.jpg" class="tooltip" title="喜鹊看花">
<img src="images/sample1.jpg"/></a></li>
    <li><a href="images/sample2.jpg" class="tooltip" title="繁花朵朵">
<img src="images/sample2.jpg"/></a></li>
```

```
44        <li><a href="images/sample3.jpg" class="tooltip" title="姹紫嫣红">
45  <img src="images/sample3.jpg"/></a></li>
46        <li><a href="images/sample5.jpg" class="tooltip" title="含苞怒放">
47  <img src="images/sample5.jpg"/></a></li>
48  </ul>
49  </div>
50  </body>
51  </html>
```

在 HTML 页面上，可以看到第 37~49 行放置了一个名为 gallery 的 div 元素，其包含一个 ul 和若干个 li 元素，每个 li 元素内部包含了一个样式类为 tooltip 的链接，指向特定的要放大显示的图片，title 属性指定将要放大显示的标题。

注意：只有将动态创建的元素（比如 tooltip 这个 div）添加到 body 区域，才会在 HTML 页面中进行显示，否则只是在内存中创建了相关的对象，并不会显示到 HTML 页面上。

第 04~31 行创建了一个匿名函数，这个函数会为每一个样式类为 tooltip 的链接元素关联 mouseover 鼠标移过、mouseout 鼠标移出以及 mousemove 鼠标在链接上移动的事件。第 07~20 行在 mouseover 事件中动态地创建了一个 div 元素，该元素内部包含了链接的图片以及链接的标题（即 title 属性值）。第 21~30 行在 mousemove 事件中不停地调整图片的 top 和 left 这两个 CSS 属性，以便设置其位置在鼠标的右下角。mouseout 事件将移除动态产生的 div 元素，从而实现了鼠标悬停时图片放大的效果。

7.22 淡出图片，淡入另一幅图片

淡出一幅图片，淡入另一幅图片在很多产品展示网站上经常出现。默认情况下，网页显示一幅标准图片，当鼠标悬停在图片上时，淡出一幅图片，然后淡入另一幅图片，以实现交叉图片显示的效果。使用 jQuery 的 fadeIn()和 fadeOut()可以轻松地实现这样的效果，代码如下：

```
01  /* fadeInfadeOutAnother.html */
02  <title>淡出图片，淡入另一张图片</title>
03  <!--加载 jQuery 加-->
04  <script type="text/javascript" src="jquery/jquery-1.9.1.js"></script>
05  <script type="text/javascript">
06   $(document).ready(function(){                    //在页就绪事件中，关联事件处理代码
07            $("#gallery").hover(function(){         //为 div 关联 hover 事件
08                $(this).find("#pic").fadeOut();     //淡出当前的图片
09            }, function() {                         //鼠标移出时
10              //判断是否需要切换显示的图片
11              if($("#pic").attr("src")=="images/sample5.jpg"){
12                  $("#pic").attr("src","images/sample2.jpg");
13              }else
```

```
14                {
15                    $("#pic").attr("src","images/sample5.jpg");
16                };
17                $(this).find("#pic").fadeIn();      //淡入设置好的图片
18            });
19  });
20  </script>
21  </head>
22
23  <body>
24  <!--进行淡入淡出的 div 元素-->
25  <div id="gallery">
26      <!--淡入淡出的图片-->
27      <img id="pic" src="images/sample5.jpg" alt="花朵" />
28  </div>
29  </body>
30  </html>
```

第27行放置了一个img元素，默认显示sample5.jpg图片。在jQuery的页就绪事件中，第07行为id为gallery的div元素关联了hover事件。

注意：jQuery的hover可以看作是onmouseover和onmouseout的联合体，它接收两个回调函数，用来处理鼠标移过和鼠标移出的事件。

第07~18行的hover事件处理器中，onmouseover事件处理代码中调用了fadeOut()函数，将当前的图片淡出显示。在onmouseout事件处理代码中，判断当前img元素显示的是哪幅图片，然后切换要显示的图片，最后调用fadeIn()淡入另一幅图片进行显示，从而达到了淡出一幅图片显示另一幅图片的效果。

7.23 页面加载时随机显示图片

在页面加载时，图片的显示是随机性的，这在电子相册中能带来比较动感的体验。随机显示主要是通过JavaScript中随机函数的灵活运用来实现。下面创建一个随机显示图片的实例，代码如下：

```
01  /* randomShowImage.html */
02  <script type="text/javascript" src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04  $(document).ready(function(){
05    $("img").hide();                          //隐藏所有图片的显示
06    var speed = 200;                          //显示的速度
07     $(window).load(function() {              //页加载事件
08        $("#content img").hide();             //隐藏图片的显示
09        timer= setInterval(function(){        //定义 setInterval 事件
```

```
10          $notLoaded = $("#content img").not(".loaded");      //获取不存在样式类 loaded 的图片
11          $notLoaded.eq(Math.floor(Math.random()*$notLoaded.length))      //随机地显示图片
12        .fadeIn()
13        .addClass("loaded");                    //为已经显示的图片添加样式类 loaded
14    if ($notLoaded.length == 0) {               //如果显示完成
15        clearInterval(timer);                   //清除定时器事件
16    }
17  }
18  , speed);                                     //指定触发的时长，以毫秒为单位
19  });
20  });
21  </script>
22
23  <style>
24   /*指定图片 span 容器的样式*/
25   #content span {
26       width: 200px;              /*宽度*/
27        height:200px;             /*高度*/
28        display:block;            /*块状显示*/
29        float: left;              /*向左浮动*/
30        overflow:hidden;          /*溢出隐藏*/
31   }
32  </style>
33  </head>
34  <body>
35  <!--图片显示的容器-->
36  <div id="content">
37   <!--定义各种不同的要显示的图片-->
38   <span><img src="images/sample1.jpg" alt="喜鹊报喜" /></span>
39   <span><img src="images/sample2.jpg" alt="姹紫嫣红" /></span>
40   <span><img src="images/sample3.jpg" alt="水墨风情" /></span>
41   <span><img src="images/sample5.jpg" alt="含苞盛放" /></span>
42  </div>
43  </body>
44  </html>
```

第36~42行代码位置放了4张图片，通过CSS控制了每一个span的显示样式。在jQuery的页就绪事件中，第08行首先让所有的图片隐藏，并不进行显示。在所有内容加载完成的load事件（第07~16行）中，定义了一个setInterval()函数，这个函数会定时以speed变量指定的时间进行触发。在回调函数内部，随机地显示一张图片。

注意：本例使用了一个并不存在的CSS样式loaded，这是一个标记样式，以供jQuery选择未被显示的图片，当图片显示出来后，会动态地为span添加一个loaded样式，以区分显示与未显示的图片。

当所有的图片都显示完成后，调用 clearInterval()来清除定时器，结束随机显示的过程。

7.24 按顺序淡入图片显示

前面介绍了随机淡入图片的显示方式，有时候也要按顺序进行淡入显示，比如依次展示同一系列的产品时，按顺序淡入图片会显得更加友好。本例效果如图 7.18 所示。

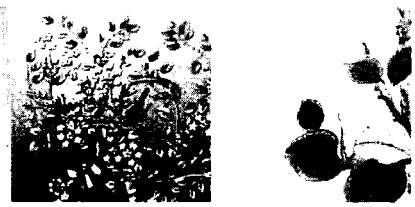

图 7.18 从左到右按顺序淡入显示图片

本例代码如下：

```
/* orderedFadeInPictures.html */
<script type="text/javascript">
    $(document).ready(function(){
        $('div#content span').hide();          //隐藏所有的图片
        function outer(){                      //定义一个函数，它返回一个闭包函数
            var a = 0;                         //定义下标变量
            function inner(){                  //定义一个内部函数，它会依次淡出图片
                if(a===$('div#content span').length){
                    return;                    //如果图片显示完成，则返回
                    }
                //依次淡出显示图片
                $('div#content span').eq(a).fadeIn(1000,function(){fader();});
                a++;                           //将计数器加 1
            }
        return inner;                          //返回内部函数
        }
        var fader = outer();                   //将函数结果赋给 fader 变量
        fader();                               //执行内部函数
    });
</script>
<style>
 /*指定图片 span 容器的样式*/
 #content span {
     width: 200px;                             /*宽度*/
      height:200px;                            /*高度*/
      display:block;                           /*块状显示*/
      float: left;                             /*向左浮动*/
```

```
28        overflow:hidden;                    /*溢出隐藏*/
29    }
30  </style>
31  </head>
32
33  <body>
34  <!--图片显示的容器-->
35  <div id="content">
36    <!--定义各种不同的要显示的图片-->
37    <span><img src="images/sample1.jpg" alt="喜鹊报喜" /></span>
38    <span><img src="images/sample2.jpg" alt="姹紫嫣红" /></span>
39    <span><img src="images/sample3.jpg" alt="水墨风情" /></span>
40    <span><img src="images/sample5.jpg" alt="含苞盛放" /></span>
41  </div>
42  </body>
43  </html>
```

第 35~41 行代码放了 4 幅从左到右排列的图片，在 jQuery 的页就绪事件中，第 04 行代码首先隐藏所有的图片，第 05~16 行定义了一个名为 outer 的函数，在 outer()函数内部定义了一个 inner()函数，这个函数将按顺序淡入图片的显示。outer()函数将返回这个 inner()函数。

注意：在 JavaScript 中，当在一个函数体内定义一个内部函数时，可以将这个函数作为一个变量或一个返回值进行调用，内部函数将保留在外部函数的作用域范围内，这种定义方式称为闭包。

从代码可以看出，outer()函数作为一个变量赋给 fader 变量，然后调用了 fader 变量，从而按从左向右的顺序依次淡入显示图片。

7.25 检测图片的 URL 是否有效

本章 7.5 节介绍过一种通过 img 元素的 load 事件处理器来检测图片 src 是否成功加载的方法，本节来介绍另一种检测图片 URL 的办法，通过编写函数直接对 img 元素的 src 地址进行测试，可以快速地对无效的 URL 进行检测。

本例创建了一个用来检测图片 URL 的通用函数，通过调用这个函数来实现对图片 URL 的检测，代码如下：

```
01  /* checkValidatedSrc.html */
02  <title>检测图片 URL 实例</title>
03  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js"
04  type="text/javascript"></script>
05  <script type="text/javascript">
06  //在页加载就绪事件中，为链接关联事件处理代码
07  $(document).ready(function(){
08  $("#btntesting").click(function(){
```

```
09        var answer=imageTest($("#googleimg").attr("src"));        //检测选中的 img 的 src
10        (answer==true)?alert("成功"):alert("失败");                //显示检测结果消息
11        });
12  });
13
14  function imageTest(url) {
15        var imageTarget = $('#urlImageTester');                    //获取隐藏的 img 元素
16         imageTarget.attr('src', url);                             //设置 img 元素的 src 属性
17        var props = ['naturalHeight', 'fileCreatedDate'];          //定义属性数组
18         var tests = [];                                           //定义测试结果数组
19         var answer;                                               //定义返回结果变量
20        for (i in props) {                                         //遍历数组元素
21              tests.push(imageTarget.attr(props[i]));  //向 test 数组中插入特定数组元素的值
22          }
23        if($.browser.msie){( //如果是 IE，如果第 2 个元素为未定义，则返回 false,否则为 true
24                  tests[1] == 'undefined') ? answer = false : answer = true;
25        }else{  //如果为其他浏览器，如果第 1 个元素值为 0，则返回 false，否则为 true
26              (tests[0] == 0) ? answer = false : answer = true;
27        }
28        return answer;                                             //返回检测结果
29  }
30  </script>
31  <style type="text/css">
32    #urlImageTester{
33          display:none;                                            /*用 CSS 隐藏测试用的 img 元素*/
34    }
35  </style>
36  </head>
37
38  <body>
39    <!--定义一个用来进行测试的隐藏的 img 元素-->
40    <img id="urlImageTester" />
41    <!--定义一个用来测试的图片-->
42    <img id="googleimg"   src="http://www.baidu.com/img/baidu_jgylogo3.gif?v=97734195.gif"
43  />
44    <br/>
45    <!---定义一个用来检测图片的链接-->
46    <a href="#" id="btntesting">检测图片链接</a>
47  </body>
48  </html>
```

第 40 行代码在 HTML 页面上放了一个 id 等于 urlImageTester 的 img 元素，通过 CSS 使其隐藏显示，这个 img 元素被用来测试其他 img 元素的 URL。第 14~29 行定义了一个名为 imageTest 的函数，这个函数接收一个要进行测试的图片的 URL，它会将图片的 URL 作为 urlImageTester 这个 img 元素的 src 属性，然后测试其中的两个属性 naturalHeight 和

fileCreatedDate，这两个属性是与 img 相关的两个属性，通过判断其属性值，就可以得知图片 URL 是否正确。imageTest 返回布尔值，成功返回 true，失败返回 false，最后在 jQuery 的页加载事件中，第 09~10 行代码对 googleimg 这个 img 元素进行了测试，得到了正确的检测结果。

7.26 强制显示图片的方法

如果图片被隐藏或是给定了一个很低的透明度，此时要将图片显示出来需要很多行代码。这时可以向 jQuery 编写一个强制显示图片的插件函数，这样就可以很轻松地显示图片了。本例代码如下：

```
01  /* forceShowPicture.html */
02  <title>强制显示图片</title>
03  <script type="text/javascript" src="jquery/jquery-1.9.1.js"></script>
04  <script type="text/javascript">
05      $(document).ready(function(){
06          $("#flower").forceShow();
07      });
08      //编写一个 jQuery 插件函数
09      jQuery.fn.forceShow = function() {
10          if (this.css('opacity') < 0.25) {          //如果透明度小于 0.25
11              this.css('opacity',1)                  //调整透明度为 1
12          }
13          return this.show().css('visibility','visible')   //设置图像可见
14      }
15  </script>
16  <style type="text/css">
17      #flower{
18          visibility:hidden;                         /*使用 CSS 让图片隐藏*/
19      }
20  </style>
21  </head>
22
23  <body>
24  <!--放一张图片，这张图片被隐藏显示-->
25  <img id="flower" src="images/sample5.jpg" />
26  </body>
27  </html>
```

第 25 行在 HTML 页面上放了一个 img 元素，然后使用 CSS 将其 visibility 设置为 hidden，表示隐藏显示。在 JavaScript 代码区中第 09~14 行编写了一个 forceShow()函数，这个函数可以强制显示图片。

注意：编写以jQuery.fn或者是$.fn开头的插件函数，表示创建的是jQuery实例函数，可以在选择器中调用实例函数。

在实例的页加载事件中，要显示隐藏的图像，只需要调用 forceShow 插件函数就可以强制地显示图片了。

7.27 实现可拖动显示的图片

在一些网站上，可以放置一些允许拖动的图片，比如一些指导性的图片，可以允许用户拖动进行显示。要实现这个效果并不复杂，只需要处理一下网页上的 mousemove、mousedown 以及 mouseup 即可。本例效果如图 7.19 所示。

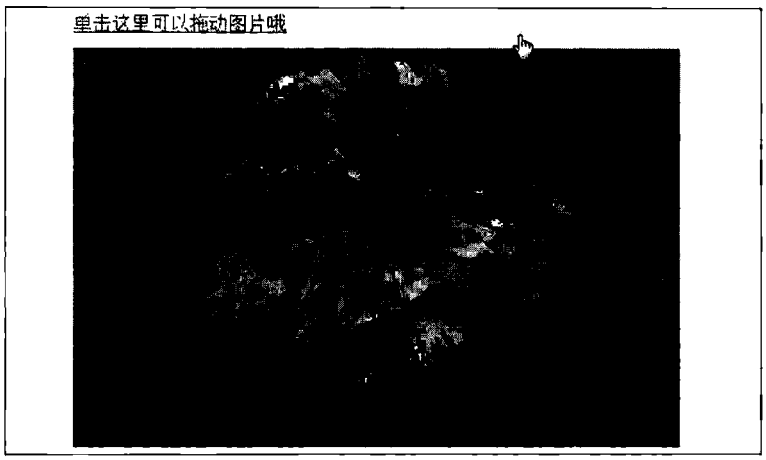

图 7.19 鼠标拖动图片

当用鼠标拖动标题栏时，可以看到图片果然跟随鼠标在进行移动，当释放鼠标左键时，图片将停止移动。

本例 JavaScript 代码如下：

```
01  /* forceShowPicture.html */
02  <script type="text/javascript" src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04      //在页就绪事件中，为拖动标题栏关联相关的事件
05      $(document).ready(function(){
06      var mover = false;                          //控制是否移动的全局变量
07      var currx = 0;                              //保存当前 x 位置的变量
08      var curry = 0;                              //保存当前 y 位置的变量
09      //为图片标题栏关联 mousedown 事件
10      $(".imgtext .title").mousedown(function(eventObject){
11          mover = true;                           //移动开始，设置移动标志
12          currx = eventObject.pageX;              //保存当前的 x 位置
13          curry = eventObject.pageY;              //保存当前的 y 位置
```

```
14        });
15        //为 window 关联 mousemove 事件，以跟踪并改变包含图片的 div 的位置
16        $(window).bind('mousemove', function(eventObject) {
17            if(!mover)                          //如果未处于移动状态，则返回
18                return;
19            var obj = $(".imgtext");            //更改包含图片的 div 的位置
20            mouseX = eventObject.pageX;         //获取当前鼠标所在的 x 位置
21            mouseY = eventObject.pageY;         //获取当前鼠标所在的 y 位置
22            x = obj.position().left;            //得到当前图片 div 所在的 x 位置
23            y = obj.position().top;             //得到当前图片 div 所在的 y 位置
24            obj.css({                           //改变当前图片 div 的位置
25                left: x + (mouseX-currx),
26                top: y + (mouseY-curry),
27            });
28            currx = eventObject.pageX;          //保存当前 x 的位置
29            curry = eventObject.pageY;          //保存当前 y 的位置
30        });
31        //为图片标题栏关联 mouseup 事件
32        $(".imgtext .title").mouseup(function() {
33            mover = false;                      //更改 mover 标记为 false
34        });
35    });
36    </script>
```

第 10 行的$(".imgtext .title")将获取 imgtext 中嵌套的 class 为 title 的一个标题栏，imgtext 这个 div 元素内部由一个标题 div 和一幅图片组成。第 10~14 行先关联了标题 div 的 mousedown 事件，当鼠标按下时，将更新 mover 这个标记变量，并将当前 x 和 y 的位置保存到全局变量中。重点是第 16~30 行关联到 window 的 mousemove 事件，这个事件处理函数将判断 mover 变量是否被设置为 true，然后更新 imgtext 这个 div left 和 top 的值，并且将当前的 *x* 和 *y* 位置保存到 currx 和 curry 全局变量。最后关联标题栏的 mouseup 事件，将 mover 设置为 false，表示停止移动图片。

第 8 章　jQuery 实现 AJAX

AJAX 是独立于 jQuery 的一门技术，它的全称是“异步 JavaScript 和 XML”，它是英文词组 Asynchronous JavaScript and XML 的首字母简写。AJAX 可以让客户端网页在不刷新的情况下与服务器端交换数据，实现对网页的局部更新。它的出现让网页的呈现更加丰富多彩，进而出现了很多完全依赖于客户端的富客户端网页。jQuery 库丰富且简化了 AJAX 的实现方法，提供了一系列强大的 API 函数，让 AJAX 操作变得简单且稳定。

本章主要涉及的知识点有：

- 使用 AJAX 动态加载外部内容。
- 加载网页并传递参数。
- 无刷新删除网页记录。
- 无刷新验证用户名和密码。
- 无刷新提交表单。
- 无刷新上传图片。

8.1　动态加载外部文件

很多网站都有日志更新的页面，这个页面显示网站的历史更新记录，多数的站长习惯于在一个文本文件中编写更新日志，在网页上通过一个异步的 AJAX 调用来显示日志内容。jQuery 的 load()函数可以轻松地满足这个要求。本例效果如图 8.1 所示。

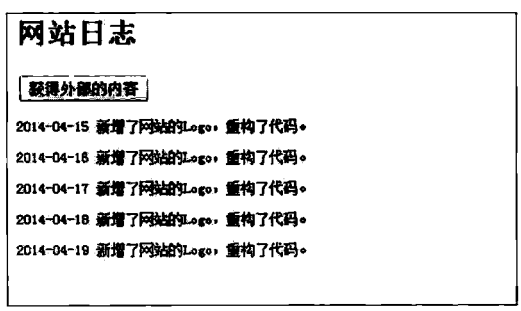

图 8.1　使用 AJAX 异步加载服务器端的网站日志

假如在服务器端有一个名为 logs.txt 的日志文件，可以使用如下代码来加载这个日志文件并显示在页面上：

```
01  /* loadingExternalFiles.html */
02  <script src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04      $(document).ready(function(){
05          $("#btn1").click(function(){                    //按钮事件处理代码
06              $('#logscontent').load('jquery/files/logs.txt');    //AJAX 加载服务器端文件
07          })
08      })
09  </script>
10  </head>
11  <body>
12  <h2>网站日志</h2>
13  <button id="btn1" type="button">获得外部的内容</button>
14  <!--用来放置服务器端文件的 div 元素-->
15  <div id="logscontent"></div>
16  </body>
17  </html>
```

第 13~15 行有一个按钮和一个 div 元素，当单击按钮时，第 05~07 行会调用 jQuery 的 load()函数，它接收要加载的外部文件的 URL 地址，将服务器端返回的内容加载到 id 为 logscontent 的 div 中。

8.2 动态加载外部网页

除了加载静态的 txt 内容之外，很多网站还需要加载一些已经定义好的 HTML 文件，或者是动态服务器语言页面（如 PHP 或 ASP.NET 页面），这都可以通过 AJAX 实现。本例将利用 load()回调函数加载一个 HTML 文件，同时还可以了解到 HTML 的加载是成功还是失败。本例效果如图 8.2 所示。

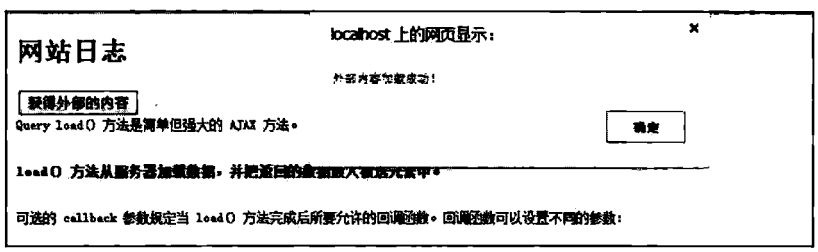

图 8.2 加载外部 HTML 内容

本例代码如下：

```
01  /* loadingExternalHtml.html */
02  <script src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04      $(document).ready(function(){
05          $("#btn1").click(function(){                    //按钮事件处理代码
```

```
06          //使用 load()函数，异步地加载服务器端的 HTML 页面，并整合到当前的 DOM 内容中
07          $('#logscontent').load("http://localhost/Chp08/jquery/files/htmldemo.html",
08          //加载回调函数，statusTxt 表示加载状态
09          function(responseTxt,statusTxt,xhr){
10              if(statusTxt=="success")
11                alert("外部内容加载成功！");              //显示加载成功消息
12              if(statusTxt=="error")                    //显示加载失败消息
13                alert("加载错误: "+xhr.status+": "+xhr.statusText);
14            });
15      })
16    })
17  </script>
18  </head>
19  <body>
20  <h2>网站日志</h2>
21  <button id="btn1" type="button">获得外部的内容</button>
22  <!--用来放置服务器端文件的 div 元素-->
23  <div id="logscontent"></div>
24  </body>
25  </html>
```

本例加载的是一个完整的 HTML 文件，第 07~14 行的 load()函数会将 HTML 页面的内容与当前网页 DOM 进行完全的整合，从而在当前的网页上显示 HTML 内容。

注意：默认情况下，jQuery 的 load()函数会使用 HTTP 的 GET 函数向服务器端发送数据请求。

8.3　加载网页文件内容并传递服务器端参数

jQuery 的 load()函数也可以向服务器端传递参数，这允许用户选择一些参数上传到服务器端，然后服务器端接收用户传递的参数。

注意：jQuery 的 load()函数传递参数时，将会自动地以 POST 方式进行提交。

本例允许用户从客户端的列表框中选择一个参数值，单击“获取参数描述”按钮时，将从服务器端的 PHP 中获取参数的描述。本例效果如图 8.3 所示。

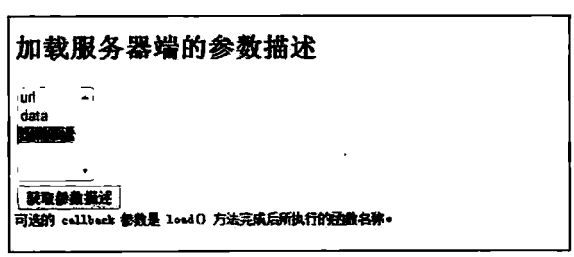

图 8.3　加载服务器端内容并传递参数

本例代码如下：

```
/* loadWithParameters.html */
<script src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript">
    $(document).ready(function(){
      $("#btn1").click(function(){                    //按钮事件处理代码
        $name = $("#parameters   option:selected").val(); //获取 select 中当前选中的参数值
        //使用 load()函数，加载服务器端的 PHP 网页，并传递客户端选择的参数
        $('#logscontent').empty().load
      ("http://localhost/Chp08/loadWithParameters_server.php",{ name : $name });
       })
    })
</script>
</head>
<body>
<h2>加载服务器端的参数描述</h2>
<label for="parameters"></label>
<select name="parameters" size="5" id="parameters">
  <option value="1" selected="selected">url</option>
  <option value="2">data</option>
  <option value="3">callback</option>
</select>
<br/>
<button id="btn1" type="button">获取参数描述</button>
<!--用来放置服务器端文件的 div 元素-->
<div id="logscontent"></div>
</body>
</html>
```

第 17~21 行在网页上放置了一个 select 列表框，这个列表框包含了 jQuery 的 load()函数所需要的几个参数，当用户选择其中一个参数并单击“获取参数描述”按钮时，将会向服务器端的 loadWithParameters_server.php 文件发送请求，并且通过键/值对的方式传递了用户选中的参数。服务器端的 PHP 文件接收到客户端的 POST 参数后，会执行一系列的条件判断，代码如下：

```
/* loadWithParameters_server.php */
<?php
switch($_POST['name']){
  case 1:
    echo '必需的 URL 参数规定您希望加载的 URL。';
    break;
  case 2:
    echo '可选的 data 参数规定与请求一同发送的查询字符串键/值对集合。';
    break;
  case 3:
    echo '可选的 callback 参数是 load() 函数完成后所执行的函数名称。';
```

```
12        break;
13    }
14    ?>
```

服务器端的 PHP 文件（第 03~13）行使用了一个 switch 语句来判断客户端 POST 过来的名为 name 的参数，然后调用 echo 语句返回输出结果。

8.4 使用 AJAX 无刷新删除网页记录

无刷新删除是目前使用非常频繁的一种 AJAX 应用，它可以让用户无刷新地删除一条服务器端的记录，就好像是删除客户端本地的一条记录一样。在目前的留言本、论坛、微博等应用中，基本都是用无刷新的记录删除方式。本例创建一个简单的留言显示页，给出一个图标允许用户单击图标，使用 AJAX 方式异步地从服务器端删除选定的记录。本例效果如图 8.4 所示。

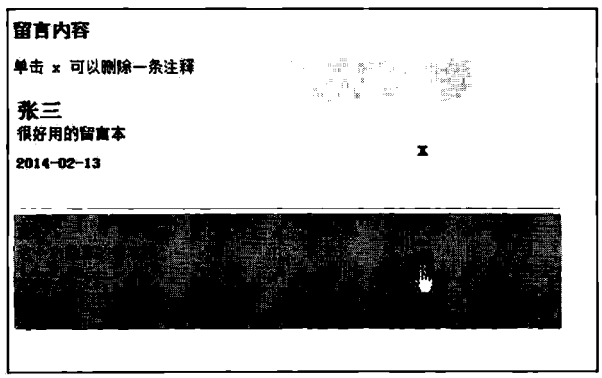

图 8.4 异步无刷新地删除记录

留言页面的 HTML 代码如下：

```
01  /* ajaxDelete.html */
02  <div id="container">
03    <h3>留言内容</h3>
04    <div id="load" align="center">
05          <!--删除时显示的异步加载进度条-->
06        <img src="images/loading.gif" width="28"
07              height="28" align="absmiddle"/> Loading...</div>
08      单击 x 可以删除一条注释<br/>
09      <br/>
10    <div class="box">
11      <div class="text"><span>张三</span><br/>
12        很好用的留言本
13          <div class="date">2014~02~13</div>
14      </div>
15      <!--删除链接，每个链接的 id 值，指向数据库中当前的记录 id-->
```

```
16        <a href="#" id="1" class="delete">x</a>
17        <div class="clear"></div>
18      </div>
19      ……
20    </div>
```

留言页面包含一个删除时异步加载的进度图片，以及若干条 class 为 box 的 div 元素，每条 div 中包含留言人姓名、留言内容和日期。

注意：每一条留言的删除链接都包含了一个 id 值，这个 id 值应该是指向数据库中该条留言的数据库 id，以便进行数据库的删除。本例出于简化的目的使用了固定的 id 编码值。

编写如下的 jQuery 代码来实现异步无刷新地删除：

```
01  /* ajaxDelete.html */
02  <script src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04  $(document).ready(function() {
05      $('#load').hide();                                  //隐藏加载图片，只在需要时显示
06  });
07  $(function() {
08      $(".delete").click(function() {                     //当删除按钮被单击时
09          $('#load').fadeIn();                            //淡出显示加载图片
10          var commentContainer = $(this).parent();        //得到当前链接所在的容器 div
11          var id = $(this).attr("id");                    //得到当前链接的 id 值
12          var string = 'id='+ id ;                        //构建参数字符串
13          $.ajax({                                        //调用$.ajax()发送异步 AJAX 请求
14              type: "POST",                               //指定提交方式为 POST
15              url: "ajax_delete_Server.php",              //服务器端的 URL
16              data: string,                               //传递的参数字符串
17              cache: false,                               //不缓存
18              success: function(){                        //成功删除后，移除留言记录
19                commentContainer.slideUp('slow', function() {$(this).remove();});
20                $('#load').fadeOut();                     //隐藏显示加载图标
21              }
22            });
23            return false;
24          });
25  });
26  </script>
```

在页就绪事件中，第 05 行首先隐藏了加载图标的显示，这个图标只在删除异步提交时才有用。第 08~24 行为所有的删除链接关联了 click 事件处理代码，它首先显示加载进度图标，然后获取当前被单击留言的 div 实例以及当前链接的 id 值。以当前的链接 id 值作为参数，第 13~22 行调用$.ajax()，向 ajax_delete_Server 文件发送一个 AJAX 请求，用来删除留言。当留言删除之后，删除留言 div，并且淡出加载图标的显示。

8.5　使用 AJAX 异步验证用户名和密码

目前多数的网站都具有用户登录框，有时候一个专业的登录框会极大地增强用户的体验，因为这往往是用户进入网站进行互动的第 1 步。使用 jQuery 的 AJAX 功能，可以对用户名与密码进行无刷新异步验证。本例将创建一个用户登录框，然后使用 AJAX 方式，异步地向后台的 PHP 页面发送验证请求，验证成功后返回成功提示消息框。本例效果如图 8.5 所示。

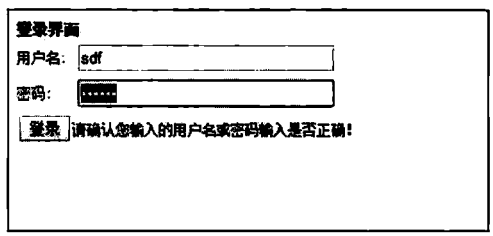

图 8.5　用户登录界面验证效果

本例构建了一个简单的 HTML 登录表单，当用户单击“登录”按钮时，将完成一系列验证，HTML 代码如下：

```
/* validateUserPassword.html */
<table>
    <tr>
        <td colspan=2 class="title">登录界面</td>
    </tr>
    <tr>
        <td>用户名:</td>
        <td><input type=text name=username id="txtusername" size=30>
            <span id="msguser"></span></td>
    </tr>
    <tr>
        <td>密码:</td>
        <td><input type=password  name=password  id="txtpassword"  size=30>
            <span id="msgpwd"></span></td>
    </tr>
    <tr>
        <td colspan=2><input type=button value=登录 id="btnsubmit">
            <span id="msg"></span></td>
    </tr>
</table>
```

在 HTML 代码中构建了一个表格，里面有文本框和按钮控件。这里没有使用表单元素，因为验证工作将由 jQuery 代码异步调用 AJAX 来实现。下面的 JavaScript 代码演示了如何对用户名和密码进行异步验证，代码如下：

```
/* validateUserPassword.html */
```

```
<script src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript" language="javascript">
    $(document).ready(function(){
        $("#btnsubmit").click(function(){                    //为按钮关联单击事件处理代码
            checklogin();                                    //调用自定义函数进行登录验证
        });
    });
        function checklogin() {                              //定义一个登录验证的函数
            if ($("#txtusername").val() == "") {             //检测用户名是否为空
                $("#msguser").html("用户名不能为空！"); //显示提示消息
                $("#txtusername").focus();
                    return false;
            }else
              {
              $("#msguser").html("");
                  }
            if ($("#txtpassword").val() == "") {                    //检测用户密码是否为空
                $("#msgpwd").html("密码框不能为空！");      //显示提示消息
                $("#txtpassword").focus();
                    return false;
            }else{
              $("#msgpwd").html("");
              }
            $.ajax({      //如果用户名和密码都不为空，则异步发送 AJAX 验证请求
                type: "POST",
                url: "validuser_Server.php",
                data: "username=" + $("#txtusername").val().toString()
                        + "&password=" + $("#txtpassword").val().toString(),
                success: function (data) {      //如果异步调用成功，则判断返回的状态值
                    if (data == "1") {          //如果返回状态值为 1，则表示登录成功
                        $("#msg").html("登录成功！");
                        return true;
                    }
                    else {                      //如果返回的状态值为 2，则登录失败
                            $("#msg").html("请确认您输入的用户名或密码输入是否正确！
                            ");
                         $("#txtusername").focus();
                        return false;
                    }
                }
            })
        }
</script>
```

在第 05~07 行的按钮单击事件中，调用了自定义的 checklogin()函数，这个函数会检测用户名和密码文本框是否为空，在不为空的前提下，第 25~42 行发送$.ajax()，将用户名和密码作为参数，发送给服务器端的 validuser_Server.php，这个 PHP 文件简单地验证了硬编码的用户名和密码，如果验证成功返回 1，如果验证失败返回 2。在运行时可以看到，如果用户名输入 admin、密码输入 123 的话，会显示登录成功的消息，如图 8.6 所示。

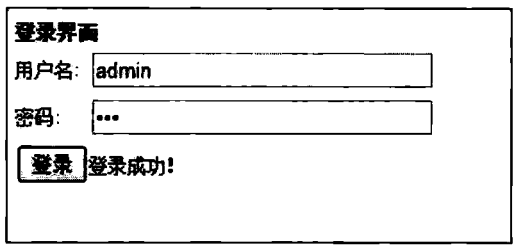

图 8.6　登录成功显示成功的消息

8.6　页面滚动时加载新的网页内容

经常可以看到一些网站没有传统的分页符号，看完一页鼠标向下拉时，就会自动加载新的内容，避免了单击翻页按钮的麻烦，可以一直浏览到底，这样的效果用 jQuery 的 AJAX 可以轻松实现。本例将演示如何实现滚动条拖动时加载新内容的效果，如图 8.7 所示。

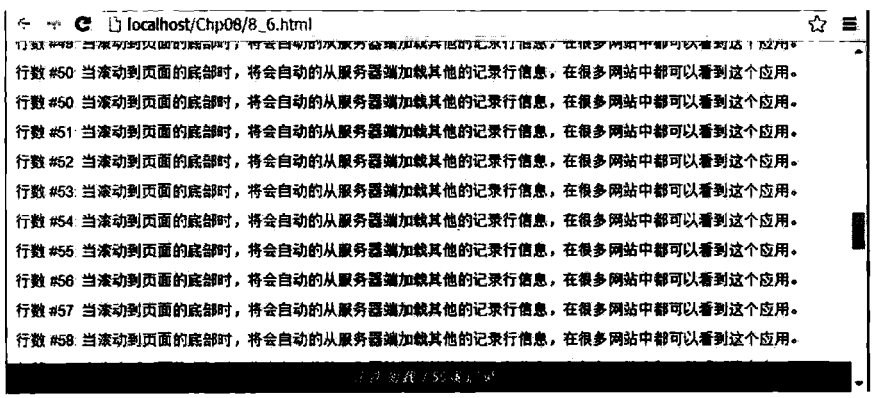

图 8.7　滚动时的 AJAX 加载

注意：滚动加载内容的核心是关联 window 的 onscroll 事件，这个事件在拖动滚动条时触发。在这个事件中判断当前滚动条的位置，如果滚动条的高度大过当前文档的高度，则向服务器端发送数据请求。

本例在 HTML 页面上放了一些简单的文字内容，在 JavaScript 中定义了如下代码，以实现滚动时加载内容：

```
01  /* ajaxScroll.html */
02  <script src="jquery/jquery-1.9.1.js" type="text/javascript"></script>
03  <script type="text/javascript">
```

```
04  $(document).ready(function() {
05      $('#loaded_max').val(50);          //为隐藏域赋值 50，表示一次加载 50 条记录
06  });
07  var loading = false;                   //全局变量，指定当前是否正在加载服务器端内容
08  $(window).scroll(function(){          //关联 window 的 onscroll 事件
09      //如果当前窗体的高度大于文档的高度
10      if(($(window).scrollTop()+$(window).height())+250)>=$(document).height()){
11          if(loading == false){
12              loading = true;                                    //设置文档加载状态
13              $('#loadingbar').css("display","block");           //显示加载提示条
14              //调用$.get，向服务器请求记录，start 参数提定起始编号
15              $.get("8_6_load.php?start="+$('#loaded_max').val(), function(loaded){
16                  $('body').append(loaded);                      //加载返回的服务器内容
17                  //在隐藏域中设置新的起始值
18                  $('#loaded_max').val(parseInt($('#loaded_max').val())+50);
19                  $('#loadingbar').css("display","none");        //隐藏状态条的显示
20                  loading = false;                               //结束加载的状态
21              });}}
22  });
23  </script>
```

id 为 loaded_max 的元素是一个 HTML 隐藏域，用来保存当前已经加载的最大记录编号。第 07 行定义的 loading 全局变量用来保存当前是否处于加载状态，可以看作一个标记变量。第 08 行关联了 window.onscroll 事件，判断当前滚动的位置是否大于当前文档的高度，如果条件成立，将 loading 标记位设置为 true，然后显示加载中的进度条。第 15 行调用$.get()函数，异步地向 ajaxScroll_Server.php 文件请求数据，传递一个参数 start 表示起始记录号。在加载完成的回调函数中，将服务器端返回的内容加载到 body 中，并且重置隐藏域中的值，以及隐藏状态条的显示，最后将 loading 重置为 false。

ajaxScroll_Server.php 文件向客户端返回一些固定的数据，代码如下：

```
01  /* ajaxScroll_Server.php */
02  <?php
03   header("Content-Type: text/html; charset=UTF-8");      //UTF8 编码
04    $maxval=1;                                            //定义起始值变量
05   if (isset($_GET['start'])){
06       $maxval=$_GET['start'];                           //获取起始值
07   }
08   $i=1;
09   while($i<=50)                                         //循环向客户端输出数据
10    {
11    echo "行数 #".$maxval.": 当滚动到页面的底部时，将会自动的从服务器端加载其他的记录行
12    信息，在很多网站中都可以看到这个应用。<br/>";
13    $i++;
14    $maxval++;
15    }
```

```
16  ?>
```

本例将循环返回50条记录，客户端会将50条记录追加到body区域。当滚动条滚动到页面底部时，将显示加载指示条，从服务器端异步加载记录，加载完成后，显示记录。

8.7 打造自己的站内搜索引擎

很多网站都有自己的站内搜索引擎，但是要从头开发一个搜索引擎是相当复杂的，而且需要耗费自己站点的服务器资源，所以很多网站或借用Google自定义的搜索服务，或使用jQuery整合Google的搜索API，来实现自己的站内搜索功能。使用jQuery，只需要简单的几行代码就可以构建一个看起来非常个性化的搜索服务，如图8.8所示。

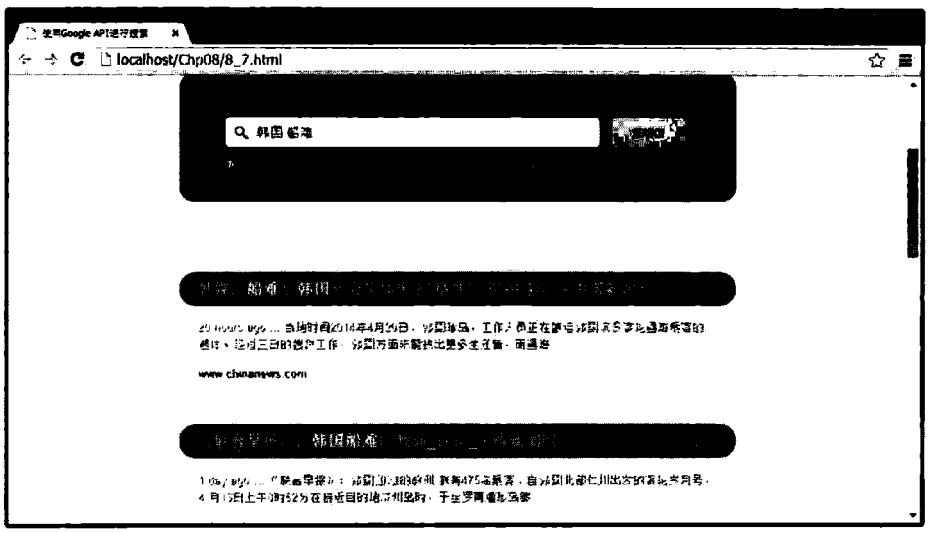

图8.8 打造自己的站内搜索引擎

用户在文本框中输入搜索关键词，单击Search按钮时，这个搜索的关键词将会作为参数传递给Google的API，然后返回一个JSON格式的搜索结果，添加到网页上。本例单击Search按钮时，会执行如下代码：

```
01  /* ajaxScroll_Server.php */
02      //搜索按钮的代码
03      $('#searchForm').submit(function(){
04          googleSearch();        //调用 googleSearch()函数进行搜索
05          return false;
06  });
```

googleSearch()是搜索的核心函数，它的实现代码如下：

```
01  /* ajaxScroll_Server.php */
02      function googleSearch(settings){                    //settings 表示配置参数
03          //如果没有提供任何参数，它将从上面的配置对象代码中取默认值
04          settings = $.extend({},config,settings);
```

```
        settings.term = settings.term || $('#s').val();  //获取搜索关键词
        if(settings.searchSite){          //使用 site:xxx.com 来限制搜索的网站域名
            settings.term = 'site:'+settings.siteURL+' '+settings.term;
        }
        //Google AJAX 的搜索 API
    var apiURL =
    'http://ajax.googleapis.com/ajax/services/search/'+
            settings.type+'?v=1.0&callback=?';
        var resultsDiv = $('#resultsDiv');
        //调用$.getJSON 提交搜索请求并获取返回的 JSON 数组
        $.getJSON(apiURL,{q:settings.term,rsz:settings.
            perPage,start:settings.page*settings.perPage},function(r){
            var results = r.responseData.results;      //获取返回结果
            $('#more').remove();
            if(results.length){
            //如果返回了结果，将其添加到 pageContainer div 中，追加到#resultDiv 中
            var pageContainer = $('<div>',{className:'pageContainer'});
                for(var i=0;i<results.length;i++){
        //添加结果对象，并添加到 pageContainer 中，这里调用了 result()函数生成结果
                    pageContainer.append(new result(results[i]) + '');
                }
                if(!settings.append){    //根据设置判断是添加显示还是全新显示
                    resultsDiv.empty(); //如果不为 append，则清空结果区域
                }
                //添加结果数据 div 到 resultDiv 中
                pageContainer.append('<div class="clear"></div>')
                             .hide().appendTo(resultsDiv)
                             .fadeIn('slow');
                var cursor = r.responseData.cursor;
                //检查是否有更多页的结果，以决定是否显示更多按钮
            if( +cursor.estimatedResultCount > (settings.page+1)*settings.perPage){
                    $('<div>',{id:'more'}).appendTo(resultsDiv).click(function(){
                        googleSearch({append:true,page:settings.page+1});
                        $(this).fadeOut();
                    });
                }
            }
            else {
                //如果没有结果，则简单的显示一行没有找到搜索结果的消息
                resultsDiv.empty();
                $('<p>',{className:'notFound',html:'没有找到搜索内容
                        '}).hide().appendTo(resultsDiv).fadeIn();
            }
        });
    }
```

googleSearch()的参数 settings 是一个配置对象，用来配置搜索的模式，比如是站内搜索还是全 Web 搜索、搜索的类型以及搜索的记录数等等。apiURL 变量保存了搜索的 URL，然后调用$.getJSON()异步地发送搜索请求，并将返回的搜索结果（JSON 数组）写入到当前的网页。这种无刷新的搜索方式，并不影响网页上的其他部分，既提供了较好的性能，也满足了网站对于搜索的需求。

8.8 异步加载并解析 XML 文件

网站开发过程中会存在大量的 XML 文件，比如一些 XML 配置文件，或者 XML 静态数据。使用 jQuery 的 AJAX 功能，可以非常轻松地完成 XML 数据解析的工作。本例将演示如何从服务器端读取 XML 文件，效果如图 8.9 所示。

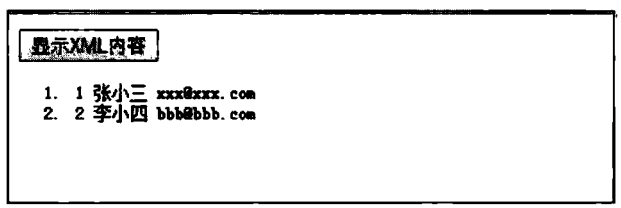

图 8.9 加载并显示 XML 内容

本例用到的 XML 文件代码如下：

```
01  /* asyncXML.xml */
02  <?xml version="1.0" encoding="UTF-8"?>
03  <stulist>
04      <student email="xxx@xxx.com">
05      <name>张小三</name>
06      <id>1</id>
07  </student>
08   <student email="bbb@bbb.com">
09      <name>李小四</name>
10      <id>2</id>
11      </student>
12  </stulist>
```

注意：XML 格式必须指定 encoding 为 UTF-8 编码，否则可能导致 XML 内容无法被正确解析。

通过如下$.ajax()代码，就可以异步地调用这个 XML 文件，并显示到网页上：

```
01  /* asyncXML.php */
02  <script src="jquery/jquery-1.9.1.js" type="text/javascript"></script>
03  <script type="text/javascript">
04   $(document).ready(function(){
05          $("#btn1").click(function(){                    //当单击按钮时，加载 XML 文件
```

```
06          $.ajax({                          //调用$.ajax 读取 XML 文件
07          url:' asyncXML.xml',              //指定读取的 URL
08          type: 'GET',                      //指定请求方式
09          dataType: 'xml',                  //这里指定数据类型为 xml
10          timeout: 1000,                    //指定超时时间
11          error: function(xml){             //异步请求错误时，显示错误消息
12                  alert('错误的加载 xml 文档'+xml);
13          },
14          success: function(xml){           //异步请求成功时
15                  //使用 jQuery 的 XML 语法查找并读取 XML 内容
16                  $(xml).find("student").each(function(i){
17                      var id=$(this).children("id");          //取 XML 的 id 子节点
18                  var id_value=$(this).children("id").text(); //取 XML 的 id 子节点的文本
19                  var name=$(this).children("name").text(); //取 XML 的 name 子节点的文本
20                  var email=$(this).attr("email");          //取 XML 的 email 属性
21                      $('<li></li>')                        //将 XML 内容写入到网页上
22                          .html(id_value+' '+name+' '+email)
23                          .appendTo('ol');
24                  });
25          }
26          });
27      });
28  });
29  </script>
30  </head>
31  <body>
32  <button id="btn1">显示 XML 内容</button>
33  <!--显示 XML 内容的 div 元素-->
34  <div id="xmlcontent">
35  <ol>
36  </ol>
37  </div>
38  </body>
39  </html>
```

第 05 行在按钮的单击事件中，通过$.ajax()来指定请求的 asyncXML.xml 文件和请求类型。当成功解析 XML 文件之后，对于每一个 XML 节点，构建了一个 li 元素，添加到 HTML 页面上的 ol 元素内部，从而显示到 HTML 页面上。

注意：解析 XML 文件时，必须指定其 dataType 为 xml，不能是 html 或 text，以避免出现无法解析的问题。

8.9　动态加载 HTML 内容到标签页中

网站通常都是由多个 HTML 页面组成，过去，当用户单击某个链接时，就加载整个页面到浏览器中。现在，有了 jQuery 的 AJAX 支持，可以异步地将 HTML 页面加载到一个网页中，这会提升网站的可访问性。本例效果如图 8.10 所示。

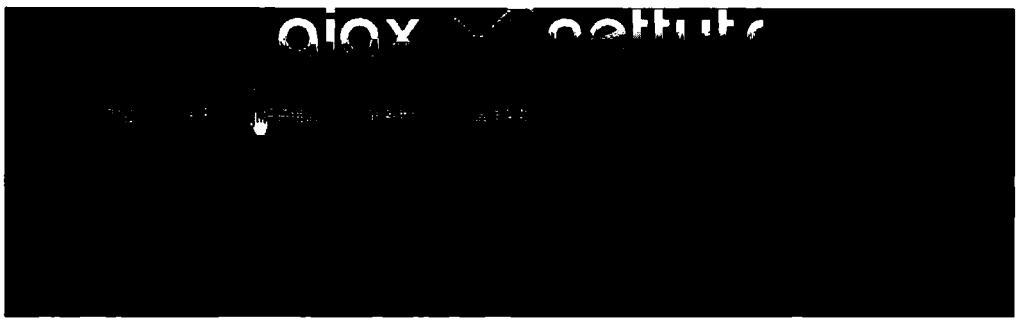

图 8.10　异步加载 HTML 内容到标签页中

本例的网站有 5 个页面，当用户单击其中一个页面时，会调用 jQuery 的 AJAX 函数，异步地移除当前标签页中的内容，然后添加新的 HTML 内容到标签页中，它不需要刷新网页，用户好像就是在一个页面中浏览一样。

本例的核心 jQuery 代码如下：

```
/* asyncHTMLtoTab.html */
$(document).ready(function() {
    var hash = window.location.hash.substr(1);      //设置或获取 href 中在"#"后面的字符串
    var href = $('#nav li a').each(function(){      //循环选中的链接元素
        var href = $(this).attr('href');            //得到链接 href 属性值
        if(hash==href.substr(0,href.length-5)){ //当 href 去掉了.html 扩展名后与 hash 比较
            var toLoad = hash+'.html #content';  //如果相同，则指定加载的 HTML 文件
            $('#content').load(toLoad)           //调用 load 加载
        }
    });
    //当导航链接单击事执行事件处理代码
    $('#nav li a').click(function(){
        var toLoad = $(this).attr('href')+' #content';      //获取当前导航链接 href+#content
        $('#content').hide('fast',loadContent);             //隐藏当前 content 中的内容
        $('#load').remove();                                //移除 load 元素
        //重新创建 load 元素进行显示
        $('#wrapper').append('<span id="load">LOADING...</span>');
        $('#load').fadeIn('normal');                        //淡入 load 元素
        //获取 hash 值
        window.location.hash = $(this).attr('href').substr(0,$(this).attr('href').length-5);
        function loadContent() {
            $('#content').load(toLoad,'',showNewContent()) //显示新内容
```

```
23              }
24              function showNewContent() {
25                  $('#content').show('normal',hideLoader());        //隐藏加载器
26              }
27              function hideLoader() {
28                  $('#load').fadeOut('normal');                     //隐藏 loading 的显示
29              }
30              return false;
31          });
32  });
```

在页就绪事件中，第 03~09 行检测 window.location.hash 的值，提取去除#之后的字符串。然后将这个字符串与当前链接的字符串（去除.html 扩展名后）比较，条件成立时，调用 div 的 load()函数，加载指定的 html 内容。

注意：在 JavaScript 中，window.location.hash 用于获取链接中的锚点字符串，例如链接中有 index.html #content，那么 window.location.hash 就可以获取到#content 字符串。

第 12~31 行是链接的单击事件处理代码，它是为链接加上#content 锚点字符串，然后显示或隐藏加载图标，并且为 window.location.hash 赋新的值。这样在每次单击不同的链接时，就可以看到不同的锚点字符串，如下：

http://localhost/Chp08/8_9/portfolio.html#terms

其中#terms 就是新添加的锚点字符串，这样就实现了页面内容的动态载入与移除。从本例可以看出，所有的网页最好放在同一个文件夹下，具有相同的层次深度，以便 window.location.hash 可以获取到正确的字符串。

8.10 使用 AJAX 无刷新异步提交表单

表单无刷新提交可以提高整个表单操作的体验，避免因为刷新操作带来的等待，但在编写上需要额外创建重定向的页面，比较复杂。有了 jQuery 中 AJAX 的辅助，可以异步无刷新地提交表单，大大提升了表单操作的用户体验。

本例构建一个 HTML 表单，这个表单允许用户向网站留言。HTML 代码如下：

```
01  /* ajaxSubmitForm.html */
02  <div class="block">
03  <h2 style="margin-left:100px">我的留言本-欢迎您的留言</h2>
04  <div class="done">
05  <b>谢谢!</b> 已经保存留言
06  </div>
07      <div class="form">
08      <form method="post" action="process.php">
09      <div class="element">
10          <label>姓名</label>
```

```
11              <input type="text" name="name" class="text" />
12          </div>
13          <div class="element">
14              <label>Email</label>
15              <input type="text" name="email" class="text" />
16          </div>
17          <div class="element">
18              <label>网站</label>
19              <input type="text" name="website" class="text" />
20          </div>
21          <div class="element">
22              <label>留言</label>
23              <textarea name="comment" class="text textarea" /></textarea>
24          </div>
25          <div class="element">
26              <!--提交按钮-->
27              <input type="submit" id="submit"/>
28             <!--加载提示-->
29              <div class="loading"></div>
30          </div>
31          </form>
32          </div>
33  </div>
```

第 08~31 行放置了一个 form 元素，构建了一个简单的留言本界面，如图 8.11 所示。

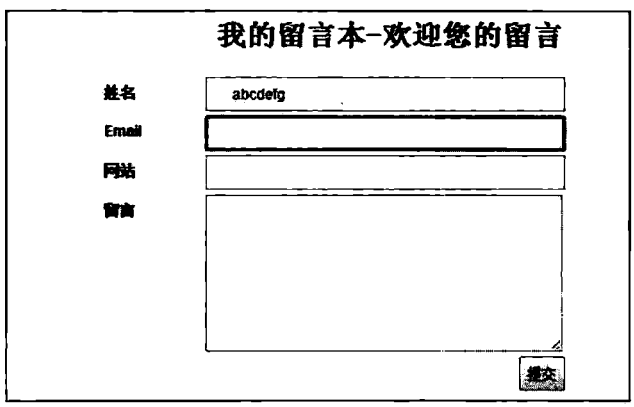

图 8.11　留言本表单界面

form 表单没有通过 action 属性指向要提交的服务器端网页，稍后将使用 jQuery 的 AJAX 代码来异步提交。表单内部使用 div+css 构建了一个简单的表单布局，接下来在页面的 head 区编写如下 JavaScript 代码来异步提交表单：

```
01  /* ajaxSubmitForm.html */
02  <script type="text/javascript" src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04  $(document).ready(function() {
```

```
05      $('#submit').click(function () {                    //为提交按钮关联事件处理代码
06          var name = $('input[name=name]');               //获取所有的表单上的数据
07          var email = $('input[name=email]');
08          var website = $('input[name=website]');
09          var comment = $('textarea[name=comment]');
10          if (name.val()=='') {       //确保输入了数据，否则应用 CSS 高亮显示
11              name.addClass('hightlight');
12              return false;
13          } else name.removeClass('hightlight');
14
15          if (email.val()=='') {      //确保输入了数据，否则应用 CSS 高亮显示
16              email.addClass('hightlight');
17              return false;
18          } else email.removeClass('hightlight');
19
20          if (comment.val()=='') {    //确保输入了数据，否则应用 CSS 高亮显示
21              comment.addClass('hightlight');
22              return false;
23          } else comment.removeClass('hightlight');
24          //组织 HTML 查询字符串属性，便于提交
25          var data = 'name=' + name.val() + '&email=' + email.val() + '&website=' +
26          website.val() + '&comment='   + encodeURIComponent(comment.val());
27          $('.text').attr('disabled','true');                 //禁止所有的表单输入
28          $('.loading').show();                               //显示加载进度
29          $.ajax({                                            //开始 AJAX 提交
30              url: "8_11.php",                                //指定服务器端文件
31              type: "GET",                                    //使用 GET 请求方式
32              data: data,                                     //传送数据
33              cache: false,                                   //不缓存此页
34              success: function (html) {                      //成功提交事件处理代码
35              if (html==1) {                                  //PHP 页面返回 1，表示成功
36                  $('.form').fadeOut('slow');                 //隐藏表单
37              $('.done').fadeIn('slow');                      //显示完成窗口
38                  } else alert('对不起，出现了未预料的异常，请重试.');
39              }
40          });
41          //取消提交按钮的默认行为
42          return false;
43      });
44  });
45  </script>
```

第 05~43 行为 submit 按钮关联了单击事件处理代码，第 06~28 行首先使用 jQuery 的选择器获取用户输入的留言信息，然后进行了简单的校验，如果没有在输入框中输入值，则会高亮显示输入框。在获取到输入值后，第 29~40 行调用$.ajax()向 8_11.php 服务器端发送

表单数据，使用 GET 提交方式，提交成功时显示留言提交成功的消息，否则显示留言保存失败的消息。成功提交后的效果如图 8.12 所示。

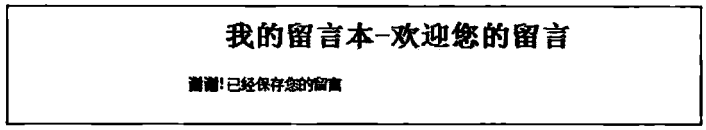

图 8.12　成功留言后的效果

从第 36 行可以看出，当留言成功后，调用 fadeOut()使表单进行了淡出隐藏，并且淡入显示了留言完成的提示消息。

8.11　使用 AJAX 无刷新上传图片

上传图片是很多网站都具有的功能，比如相册网站或者是产品展示网站，都需要上传图片，现在很多的博客也需要上传头像图片。传统的无 AJAX 上传图片，需要刷新页面，而且在上传过程中，用户除了通过浏览器的进度条来感知上传过程外，无法准确地获取上传进度，使用 AJAX 技术，可以实现无刷新地上传图片。本例将创建一个简单的图片上传网页，后台使用 PHP 来处理图片。本例效果如图 8.13 所示。

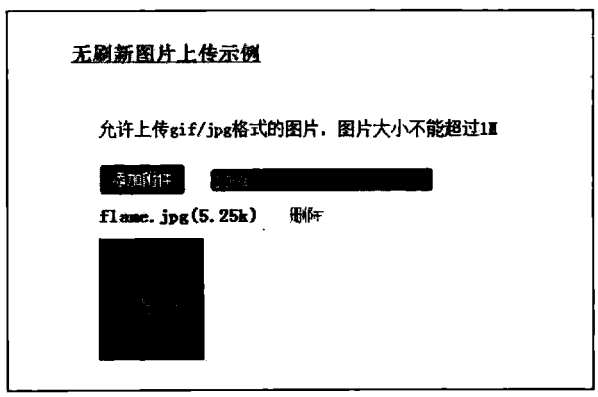

图 8.13　AJAX 无刷新上传图片

从图中可以看出，上传图片不仅是无刷新上传，而且还具有进度显示。除此之外，还可以进行异步删除，单击“删除”按钮，会在服务器端删除文件。本例需要构建一个简单的 HTML 上传界面，代码如下：

```
01  /* ajaxUploadImages.html */
02  <body>
03  <div id="main">
04      <!--网页标题-->
05    <h2 class="top_title"><a href="l">无刷新图片上传实例</a></h2>
06    <!--网页布局 div-->
07    <div class="demo">
```

```
08          <p>允许上传 gif/jpg 格式的图片，图片大小不能超过 1M </p>
09            <div class="btn">
10              <span>添加附件</span>
11              <!--上传用的 file 控件-->
12              <input id="fileupload" type="file" name="mypic">
13          </div>
14          <!--显示加载进度-->
15          <div class="progress">
16            <span class="bar"></span><span class="percent">0%</span >
17            </div>
18          <!--显示已经上传的文件名-->
19          <div class="files"></div>
20          <!--显示已经上传的图片-->
21          <div id="showimg"></div>
22      </div>
23        <br><br>
24  </div>
25  </body>
26  </html>
```

第 12~22 行包含了 file 控件、进度条以及显示文件名和图片预览的 div。这里并没有表单 form 标签，也不包含 MIME 编码方式，这些都在后面的 JavaScript 代码中进行设置。

注意：对于上传文件的表单，必须要指定 form 表单中 enctype 的 MIME 类型为 multipart/form-data，否则表单不能用于文件上传。

本例使用了一个知名的 jQuery 插件来处理表单的无刷新上传，这个插件就是 jquery.form.js，它可以非常容易地、无侵入地升级 HTML 表单以支持 AJAX。接下来看看 JavaScript 处理图片上传的代码：

```
01  /* ajaxUploadImages.html */
02  <!--添加对 jQueryForm 的引用-->
03  <script type="text/javascript" src="jquery/jquery.form.js"></script>
04  <script type="text/javascript">
05  $(function () {
06      var bar = $('.bar');                    //获取上传进度条 span
07      var percent = $('.percent');            //获取显示上传百分比的 span
08      var showimg = $('#showimg');            //显示图片的 div
09      var progress = $(".progress");          //显示进度的 div
10      var files = $(".files");                //文件上传控件 input 元素
11      var btn = $(".btn span");               //按钮文本
12      //调用 wrap()函数，为 id 为 demo 的 div 外层添加 form 元素
13      //指定 enctype 为文件类型，action 指定为 PHP 文件
14      $(".demo").wrap("<form id='myupload' action='8_12.php' method='post'
15              enctype='multipart/form-data'></form>");
16      $("#fileupload").change(function(){     //当上传文件改变时，触发事件
17          $("#myupload").ajaxSubmit({         //调用 jquery.form 插件的 ajaxSubmit()提交
```

```
18                  dataType:  'json',              //返回数据类型为 json
19                  beforeSend: function() {        //发送数据之前，要执行的代码
20                  showimg.empty();                //清空图片预览区
21                      progress.show();            //显示进度
22                  var percentVal = '0%';          //显示进度百分比
23                  bar.width(percentVal);          //设置进度的宽度
24                  percent.html(percentVal);       //设置进度值
25                      btn.html("上传中...");      //指定显示中
26              },
27              //更新进度条事件处理代码
28              uploadProgress: function(event, position, total, percentComplete) {
29                  var percentVal = percentComplete + '%';
30                  bar.width(percentVal)           //更新进度条的宽度
31                  percent.html(percentVal);
32              },
33                  success: function(data) {       //图片上传成功时
34                      //获取服务器端返回的文件数据
35                      files.html("<b>"+data.name+"("+data.size+"k)</b>
36                              <span class='delimg' rel="+data.pic+">删除</span>");
37                      var img = "files/"+data.pic;                //得到文件路径
38                      showimg.html("<img src="+img+">");          //显示上传的图片预览
39                      btn.html("添加附件");
40                  },
41                  error:function(xhr){                            //图片上传失败时
42                      btn.html("上传失败");
43                      bar.width('0')                              //重置进度条
44                      files.html(xhr.responseText);               //显示错误文本
45                  }
46              });
47      });
48      $(".delimg").live('click',function(){       //为删除按钮关联事件处理代码，这里用了 live
49          var pic = $(this).attr("rel");          //得到图片路径
50          //向服务器发送删除请求
51          $.post("8_12.php?act=delimg",{imagename:pic},function(msg){
52              if(msg==1){
53                  files.html("删除成功.");
54                  showimg.empty();
55                  progress.hide();                //重置进度条，并清空图片预览
56              }else{
57                  alert(msg);
58              }
59          });
60      });
61  });
62  </script>
```

第 14~15 行动态地为 div 元素外层包裹了一层<form>标签，指定其 enctype 和 action 属性，这样就可以利用 jquery.form 进行无刷新的内容上传。第 17~26 行使用 jquery.form 的 ajaxSubmit()来异步地提交表单，它具有好几个事件，可以控制表单提交的细节，比如 beforeSend，在发送数据之前触发，就可以初始化界面的显示。第 28~32 行的 uploadProgress 事件可以处理上传的进度。第 33~46 行的 success 和 error 可以在上传成功或失败时对界面进行相应地控制。当上传成功后，第 48~59 行动态地添加了一个“删除”链接，然后使用 jQuery 的 live 为链接关联事件处理代码。从代码可以看出，无论是提交还是删除，都异步地提交到 8_12.php 这个服务器端文件，这个文件处理图片的上传和删除操作，详细代码可以参考本书配套的代码。

8.12 使用 AJAX 无刷新验证 PHP 会话是否有效

在编写 PHP 应用程序时，如果服务器端的会话过期，能够通过无刷新的 AJAX 技术异步地提醒用户，可以避免用户在不知道会话已经过期的情况下导致的误操作。本例将使用 JavaScript 的 setInterval()一次又一次地执行一个异步 AJAX 检查，来实现这个功能。本例创建一个 PHP 的页面，这个页面启动时便开启了一个会话。

注意：一般会在用户注册登录时，开启用户的会话。用户注销时，清除用户的会话。本例为了简单起见，在页面打开时直接开启了一个会话。

开启会话的代码如下：

```
01  /* AjaxChkSession.php */
02  <?php
03  session_start();                          //开启一个会话
04  $_SESSION['username'] ="demo";            //设置会话变量 username
05  ?>
```

只要会话变量 username 存在值，则表示会话还处于有效状态，为了使用 AJAX 方式异步地检查会话的有效状态，在页面上使用了如下的检测代码：

```
01  /* AjaxChkSession.php */
02  <script type="text/javascript"
03  src="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js"></script>
04  <!--确保会话处于有效状态下，执行检测代码-->
05  <?php if(!empty($_SESSION['username'])) { ?>
06  <script type="text/javascript">
07  var check_session;                                    //定义全局变量
08  function CheckForSession() {
09          var str="chklogout=true";                     //传递检查字符串
10          jQuery.ajax({
11                  type: "POST",
12                  url: "8_13_chk_session.php",          //发送对服务器端页面的请求
13                  data: str,
```

```
14                      cache: false,
15                      success: function(res){
16                          if(res == "1") {                    //如果返回 1，则提示会话过期
17                          alert('您的会话已经过期了!');
18                          }
19                      }
20              });
21              }
22  check_session = setInterval(CheckForSession, 5000);    //定时的检查是否过期
23  </script>
24  <?php } ?>
25  </head>
```

第 05~24 行表示仅在会话有效的状态下，才执行会话检测，否则对于已经过期的会话，检测就没有太大的意义了。在 JavaScript 代码内部，第 10~21 行向 8_13_chk_session.php 页面定时发送请求，判断该 PHP 页面检测的会话是否存在，如果存在则返回 0 否则返回 1，当返回 1 时，显示一个警告框，提示会话已经过期。8_13_chk_session.php 的代码如下：

```
01  /* 8_13_chk_session.php */
02  <?php
03  session_start();
04  $name = $_SESSION["username"];          //得到用户名
05  if($name == '')
06  {
07          echo "1";                      //会话已经过期
08  } else {
09          echo "0";                      //会话没有过期
10  }
11  ?>
```

PHP 代码通过检测会话变量 username 来判断是否过期，在运行实例时，如果单击了“登出”按钮，会话被注销，页面上会自动弹出警告提示框，如图 8.14 所示。

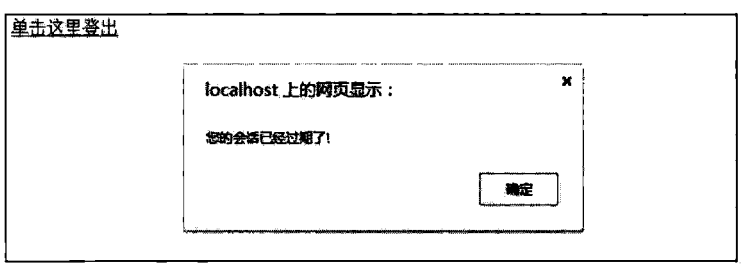

图 8.14　会话过期提示消息

8.13　在 AJAX 异步调用时显示加载指示器

对于长时间地异步调用，最好显示一个加载的指示器，让用户知道当前的异步调用正

在进行，不然用户无法了解当前进度，还有可能因为进一步的操作而出现错误。本例创建一个异步加载 AJAX 页面，在加载期间，会显示如图 8.15 所示的进度指示器。

图 8.15　异步加载进度指示器

首先创建一个简单的 HTML 文件，然后添加如下代码实现异步加载时的进度指示器效果：

```
/* ajaxIndicator.html */
<script   type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
<script type="text/javascript">
    function ajaxindicatorstart(text)                    //开始 AJAX 加载指示器
    {
        //如果不存在加载 div，则动态创建并且显示加载 div
        if($('body').find('#resultLoading').attr('id') != 'resultLoading'){
        jQuery('body').append('<div id="resultLoading" style="display:none">
                <div><img src="images/loading.gif"><div>'+text+'</div>
                </div><div class="bg"></div></div>');
        }
        $('#resultLoading').css({                        //为 div 应用遮罩样式
            'width':'100%',
            'height':'100%',
            'position':'fixed',
            'z-index':'10000000',
            'top':'0',
            'left':'0',
            'right':'0',
            'bottom':'0',
            'margin':'auto'
        });
        $('#resultLoading .bg').css({                    //为 div 中的背景层应用遮罩样式
            'background':'#000000',
            'opacity':'0.7',
            'width':'100%',
```

```
28                  'height':'100%',
29                  'position':'absolute',
30                  'top':'0'
31              });
32              $('#resultLoading>div:first').css({          //为 div 中的文字层应用样式
33                  'width': '250px',
34                  'height':'75px',
35                  'text-align': 'center',
36                  'position': 'fixed',
37                  'top':'0',
38                  'left':'0',
39                  'right':'0',
40                  'bottom':'0',
41                  'margin':'auto',
42                  'font-size':'16px',
43                  'z-index':'10',
44                  'color':'#ffffff'
45              });
46              $('#resultLoading .bg').height('100%');      //指定背景的高度
47              $('#resultLoading').fadeIn(300);             //淡入背景的显示
48              $('body').css('cursor', 'wait');             //更改鼠标光标
49          }
50          function ajaxindicatorstop()                     //定义加载器停止的函数
51          {
52              $('#resultLoading .bg').height('100%');      //背景高度
53              $('#resultLoading').fadeOut(300);            //淡出显示
54              $('body').css('cursor', 'default');          //恢复鼠标光标
55          }
56          function callAjax()                              //异步调用 AJAX
57          {
58              $.ajax({
59                  type: "GET",
60                  url: "8_14.php",                         //指定调用的服务器文件地址
61                  cache: false,
62                  success: function(res){
63                          $('#ajaxcontent').html(res);
64                  }
65              });
66          }
67      $(document).ajaxStart(function () {                  //关联 ajaxStart 事件
68          ajaxindicatorstart('正在加载，请稍候..');          //显示 AJAX 进度指示器
69      }).ajaxStop(function () {
70          ajaxindicatorstop();                             //隐藏 AJAX 进度指示器
71      });
72  </script>
```

```
73  </head>
74  <body><div class="container">
75  <h1>在异步加载 AJAX 内容时，显示加载指示器</h1>
76  <!--异步加载时显示进度指示器-->
77  <a href="javascript:;" class="button" onclick="callAjax();">异步调用 AJAX</a>
78  <!--从服务器端返回的 XML 内容-->
79  <p id="ajaxcontent"></p>
80  </div>
81  </body>
82  </html>
```

由代码可以看出，本例关联了 ajaxStart 事件和 ajaxStop 事件。

注意：无论在何时发送 AJAX 请求，jQuery 都会检查是否存在其他 AJAX 请求。如果不存在，则 jQuery 会触发该 ajaxStart 事件。此时，由.ajaxStart()函数注册的任何函数都会被执行。

第68行调用了自定义的函数ajaxindicatorstart()显示一个遮罩效果的指示器，当ajaxStop事件被触发时，第70行隐藏这个指示器，从而实现了 AJAX 异步加载时的指示器显示效果。

8.14 在 AJAX 异步调用时处理 JSON 数据

JSON 是 JavaScript 对象表示法，即 JavaScript Object Notation，是存储和交换文本信息的语法，有些类似 XML，但是 JSON 格式比 XML 更小、更快、更易解析。JSON 格式常常是 jQuery AJAX 首选的数据格式，它独立于语言，具有自我描述性，更易理解，是轻量级的文本数据交换格式。

注意：JSON 使用 JavaScript 语法来描述数据对象，但是 JSON 是语言独立的，JSON 的解析器以及 JSON 库都可以支持多种不同的编程语言。

本例演示如何在 PHP 中使用 JSON 格式，并且在 jQuery AJAX 中获取和解析 JSON 数据。本例效果如图 8.16 所示。

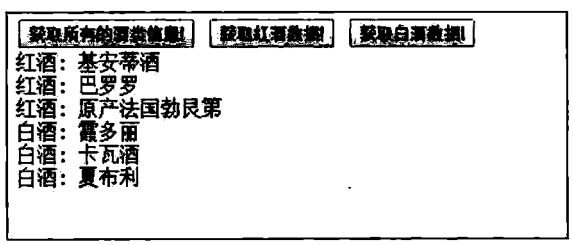

图 8.16 AJAX 和 JSON 数组应用

本例构建了 3 个 HTML 按钮，用于请求酒类信息，代码如下：

```
01  /* ajaxJSON.html */
02  <body>
```

```
<body>
  <form method="post" action="">
      <button value="all" type="submit">获取所有的酒类信息!</button>
      <button value="red" type="submit">获取红酒数据!</button>
      <button value="white" type="submit">获取白酒数据!</button>
  </form>
  <div id="wines">
  <!-- 从服务器端返回的数据将显示在这个 div 中-->
  </div>
</body>
</html>
```

第 04~08 行构建了一个表单，表单上仅包含 3 个提交按钮，表单的 action 没有指定任何页面，将通过 jQuery AJAX 来异步获取服务器端的数据，代码如下：

```
/* ajaxJSON.html */
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"
type="text/javascript"></script>
<title>AJAX JSON 格式使用实例</title>
<script type="text/javascript">
$(document).ready(function(){
  $(':submit').on('click', function() {          //当按钮被单击时触发单击事件
      var button = $(this).val();
      $.ajax({                                   //开始 AJAX 调用
          url: '8_15.php',                       //jQuery 加载服务器端的 PHP 文件
          data: 'button=' + $(this).val(),       //发送按钮的酒类值
          dataType: 'json',                      //选择 JSON 数据类型
          success: function(data)                //data 变量包含了从服务器端返回的数据
          {
              $('#wines').html('');              //清除结果区域的 div 内容

              if (button == 'all') {             //当单击 all 按钮时，显示所有的酒类
                  for (var i in data.red) {
                      $('#wines').append('红酒: ' + data.red[i] + '<br/>');
                  }
                  for (var i in data.white) {
                      $('#wines').append('白酒: ' + data.white[i] + '<br/>');
                  }
              }
              else if (button == 'red') {        //如果单击的是红酒按钮，则显示红酒
                  for (var i in data) {
                      $('#wines').append('红酒: ' + data[i] + '<br/>');
                  }
              }
              else if (button == 'white') {      //如果单击的是白酒按钮，则显示白酒
                  for (var i in data) {
                      $('#wines').append('白酒: ' + data[i] + '<br/>');
                  }
```

```
34                      }
35                  }
36              });
37              return false;                          //确保页面不会因为提交而被刷新
38      });
39  });
40  </script>
41  </head>
```

第 09~38 行调用$.ajax()向 8_15.php 请求数据，并且返回 JSON 类型的数据，第 15~34 行解析返回的 JSON 数组，将 JSON 数组的内容写到网页上。对于返回的 JSON 数据，jQuery 直接就可以循环取出。

服务器端的 PHP 代码使用 json_encode()将 PHP 数组解析成 JSON 数组，通过 print()函数传回到客户端，代码如下：

```
01  /* 8_15.php */
02  <?php
03  $button = $_GET['button'];                  //获取所所单击的按钮的值
04  //读取红酒和白酒数据
05  $red = array('基安蒂酒', '巴罗罗', '原产法国勃艮第');
06  $white = array('霞多丽', '卡瓦酒', '夏布利');
07  $winetable = array(                         //合并红酒和白酒到一个多维数组中
08          "red" => $red,
09          "white" => $white,
10  );
11  if ($button == "red") {
12      print json_encode($red);                //使用 json_encode()转换 JSON 数据
13  }
14  else if ($button == "white") {
15      print json_encode($white);
16  }
17  else {
18      print json_encode($winetable);
19  }
20  ?>
```

由代码可以看出，第 05~10 行构建 PHP 数组，第 11~19 行调用 json_encode()转换 PHP 数组为 JSON 数组，最后输出给客户端。

8.15 解析 XML 数据并加载到 HTML 表格

XML 在网站中的作用日益重要，作为一种数据交换格式，它可以在不同的语言之间共享数据。比如 ERP 产生的 XML 数据，可以被 jQuery 读入，解析后，再呈现到用户界面上。下面是一个稍稍复杂的类似二维表格的 XML 实例，代码如下：

```
01  /* 8_16.xml */
```

```
<?xml version="1.0" encoding="UTF-8"?>
<tabledata>
    <row>
        <column>使用 AJAX 无刷新异步提交表单</column>
        <column>使用 AJAX 无刷新上传图片实例</column>
        <column>使用 AJAX 无刷新验证 PHP 会话是否有效</column>
    </row>
    <row>
        <column>在 AJAX 异步调用时显示加载指示器</column>
        <column>在 AJAX 异步调用时处理 JSON 数据</column>
        <column>解析 XML 数据并加载到 HTML 表格</column>
    </row>
    <row>
        <column>加载网页文件内容并传递服务器端参数</column>
        <column>使用 AJAX 无刷新删除网页记录</column>
        <column>使用 AJAX 异步验证用户名和密码</column>
    </row>
</tabledata>
```

在这个 XML 实例中，节点<row>表示一行数据，下面的子节点<column>表示一列数据，接下来将通过 jQuery AJAX 来将这个 XML 中的数据呈现在一个 HTML 表格内，代码如下：

```
/*loadXMLToTable.html*/
<div id="tableWrapper">
<button id="loadTable">加载 XML 文件</button>
<script type="text/javascript">
$('#loadTable').click(function() {                          //为按钮关联单击事件处理代码
    $.get('8_16.xml', function(data) {                      //使用$.get 获取服务器端数据
        //在 HTML 页面上创建一个新的 HTML 表格
        $('div#tableWrapper').append('<table id="ajaxTable"></table>');
        $(document).ready(function() {                      //页加载就绪时
            //寻环子节点中的行
            $(data).children('tabledata').children('row').each(function() {
                var thisRow = this;
                $('table#ajaxTable').append('<tr></tr>');           //添加 tr 标签
                    $(thisRow).children('column').each(function() { //循环 XML 中的列
                        var thisColumn = $(this).text();            //添加 td 标签
                        $('table#ajaxTable').children('tbody').children('tr').last().
                                append('<td>' + thisColumn + '</td>');
                    });
            });
        });
    });
});
</script>
</body>
```

第 06~21 行向 8_16.xml 发送了一个 get 请求，然后根据返回的 XML 数据进行循环遍

历，第 12~17 行根据 row 和 column 节点来创建一个新的 HTML 表格，比如 row 创建<tr>标签，column 创建<td>标签，从而创建了一个根据 XML 文件呈现的 HTML 表格，如图 8.17 所示。

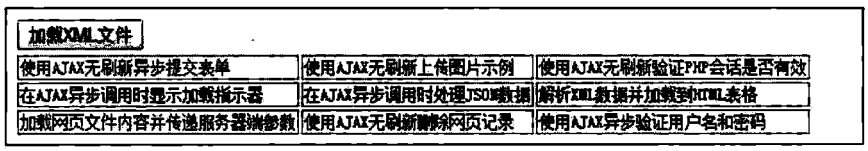

图 8.17　加载并显示 XML 文档到 HTML 表格

8.16　jQuery AJAX 错误的处理方法

由于异步操作涉及到与服务器的数据交换，因此很可能因为某种不明原因产生错误。错误的原因可能是因为服务器端的数据产生了误操作，此时异步调用成功，但是数据有错；另一种常见的是网络错误，导致异步调用本身失败。无论是哪种错误，都需要告之用户，以便用户进行下一步地处理。下面的异步调用代码演示了这两种错误的简单处理方式：

```
01  /*ajaxHandleException.html*/
02  <script type="text/javascript">
03  $.ajax({
04      url: '',
05      type: 'POST',
06      dataType: 'json',
07      data: {},
08      beforeSend: function(){
09          $('#load-msg').show();
10      },
11      success: function(data) {              //异步调用成功
12          $('#load-msg').hide();
13          if(data.errorStatus){              //异步调用成功，但是数据产生了错误
14              alert(data.errorMsg || '产生了数据错误');
15          }else{
16              //如果没有产生错误，则在这里编写代码
17          }
18      },
19      error: function() {                    //由于网络或某些原因导致的异步调用失败
20          $('#load-msg').hide();
21          alert('异步调用产生的错误 d');
22      }
23  });
24  </script>
25  </head>
```

第 11~17 行的$.ajax success 事件中，通过返回数据中的 errorStatus 属性，来判断异步调用成功后，数据是否产生了错误。如果数据产生了错误，第 13~17 行使用 alert()弹出提

示信息。如果异步调用失败，第 19~22 行触发 error 事件处理代码，显示异步调用错误消息。

8.17　在页面级创建全局的 AJAX 监听器以及状态指示器

可以在页面级创建全局性的 AJAX 调用监听器，当 AJAX 开始时，显示一个动画图标并显示一些提示消息来提示异步调用已经开始，当完成异步调用后再隐藏指示器。这些通过页面级别的全局 AJAX 事件来实现，代码如下：

```
01  /* ajaxMonitorIndictor.html*/
02  <title>创建全局性的页面监听器</title>
03  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"
04  type="text/javascript"></script>
05  <script type="text/javascript">
06      $(function(){
07          $("body").ajaxStart(function(){   //在 AJAX 请求发起前执行事件
08              $('#ajax_status_div').html("已经发送了 AJAX 请求");   //更新文本消息
09              $('.activity_indicator').show();                      //显示 AJAX 的活动图像
10          });
11          //全局性的 AJAX 成功完成时执行的事件
12          $("body").ajaxSuccess(function(event,xmlHttp,options){
13              var status = xmlHttp.statusText;                      //获取状态文本消息
14              var url = options.url;                                //得到 URL
15              var data = options.data;                              //得到异步数据
16              $('.activity_indicator').hide();                      //隐藏 AJAX 活动图像
17          $('#ajax_status_div').html("URL : "+url+"  <br/>  Status : "+status);//更新文本消息
18          });
19      });
20  </script>
21  </head>
```

为了创建全局性的 AJAX 监听器，代码第 07 行为 body 关联了 ajaxStart 事件和 ajaxSuccess 事件，通过这两个事件，可以全局性地监控页面级别的 AJAX 事件，从而实现一些特殊的提示效果。

8.18　级联 AJAX 数据异步加载

级联加载，是指有一个下拉列表框，当选中不同的列表项时，就会通过 AJAX 来异步加载服务器端的数据，并将数据显示到另一个控件或者另外一个下拉列表框内部，这就是级联选择。要实现这样的效果并不复杂，HTML 代码如下：

```
01  /* cascadeAjaxLoad.html*/
02  <body>
03   <!-- 行为 div ，需要有一个 URL 指示要加载的服务器路径，
```

```
04                    load-data-container 指示要将数据加载到的目标 div 的 id 值-->
05  <div id="parentDivId" class="geo" url="/loadGeoDistrict" load-data-container="#childDivId" >
06          <!--select 一般用来给出 URL 的查询字符串参数-->
07          <select>
08            <option value="1" selected="selected">1</option>
09            <option value="2">2</option>
10            <option value="3">3</option>
11            <option value="4">4</option>
12        </select>
13    </div>
14    <!-- 下面是要级联加载的目标 div -->
15    <div id ="childDivId">
16    </div>
17  </body>
```

第 05 行包含了一个 class 为 geo 的 div 元素，其 url 和 load-data-container 都是附加的，用来提供特定的识别值，div 内部的 select 控件用来为 div 中的 url 属性提供参数值，以便于级联加载。完成 AJAX 级联加载的 JavaScript 代码如下：

```
01  /* cascadeAjaxLoad.html*/
02  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"
03  type="text/javascript"></script>
04  <script type="text/javascript">
05  $("document").ready(function(){
06      $(".geo").on("change", function() {    //当 div 内部的 select 的 change 触发时执行代码
07            var currElement =  $(this);      //当前的 div 元素
08            //当前的 div 元素指向的 load-data-container 元素
09            var loadDataContainer = $(currElement.attr("load-data-container"));
10            loadDataContainer.empty();                  //清除目标容器元素值
11            loadDataContainer.trigger("change");        //需要一个 trigger 来清除所有内容
12            var url = currElement.attr("url");          //得到 URL
13            //得到 div 中的第 1 个元素，即第 1 个 select 元素
14            var firstSelectElement = currElement.find("select:first");
15            //得到选中的元素值
16            var geoId = $(firstSelectElement).find(":selected").attr("value");
17            if(geoId) {     //如果元素值存在，则调用 loaddata 异步加载
18               loadData(url, "get", geoId).done(function(data) {
19                    loadDataContainer.html(data);     //加载的数据显示到子元素中
20               });
21            }
22      });
23      /**
24       *调用$.ajax()异步加载数据，并返回成功加载的数据
25       * */
26      function loadData(url, type, geoId) {
27          return $.ajax(
28              {
29                  url : url,                  //加载路径
```

```
30                  type : type,            //类型
31                  data : {id : geoId}     //查询字符串参数
32              }
33          );
34      }
35  });
36  </script>
```

第 06 行为 div 关联了 change 事件，这样当 select 改变了选择值时，也会冒泡到这个事件。然后在第 12 行中检测 load-data-container 和 url 属性，第 14~20 行获取 select 控件中被选中的值，以此作为参数，调用 loadData()自定义函数，这个函数将调用$.ajax()向指定的 url 发送数据请求，传入 id 值进行查询。服务器返回数据后，将服务器端的数据加载到 load-data-container 指定的 id 元素中。

8.19　取消 AJAX 异步请求

在异步提交过程中，有时可能要取消异步提交，$.ajax()函数返回它创建的 XMLHttpRequest 对象。可以通过调用这个对象的 abort()函数来取消异步请求。

注意：通常 jQuery 只在内部处理并创建 XMLHttpRequest 对象，但用户也可以通过 xhr 选项来传递一个自己创建的 xhr 对象。返回的对象通常已经被丢弃了，但依然提供一个底层接口来观察和操控请求。比如说，调用对象上的.abort()可以在请求完成前挂起请求。

下面的代码演示了如何取消异步请求：

```
01  /* cascadeAjaxLoad.html*/
02  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"
03  type="text/javascript"></script>
04  <script type="text/javascript">
05   var req = $.ajax({
06       type: "POST",
07       url: "请求的地址",
08       data: "id=1",
09       success: function(){
10       //当加载成功时，执行事件处理代码
11       }
12       });
13       //可以调用 abort()函数取消异步加载
14       req.abort()
15  </script>
16  </head>
```

从代码可以看出，第 05 行将$.ajax()的结果保存到 req 变量中，然后通过调用 req.abort()就中断了 AJAX 的异步请求。

第 9 章　jQuery 常用算法

在使用 jQuery 的过程中，除了掌握 jQuery 的选择器与 UI 应用等技巧外，熟悉一些常用的 jQuery 算法以及编程小技巧更有助于加速功能的实现。

本章主要涉及的知识点有：

- jQuery 的选择器算法。
- 在 jQuery 中对列表排序。
- 查找或移除数组中特定的值。
- 随机选择页面元素。
- 序列化表单到 JSON 数据。

9.1　jQuery 遍历算法

使用 jQuery 的$(selector).each()函数可以循环遍历选中的子元素，而 jQuery 的$.each()函数与之不同，$.each()函数可以遍历任何集合，比如对象或者数组，它接收要遍历的集合以及一个回调函数，回调函数每次传递一个数组的下标和这个下标所对应的数组的值。

本例代码如下：

```
/* jQueryEach.html */
<script src="jquery/jquery-1.9.1.js"></script>
</head>
<body>
  <div id="arr"></div>
  <div id="obj"></div>
<script type="text/javascript">
    var arr = [ "张三", "李四", "王五", "赵六", "贺八" ];          //定义一个数组
    var obj = { name:"张三", age:22, gender:"男", country:"中国", city:"上海" }; //定义一个对象
    $.each(arr, function() {                                    //对数组进行循环
      $("#arr").append("数组元素是" + this + "<br/>"); //this 指向为数组的值，如 one, two
       return (this != "赵六");                                 //如果 this=赵六则退出遍历
   });
    $.each(obj, function(i, val) {      //循环对象，i 指向键，val 指定值
      $("#obj").append(i+ " – " + val+"<br/>");

    });
```

```
18  </script>
19  </body>
20  </html>
```

第 08~09 行定义了一个数组和一个对象，第 10~17 行使用$.each()函数来遍历其中的元素，对于数组的遍历来说，回调函数没有任何参数，this 表示数组的元素。对于对象类型的遍历来说，回调函数具有 i 和 val 参数，其中 i 表示对象类型的属性名称，val 表示对象类型的属性值，本例效果如下：

```
数组元素是张三
数组元素是李四
数组元素是王五
数组元素是赵六
name    张三
age – 22
gender    男
country    中国
city    上海
```

可以看到，jQuery 的$.each()已经正确地循环出了对象和数组元素。

注意：对于$.each()，如果要中断或继续循环，可以使用 return 语句，返回'false'将停止循环，就像在普通的循环中使用“break”；返回“true”跳至下一个循环，就像在普通的循环中使用“continue”。

9.2　jQuery 祖先算法

因为 DOM 具有层次结构，因而 jQuery 也提供了一系列的节点遍历算法，以及节点的父节点、祖先节点的取值函数。常用的有如下几个函数。

- parent()函数：返回被选元素的直接父元素，该函数只会向上一级的 DOM 树进行遍历。
- parents()函数：返回被选元素的所有祖先元素，它一路向上直到文档的根元素（<html>）。
- parentsUntil()函数：返回介于两个给定元素之间的所有祖先元素。

在实际编程时，这 3 个函数可以实现层次调用，比如 parent().parent()就可以取到父级的父级元素。下面的实例演示了如何使用这几个函数来选择父级和祖先级的节点：

```
01  /* jQueryAncestors.html */
02  <body>
03      <div id="div1">                    <!--根节点-->
04        <div id="div2">                  <!--包含子节点的父节点-->
05            <p></p>
06        </div>
07        <div id="div3" class="a">        <!--包含子节点的父节点-->
08            <p></p>
```

```
09          </div>
10          <div id="div4">                    <!--包含子节点的父节点-->
11              <p></p>
12          </div>
13      </div>
14      <div id="result"></div>
15      <script type="text/javascript">
16          var ctl=$("p").parent();        //取得 div2、div3、div4 的集合
17          $("#result").append("p 元素的父元素列表：");
18          ctl.each(function(index,domEle){
19              $("#result").append(domEle.id+" ");
20          });
21          $("#result").append("<br/>类为 a 的元素的父元素：");
22          ctl=$('p').parent('.a');        //取得 div3 单一元素
23          ctl.each(function(index,domEle){
24              $("#result").append(domEle.id+" ");
25          });
26          ctl=$('p').parent().parent(); //取得 div1，父节点的父节点即为 div1
27          $("#result").append("<br/>p 元素的祖父元素：");
28          ctl.each(function(index,domEle){
29              $("#result").append(domEle.id+" ");
30          });
31          ctl=$('p').parents();           //取得 div1、div2、div3、div4
32          $("#result").append("<br/>p 元素的父元素列表：");
33          ctl.each(function(index,domEle){
34              $("#result").append(domEle.id+" ");
35          });
36          ctl=$('p').parents('.a');       //取得 div3
37          $("#result").append("<br/>p 元素的父元素列表中 class 为 a 的元素：");
38          ctl.each(function(index,domEle){
39              $("#result").append(domEle.id+" ");
40          });
41          ctl=$("p").parentsUntil("p");  //取得两个 p 元素之间的所有祖先元素
42          $("#result").append("<br/>2 个 p 元素之间的所有祖先元素：");
43          ctl.each(function(index,domEle){
44              $("#result").append(domEle.id+" ");
45          });
46      </script>
47  </body>
```

第 03~13 行构建了由几个 div 和 p 元素组成的简单层次结构。第 16~45 行依次调用了 parents()、parent()以及 parentsUntil()等函数来获取相关的祖先元素，运行效果如下：

p 元素的父元素列表：div2 div3 div4
类为 a 的元素的父元素：div3
p 元素的祖父元素：div1

```
p 元素的父元素列表：div4 div3 div2 div1
p 元素的父元素列表中 class 为 a 的元素：div3
2 个 p 元素之间的所有祖先元素：div4 div3 div2 div1
```

9.3 jQuery 后代算法

使用jQuery的后代算法，可以取出某个DOM节点的子元素、孙元素或曾孙元素。jQuery不仅可以向上遍历DOM树，还可以向下遍历DOM树，以查找元素的后代。

jQuery中可以使用如下两个函数来实现后代查找。

- children()函数：返回被选元素的所有直接子元素，该函数只会向下一级DOM树进行遍历。
- find()函数：返回被选元素的后代元素，一路向下直到最后一个后代。

注意：实际工作中通常称children()为获取子节点，find()函数为获取后代节点。

本例将使用后代算法，获取当前某元素的子元素，以及叶子节点的元素，代码如下：

```
/* jQueryChildren.html */
<body>
    <div id="div1">                    <!--根节点-->
     <div id="div2">                   <!--包含子节点的父节点-->
        <p id="p1"></p>
     </div>
     <div id="div3" class="a">         <!--包含子节点的父节点-->
        <p id="p2"></p>
     </div>
     <div id="div4">                   <!--包含子节点的父节点-->
        <p id="p3"></p>
     </div>
    </div>
    <div id="result"></div>
    <script type="text/javascript">
        var ctl=$("#div1").children();      //取得当前元素子一级子元素的集合
        $("#result").append("div1 元素的子元素列表：");
        ctl.each(function(index,domEle){
            $("#result").append(domEle.id+" ");
        });
        $("#result").append("<br/>div1 中子元素 id 为 div3 的元素列表：");
        var ctl=$("#div1").children(".a"); //取得当前元素的子元素中，class 为 a 的子元素
        ctl.each(function(index,domEle){
            $("#result").append(domEle.id+" ");
        });
        var ctl=$("#div1").find("*");      //取得当前元素的所有子元素，直到最叶级元素
        $("#result").append("<br/>div1 元素的子元素列表：");
```

```
28              ctl.each(function(index,domEle){
29                  $("#result").append(domEle.id+" ");
30              });
31              var ctl=$("#div1").find("p");       //取得当前元素中，所有子元素中的 p 元素
32              $("#result").append("<br/>div1 元素的 p 子元素列表：");
33              ctl.each(function(index,domEle){
34                  $("#result").append(domEle.id+" ");
35              });
36          </script>
37      </body>
```

本例分别演示了使用 children()和 find()查找 div1 元素的子元素，在 HTML 中 div1 元素是根元素，实例的输出结果如下：

```
div1 元素的子元素列表：div2 div3 div4
div1 中子元素 id 为 div3 的元素列表：div3
div1 元素的子元素列表：div2 p1 div3 p2 div4 p3
div1 元素的 p 子元素列表：p1 p2 p3
```

从结果可以看出，使用 children()只取出了一级的子元素，而使用 find()可以取出当前节点下的所有子节点、子节点的子节点等等，直到取到最叶级的节点。

9.4 jQuery 同胞算法

jQuery 不仅能获取上一级或下一级的节点，还能获取同级的节点，比如兄弟节点、下一个节点或上一个节点等等。获取同级节点，jQuery 提供了很多函数，如 siblings()、next()、nextAll()、nextUntil()、prev()、prevAll()、prevUntil()等。本例演示了如何使用 jQuery 的同胞算法来获取相关的节点：

```
01  /* jQueryFellow.html */
02  <body>
03      <div id="div1">                    <!--根节点-->
04       <div id="div2">                   <!--包含子节点的父节点-->
05          <p id="p1"></p>
06       </div>
07       <div id="div3" class="a">         <!--包含子节点的父节点-->
08          <p id="p2"></p>
09       </div>
10       <div id="div4">                   <!--包含子节点的父节点-->
11          <p id="p3"></p>
12       </div>
13      </div>
14      <div id="result"></div>
15      <script type="text/javascript">
16          var ctl=$("#div2").siblings();      //div2 的兄弟元素
17          //$("#div2").siblings("p");         //可以过滤选中 div2 中的 p 元素
```

```
            $("#result").append("div2 的兄弟元素：");
            ctl.each(function(index,domEle){
                $("#result").append(domEle.id+" ");
            });
            $("#result").append("<br/>div2 元素的下一个元素：");
            ctl=$('#div2').next();              //取得 div2 的下一个同胞元素，仅返回一个元素
            ctl.each(function(index,domEle){
                $("#result").append(domEle.id+" ");
            });
            ctl=$('#div2').nextAll();           //取得 div2 后面的所有的同胞元素
            $("#result").append("<br/>div2 后面的所有同级元素：");
            ctl.each(function(index,domEle){
                $("#result").append(domEle.id+" ");
            });
            ctl=$('#div2').nextUntil('#div4');  //div2 和 div4 之间的同级元素
            $("#result").append("<br/>div2 和 div4 之间的同级元素：");
            ctl.each(function(index,domEle){
                $("#result").append(domEle.id+" ");
            });

            $("#result").append("<br/>div3 元素的上一个同级元素：");
            ctl=$('#div3').prev();              //取得 div3 的上一个同胞元素，仅返回一个元素
            ctl.each(function(index,domEle){
                $("#result").append(domEle.id+" ");
            });
            ctl=$('#div4').prevAll();           //取得 div4 上面的所有同胞元素，返回多个元素
            $("#result").append("<br/>div4 之上的所有同级元素：");
            ctl.each(function(index,domEle){
                $("#result").append(domEle.id+" ");
            });
            ctl=$('#div4').prevUntil('#div2');  //div2 和 div4 之间的同级元素
            $("#result").append("<br/>div4 和 div2 之间的同级元素：");
            ctl.each(function(index,domEle){
                $("#result").append(domEle.id+" ");
            });
        </script>
    </body>
```

本例使用了 jQuery 提供的获取同胞元素的相关函数，可以通过结果比较这些函数的不同之处，结果如下：

```
div2 的兄弟元素：div3 div4
div2 元素的下一个元素：div3
div2 后面的所有同级元素：div3 div4
div2 和 div4 之间的同级元素：div3
div3 元素的上一个同级元素：div2
```

```
div4 之上的所有同级元素：div3 div2
div4 和 div2 之间的同级元素：div3
```

提示：nextAll()或 prevAll()获取所有向下或向上的同级节点，而 next()或 prev()只获取一个子节点，siblings()则获取所有相关的兄弟元素。

9.5 jQuery 过滤算法

过滤是指从一组元素中有条件地筛选出指定的元素，jQuery 为过滤算法提供了 3 个基本函数，分别是 first()、last()和 eq()。另外还可以使用 filter()和 not()选择匹配或不匹配某项指定标准的元素。本例演示了如何使用 jQuery 的过滤算法来过滤一个 DOM 树结构：

```
/* jQueryFilter.html */
<body>
    <div id="div1">                    <!--根节点-->
      <div id="div2">                  <!--包含子节点的父节点-->
        <p id="p1"></p>
      </div>
      <div id="div3" class="a">        <!--包含子节点的父节点-->
        <p id="p2"></p>
      </div>
      <div id="div4">                  <!--包含子节点的父节点-->
        <p id="p3"></p>
      </div>
    </div>
    <div id="result"></div>
    <script type="text/javascript">
        var ctl=$("#div1 div").last();                  //div1 中的最后一个 div 元素
        $("#result").append("最后一个 div 元素：");
        ctl.each(function(index,domEle){
            $("#result").append(domEle.id+" ");
        });
        var ctl=$("#div1 div").first();                 //div1 中的第一个 div 元素
        $("#result").append("<br/>第一个 div 元素：");
        ctl.each(function(index,domEle){
            $("#result").append(domEle.id+" ");
        });
        var ctl=$("#div1 div").eq(1);                   //索引等于 1 的 div 元素
        $("#result").append("<br/>索引等于 1 的 div 元素：");
        ctl.each(function(index,domEle){
            $("#result").append(domEle.id+" ");
        });
        var ctl=$("#div1 div").filter(".a");            //class 为 a 的 div 元素
        $("#result").append("<br/>具有 class 为 a 的 div 元素：");
```

```
33              ctl.each(function(index,domEle){
34                  $("#result").append(domEle.id+" ");
35              });
36      </script>
37  </body>
```

本例使用 first()获取第 1 个元素、last()获取最后 1 个元素、eq()获取指定索引位置的元素、filter()过滤 class 为 a 的元素，从而得到如下结果：

```
最后一个 div 元素：div4
第一个 div 元素：div2
索引等于 1 的 div 元素：div3
具有 class 为 a 的 div 元素：div3
```

9.6　对一个列表进行排序

进行网站开发时难免会遇到需要排序的场合，jQuery 提供了很多插件可以实现列表排序，但有时需要自己来完成一些排序工作。JavaScript 的数组提供了 sort()函数，可以借用这个函数，来对 jQuery 选中的元素进行排序。

本例创建一个简单的 ul 元素，然后使用 jQuery 对这个 ul 元素中的列表项进行排序，以产生排序后的结果。代码如下：

```
01  /* listSort.html */
02  <script src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04      $(document).ready(function(){
05          $("#lnk").click(function(){
06              var mylist = $('ul');                    //获取 ul 对象
07          var listitems = mylist.children('li').get(); //获取 ul 中的所有 li 元素,使用 get 转换为数组
08              listitems.sort(function(a, b) {      //调用 Array 对象的 sort()函数进行排序
09                  var compA = $(a).text().toUpperCase();//转换字符值为大写
10                  var compB = $(b).text().toUpperCase();
11                  //比较 2 个字符串，如果大于则返回 1，等于返回 0，于小返回-1
12                  return (compA < compB) ? -1 : (compA > compB) ? 1 : 0;
13              })
14              //通过循环 js 数组，在回调函数中将排序后的值添加到 mylist 中
15              $.each(listitems, function(idx, itm) { mylist.append(itm); });
16          });
17      });
18  </script>
19  </head>
20
21  <body>
22  <!--用来排序的无序列表元素-->
23  <ul id="ul1">
```

```
24      <li>Apple</li>
25      <li>Orange</li>
26      <li>Bananas</li>
27      <li>Peach</li>
28    </ul>
29    <!--单击按钮来进行排序-->
30    <a href="#" id="lnk">单击这里排序</a>
31    </ul>
32    </body>
33    </html>
```

第 07 行获取 ul 中的所有 li 元素，然后通过 get()函数将其转换为 DOM 元素数组，此时就可以使用 JavaScript 本身的 sort()函数来对元素进行排序了。第 08~13 行的回调函数指定了排序的具体细节，然后将排序后的数据重新添加到 ul 元素中，这里使用了$.each()进行循环添加。

9.7 实现 JSON 对象数组的排序

在进行 AJAX 异步调用时，通常会从服务器端返回一个 JSON 数组，可以使用 jQuery 对返回的数组进行排序，从而呈现出排序后的结果。本例构建了一个人事列表，这个表的数据来自一个 JSON 数组，通过对这个人事表中的员工工号和员工姓名进行排序，最终产生如图 9.1 的效果。

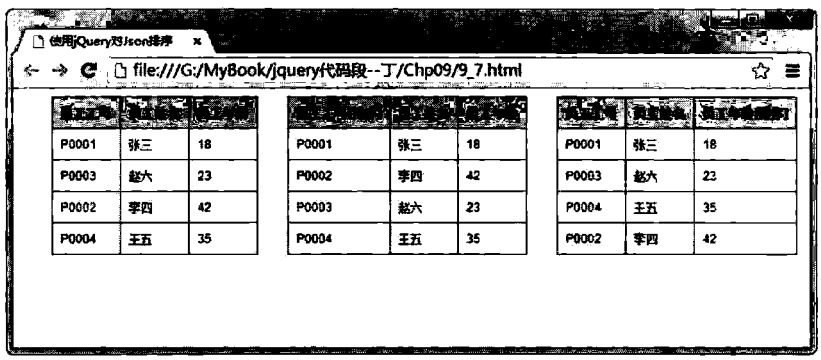

图 9.1 排序的结果

本例初始时，HTML 页面上仅仅包含 3 个带标题栏的 HTML 表格，代码如下：

```
01  /* jsonArraySort.html */
02  <div id="tbllayout">
03    <!--未经排序的 HTML 表格-->
04    <table class="gridtable" id='ia' border="1">
05    <tr>
06      <th>员工工号</th><th>员工姓名</th><th>员工年龄</th>
07    </tr>
```

```
</table>
<!--按员工工号排序的 HTML 表格-->
<table class="gridtable" id='ib' border="1">
 <tr>
   <th>员工工号[排序]</th><th>员工姓名</th><th>员工年龄</th>
 </tr>
</table>
<!--按员工年龄排序的 HTML 表格-->
<table class="gridtable" id='ic' border="1">
  <tr>
   <th>员工工号</th><th>员工姓名</th><th>员工年龄[排序]</th>
  </tr>
</table>
</div>
```

在 jQuery 代码中，构造了一个简单的 JSON 数组，然后使用数组的 sort()函数，对数组内容按选择的字段进行排序，最后显示到 HTML 表格中：

```
/* jsonArraySort.html */
<script type="text/javascript">
    $(function(){
        //构造一个 JSON 数组，用于包含员工信息
        var peoples =[{'card_id':'P0001','p_name':'张三','p_age':18},
            {'card_id':'P0003','p_name':'赵六','p_age':23},
            {'card_id':'P0002','p_name':'李四','p_age':42},
            {'card_id':'P0004','p_name':'王五','p_age':35}];
        $.each(peoples, function(index, value)                    //遍历 JSON 数组
        {    //将数组中的 JSON 对象显示到表格中
            $('#ia').append('<tr><td>' + value.card_id +
                '</td><td>' + value.p_name +
                '</td><td>' + value.p_age + '</td></tr>');
        });
        var card_id_peoples = peoples.sort(function(a, b) {   //按 card_id 字段进行排序
            if(a.card_id < b.card_id) return -1;
            if(a.card_id > b.card_id) return 1;
            return 0; } );
        $.each(card_id_peoples, function(index, value){       //显示到 HTML 表格中
            $('#ib').append('<tr><td>' + value.card_id +
                '</td><td>' + value.p_name +
                '</td><td>' + value.p_age + '</td></tr>')
        });
        var p_age_peoples = peoples.sort(function(a, b){      //按年龄进行排序
            return (a.p_age - b.p_age);
        });
        $.each(p_age_peoples,function(index, value){          //显示到 HTML 表格中
            $('#ic').append('<tr><td>' + value.card_id +
```

```
                '</td><td>' + value.p_name +
                '</td><td>' + value.p_age + '</td></tr>')
        });});
    </script>
```

第 05~08 行构建了一个简单的 JSON 数组，这个 JSON 数组由 4 条员工信息组成。接下来对这 4 个 JSON 数组分别按员工工号与员工年龄进行排序。排序之后，使用$.each()函数将之添加到 HTML 表格中。

9.8 将 12343778 转成 12.343.778 的形式

正则表达式可以按照一定的模式来修改字符串，使用正则表达式可以实现“将 12343778 转成 12.343.778”的效果。本例代码如下：

```
/* dataFormatConvert.html */
<script src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    var delimiter = '.';
    //使用正则表达式来替换标准字符串
    var fmtdata=$('#result').html().toString().replace
    (new RegExp("(^\\d{"+($('#result').html().toString().length%3||-1)+"}
        )(?=\\d{3})"),"$1"+delimiter).replace(/(\d{3})(?=\d)/g,"$1"+ delimiter);
    $("#fmtresult").html(fmtdata);
});
</script>
</head>
<body>
  <div id="result">12343778</div>
  <div id="fmtresult"></div>
</body>
</html>
```

第 08~09 行使用了 RegExp 将原来 div 中的 HTML 字符串替换为使用正则表达式替换的字符串，其中 delimiter 指定了分隔符，可以替换这个分隔符，也可以替换 div 中的数字，从而得到自己想要的效果。

9.9 模拟抽奖程序

很多公司一旦具有大型活动或节日时，都会创建一些抽奖程序来进行抽奖。抽奖的原理就是从一系列候选人中随机地选出一名幸运者，这里面有很多随机因素，因此实现这个抽奖程序的关键是随机函数的应用。本例演示一个简单的抽奖程序，使用了 jQuery 中的随

机函数，本例效果如图 9.2 所示。当单击“开始”按钮后，字幕开始不停地切换人员名称，单击“停止”按钮时，会选中抽奖人。

图 9.2　随机抽奖

本例代码如下：

```
01  /* getAwards.html*/
02  <script src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04    //构造抽奖的姓名，以逗号为分隔
05    var alldata="王大富,张三甲,李仁杰,刘三张,张二李,李四丁,丁四吴,吴若刘,刘大金,王小富"
06    var alldataarr=alldata.split(",")          //调用 split()将字符串分隔为数组
07    var num=alldataarr.length-1                //设置随机种子
08    var timer                                  //保存 setInterval()的返回值
09    function getramdom(min,max){               //创建一个随机函数，用于产生数组长度的随机数
10        return parseInt(Math.random()*(max-min+1));
11    }
12    function change(){     //构建一个定时执行的函数，获取随机数显示到 div 中
13        $("#award").html(alldataarr[getramdom(0,num)]);
14    }
15    function  start(){      //当单击“开始”按钮时，定时执行随机函数
16        clearInterval(timer);
17        timer=setInterval('change()',200);    //开始进行抽奖，指定变更时间为 200
18    }
19    function ok(){          //当单击“停止”按钮时，停止定时函数
20        clearInterval(timer);
21    }
22  </script>
23  </head>
24  <body>
25  简单的随机抽奖实例：<div id="award"></div>
26  <button onclick="start()" accesskey="s">开始(<U>S</U>)</button>
27  <button onclick="ok()"  accesskey="o">停止(<U>O</U>)</button>
28  </body>
29  </html>
```

第 05 行定义了一个字符串 alldata，包含随机抽奖的人员名称。通过调用字符串对象的

split()函数，以逗号分隔来构建一个字符串数组。第 09 行的 getramdom()函数将构建数组元素的个数，来产生随机数，这个随机数将作为下标，随机地选取中奖人员。当用户单击“开始”按钮时，setInterval()函数被调用来随机地选取中奖人员名称。单击“停止”按钮时，将调用 clearInterval()函数来清除定时器。

9.10 实现冒泡算法

除了使用 Array 对象进行排序之外，在编写 jQuery 代码时，也可以编写自己的排序算法。冒泡排序是排序算法中一种比较简单的从小到大的排序函数，冒泡排序重复地对要排序的元素进行循环比较，一次比较两个元素，按从小到大进行排列。本例效果如图 9.3 所示。

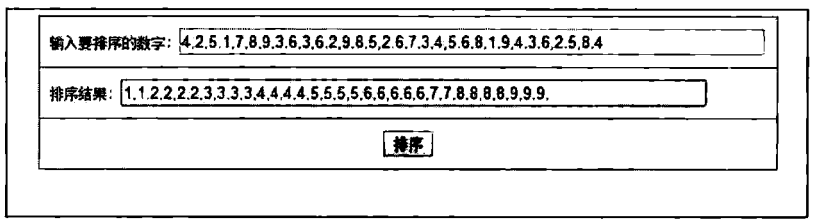

图 9.3 冒泡排序的结果

注意：冒泡排序就是把小的元素往前调或者把大的元素往后调。比较是在相邻的两个元素之间进行，交换也发生在这两个元素之间。

本例代码如下：

```
01  /* bubbleSort.html*/
02  <script src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04  $(document).ready(function(){
05      $("#btnsort").click(function(){
06          doSort();          //单击按钮时，调用 doSort 进行冒泡排序
07      });
08  });
09
10  function doSort() {
11      nanExists = false;                                    //是否具有无效数字的标记
12      inputString = $("input[name='numbers']").val();       //要排序的内容
13      inputNumbers = inputString.split(",");                //以逗号为分隔符构建数组
14      for (var i = 0; i < inputNumbers.length; i++) {       //循环数组，
15        inputNumbers[i] = parseInt(inputNumbers[i], 10);    //判断是否存在非法数字
16        if (isNaN(inputNumbers[i])) {                       //如果存在非法数字
17          nanExists = true;                                 //设置 nanExists 为 true
18          break;                                            //中断循环
19        }
20    }
```

```
21        //调用 bubbleSort()函数，对数组进行冒泡排序
22        inputNumbers = bubbleSort(inputNumbers, 0, inputNumbers.length - 1);
23        if (nanExists)                                    //如果存在非法数字，则输出提示
24          $("input[name='answers']").val("无效的输入，必须是以逗号分隔的数字才能进行排序");
25        else
26          //如果不存在非法数字，则显示排序后的结果
27          $("input[name='answers']").val(resultString(inputNumbers, 0));
28    }
29
30    function resultString(inputArray, num) {          //用来将数组转换为逗号分隔的字符串
31        if ((inputArray.length  - 1) >= num)
32        return (inputArray[num] + "," + resultString(inputArray,(num + 1)));
33        else return "";
34    }
35
36    function bubbleSort(inputArray, start, rest) {    //冒泡排序算法的实现
37     for (var i = rest-1; i>=start; i--) {            //由两个循环组成，不断地循环未排序字符串
38       for (var j=start; j<=i; j++) {
39           if (inputArray[j+1]<inputArray[j]) {
40               var tempValue=inputArray[j];
41               inputArray[j]=inputArray[j+1];        //交换数据的位置，从小到大排列
42               inputArray[j+1]=tempValue;
43           }
44       }
45     }
46       return inputArray;                             //返回排序后的数组
47    }
48    </script>
49    </head>
50    <body>
51    <form>
52    <!--构建排序实例使用到的 HTML 表单-->
53    <table border=1   class="gridtable" >
54    <td>输入要排序的数字: <input type=text name=numbers size=75
55    value="4,2,5,1,7,8,9,3,6,3,6,2,9,8,5,2,6,7,3,4,5,6,8,1,9,4,3,6,2,5,8,4"></td>
56    <tr><td>排序结果: <input type=text name=answers size=75></td>
57    </tr><tr><td colspan=2 align=center>
58    <input type=button value="排序" id="btnsort"></td>
59    </tr></table></form></body></html>
```

整段代码实现的核心是第36~47行的bubbleSort()，它使用了经典的冒泡排序算法，在嵌套循环中不停地交换元素在数组中的位置，以实现从小到大的排序效果。第10~28行的doSort()获取HTML表单上的数字，将其分割成数组，然后调用bubbleSort()进行冒泡排序，并将结果又合并成逗号分隔的字符串，显示到HTML的表单中。

9.11 查询数组中特定数值

如果存在一个数组，要想获取数组中符合指定条件的值，可以使用$.each()对数组进行循环，从而取出想要的值。不过 jQuery 提供了功能强大且好用的 grep()函数，专门用来从数组中筛选符合条件的数组元素。

本例演示如何使用 grep()函数来提取数组中特定的元素值：

```
/* findArrayValues.html*/
<title>从数组中筛选元素</title>
<style type="text/css">
  #result{
      font-size:12pt;
  }
</style>
<script src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript">
 $(function(){
  var alldata="王大富,张三甲,李仁杰,刘三张,张二李,李四丁,丁四吴,吴若刘,刘大金,王小富"
  var alldataarr=alldata.split(",")          //调用 split()将字符串分隔为数组
  $("#result").html("员工姓名中包含\"李\"字的有：");
  //grep 接收要过滤的数组以及一个回调函数
  $.grep(alldataarr,function(val,key){       //回调函数的 val 表示元素值，key 表示元素下标
    if(val.indexOf("李")!==-1){
      $("#result").append(val+",");
    }
  });
  $("#result").append("<br/>下标小于 2 的员工是：");
  var datareskey=$.grep(alldataarr,function(val,key){
      return key>2;          //条件是下标大于 2
  },true);                   //因为使用了第 3 个参数 true，表示反向结果，下标的值小于 2
  $.each(datareskey,function(key,val){   //循环输出结果值
      $("#result").append(val+",");
  });
 });
</script>
```

第 15 行的第 1 个 grep()提取数组元素中包含“李”字的人员名称，第 21 行的第 2 个 grep，指定反向地提取条件数据，因此 key>2 实际上取的是 key<2 的数据，查询结果如下：

```
员工姓名中包含"李"字的有：李仁杰,张二李,李四丁,
下标小于 2 的员工是：王大富,张三甲,李仁杰,
```

9.12 从数组中移除特定数值

JavaScript的Array对象提供了splice()，可以用来删除元素，也可以向数组添加新元素。使用splice()需要知道元素的索引，但有时可能无法获取元素的索引，此时可以利用jQuery的inArray()函数查找到特定元素值的索引，再进行删除。

本例演示了如何使用jQuery和Array对象来删除特定的数组元素，代码如下：

```
01  /* removeArrayValues.html*/
02  <script src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04   $(function(){
05    var alldata="王大富,张三甲,李仁杰,刘三张,张二李,李四丁,丁四吴,吴若刘,刘大金,王小富"
06    var alldataarr=alldata.split(",")          //调用 split()将字符串分隔为数组
07    var idx=[];
08    //grep 接收要过滤的数组以及一个回调函数
09    $.grep(alldataarr,function(val,key){       //回调函数的 val 表示元素值，key 表示元素下标
10       if(val.indexOf("李")!==-1){
11          idx.push(val);
12       }
13    });
14    $.each(idx,function(key,val){
15          alldataarr.splice($.inArray(val,alldataarr),1);       //依次移除姓李的员工记录
16    });
17    alldataarr.splice($.inArray('张三甲',alldataarr),1);       //删除张三甲
18    $.each(alldataarr,function(key,val){                       //循环输出结果值
19          $("#result").append(val+",");
20    });
21   });
22  </script>
```

第09行使用$.grep()提取数组中包含“李”的人员姓名，将其添加到idx数组中。第14行循环idx数组，使用$.inArray()获取数组值的下标，然后调用splice()将其从数组中移除。第17行在第2个splice()语句中，直接删除了“张三甲”元素，循环输出结果如下：

```
王大富,刘三张,丁四吴,吴若刘,刘大金,王小富,
```

9.13 根据指定正则表达式识别超链接

使用jQuery，可以很方便地查询出网站上的链接，只需要使用$("a")即可取出页面上的所有链接。但实际开发中经常需要对特定的链接进行过滤或为特定的链接添加额外的功能，比如为特定的链接应用不同的CSS。本例演示如何为链接应用正则表达式匹配，当单击非本地链接时，让链接在新窗口中打开。代码如下：

```
01  /* findUrlByReg.html*/
```

```
<script src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript">
  $(document).ready(function(){
        var pattern = /^\http/;                          //一个简单的 http 前缀正则表达式
       $('a').each(function () {                         //循环链接
        var href = $(this).attr('href');                 //获取 href
       if (href != undefined && href.match(pattern)) {  //匹配正则表达式
          $("#result").append(href+"<br/>");             //在 div 中显示 http 链接
      };
      var root = location.protocol + '//' + location.host; //判断是否为本地服务器地址
      $('a').not(':contains(root)').click(function(){       //如果不匹配本地地址，则加_blank
          this.target = "_blank";
      });
})
  });
</script>
</head>

<body>
<!--实例链接-->
<a href="www.21cn.com">21cn</a>
<a href="/9_13.html">链接实例</a>
<a href="http://www.google.com">谷歌</a>
<a href="http://www.youku.com">优酷</a>
<br/>
<br/>
<!--结果显示区域-->
<div id="result"></div>
</body>
</html>
```

本例有 4 个链接，第 05~09 行通过正则表达式匹配所有以 http 开头的链接，显示到 result 区域中。当单击链接时，判断是否为本地服务器链接，如果不是，将在一个新弹出的窗口中打开链接。

9.14 验证 Email 地址的正确性

在一些表单数据的输入场合，经常要求用户输入有效的电子邮件地址，如果用户输入错误的邮件地址，则不允许用户进行下一步的操作。使用 jQuery 的正则表达式，不需要借助任何第三方的验证插件也可以轻松实现这个验证。

注意：W3Cschool 提供了关于正则表达式的详细介绍，请参考 http://www.w3school.com.cn/jsref/jsref_obj_regexp.asp 网址，以便获取关于正则表达式修饰符的更多详细信息。

本例演示了如何验证 Email 地址的正确性，代码如下：

```
/* emailValidation.html*/
<script src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript">
  $(document).ready(function(){
    $("#iptmail").blur(function(){                    //文本框失去焦点时，触发事件
    var search_str = new RegExp(/^[\w\-\.]+@[\w\-\.]+(\.\w+)+$/);   //指定正则表达式
     var email_val = $("#iptmail").val();                          //获取邮件地址
     if(!search_str.test(email_val)){                 //测试邮件地址是否正确
         $("#txt").css("color","red");                //如果不正确显示红色消息
        $("#txt").html("请输入正确的邮件地址!");      //如果不正确提示错误消息
        $('#iptmail').focus();                        //重新设置输入焦点
     }
     else{
          $("#txt").css("color","green");             //如果正确显示绿色消息
     $("#txt").html("邮件地址正确");                  //如果正确提示正确消息
     }
    });
  });
</script>
</head>
<body>
请输入 Email:
<input type="text" id="iptmail" size="60" />
<!--显示验证消息文本-->
<span id="txt"></span>
</body>
</html>
```

本例构建了一个 RegExp 对象，其中包含了一个检测邮件的正则表达式，即字符串中应该包含@字符，并且包含一个.域名，否则会被判定为非法的邮件地址。本例调用了 RegExp 的 test()函数对文本框中输入的邮件地址进行验证。如果邮件地址错误，则在提示消息框中显示红色错误消息，如果邮件地址正确，则显示绿色消息。本例效果如图 9.4 所示。

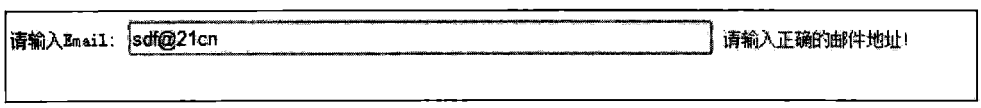

图 9.4　邮箱地址验证

9.15　动态统计字符个数

字符统计在 Word 上用得比较多，但是现在随着微博的兴起，很多网站也开始对用户输入的数量进行限制，此时字符统计功能就显得非常有用了。本例构建了一个类似微博的 140 字输入的窗口，每当用户输入字符时就统计当前的字符个数。当用户输入满 140 时，

就用红色背景提醒用户已经超过了最大可输入值。本例效果如图 9.5 所示。

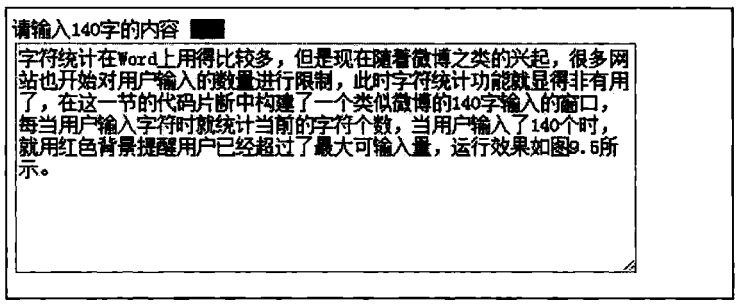

图 9.5　字符统计功能

本例代码如下：

```
01  /* wordCounts.html*/
02  <script type="text/javascript">
03  $(document).ready(function(){
04   $(".countchars").each(function(){                    //循环选中 countchars 类中的元素
05        var length = $(this).val().length;              //得到字符长度
06        $(this).parent().find('#charlength').html('<b>'+length+'</b>'); //显示字符个数
07        $(this).keyup(function(){                                  //为其关联 keyup 事件
08             var new_length = $(this).val().length;                //得到新长度
09             $(this).parent().find('#charlength').html('<b>'+new_length+'</b>'); //显示新的长度
10             if (new_length>= "140") {                             //如果字符个数大于 140
11                 $('#charlength').css('background', 'red');        //显示红色背景
12             }else
13             {
14                 $('#charlength').css('background','white');       //否则显示白色背景
15             }
16         });
17     });
18  });
19  </script>
20  </head>
21  <body>
22  <div id="gusbooks">
23     <span>请输入 140 字的内容</span>
24     <!--字符个数提示框-->
25     <span id="charlength"></span>
26     <br/>
27     <!--构建一个字符统计的文本框-->
28     <textarea name="txt" cols="60" rows="10" class="countchars"></textarea>
29  </div>
30  </body>
31  </html>
```

第 05 行通过调用字符串的 length 属性来统计字符个数，然后为控件关联了 keyup 事件，

当键盘弹开时触发这个事件。在这个事件触发期间，第09~15行不断地统计字符个数并写到span元素内部，以便显示最新的字符个数。当字符长度大于或等于140时，通过CSS设置红色背景来显示字符个数，以提醒用户超过了最大输入值。

9.16 使用jQuery验证用户年龄

有些网站只允许大于18岁的用户才能进入（如游戏类、杂志类网站），此时就需要根据用户的生日和当前的日期进行验证，以判断用户是否具有准入的条件。为了实现这样的效果，必须理解JavaScript的日期类型。本例构造了一个允许用户输入生日的表单，当用户提交表单时，将用户的生日与当前日期进行比较，以确保用户在18岁以上。本例效果如图9.6所示。

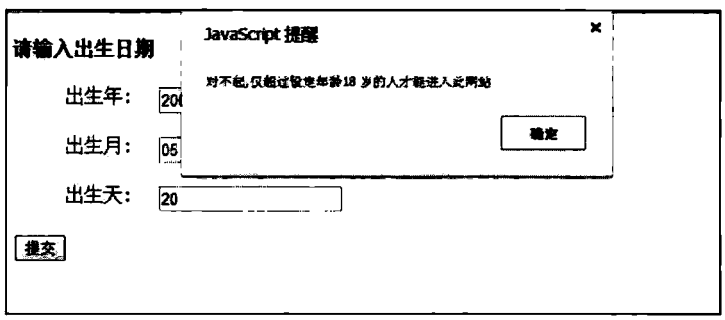

图9.6 验证用户年龄

本例代码如下：

```
01  /* validateAge.html*/
02  <body>
03  <form name="demoForm" id="demoForm" action="">
04  <!--在隐藏字段中保存了最小允许年龄-->
05   <input type="hidden" name="requiredAge" id="requiredAge" value="18">
06  <h4>请输入出生日期</h4>
07  <p><span>出生年:</span>
08  <input type="text" name="birthYear" id="birthYear" value=""></p>
09  <p><span>出生月:</span>
10  <input type="text" name="birthMonth" id="birthMonth" value=""></p>
11  <p><span>出生天:</span>
12  <input name="birthDay" type="text" id="birthDay" value=""></p>
13  <input type="submit" name="submit" value="提交">
14  </form>
15  <script type="text/javascript">
16        $("#demoForm").submit(function(){                    //关联 submit 事件
17              var day = $("#birthDay").val();                //出生日
18              var month = $("#birthMonth").val();            //出生月
```

```
19              var year = $("#birthYear").val();                    //出生年
20              var age   = $("#requiredAge").val();                 //最小年龄
21              if (day == "" || month == "" || year == "") {        //对输入进行验证
22                  alert("你必须在所有的字段中输入数值");
23                  return false;
24              }
25              var mydate = new Date();                             //构建新的日期对象
26              mydate.setFullYear(year, month-1, day);              //将输入的数据转换成出生日期
27              var currdate = new Date();                           //构建当前日期类型
28              //将当前日期减去 age，得到至少要求的日期
29              currdate.setFullYear(currdate.getFullYear() - age);
30              //如果当前日期小于输入的日期，则不允许用户进入网站
31              if ((currdate - mydate) < 0){
32                  alert("对不起,仅超过设定年龄" + age + " 岁的人才能进入此网站");
33                  return false;
34              }
35              alert("成功! 正在提交表单...");
36              return true;
37  });
38  </script>
```

第 05 行在表单中设计了一个隐藏域，这个隐藏域包含了可以进入网站的年龄最小值。第 16~37 行在表单的 submit 事件中，先行获取表单上输入的年、月、日，然后验证用户输入值的正确性。第 29 行根据用户输入的值，调用 setFullYear()构造一个当前输入的日期类型。

注意：setFullYear()包含了 year、month、day 这 3 个参数，但是表示月份的值介于 0~11 之间，因此需要在当前输入的月份上减 1。

9.17 按照首字母进行元素排序

JavaScript 数组中的 sort()函数可以按照指定的或默认的方式进行字符串的排序，不过它只用字符的 Unicode 编码比较字符串，而不考虑当地的排序规则。以这种函数生成的顺序不一定是正确的。例如，在西班牙语中，字符“ch”通常作为出现在字母“c”和“d”之间的字符来排序。

本例将演示如何通过本地排序规则对字符串进行排序，代码如下：

```
01  /* sortByalphabetical.html*/
02  <title>按照首字母进行元素排序</title>
03  <script src="jquery/jquery-1.9.1.js"></script>
04  <script type="text/javascript">
05    $(document).ready(function(){
06          $("#btnsort").click(function(){                    //单击按钮进行排序
07          sortLocal($("ul"));                                //调用 sortLocal()进行排序
```

```
08          });
09      });
10      //定义一个用于排序的函数 sortLocal()
11      function sortLocal(sort_container, sort_item) {
12          var list_items = sort_container.children(sort_item).get();      //得到子元数数组
13          list_items.sort(function(a, b) {                    //调用 localCompare()进行排序
14              return $(a).text().toUpperCase().localeCompare($(b).text().toUpperCase());
15          });
16          $.each(list_items, function(index, item) {      //循环输出排序后的数组
17              sort_container.append(item);
18          });
19      }
20  </script>
21  </head>
22
23  <body>
24  <button id="btnsort">单击这里排序</button>
25  <ul>
26      <li>我的中国心</li>
27      <li>你的中国梦</li>
28      <li>人们的中国梦</li>
29      <li>中华民族</li>
30      <li>伟大的中国</li>
31  </ul>
32  </body>
33  </html>
```

HTML 页面上有一个按钮，单击这个按钮将会调用函数 sortLocal()进行本地排序。第 11~19 行的 sortLocal()函数内部调用了 localeCompare()函数来实现本地化的排序工作。

注意：实现本地化字符串排序的关键是使用 localeCompare()函数，它考虑了默认的本地排序规则。

9.18 获取 URL 地址的 Hash 参数

URL 地址的 Hash 参数是指 URL 中“#”后面的字符，作为浏览器的锚点，它在很多地方被使用。通过 window.location.hash 可以获取和设置 Hash 字符串，本例效果如图 9.7 所示。

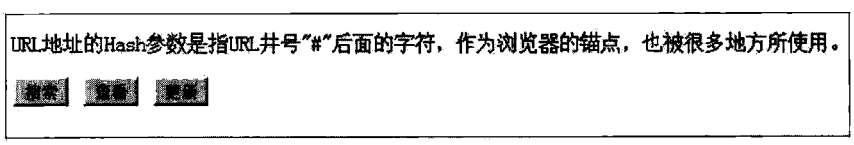

图 9.7 更新浏览器的 Hash 字符

注意：#是用来指导浏览器动作的，对服务器端完全无用。所以，HTTP 请求中不包括#，且改变#后的字符串不触发网页重载。

本例演示了如何获取和设置 URL 栏的 Hash 字符。

```
01  /* getUrlHash.html*/
02  <title>更改浏览器的 Hash 字符</title>
03  <script src="jquery/jquery-1.9.1.js"></script>
04  <script type="text/javascript">
05    $(document).ready(function(){
06      //单击按钮时，更新 URL 的 Hash 值，在进行 AJAX 调用时非常有用
07      $('button').click(function(){
08          var hash=this.id;                                    //得到 id 值
09          location.hash = hash;                                //更新 URL 栏上的 Hash 值
10          var param = document.URL.split('#')[1];              //得到 Hash 字符串
11          $("#result").html("当前的 Hash 字符串:"+param);      //显示 Hash 字符串
12      });
13    });
14  </script>
15  </head>
16  <body>
17  <p>URL 地址的 Hash 参数是指 URL 井号"#"后面的字符，作为浏览器的锚点，也被很多地方所
18      使用。</p>
19  <button id="search">搜索</button>
20  <button id="list">查看</button>
21  <button id="update">更新</button>
22  <!--显示结果-->
23  <div id="result"></div>
24  </body>
25  </html>
```

第 19~21 行在 HTML 放置了几个按钮，当在客户端单击按钮时，第 07~12 行调用 location.hash 将当前的按钮的 id 值设置为 Hash 值，然后调用 split()函数提取 Hash 字符串（即去掉#号之后的字符串），最后将其显示到 div 上。

注意：更新 Hash 字符串对于 AJAX 的调用非常有用，当更改了页面的显示又希望具有一个标记时，就可以在 URL 后面添加一个 Hash 字符，以便识别。

9.19 避免多行文本溢出的算法

很多网站都有文章列表，列表中显示了文章的部分内容，更多的内容显示省略号，单击“显示更多”链接会将其余的内容显示出来，这样即可以让用户看到部分内容，又节省了空间。CSS 具有 text-overflow:ellipsis 属性，可以实现当单行文本溢出显示区域时，显示省略号，但是这个属性不支持多行文本。本例效果如图 9.8 所示。

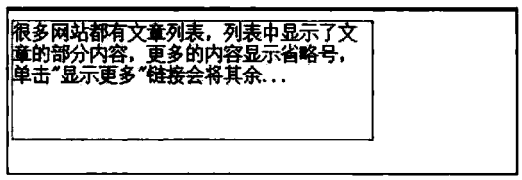

图 9.8 避免多行文本溢出

本例代码如下：

```
/* multiLineoverflow.html */
<script src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $("#txt").ellipsis(50);
});
/* 文本溢出显示省略号 */
/* 用法：$(".box").ellipsis(50) */
/*box为需要实现的Class*/
(function($) {
    jQuery.fn.ellipsis = function(maxwidth) {
        this.each(function() {                          //循环选中的控件
            if ($(this).text().length > maxwidth) {     //如果控件的text的长度大于maxwidth
                //则截取maxwidth的长度
                $(this).text($(this).text().substring(0, maxwidth));
                $(this).html($(this).html() + '...');   //在当前文本后面添加省略号
            }
        });
    }
})(jQuery)
</script>
```

第 10~20 行创建了一个 jQuery 的实例函数，它会判断当前选中的控件中文本长度是否大于传入的 maxwidth 的长度，如果条件成立则截取 maxwidth 的文本长度，在后面的 div 中添加省略号。

9.20 随机选择一个元素

网站开发经常需要随机的选择元素，比如从显示的产品中随机地为用户弹出一个推荐产品，或者是随机地显示用户收到的消息。这些都可以利用 jQuery 中随机选择元素的功能来实现。本例演示了如何随机地选择水果列表中的水果，代码如下：

```
/* multiLineoverflow.html */
<script src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript">
    $(document).ready(function(){
```

```
05              $("#lnk").click(function(){
06             var mylist = $('ul');                          //获取 ul 对象
07             var listitems = mylist.children('li').get(); //获取 ul 对象中所有 li 元素，使用 get()转换
08              var num=listitems.length-1;                   //获取数组长度
09              var ele=listitems[getramdom(0,num)];          //调用随机函数获取数组中的元素
10               $("#result").html("随机选择的元素为："+$(ele).text());      //显示随机数结果
11               });
12        });
13     function getramdom(min,max){          //创建一个随机函数，用于产生数组长度的随机数
14         return parseInt(Math.random()*(max-min+1));
15     }
16  </script>
17  </head>
18  <body>
19  <!--用来进行随机选择的元素-->
20  <ul id="ul1">
21     <li>苹果</li>
22     <li>橘子</li>
23     <li>香蕉</li>
24     <li>桃子</li>
25  </ul>
26  <a href="#" id="lnk">单击这里随机选择</a>
27  <!--结果显示部分-->
28  <div id="result"></div>
29  </body>
30  </html>
```

第 13~15 行定义了一个函数 getramdom()，它可以根据最小和最大的范围来获取一个随机整数。第 05~11 行在链接的单击事件中，将 jQuery 结果中的 li 元素转换成原生的 JavaScript 数组，然后调用 getramdom()就可以算出一个随机的数组下标，最后将这个下标显示到结果区域中，从而实现了随机选择元素的效果。

9.21 替换&bnsp;空字符的方法

在 HTML 中，空格一般是用&bnsp 表示的，某些应用中，常常需要将 保存的空格替换为空白字符串，以便插入到数据库中。这样的功能可以借助 JavaScript 的 replace()函数实现，本例演示了这种替换函数：

```
01  /* replacenbsp.html */
02  <title>替换&amp;bnsp;空字符的函数</title>
03  <script src="jquery/jquery-1.9.1.js"></script>
04  <script language="javascript">
05      $(document).ready(function(){
06          var test=$("#txt").html();          //获取 div 中的 HTML 值
```

```
07              test=test.replace(/ /ig, ""); //调用 replace()替换 为空白，/ig 是全局标志
08              $("#txt").append("<br/>"+test);  //追加到 div 中
09          });
10      </script>
11      </head>
12      <body>
13      <div id="txt">这是 一个 小小的测试</div>
14      </body>
15      </html>
```

本例通过替换 div 元素中的 来实现消除 HTML 空白字符串的效果，其中第 07 行使用了 replace()函数，这个函数是原生的 JavaScript 函数。

注意：replace()的第 1 个参数可以是一个子字符串，也可以是一个正则表达式，实例中的/ig 是指忽略大小写，并且执行全方位的搜索替换，可以通过正则表达式来创建更灵活的替换条件。

通过应用 replace()，轻松地替换掉了 div 中的 ，同样的方式可以替换很多 HTML 中其他的字符串。

9.22　序列化表单到 JSON 数据

可以通过将 form 表单的数据序列化为 JSON 数据后，再提交到后台，从而可以通过 AJAX 操作表单并处理返回的数据。jQuery 的$("form").serialize()提供了这个功能。这个函数可以收集表单上的数据，以名/值对的方式进行组织，将表单上的数据轻松地通过 AJAX 发送给服务器。

本例创建了一个简单的 HTML 表单，当用户单击“提交”按钮时，会调用 serialize()函数来序列化表单数据，然后可以看到被序列化后的表单数据，代码如下：

```
01  /* serializeJSON.html */
02  <script src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04      $(document).ready(function(){                    //在页就绪事件中，为表单提交关联处理代码
05          $('form').submit(function() {                //关联 submit 事件处理器
06              $("#result").html(($(this).serialize()));  //序列化表单数据
07              return false;                            //禁止表单提交的默认行为
08          });
09      });
10  </script>
11  </head>
12
13  <body>
14  <!--构建一个表单，表单提交方法为 post-->
15  <form method="post">
```

```
16      <!--构建2个文本域和一个隐藏域-->
17      <div><input type="text" name="a" value="1" id="a" /></div>
18      <div><input type="text" name="b" value="2" id="b" /></div>
19      <div><input type="hidden" name="c" value="3" id="c" /></div>
20      <!--多行文本域-->
21      <div><textarea name="d" rows="8" cols="40">4</textarea></div>
22      <!--下拉选择域-->
23      <div><select name="e">
24        <option value="5" selected="selected">5</option>
25        <option value="6">6</option>
26        <option value="7">7</option>
27      </select></div>
28      <div>
29        <!--复选框选项-->
30        <input type="checkbox" name="f" value="8" id="f" /><span>单击复选框进行选择
31    </span>
32      </div>
33      <div>
34        <input type="submit" name="g" value="提交" id="g" />
35      </div>
36    </form>
37    <!----显示序列化表单后的结果-->
38    <div id="result"></div>
39    </body>
40  </html>
```

本例构建了一个HTML的表单，可以看到它包含丰富的HTML表单控件，有文本框、隐藏域、多行文本框、下拉列表框、复选框等等。

注意：要实现表单序列化，每个控件都必须指定name和value属性，序列化功能将根据这两个属性来构建字符串。

当用户单击“提交”按钮后，第06行会调用serialize()来序列化表单数据，在div中产生了如下输出结果：

```
a=1&b=2&c=3&d=4&e=5&f=8
```

从结果可以看出，表单序列化后的结果完全符合$.ajax的data属性要求的格式，这样就可以直接用serialize来序列化表单数据，用AJAX方式进行提交，而不用手工书写代码来构建字符串。

9.23 获取页面加载时间

了解页面加载时间，对于监测网络的响应效率非常有用，由于HTML页面是从上到下进行解析的，而jQuery本身的ready事件仅在页面就绪而HTML的元素并没有下载完成时

触发，因此在 ready 事件中监测加载时间是不准确的。window.load 事件是在所有的 HTML 元素（包含 iframe）加载完成后才会被触发，所以使用它是最好的方法。本例效果如图 9.9 所示，在 iframe 中包含了 Adobe 公司的网页后，笔者的电脑用时 41 秒。

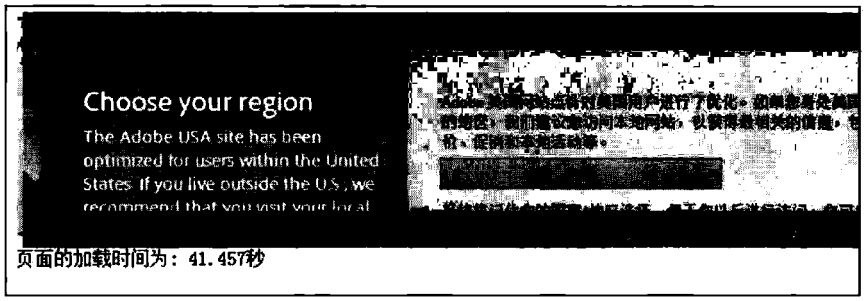

图 9.9　计算页面加载时间

本例代码如下：

```
01  /* getLoadingTime.html */
02  <title>获取页面加载时间</title>
03  <script src="jquery/jquery-1.9.1.js"></script>
04  <script type="text/javascript">
05    var beforeload = (new Date()).getTime();
06    function getPageLoadTime(){
07          var afterload = (new Date()).getTime();       //计算页面已经加载完成后的当前时间
08            seconds = (afterload-beforeload) / 1000;  //计算加载前和加载后的时间差
09          //将结果放到 div 中
10          $('#load_time').text('页面的加载时间为：  ' + seconds + '秒');
11           }
12      window.onload = getPageLoadTime;              //为 window.onload 关联事件
13  </script>
14  </head>
15
16  <body>
17    <!--调用 iframe 加载一个外部网站-->
18    <iframe src="http://www.adobe.com/" id="tag" name="tag"
19              allowTransparency="true" width="600"
20              height="200" scrolling="no" frameborder="0">
21    </iframe>
22    <!--显示结果加载时间-->
23    <div id="load_time"></div>
24  </body>
25 </html>
```

第 05 行定义了一个全局变量 beforeload，它会在页面开始加载时被调用，用来获取页面加载的时间。第 12 行在 window.onload 事件触发时调用 getPageLoadTime()函数，它将计算加载完成后的时间，计算加载前和加载后的时间差，并显示到页面上。

9.24 将单个句子打断显示

在一些网站中经常可以看到一些链接，它们包含加粗显示的标题、标题的描述等等，实际上实现这样的功能并不需要编写两行的 HTML 代码，只需要灵活地使用 jQuery 对单行文字进行打断即可。本例效果如图 9.10 所示。

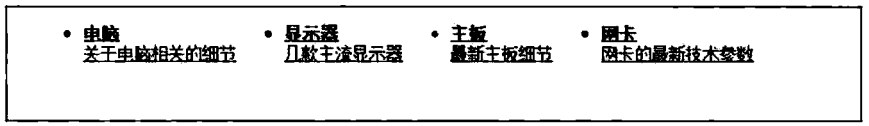

图 9.10 打断显示的效果

本例代码如下：

```
01  /* breakLine.html */
02  <title>首个单词自动换行</title>
03  <style type="text/css">
04      body{
05          font-size:14px;
06          font-family:Verdana, Geneva, sans-serif;
07      }
08      .firstWord {
09          display: block;     /*首字母 block 显示，将打断行*/
10          font-weight:bold;   /*加粗显示字体*/
11          }
12       li{
13          float: left;        /*向左浮动显示*/
14          margin: 0 20px;
15          }
16  </style>
17  <script src="jquery/jquery-1.9.1.js"></script>
18  <script type="text/javascript">
19    $(document).ready(function(){
20       $('ul.nav > li > a').each(function() {          //循环 li 中的链接元素
21            var obj = $(this);                         //得到当前的链接元素
22            var text = obj.html();                     //获取链接元素中的 HTML
23            var parts = text.split(' ');               //用空格将 HTML 分成多个元素的数组
24            //为第 1 个元素应用 span 标签与样式
25            var replace = '<span class="firstWord">'+parts[0]+'</span>';
26            parts.shift();                             //删除数组中的第 1 个元素
27            $.each(parts, function(key, value) {
28                 replace += ' '+value;                 //合并除第 1 个之外的多个子元素
29            });
30            obj.html(replace);                         //重新设置元素内容的 HTML
31       });
```

```
    });
</script>
</head>

<body>
<ul class="nav">
    <!--链接列表-->
    <li><a href="#">电脑 关于电脑相关的细节</a></li>
    <li><a href="#">显示器 几款主流显示器</a></li>
    <li><a href="#">主板 最新主板细节</a></li>
    <li><a href="#">网卡 网卡的最新技术参数</a></li>
</ul>
</body>
</html>
```

在这个实例中，ul 元素中包含了多个链接，每个链接都由一个主要关键字和描述组成，关键字与描述之间用空格分隔。在 jQuery 代码中，通过循环遍历链接，取出链接文本，然后调用字符串的 split()函数将其分隔为数组，最后为数组的第 1 个元素应用标签，这个标签的 CSS 样式会使得链接文本折断显示。

9.25 限制只能输入中文、英文或数字

在编写让用户输入数据的表单时，经常会限制用户的输入，比如有的文本框只能输入中文，有的文本框只能输入英文，还有的只能输入数据。使用正则表达式，可以轻松地实现这样的效果。本例演示了如何使用正则表达式对用户的输入表单进行限制，代码如下：

```
/* enforceInput.html */
<script src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript">
$(function(){
    var $input =$("#txt"),$button=$("#btnok");          //获取按钮和文本框实例以进行缓存
    $button.click(function(){                           //按钮单击时检测输入内容
    if(/[\s><,._\。\[\]\{\}\?\/\+\=\|\'\\\":;\~\!\@\#\*\$\%\^\&`\uff00-\uffff)(]+/.test( $input.val())) {
            $input.select();                            //如果验证失败则选中文本
             alert("输入字符只能是中文英文-");          //提示错误消息
        }else{
             alert("输入正确！");                       //如果验证通过则显示正确消息
        }
    });
});
</script>
</head>
<body>
<!--允许用户输入文本的文本框-->
```

```
19  <input type="text" value="" id="txt">
20  <input type="button" value="确定" id="btnok">
21  </body>
22  </html>
```

本例要求用户输入的是英文和中文，如果输入符号，则会显示如图 9.11 所示的信息。

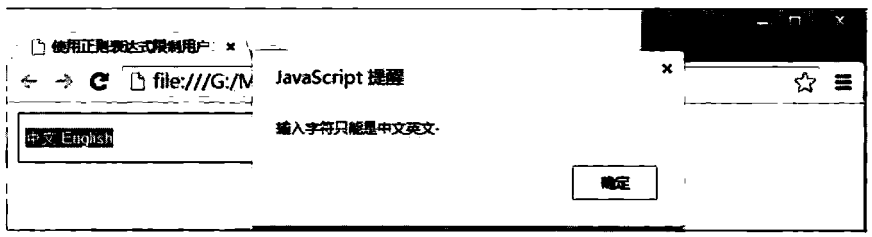

图 9.11　输入验证效果

9.26　日期类型与时间戳的转换

时间戳是自 1970 年 1 月 1 日（00:00:00 UTC/GMT）以来的秒数，它也被称为 Unix 时间戳（Unix Timestam、Unix epoch、POSIX time、Unix timestamp）。时间戳常用来计算时间差异，或是某段代码执行所需要的时间。在 JavaScript 中，可以通过 Date 类型的 valueOf() 来计算时间戳，代码如下：

```
document.write(new Date().valueOf());
```

注意：valueOf()返回的是日期时间对象的原始值，以毫秒为单位，一般将毫秒乘以 1000 来转换为秒。建议使用 JavaScript 的 Date.parse()来获取时间戳。

为了简化在 JavaScript 中创建日期类型向时间戳类型的转换步骤，本例将创建一个 jQuery 插件函数，它可以自动将传入的参数转换成日期时间类型，再转换成时间戳类型，代码如下：

```
01  /* dateToTimeConvert.html */
02  <script src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04    $(document).ready(function(){
05          var mydate=new Date();                          //构建日期类型
06          mydate.getDate();                               //获取当前日期时间
07         var today=$.Date.timestamp(                      //计算时间戳
08          {year:mydate.getFullYear(),                     //得到当前年
09          month:mydate.getMonth()+1,                      //得到当前月
10          date: mydate.getDay(),                          //得到当前天
11          hour:mydate.getHours(),                         //得到当前时
12          minute:mydate.getMinutes(),                     //得到当前分
13          second: mydate.getSeconds()});                  //得到当前秒
14          $("#result").append("当前时间戳"+today+"<br/>"); //显示当前日期时间戳
```

```
    });
    $.extend({                                          //构建 jQuery 实例插件函数
      Date:{                                            //扩展 Date 对象
          timestamp: function( options ) {              //添加一个 timestamp 对象
              options = options||{};                    //参数选项
    //从 options 中获取年、月、日、时、分、秒，构建新日期类型，用 Date.parse()转成时间戳
              var date =[ options.year || '3000', options.month || '01', options.date || '01'],
                  time=[ options.hour || '0', options.minute || '0', options.second || '0' ],
                  value=Date.parse( new Date( [date.join('-'),time.join(':')].join(' ')));
              if( !isNaN( value ) ) return value;
              return null;
          }
      }
});
</script>
```

本例为 jQuery 中的 Date 类型构建了一个新的函数 timestamp()，这个函数将调用 Date.parse()将日期类型转换成时间戳类型。在页就绪事件中，通过调用这个函数，获取当前日期的时间戳。本例效果如下：

```
当前时间戳 1396479831000
```

9.27　使用数组模拟打字效果

网页上的打字效果，相信有过上网经验的人并不陌生，使用 jQuery 可以轻松地实现一种单词的打字效果，而非单个字母的打字效果。本例演示了如何实现逐个单词的打字效果：

```
/* TypeMachine.html */
<script src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript">
$(document).ready(function () {
    var $el = $('div'),
        text = $el.text(),                  //获取文本内容
        speed = 1000;                       //指定打字的速度
    $el.empty();                            //清空原来的文本内容
    var wordArray = text.split(' '),        //使用空格将文本内容分割为数组
        i = 0;
    INV = setInterval(function () {         //循环输出
        if (i >= wordArray.length - 1) {
            clearInterval(INV);             //如果输出完成则清除计时器
        }
        $el.append(wordArray[i] + ' ');     //添加单词到 div 中
        i++;
    }, speed);                              //speed 指定加载速度
});
```

```
</script>
</head>
<body>
<div>jQuery code snippet which outputs each word in a sentence
at a specified interval. You can change the speed at
which is outputs by the speed parameter in milliseconds.
 You can view the demo or load the code in jsfiddle below.</div>
</body>
</html>
```

从代码可以看出，由于英文的单词与单词之间是通过空格来区分的，因而在 JavaScript 代码中，通过 split()函数将英文内容分割为多个数组，然后对这个数组进行循环。这个循环是在 setInterval()函数中进行的，它会每隔一段时间输出一个单词，然后递增计数器变量 i，当 i 的值与数组长度相等时，则调用 clearInterval()停止计时器，表示打字操作结束。

9.28 获取数组中特定索引的最高值

假定有一个 JSON 数组，里面包含了多个 JSON 对象作为数组元素，如果要获取某个索引值的最大值，可以使用$.each()来循环计算。本例演示了如何获取数组中特定索引的最高值：

```
/* getHightIndexFromArray.html */
<script src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript">
 $(document).ready(function(){
     function getHighestVal(data, index)                    //定义一个获取最大值的函数
     {
          var max=-1;                                       //保存返回值的全局变量
          $.each(data, function (i,v)                       //循环传入的数组
          {
               thisVal = v[index];                          //得到特定索引的值
               max = (max < thisVal) ? thisVal : max;       //比较并将最大值赋给 max
          });
          return max;                                       //将 max 作为最大值返回给调用端
     }
     var highest = getHighestVal(data, '橘子');             //调用函数，得到橘子的最大个数
     $("#result").append("橘子的最大个数是："+highest); //输出结果值
 });
 var data = Array();                                        //构建一个数组
//为数组元素赋 JSON 对象值
data[0] = {"苹果":1, "桃子":2, "橘子":3};
data[1] = {"苹果":3, "桃子":3, "橘子":5};
data[2] = {"苹果":4, "桃子":1, "橘子":6};
 </script>
```

```
24  </head>
25  <body>
26  <!--显示结果的 DIV 元素-->
27  <div id="result"></div>
28  </body>
29  </html>
```

第 20~22 行构建了一个名为 data 的数组，它包含 3 个元素，每个元素都是一个 JSON 对象，也可以将之看作键/值对组成的数组。在页就绪事件中，第 05~14 行定义了一个名为 getHighestVal()的函数，它接收数组变量和索引值，可以获取这个数组中特定索引的最大值。第 15 行调用这个函数，获取橘子的最大个数。本例效果如下：

```
橘子的最大个数是：6
```

9.29　清除所有的表单内容

在编写表单应用时，清除表单内容是经常要处理的工作，一般的做法是使用 jQuery 选择器来一个一个地清除，这种方式效率有些低下。本例编写了一个通用函数，这个函数可以一次性地清除所有的表单内容，代码如下：

```
01  /* clearFormData.html */
02  <title>清除表单的通用函数</title>
03  <script src="jquery/jquery-1.9.1.js"></script>
04  <script type="text/javascript">
05      $(document).ready(function(){
06          clearForm($("form"));                    //调用 clearForm()清除表单元素
07      });
08      function clearForm(form) {
09          $(':input', form).each(function() {      //循环表单中的所有 input 控件
10              var type = this.type;                //得到控件的类型
11              var tag = this.tagName.toLowerCase(); //得到 tagName
12              //清除文本框、密码输入框和多行文本框的内容
13              if (type == 'text' || type == 'password' || tag == 'textarea')
14                  this.value = "";
15              //将复选框和单选框的 checked 设置为 false
16              else if (type == 'checkbox' || type == 'radio')
17                  this.checked = false;
18              //将下拉列表框的 selectedIndex 设置为-1，表示不选中任何元素
19              else if (tag == 'select')
20                  this.selectedIndex = -1;
21          });
22      };
23  </script>
```

第 08~22 行定义了一个通用的名为 clearForm()的函数，这个函数接收一个表单对象作

为参数。在函数体内循环 input 标签，首先判断其 type，如果是 text、password 或 textarea，则直接设其 value 为空；如果是 checkbox 或 radio，则设其 checked 为 false；如果是 select，则设置其 selectedIndex 为-1，表示不选中任何元素。这样在调用这个函数时，只需要传递一个选中的表单对象，就可以轻松地删除所有的表单内容了。

9.30 用 jQuery 删除空白标签和具有非中断空格的标签

自动工具产生的 HTML 代码经常会出现一些空白的标签，或者是没有任何内容的标签，这会造成网页体积过大、浏览速度减慢。为此，可以借助 jQuery 来检查 HTML 代码，删除其中包含的空白标签以及无内容的标签。本例代码如下：

```
01  /* clearBankTag.html */
02  <script src="jquery/jquery-1.9.1.js"></script>
03  <script type="text/javascript">
04    $(document).ready(function(){
05      $('p,div').each(function() {              //循环选中 p 和 div 元素
06          var $p = $(this),
07              txt = $p.html();                  //得到其 html 内容
08          if (txt==' ') {                  //如果 html 内容是 
09              $p.remove();                      //则移除该元素
10          }
11      });
12      $('p,div')
13          .filter(function() {                  //过滤 p 和 div 元素中 text 为空的元素
14              return $.trim($(this).text()) === ''
15          })
16          .remove()                             //移除这些元素
17    });
18  </script>
19  </head>
20  <body>
21  <p>
22      <!--定义一些为空的标签-->
23    <span> </span>
24    <span></span>
25      <p></p>
26      <p> </p>
27      <div> </div>
28    <span>正工的文本</span>
29  </p>
30  </body>
31  </html>
```

第 05~16 行循环选中的元素并获取其中的 HTML 内容，如果为 ，则移除元素。

第 13 行使用 filter()过滤元素内容为空白的标签并进行移除。浏览这个页面时，可以看到只有一个 span 元素存在，其他都被移除了，如图 9.12 所示。

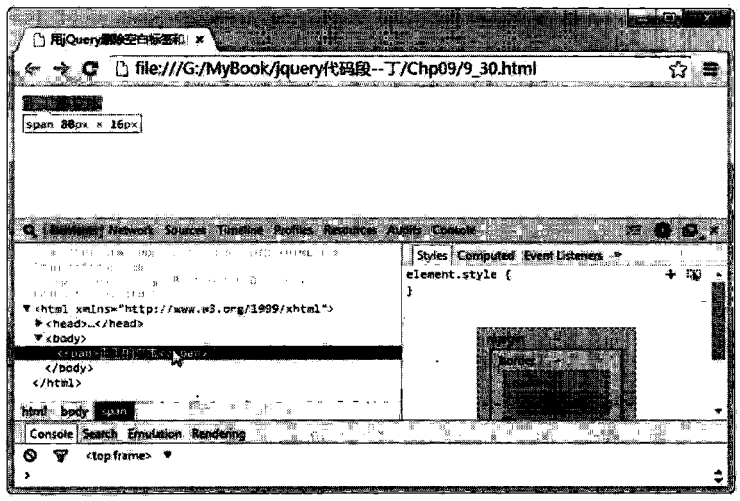

图 9.12　移除空白标签后的效果

第 10 章　jQuery Mobile 移动开发技巧

jQuery Mobile 是一个基于 jQuery 的移动开发平台，用这个平台可以开发出支持多种移动系统的应用，比如用 jQuery Mobile 开发的 iOS 应用，可以轻松地部署到 Android 系统上，基本上只需要编写一套代码，就可以在多个移动平台上使用。jQuery Mobile 让移动开发变得更加容易，开发人员只需要对 HTML、CSS 和 jQuery 具有一定的了解，就可以着手开发移动 APP 了。

本章主要涉及的知识点有：

- jQuery Mobile 的多页面显示视图。
- jQuery Mobile 的转场效果。
- 不同的导航栏的应用。
- 可折叠列表框的应用。
- 动态加载和切换页面。
- 动态添加列表框的菜单。

10.1　让页面自适应屏幕宽度

使用 jQuery Mobile 编写移动应用时，首先要注意的就是屏幕的宽度和高度要与设备的大小自动适应。移动设备的大小种类非常多，如果设置了固定的大小，会导致其他的设备在显示上有差异。jQuery Mobile 提供了名为 viewport 的 meta，通过设置该标记，就可以让 jQuery Mobile 设计的页面自适应屏幕。本例代码如下：

```
01  /* autofitDevice.html */
02  <!DOCTYPE html>
03  <html>
04    <head>
05      <meta charset="utf-8">
06     <title>自适应宽度的页面</title>
07     <!--使用名为 viewport 的 meta 值，其 content 指定自适应设备的宽度-->
08      <meta name="viewport"  content="width=device-width,  initial-scale=1">
09      <link href="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css"
```

```
10        rel="stylesheet" type="text/css"/>
11        <script src="http://code.jquery.com/jquery-1.6.4.min.js" type="text/javascript"></script
12        <script src=http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.js
13          type="text/javascript"></script>
14        </head>
15    <body>
16      <div data-role="page">
17      <div data-role="header">
18        <h1>我的实例</h1>
19      </div><!-- /页面头区域 -->
20      <div data-role="content">
21        <p>Hello world</p>
22      </div><!-- /页面内容  -->
23      <div data-role="footer" data-position="fixed">
24          <h4>页面脚注</h4>
25      </div><!-- /脚注 -->
26     </div><!-- /页面标签 -->
27    </body>
28    </html>
```

第08行的device-width表示的是所使用设备的宽度，而initial-scale指定缩放的等级，这个设置没有禁用用户缩放页面的权限，因而可以带来较好的可访问性。

10.2 在移动设备页面中创建多个显示视图

移动应用一般都具有多个显示页面，比如首页用来显示系统的导航信息，当用户单击不同的链接时，将导航到不同的页面。在jQuery Mobile中，通过定义多个data-role为page的页面，可以轻松地实现多页面视图效果。本例效果如图10.1所示。

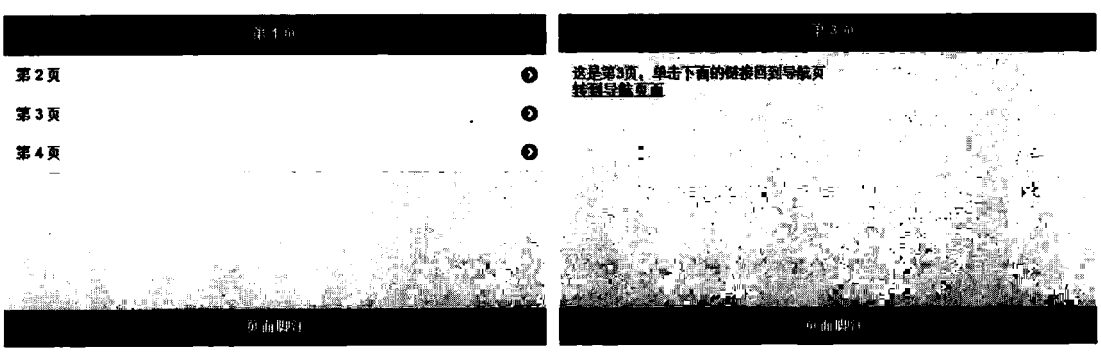

图10.1 多个页面效果

在这个实例中，第1页包含了列表链接，当单击不同的页面链接时，将进入到不同的页面，当单击每一页的“转到导航页面”链接时，将回到首页。本例代码如下：

```
01  /* multiView.html*/
```

```
<meta charset="utf-8">
<title>多页面的 jQuery Mobile 效果实例</title>
<!--指定设备的宽度-->
<meta name="viewport"  content="width=device-width,  initial-scale=1">
<link href="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css" rel="stylesheet"
type="text/css"/>
<script src="http://code.jquery.com/jquery-1.6.4.min.js" type="text/javascript"></script>
<script src="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.js"
type="text/javascript"></script>
</head>
<body>
<div data-role="page" id="navpage"><!--定义导航页-->
    <div data-role="header">
        <h1>第 1 页</h1>
    </div>
    <div data-role="content">          <!--导航列表，每一个链接指向一个 page id-->
        <ul data-role="listview">
            <li><a href="#page2">第 2 页</a></li>
            <li><a href="#page3">第 3 页</a></li>
            <li><a href="#page4">第 4 页</a></li>
        </ul>
    </div>                              <!--页脚，data-position 指示在页面的底部-->
    <div data-role="footer" data-position="fixed">
        <h4>页面脚注</h4>
    </div>
</div>
<div data-role="page" id="page2">      <!--定义第 2 个页面，id 值为 page2-->
    <div data-role="header">
        <h1>第 2 页</h1>
    </div>
    <div data-role="content">          <!--定义第 2 个页面的内容-->
        这是第 2 页，单击下面的链接回到导航页<br/>
       <a href="#navpage">转到导航页面</a><!--返回到导航页面-->
    </div>
    <div data-role="footer" data-position="fixed">
        <h4>页面脚注</h4>
    </div>
</div>
<div data-role="page" id="page3">      <!--定义第 3 个页面的内容-->
    <div data-role="header">
        <h1>第 3 页</h1>
    </div>
```

```
44        <div data-role="content">
45            这是第 3 页，单击下面的链接回到导航页<br/>
46          <a href="#navpage">转到导航页面</a><!--返回到导航页面-->
47        </div>
48        <div data-role="footer" data-position="fixed">
49            <h4>页面脚注</h4>
50        </div>
51    </div>
52    <div data-role="page" id="page4">        <!--定义第 4 个页面的内容-->
53        <div data-role="header">
54            <h1>第 4 页</h1>
55        </div>
56        <div data-role="content">
57            这是第 4 页，单击下面的链接回到导航页<br/>
58          <a href="#navpage">转到导航页面</a><!--返回到导航页面-->
59        </div>
60        <div data-role="footer" data-position="fixed">
61            <h4>页面脚注</h4>
62        </div>
63    </div>
64    </body>
65    </html>
```

在 jQuery Mobile 中，data-role 是一个非常重要的角色，它用来指定当前的元素所代表的角色。在代码中，每一个页面由一个 data-role 为 page 的 div 表示，它里面可以包含页标题 header、页面内容区域 content 以及页脚区域 footer，其中页面的内容区域 content 可以包含任何其他的 HTML 元素，从而形成 jQuery Mobile 的用户界面。

注意：在 content 容器中，可以加入标准 HTML 元素，如<p>、 、<ul>等，也可以在 jQuery Mobile 的样式表之后加入自己的样式来创建自定义的布局。

本例为每个页面分配了一个 id 值，通过链接元素 a 的 href 属性指向这个 id 值，jQuery Mobile 就会自动完成单击时的页面切换，就好像同时启动了多个 HTML 页面一样，从而实现了多个页面的 jQuery Mobile 效果。

10.3 创建对话框显示效果

对话框是移动应用中常见的需求，比如弹出一个提示消息框，或是一个多项选择框。在 jQuery Mobile 中，只需要为链接添加 data-rel="dialog"属性，就可以将链接的页面显示成对话框。本例演示了如何在页面中弹出一个对话框，效果如图 10.2 所示。

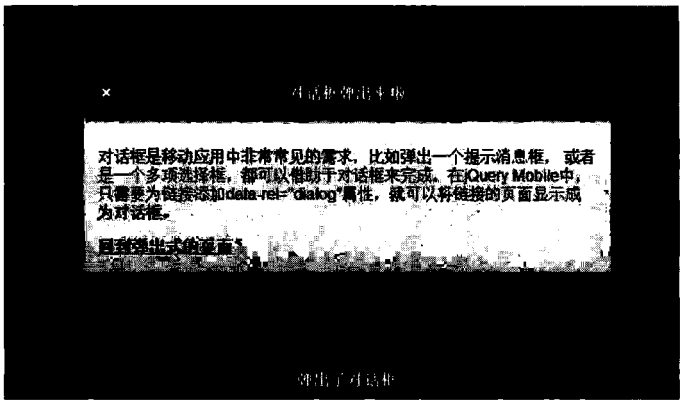

图 10.2　jQuery Mobile 弹出式对话框

当用户单击主页上的“弹出一个对话框”链接时，将弹出如图 10.2 所示的对话框，下面是本例代码：

```
/* multiView.html*/
<!--定义一个主页-->
<div data-role="page" id="pageone">
  <div data-role="header">
    <h1>jQuery Mobile 对话框效果</h1>
  </div>
  <div data-role="content">
    <p>请单击下面的按钮弹出一个对话框！</p>
    <!--这里使用 data-rel 指定将弹出一个对话框-->
    <a href="#pagetwo" data-rel="dialog">弹出一个对话框</a>
  </div>
  <div data-role="footer" data-position="fixed">
  <h1>对话框效果</h1>
  </div>
</div>
<!--被用来作为弹出式窗口显示的页面内容-->
<div data-role="page" id="pagetwo">
  <div data-role="header">
    <h1>对话框弹出来啦</h1>
  </div>
  <!--弹出式对话框的内容-->
  <div data-role="content">
    <p>对话框是移动应用中非常常见的需求，比如弹出一个提示消息框，
    或者是一个多项选择框，都可以借助于对话框来完成，在 jQuery Mobile 中，
    只需要为链接添加 data-rel="dialog"属性，就可以将链接的页面显示成为对话框。</p>
    <!--使用链接返回到前一页-->
    <a href="#pageone">回到弹出式的页面</a>
  </div>
  <div data-role="footer" data-position="fixed">
```

```
30    <h1>弹出了对话框</h1>
31    </div>
32  </div>
```

第10行为a元素指定了data-rel="dialog"属性之后，jQuery Mobile就会在一个弹出式窗口中显示链接到的页面，单击页面左上角的关闭按钮可以关闭对话框，也可以通过页面上的返回链接返回到前一页，从而形成流畅的弹出/关闭效果。

10.4 从外部文件中加载页面内容

当构建一个较复杂的应用时，在单一页面上可能有数十个页面，这样的页面特别臃肿。jQuery Mobile允许加载外部的页面链接，即通过链接指向一个外部页面，它的内容会加载到DOM中，所有的jQuery Mobile组件都会自动进行初始化。

注意：jQuery Mobile使用AJAX来请求外部页面，读取时会显示一个提示框，如果AJAX请求失败，jQuery Mobile会显示一个错误提示框，但很快会消失并不会影响用户继续使用。

本例演示了如何从外部文件加载内容，代码如下：

```
01  /* showExternalFile.html*/
02    <div data-role="page">
03    <div data-role="header">
04     <h1>我的实例</h1>
05    </div><!-- /页面头区域 -->
06    <div data-role="content">
07    <a href="10_4_1.html">显示外部页面内容</a>
08    </div><!-- /页面内容 -->
09    <div data-role="footer" data-position="fixed">
10      <h4>页面脚注</h4>
11    </div><!-- /脚注 -->
12   </div><!-- /页面标签 -->
```

上述代码构建了一个简单页面，在页面的内容区域，第07行包含一个指向10_4_1.html的外部链接。10_4_1.html是一个独立的HTML页面，它不需要包含对jQuery Mobile库或CSS的引用，它的HTML元素将被插入到宿主页面的DOM树中。10_4_1.html的代码如下：

```
01  /* 10_4_1.html*/
02  <!DOCTYPE html>
03  <html>
04    <head>
05      <meta charset="utf-8">
06      <meta name="viewport"  content="width=device-width,  initial-scale=1">
07      <title>外部页面</title>
```

```
08    </head>
09  <body>
10    <div data-role="page"   data-add-back-btn="true">
11    <div data-role="header">
12     <h1>这是来自外部的页面</h1>
13    </div><!-- /页面头区域 -->
14    <div data-role="content">
15     <p>这个页面来自 10_4_1.html，它会被另一个页面以 AJAX 方式载入</p>
16    </div><!-- /页面内容  -->
17    <div data-role="footer" data-position="fixed">
18      <h4>外部页面</h4>
19    </div><!-- /脚注 -->
20   </div><!-- /页面标签 -->
21  </body>
22  </html>
```

可以看到这是一个中规中矩的页面，第 10 行包含一个名为 data-add-back-btn 的属性，表示将在页面标题部分自动显示一个回退按钮，方便回到来源页面。本例效果如图 10.3 所示。

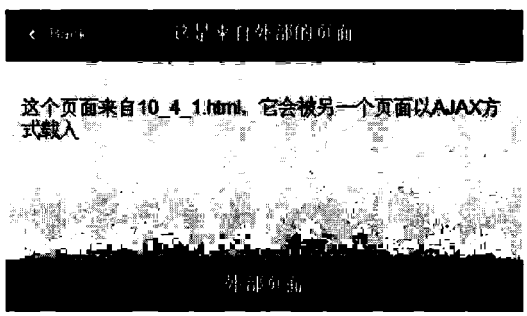

图 10.3　显示一个外部页面

从图中可以看到，外部页面包含了一个回退按钮（Back），单击这个按钮将回到宿主页面。如果宿主页面上的链接指向一个不存在的页面，jQuery Mobile 将显示一行错误消息，并且立即消失，不影响用户的其他操作。

10.5　为页面添加转场效果

jQuery Mobile 为多个页面之间的切换提供了丰富的转场效果，而且还允许开发人员添加自定义的转场效果。通过引入转场效果，可以让整个应用程序变得更加动感，增强了应用的吸引力。jQuery Mobile 使用 CSS 来实现转场效果，它提供了多种预置的转场特效。本例演示了如何在页面切换时应用不同的转场效果，效果如图 10.4 所示。当单击不同的链接时，将以相应的转场效果显示页面。

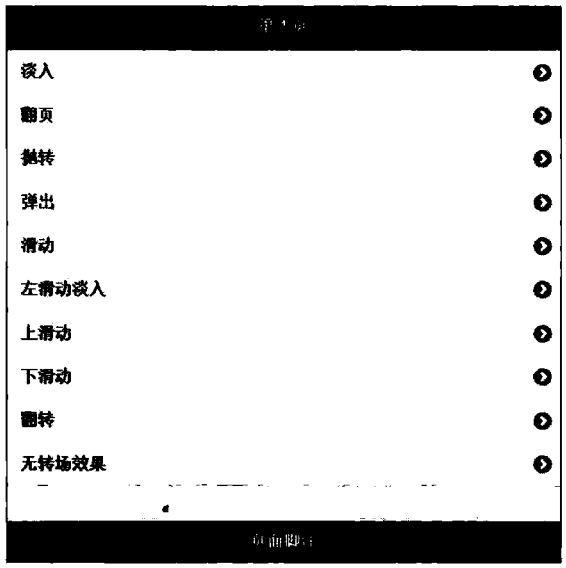

图 10.4　页面转场加载效果

本例代码如下：

```
01  /* pageTransition.html*/
02  <div data-role="page" id="navpage"><!--定义导航页-->
03      <div data-role="header">
04          <h1>第 1 页</h1>
05      </div>
06      <div data-role="content">     <!--使用不同的转场效果来加载外部链接-->
07              <li><a href="#page2" data-transition="fade">淡入</a></li>
08              <li><a href="#page2" data-transition="flip">翻页</a></li>
09              <li><a href="#page2" data-transition="flow">抛转</a></li>
10              <li><a href="#page2" data-transition="pop">弹出</a></li>
11              <li><a href="#page2" data-transition="slide">滑动</a></li>
12              <li><a href="#page2" data-transition="slidefade">左滑动淡入</a></li>
13              <li><a href="#page2" data-transition="slideup">上滑动</a></li>
14              <li><a href="#page2" data-transition="slidedown">下滑动</a></li>
15              <li><a href="#page2" data-transition="turn">翻转</a></li>
16              <li><a href="#page2" data-transition="none">无转场效果</a></li>
17      </div>                          <!--页脚，data-position 指示在页面的底部-->
18      <div data-role="footer" data-position="fixed">
19          <h4>页面脚注</h4>
20      </div>
21  </div>
```

第 07~16 行为每一种转场效果都定义了一个列表视图的链接，当单击链接时，将以不同的转场方式加载第 2 个页面。

10.6 设置全局默认的转场效果

除了为每个样式应用转场效果之外，还可以通过 jQuery Mobile 的全局参数 defaultPageTransition 设置默认的转场效果。

注意：jQuery Mobile 全局参数允许开发者更改其框架的默认事件行为，可以扩展 jQuery Mobile 的初始化事件、创建自定义命名空间、页面初始化等等。它在 jQuery Mobile 初始化时使用 $.mobile 进行设置。

本例演示了如何通过设置 defaultPageTransition 参数来改变默认的转场效果，如图 10.5 所示。

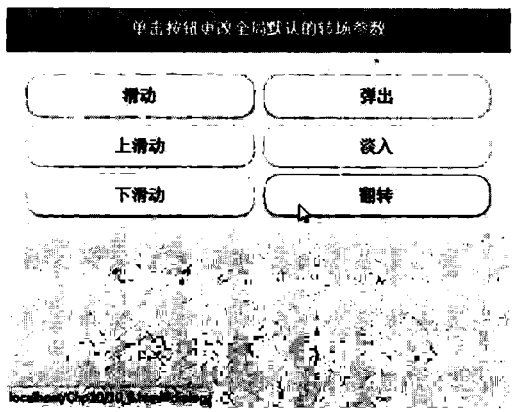

图 10.5 单击不同的按钮实现全局翻转

本例代码如下：

```
/* defaultTransition.html*/
<script>
  $(document).ready(function(){
    $(".slide").bind("click",function(){
      $.mobile.defaultPageTransition="slide";          //更改为滑动
    });
    $(".slideup").bind("click",function(){
      $.mobile.defaultPageTransition="slideup";        //更改为上滑动
    });
    $(".slidedown").bind("click",function(){
      $.mobile.defaultPageTransition="slidedown";      //更改为下滑动
    });
    $(".pop").bind("click",function(){
      $.mobile.defaultPageTransition="pop";            //更改为弹出式
    });
    $(".fade").bind("click",function(){
      $.mobile.defaultPageTransition="fade";           //更改为淡入
```

```
18        });
19        $(".flip").bind("click",function(){
20          $.mobile.defaultPageTransition="flip";              //更改为翻转
21        });
22      });
23      </script>
24    </head>
25    <body>
26    <div data-role="page">
27      <div data-role="header" data-theme="b">
28        <h1>单击按钮更改全局默认的转场参数</h1>
29      </div>
30      <div data-role="content">
31        <div class="ui-grid-a">
32          <div class="ui-block-a">
33            <a href="#dialog" data-role="button" data-rel="page" class="slide">滑动</a>
34            <a href="#dialog" data-role="button" data-rel="page" class="slideup">上滑动</a>
35            <a href="#dialog" data-role="button" data-rel="page" class="slidedown">下滑动</a>
36          </div>
37          <div class="ui-block-b">
38              <a href="#dialog" data-role="button" data-rel="page" class="pop">弹出</a>
39            <a href="#dialog" data-role="button" data-rel="page" class="fade">淡入</a>
40            <a href="#dialog" data-role="button" data-rel="page" class="flip">翻转</a>
41          </div>
42        </div>
43      </div>
44      </div>
45    </div>
46    <div data-role="page" id="dialog"><!-- dialog-->
47      <div data-role="header" data-theme="e">
48        <h1>实例数据</h1>
49      </div><!-- /header -->
50      <div data-role="content" data-theme="e">
51        <p>通过在链接上使用<code>data-transition</code>标记，可以实现动画转场效果</p>
52        <p>由于转场使用了 CSS 变换，因此在很多设备上需要硬件的支持</p>
53        <p>你是如何考虑的?</p>
54        <a href="#" data-role="button" data-theme="b" data-rel="back">返回</a>
55      </div>
56    </div>
57    </body>
58    </html>
```

第 32~41 行创建了一排按钮，分别用来展示单击时不同的转场效果。在页面就绪事件中，第 04~21 行通过为每个按钮关联事件处理代码，设置$.mobile.defaultPageTransition 的

默认效果。

10.7 定制显示回退按钮

在 jQuery Mobile 中，可以通过在页面上设置 data-add-back-btn="true"来显示回退按钮，不过当页面很多时，如果需要统一地为所有页面都添加这个功能，可能代码量会增大很多。本例通过在 mobileinit 事件中增加全局选项来为所有的页面增加回退按钮，效果如图 10.6 所示。

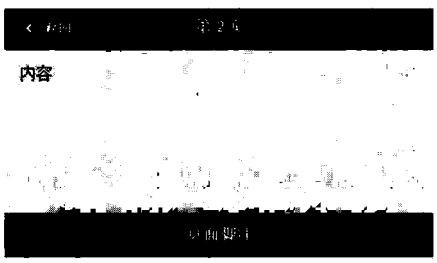

图 10.6 定制显示的回退按钮

本例代码如下：

```
01  /* backButton.html*/
02      <script type="text/javascript">
03          $(document).bind("pageinit", function () {  //在页面初始化事件中设置全局选项
04              $.mobile.page.prototype.options.backBtnText = "返回";  //设置回退按钮文本
05              $.mobile.page.prototype.options.addBackBtn = true;     //为每一页添加回退按钮
06          });
07      </script>
```

第 04~05 行通过设置全局参数的 backBtnText 值和 addBackBtn 值，就可以定制回退按钮。

同样的方法，可以通过设置 closeBtnText 来指定关闭按钮的文本，jQuery Mobile 本身提供了很多全局选项，灵活应用这些选项可以实现很多有用的效果。

10.8 在标题栏中添加导航栏

移动应用都要有一个非常明显的导航栏，来给用户提供方便。在 jQuery Mobile 中，使用 data-role="navbar"可以轻松地实现导航栏。本例演示了如何在标题栏中添加一个导航栏，并允许用户进行搜索和查找。本例效果如图 10.7 所示。

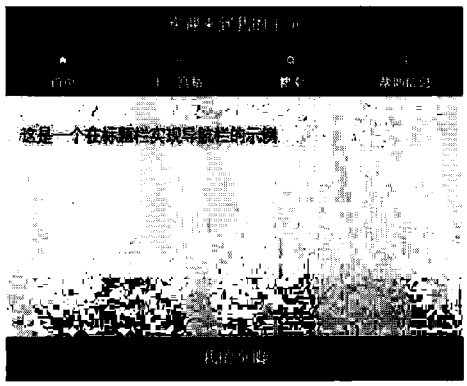

图 10.7 标题栏的导航效果

本例代码如下：

```
01  /* navInTitle.html*/
02  <!--在标题栏实现导航效果-->
03  <div data-role="page" id="pageone">
04    <div data-role="header">
05      <h1>欢迎来到我的主页</h1>
06      <!--指定导航条效果-->
07      <div data-role="navbar">
08        <ul>
09          <!--导航工具栏-->
10          <li><a href="#" data-icon="home"
11                class="ui-btn-active ui-state-persist">首页</a></li>
12          <li><a href="#" data-icon="grid">十二宫格</a></li>
13          <li><a href="#" data-icon="search">搜索</a></li>
14          <li><a href="#" data-icon="info">帮助信息</a></li>
15        </ul>
16      </div>
17    </div>
18    <div data-role="content">
19      <p>这是一个在标题栏实现导航栏的实例</p>
20    </div>
21    <div data-role="footer" data-position="fixed">
22      <h1>我的页脚</h1>
23    </div>
24  </div>
```

第 07 行设置 data-role="navbar"表示要在标题栏放置一个导航栏，第 10~14 行中的 data-icon 指定按钮要显示的图标。

10.9 在页脚区域添加导航栏

网页的导航栏一般在上方，而手机 APP 的导航栏一般在下方，这主要是考虑到单手操

作的便捷性。在页脚区域添加导航栏只需要在页脚区域编写带有 data-role="navbar"属性的 div 即可。本例演示了如何在页脚区域放置导航栏，效果如图 10.8 所示。

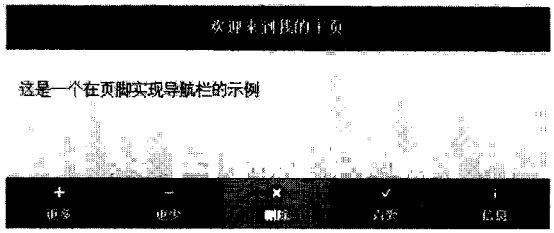

图 10.8　页脚的导航栏效果

本例代码如下：

```
01  /* navInFooter.html*/
02  <!--在页脚区域实现导航效果-->
03  <div data-role="page" id="pageone">
04    <div data-role="header">
05      <h1>欢迎来到我的主页</h1>
06    </div>
07    <div data-role="content">
08      <p>这是一个在页脚实现导航栏的实例</p>
09    </div>
10    <!--页脚区域-->
11    <div data-role="footer" data-position="fixed">
12      <div data-role="navbar">      <!---在页脚放置导航栏-->
13        <ul>
14          <li><a href="#" data-icon="plus" >更多</a></li>
15          <li><a href="#" data-icon="minus">更少</a></li>
16          <li><a href="#" data-icon="delete">删除</a></li>
17          <li><a href="#" data-icon="check">喜爱</a></li>
18          <li><a href="#" data-icon="info">信息</a></li>
19        </ul>
20      </div>
21    </div>
22  </div>
```

不管将标题放在页脚中还是在标题栏中，都是使用了 data-role 和 data-icon 这两个属性，在使用方式上一样，效果不同罢了。

10.10　添加可折叠的导航按钮

当需要提供的功能较多时，可以创建可折叠分组的导航按钮，这样可以节省屏幕空间。jQuery Mobile 中只需要指定 data-role="collapsible"即可创建一个可折叠的面板。本例创建一个可折叠的图书查询系统，通过可折叠的面板能查到不同的分类，效果如图 10.9 所示。

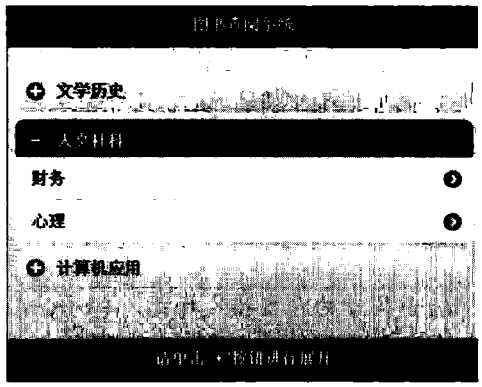

图 10.9 可折叠的导航面板

本例代码如下：

```
/* collapsibleNav.html*/
<div data-role="page" id="pageone">
  <div data-role="header">
    <h1>图书查阅系统</h1>
  </div>
  <!--创建一个可折叠的导航面板-->
  <div data-role="content">
    <div data-role="collapsible" data-theme="e">
    <h4>文学历史</h4>
    <ul data-role="listview">
      <li><a href="#">明代</a></li>
      <li><a href="#">宋代</a></li>
    </ul>
  </div>
    <!--显示人文社科的可折叠面板-->
    <div data-role="collapsible" data-theme="b" data-collapsed="false" >
    <h4>人文社科</h4>
    <ul data-role="listview">
       <li><a href="#">财务</a></li>
       <li><a href="#">心理</a></li>
    </ul>
    </div>
    <!--显示计算机应用的可折叠面板-->
    <div data-role="collapsible" data-theme="e">
    <h4>计算机应用</h4>
    <ul data-role="listview">
      <li><a href="#">软件开发</a></li>
      <li><a href="#">数据库</a></li>
      <li><a href="#">移动开发</a></li>
    </ul>
    </div>
```

```
32      </div>
33      <div data-role="footer" data-position="fixed">
34        <h1>请单击“+”按钮进行展开</h1>
35      </div>
36    </div>
```

第 08、16 和 24 行先是通过 data-role="collapsible"创建可折叠的 div，然后在其中通过 data-role="listview"创建列表框。

注意：第 08 行的 data-theme 用于为元素指定样式，jQuery Mobile 内置了一系列的样式，通过 data-theme 指定 a~e 之间的字母即可以使用预定义样式，也可以使用 jQuery Mobile 的样式构建器创建自定义的样式。

当展开图书大类时，便会显示不同的图书子类，其中第 16 行的 data-collapsed="false" 表示默认不折叠，也就是默认将显示“人文社科”分类。

10.11 实现可折叠的输入表单

折叠就是为了节省可用区域，表单也可以放在可折叠区域中，当用户单击按钮时，才将表单展开供用户输入信息，这样可以大大节省空间。本例演示了如何创建一个可折叠的表单，效果如图 10.10 所示。单击顶部的按钮，会将整个表单折叠起来，再次单击按钮，则表单展开，这样可以在屏幕上放置其他可显示的内容，节省了屏幕空间。

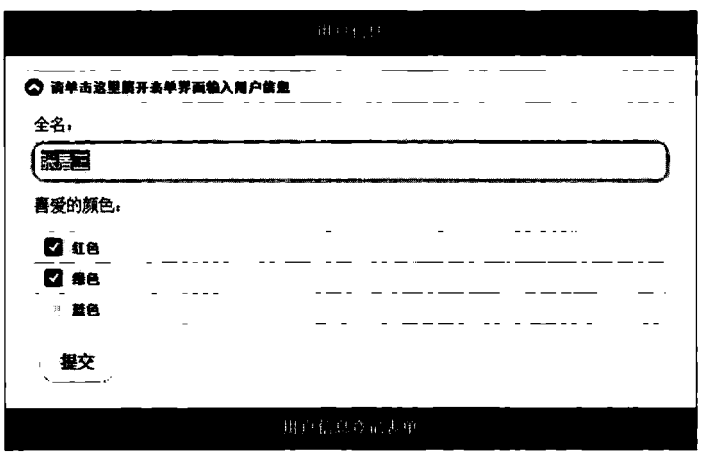

图 10.10 可折叠的表单效果

本例代码如下：

```
01  /* collapsibleForm.html*/
02  <div data-role="page" data-theme="d">
03    <div data-role="header">
04    <h1>用户信息</h1>
05    </div>
```

```
06      <!--创建一个可折叠的表单-->
07    <div data-role="content">
08      <form method="post" action="demoform.php">
09        <!--将所有的表单内容包含在一个 fieldset 中，然后为 fieldset 应用折叠-->
10        <fieldset data-role="collapsible" data-collapsed-icon="arrow-d"
11          data-expanded-icon="arrow-u" data-mini="true">
12          <!--折叠按钮的标题-->
13          <legend>请单击这里展开表单界面输入用户信息</legend>
14            <label for="name">全名：</label>
15            <input type="text" name="text" id="name">
16            <p>喜爱的颜色：</p>
17           <!--将控件进行垂直分组-->
18          <div data-role="controlgroup">
19            <label for="red">红色</label>
20            <input type="checkbox" name="favcolor" id="red" value="red">
21            <label for="green">绿色</label>
22            <input type="checkbox" name="favcolor" id="green" value="green">
23            <label for="blue">蓝色</label>
24            <input type="checkbox" name="favcolor" id="blue" value="blue">
25          </div>
26        <input type="submit" data-inline="true" value="提交">
27        </fieldset>
28      </form>
29    </div>
30    <!--页脚区域-->
31    <div data-role="footer" data-position="fixed">
32      <h1>用户信息登记表单</h1>
33    </div>
34  </div>
```

从第 08 行开始的 form 元素内部结构可以看出，所有的 jQuery Mobile 表单元素都放在一个 fieldset 中，而且为 fieldset 指定了 data-role="collapsible"，这样使得表单内容可以被折叠起来。其中 data-collapsed-icon 和 data-expanded-icon 用来指定自定义的折叠图标，data-mini 用来指定用较小的折叠样式显示其外观。

10.12 实现手风琴样式的折叠面板

手风琴样式的折叠面板是指当展开一个栏位时，其他的栏位自动折叠，这样的效果在展示局部信息时特别有用，比如我们最常见的 QQ 面板就是手风琴样式的。本例采用了手风琴样式的折叠面板显示当前流行的平板设备，效果如图 10.11 所示。

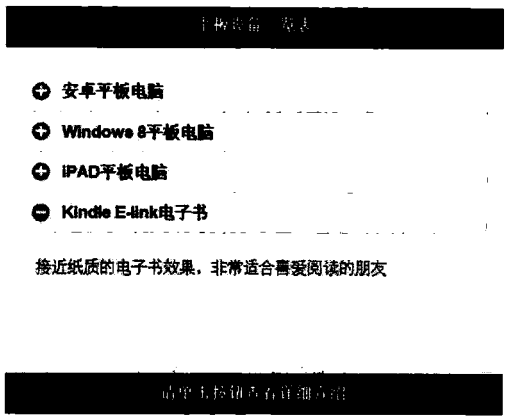

图 10.11　手风琴效果的折叠面板

本例代码如下：

```
/* collapsibleCool.html*/
<div data-role="page" id="pageone">
  <div data-role="header">
    <h1>平板设备一览表</h1>
  </div>
  <!--内容区域-->
  <div data-role="content">
    <!--创建一个折叠组，在这个组内的所有折叠效果具有手风琴效果-->
    <div data-role="collapsible-set">
      <div data-role="collapsible">
        <h3>安卓平板电脑</h3>
         <p>价钱相当实惠，软件相当多，硬件可圈可点，是广大平民的最爱</p>
      </div>
      <div data-role="collapsible">
        <h3>Windows 8 平板电脑</h3>
        <p>微软的后启之秀，但是冒似喜欢的人不多，适合商务人士</p>
      </div>
      <div data-role="collapsible">
        <h3>iPAD 平板电脑</h3>
        <p>苹果的高质量，iPAD 仍然是很多高品位人士的首选</p>
      </div>
      <div data-role="collapsible">
        <h3>Kindle E-link 电子书</h3>
        <p>接近纸质的电子书效果，非常适合喜爱阅读的朋友</p>
      </div>
    </div>
  </div>
   <!--页脚区域-->
  <div data-role="footer"   data-position="fixed">
    <h1>请单击按钮查看详细介绍</h1>
```

```
31      </div>
32  </div>
```

本例将所有的折叠面板放在了一个折叠组内部，每个折叠组内部的折叠面板都具有手风琴效果，也就是说任意一个面板展开时其他面板会自动叠起。

10.13 使用网格打造简单的九宫格界面

九宫格界面是Windows Phone手机上的一种经典界面，本例使用jQuery Mobile的网格布局特性来打造一个九宫格界面。网格布局是jQuery Mobile提供的基于CSS的列布局方案，如果需要类似网站表格那样的布局时，使用网格就非常方便。

注意：jQuery Mobile不推荐在移动设备上使用列布局，由于移动设备的屏幕宽度有限，因此有可能得不到想要的效果。网格中的列是等宽的（总宽是100%），没有边框、背景、外边距或内边距。

jQuery Mobile的网格布局只需要设置CSS即可，它内置了一些布局元素，当然也可以自定义布局方法。本例效果如图10.12所示。

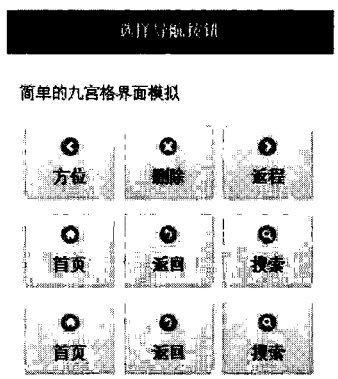

图10.12 九宫格模拟效果

本例代码如下：

```
01  /* nineUI.html*/
02  <!--指定页面使用内置的 e 主题-->
03  <div data-role="page" id="pageone" data-theme="e">
04    <div data-role="header">
05    <h1>选择导航按钮</h1>
06    </div>
07    <div data-role="content">
08     <p>简单的九宫格界面模拟</p>
09     <div class="ui-grid-b">   <!---使用 ui-grid-b 来布局界面，表示必须有 3 列 -->
10       <div class="ui-block-a"> <!--第 1 行 第 1 列-->
11          <a href="#" data-role="button" data-corners="false" data-icon="arrow-l"
```

```
12              data-iconpos="top">方位</a>
13          </div>
14          <div class="ui-block-b"> <!--第 1 行 第 2 列-->
15            <a href="#" data-role="button" data-corners="false" data-icon="delete"
16              data-iconpos="top">删除</a>
17          </div>
18          <div class="ui-block-c"> <!--第 1 行 第 3 列-->
19            <a href="#" data-role="button" data-corners="false"
20              data-icon="arrow-r" data-iconpos="top">返程</a>
21          </div>
22          <!--开始第 2 行-->
23          <div class="ui-block-a"> <!--第 2 行 第 1 列-->
24              <a href="#" data-role="button" data-corners="false"
25              data-icon="home" data-iconpos="top">首页</a>
26          </div>
27          <div class="ui-block-b"> <!--第 2 行 第 2 列-->
28            <a href="#" data-role="button" data-corners="false"
29              data-icon="back" data-iconpos="top">返回</a>
30          </div>
31          <div class="ui-block-c"> <!--第 2 行 第 3 列-->
32            <a href="#" data-role="button" data-corners="false"
33              data-icon="search" data-iconpos="top">搜索</a>
34          </div>
35          <!--开始第 3 行-->
36          <div class="ui-block-a"> <!--第 3 行 第 1 列-->
37            <a href="#" data-role="button" data-corners="false"
38              data-icon="home" data-iconpos="top">首页</a>
39          </div>
40          <div class="ui-block-b"> <!--第 3 行 第 2 列-->
41            <a href="#" data-role="button" data-corners="false"
42              data-icon="back" data-iconpos="top">返回</a>
43          </div>
44          <div class="ui-block-c"> <!--第 3 行 第 3 列-->
45            <a href="#" data-role="button" data-corners="false"
46              data-icon="search" data-iconpos="top">搜索</a>
47          </div>
48        </div>
49      </div>
50  </div>
```

本例使用 jQuery Mobile 的类 ui-grid-b 来定义 3 列的网格布局，第 10~49 行分别指定了 ui-block-a、ui-block-b 和 ui-block-c 这 3 列。

注意：这些列将依次并排浮动，因此当指定了 ui-grid-a|b|c|d|e 时，必须要指定相应的 ui-block-a|b|c|d|e，以便浮动显示相应的列。

在模拟九宫格的列中使用了按钮，data-corners="false"表示不显示圆角，data-iconpos="top"表示图标显示在上方。

10.14 可动态隐藏的页眉和页脚

jQuery Mobile标准的页面显示格式是页眉、中间内容区、页脚区，这也是众多移动应用标准的显示风格。但是当屏幕的内容很多时，页眉和页脚无疑占据了过多的空间。如果能够全屏显示内容区，只在需要时显示页眉和页脚，就非常方便了，比如一些电子阅读类APP中，阅读图书时没有页眉和页脚，单击屏幕时才会出现。jQuery Mobile提供了动态隐藏页眉和页脚的功能，代码如下：

```
01  /* visibleFooterHeader.html*/
02  <!--定义标准的 jQuery Mobile 页面-->
03  <div data-role="page" data-theme="b">
04    <!--固定页眉，并且具有自适应全屏效果展示-->
05    <div data-role="header" data-position="fixed" data-fullscreen="true">
06      <h1>欢迎使用全屏效果的页眉和页脚</h1>
07    </div>
08    <!--页面内容区域-->
09    <div data-role="content"><br><br>
10        <p>当按住此区域时，就可以看到页眉和页脚了</p>
11        <p>当按住此区域时，就可以看到页眉和页脚了</p>
12        <p>当按住此区域时，就可以看到页眉和页脚了</p>
13        <p>当按住此区域时，就可以看到页眉和页脚了</p>
14        <p>当按住此区域时，就可以看到页眉和页脚了</p>
15        <p>当按住此区域时，就可以看到页眉和页脚了</p>
16        <p>当按住此区域时，就可以看到页眉和页脚了</p>
17    </div>
18     <!--页脚区域，并且具有全屏自适应内容-->
19    <div data-role="footer" data-position="fixed" data-fullscreen="true">
20      <h1>滚动条移动时，会自动显示</h1>
21    </div>
22  </div>
23  </div>
```

为了让页眉和页脚能够自动隐藏，并且在滚动或滑动屏幕时显示，第05行设置了data-fullscreen属性为true。

10.15 最简单的手机相册

本例实现一个简单的手机相册，它使用了jQuery Mobile的Listview控件来实现相册的

列表展示效果，首页运行效果如图 10.13 所示。

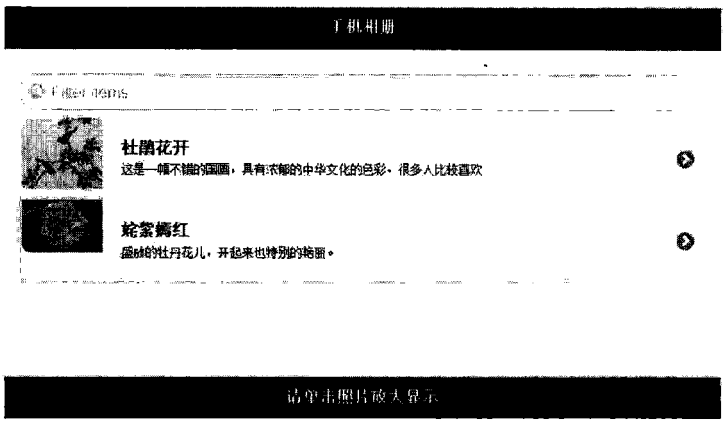

图 10.13　手机相册列表

打开页面时，显示的是一个图片缩略图的列表视图，当单击某中的某一幅图片时，将会进入到全屏放大的图片页面，如图 10.14 所示。

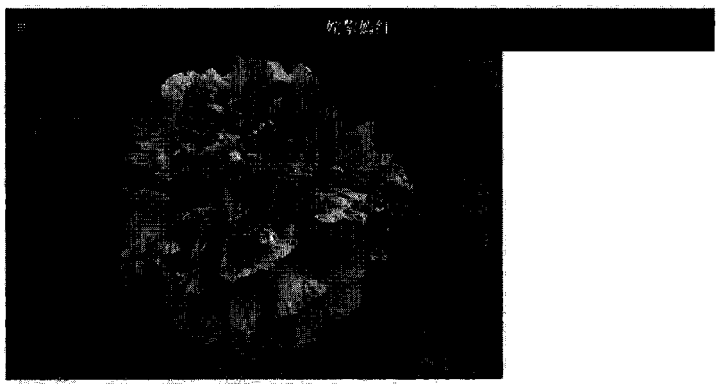

图 10.14　全屏预览窗口

在全屏预览之后，单击左上角的返回按钮，即可返回到首页。首页实际上使用了一个 jQuery Mobile 的列表视图控件，代码如下：

```
01  /* mobilePhotos.html*/
02  <!--相册首页-->
03  <div data-role="page" id="photos">
04    <div data-role="header" data-position="fixed">
05      <h1>手机相册</h1>
06    </div>
07    <!--内容区域-->
08    <div data-role="content">
09      <h2></h2>
10      <!--定义一个 ul，指定其 data-role 为 listview、data-filter 为 true 表示显示搜索框-->
11      <ul data-role="listview" data-inset="true" data-filter="true">
```

```
12          <li>
13            <a href="#flower1">                                    <!--链接到明细的页面-->
14            <img src="images/sample1.jpg" width="100" height="72">  <!--显示缩略图-->
15            <h2>杜鹃花开</h2>
16            <p>这是一幅不错的国画，具有浓郁的中华文化的色彩，很多人比较喜欢</p>
17            </a>
18          </li>
19          <li>
20            <a href="#flower2">                                    <!--链接到明细的页面-->
21            <img src="images/sample5.jpg" width="100">   <!--显示缩略图-->
22            <h2>姹紫嫣红</h2>
23            <p>盛放的牡丹花儿，开起来也特别的艳丽。</p>
24            </a>
25          </li>
26        </ul>
27      </div>
28      <!--页脚区域-->
29      <div data-role="footer" data-position="fixed">
30        <h1>请单击照片放大显示</h1>
31      </div>
32    </div>
```

第11~26行定义了一个Listview，data-inset表示显示圆角列表框，data-filter会自动为列表框添加过滤效果。在每个li元素内部都显示了缩略图以及图片的链接，链接到相应的详细信息页面。

本例构建了两个明细页面，用来显示放大的图片，以flower1页面为例，代码如下：

```
01  /* mobilePhotos.html*/
02  <!--显示放大图片的明细页面-->
03  <div data-role="page" id="flower1">
04      <header data-role="header">
05          <h1>杜鹃花开</h1>
06          <!--在页面的标题栏显示返回链接，指定缩略图为 grid，data-iconpos 为不显示文本-->
07          <a href="#photos" data-icon="grid" data-iconpos="notext">返回</a>
08      </header>
09      <img src="images/sample1.jpg" class="fullscreen" alt="杜鹃花开" />
10  </div>
```

在明细页中，第04行的header区设置了一个不显示文本的返回按钮，然后直接在div中设置了一个img元素，通过class来控制其max-height和max-width各为100%，这样使得图片可以按原尺寸显示。

10.16 在列表框上添加气泡提示

在jQuery中，列表框上还可以出现类似于邮件个数的气泡提示功能。这个功能是通过

CSS 来控制的，只需要在列表项上添加一个 class 为 ui-li-count 的 span 即可实现。本例演示了如何创建一个具有邮件数量提示的气泡效果，如图 10.15 所示。

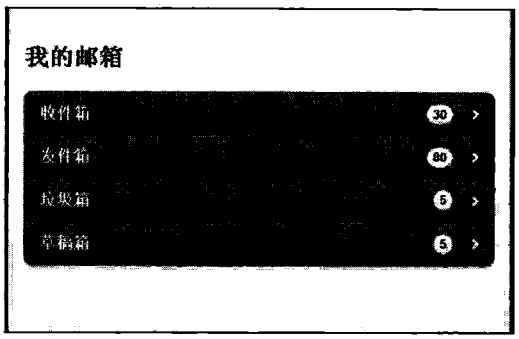

图 10.15　具有邮件数量提示的列表

本例代码如下：

```
01  /* listviewCount.html*/
02  <div data-role="page" id="pageone">
03    <div data-role="content">
04      <h2>我的邮箱</h2>
05      <!--创建一个列表框，用来显示邮箱列表-->
06      <ul data-role="listview" data-inset="true" data-theme="b">
07        <!--为每个邮箱定义邮箱数量的气泡提示-->
08        <li><a href="#">收件箱<span class="ui-li-count">30</span></a></li>
09        <li><a href="#">发件箱<span class="ui-li-count">80</span></a></li>
10        <li><a href="#">垃圾箱<span class="ui-li-count">5</span></a></li>
11        <li><a href="#">草稿箱<span class="ui-li-count">5</span></a></li>
12      </ul>
13    </div>
14  </div>
```

第 08~11 行通过添加 class 为 ui-li-count 的 span，就可以出现气泡提示效果了。

10.17　在列表框上添加自定义按钮

在使用 jQuery Mobile 的 Listview 控件时，可以更加灵活地为每个列表项定制显示的风格，而不是仅在链接元素内部创建要显示的样式。本例将会在每个列表项中添加一个按钮，当单击按钮时，可以进入到下载窗口，单击链接时，又会进入到列表的网站浏览窗口，从而实现了增强的列表框效果。本例代码如下：

```
01  /* listviewButton.html*/
02  <!--构建浏览器下载列表页-->
03  <div data-role="page" id="pageone" data-theme="e">
04    <div data-role="content">
05      <h2>软件下载</h2>
```

```
06      <!--构建圆角显示的列表框-->
07      <ul data-role="listview" data-inset="true" data-filter="true">
08        <!--显示一个分隔条-->
09        <li data-role="divider">市面上主流的浏览器</li>
10        <li>                                    <!--列表框的列表项-->
11          <a href="#">
12          <img src="images/chrome.png">
13          <h2>谷哥浏览器</h2>
14          <p>谷哥Chrome浏览器是由Google公司推出的一款功能强大的,兼容性强的浏览器,
15  Chrome是一款大众喜欢的浏览器</p>
16          </a>
17          <!--创建一个分隔显示的按钮，允许用户下载浏览器-->
18        <a href="#download" data-rel="dialog" data-transition="pop">单击这里下载</a>
19        </li>
20        <li>                                    <!--列表框的列表项-->
21          <a href="#">
22          <img src="images/firefox.png">
23          <h2>火狐浏览器</h2>
24          <p>来自Mozilla的浏览器，是一款开源的，非常标准化的浏览器，是广大网页设计者
25  常用来测试的浏览器，轻量小巧是其特色。</p>
26          </a>
27          <!--创建一个分隔显示的按钮，允许用户下载浏览器-->
28          <a href="#download" data-rel="dialog" data-transition="pop">单击这里下载
29         </a>
30        </li>
31      </ul>
32    </div>
33  </div>
```

页面内定义了一个 data-role 为 Listview 的控件，其中 data-inset 指定显示圆角，data-role 为 divider 创建了一个分隔条。在每个列表项的 li 元素内部，并列排放着两个链接，第 1 个显示浏览器的相关资讯，第 2 个显示一个下载的对话框链接。运行效果如图 10.16 所示。

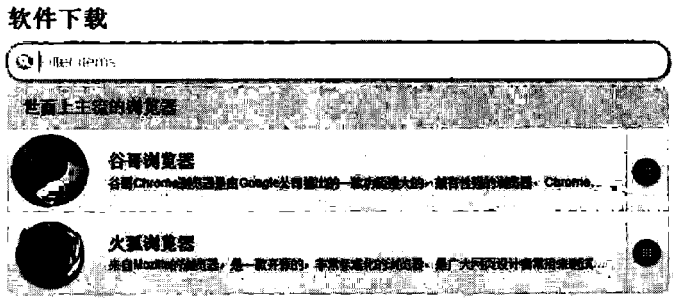

图 10.16 增强效果的列表框

在单击“单击这里下载”的图标时，将会弹出下载对话框，其中有两个内联显示的按钮，允许用户下载或取消下载，代码如下：

```
01  /* listviewButton.html*/
02  <div data-role="page" id="download">
03    <div data-role="content">
04    <h3>请单击这里下载</h3>
05      <p>下载或者是取消下载软件</p>
06      <!--定义 2 个内联显示的按钮，data-rel 指定无论其链接内容，都将会返回到前一页-->
07      <a href="#" data-role="button" data-rel="back"
08      data-icon="check" data-inline="true" data-mini="true">下载</a>
09      <a href="#" data-role="button" data-rel="back"
10      data-inline="true" data-mini="true">取消</a>
11    </div>
12  </div>
```

第 07~10 行添加了两个按钮，如果 href 属性指向了具体的下载位置，将开始下载软件。

注意：如果在按钮或链接中添加了 data-rel="back"属性，那么对于该链接的任何点击行为都是后退行为，此时会无视链接的 href，后退到浏览器历史中的上一个地址。

下载页面的效果如图 10.17 所示。

图 10.17　下载页面

本例只是简单地返回到前一页，并不会真的发生下载行为，也可以去掉 data-rel 属性，并为 href 属性指定具体的下载链接，方便下载软件。

10.18　为列表添加自定义的缩略图图标

对于 Listview 来说，可以通过为列表项的 li 元素更改 data-icon 属性值的方式来更改其显示的链接图标，当然也可以为 img 元素应用 ui-li-icon 来为每个列表项应用一幅自定义的图标。本例演示了如何添加自定义的列表项图标以及如何更改列表项默认的图标，代码如下：

```
01  /* listviewIcon.html*/
02  <div data-role="page">
```

```
    <div data-role="header" data-position="fixed">
        <h1>选择感兴趣的栏目进行查看</h1>
    </div>
    <div data-role="content">
    <!--在内容区中添加一个列表项-->
    <ul data-role="listview" data-inset="true">
        <!--使用 class 为 ui-li-icon 为图片应用缩略图效果-->
        <li><img src="images/firefox.png" class="ui-li-icon">火狐浏览器，非常好用的
小巧的浏览器</li>
        <!--更改列表项默认的图标为加号-->
        <li data-icon="plus">
        <a href="http://www.google.com">
        <!--为图标应用 class 类 ui-li-icon 设置缩略图样式-->
            <img src="images/chrome.png" class="ui-li-icon">谷哥浏览器，不用多说了，
搜索起家的，现在很强
        </a>
        </li>
        <!--更改列表项默认的图标为提示符号-->
        <li data-icon="info">
        <a href="http://www.baidu.com">
        <!--为图标应用 class 类 ui-li-icon 设置缩略图样式-->
            <img src="images/us.png" class="ui-li-icon"> 美国硅谷高科技公司林立
        </a>
        </li>
        <!--隐藏显示默认的图标-->
        <li data-icon="false">
        <a href="http://www.21cn.com">
        <!--为图标应用 class 类 ui-li-icon 设置缩略图样式-->
            <img src="images/germany.png" class="ui-li-icon"> 德国 SAP ERP 软件中
            的老大
        </a>
        </li>
    </ul>
    </div>
    <div data-role="footer" data-position="fixed">
        <h4>世界之最</h4>
    </div>
</div>
```

从代码中可以看出，本例为 li 元素应用了 data-icon 来改变列表右侧显示的图像，而在列表的左侧用 img 元素显示了一个缩略图，使用的类为 ui-li-icon，使得图标会在列表项的缩略图上进行显示，效果如图 10.18 所示。

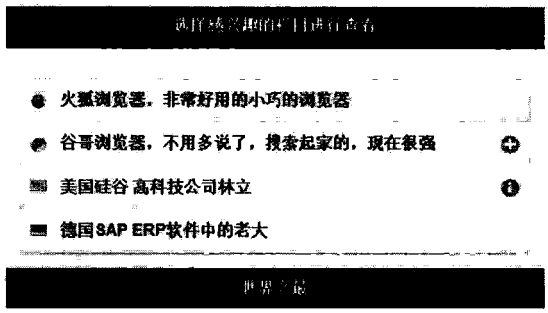

图 10.18　更改列表的图标和缩略图

从图中可以看出，预定义的 ui-li-icon 类可以将图标放到列表项的合适位置上，而 data-icon 属性指定不同的图标值，就可以改变列表项的导向图标。

10.19　创建列表日历的效果

大多数的移动应用都有日历软件，日历软件按日期依次列出了相关的事件。使用 jQuery Mobile 的 Listview，也可以非常方便地实现日历效果。本例演示了如何使用 Listview 来创建一个行事日历，效果如图 10.19 所示。

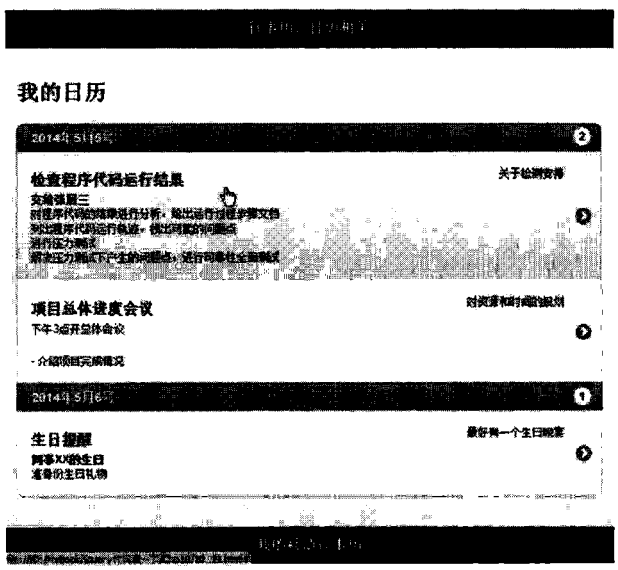

图 10.19　行事日历

本例代码如下：

```
01  /* listviewCalendar.html*/
02  <div data-role="page" id="pageone">
03      <div data-role="header" data-position="fixed">
04          <h1>行事历，日历相关</h1>
05      </div>
```

```
06    <!--页面内容区域，日历的详细信息列表-->
07    <div data-role="content">
08      <h2>我的日历</h2>
09      <!--编写一个列表-->
10      <ul data-role="listview" data-inset="true">
11        <!--列表题头，用于分隔每一天-->
12        <li data-role="list-divider">2014 年 5 月 5 号<span
13                  class="ui-li-count">2</span></li>
14        <li><a href="#">
15          <h2>检查程序代码运行结果</h2>
16          <p><b>交给张居三</b></p>
17          <p>对程序代码的结果进行分析，给出运行过程步骤文档</p>
18          <p>列出程序代码运行轨迹，找出可能的问题点</p>
19          <p>进行压力测试</p>
20          <p>解决压力测试下产生的问题点，进行可靠性全面测试</p>
21           <!--显示在列表右侧的文本-->
22          <p class="ui-li-aside">关于检测安排</p></a>
23        </li>
24        <li><a href="#">
25          <h2>项目总体进度会议</h2>
26          <p>下午 3 点开总体会议</p>
27          <br>
28          <p>- 介绍项目完成情况</p>
29           <!--显示在列表右侧的文本-->
30          <p class="ui-li-aside">对资源和时间的规划</p></a>
31        </li>
32        <!--列表题头，用于分隔每一天-->
33        <li data-role="list-divider">2014 年 5 月 6 号<span
34                  class="ui-li-count">1</span></li>
35        <li><a href="#">
36          <h2>生日提醒</h2>
37          <p><b>同事 XX 的生日</b></p>
38          <p>准备份生日礼物</p>
39          <p class="ui-li-aside">最好有一个生日晚宴</p></a>
40        </li>
41      </ul>
42    </div>
43    <div data-role="footer" data-position="fixed">
44      <h1>我的移动行事历</h1>
45    </div>
46  </div>
```

第 10~41 行在页面的内容区创建了一个 Listview 列表框。在这个列表框的内部，使用 data-role="list-divider"作为分隔符，它用来分隔每一天的事件日志。从图中可以看出 5 月 5 号有两个事件，这里使用了 Listview 的气泡提示效果，每一个事件用一个 li 元素表示，并

使用 class=“ui-li-aside”在列表框的最右侧显示一个日历标题。

10.20 动态创建 listview 列表项

在实际的程序开发过程中，多数情况下都需要动态地向列表框中添加内容。jQuery Mobile 基于 jQuery 创建而来，因此多数语法都与 jQuery 相似，只是 jQuery Mobile 对于控件仍然有一些特定的语法。本例演示了如何通过 jQuery 代码来向 Listview 列表框添加列表项，效果如图 10.20 所示。

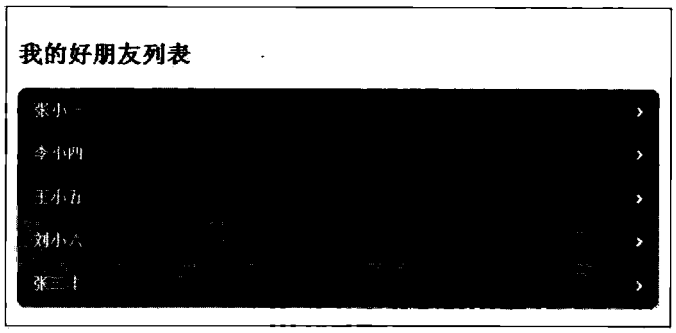

图 10.20 动态加载列表项

本例代码如下：

```
01  /* listviewDynCreate.html*/
02  <div data-role="page" id="pageone">
03    <div data-role="content">
04      <h2>我的好朋友列表</h2>
05      <!--创建一个列表框，这个列表框中不包含任何的元素-->
06      <ul data-role="listview" data-inset="true" data-theme="b" id="myfrilst">
07      </ul>
08    </div>
09    <script type="text/javascript">
10          var myfriends=["张小三","李小四","王小五","刘小六","张三丰"]; //定义一个数组
11          var friendList = $("#myfrilst");                          //获取页面上的 listview
12           $("#pageone").bind("pageshow",function(){                //绑定 pageshow 事件
13             var str = "";
14             for(var i=0, len=myfriends.length; i<len; i++) {       //循环数组
15                   str += "<li><a href=\"#\">"+myfriends[i]+"</a></li>";  //构建列表项
16             }
17             friendList.html(str);                                  //向 listview 添加列表项
18             friendList.listview("refresh");                        //刷新列表项
19           });
20    </script>
21  </div>
```

第06行构建了一个空白的Listview控件，第09~20行定义了一段jQuery代码，其中绑定了页面的pageshow事件，在这个事件触发时，通过循环来构建列表项。第17~18行调用listview.html将列表项添加到列表中，最后刷新列表项，显示动态添加的结果。

注意：实例中将JavaScript代码放到了page元素内部，这样当页面被作为对话框或者是跳转页应用时，JavaScript才可以正确执行。

10.21 动态加载和切换页面

前面的实例所看到的都是使用链接a元素的href属性来指定切换的目标页面，实际开发中，页面的链接经常是不固定的。比如一个登录按钮，需要根据用户输入的用户名和密码的验证结果，来决定切换到哪一页，可以是AJAX异步加载的一个服务器端页面，也可以是一个已经存在的页面。

本例创建一个登录页面，当用户登录成功时，将调用loadPage()来动态加载index.php首页，当用户登录失败时，切换到一个已经定义好的登录失败页面。本例代码如下：

```
01  /* dynamicLoadSwitch.html*/
02      <!--用户登录页面-->
03     <div data-role="page" id="pagelogin"  data-title="登录">
04       <div data-role="header" data-position="fixed">
05           <h1 id="headertitle">登录窗口</h1>
06       </div>
07       <div data-role="content">
08           <div id="login_form">
09             <input id="txtusername" name="txtusername"
10                       value="" placeholder="用户名" type="text">
11             <input id="txtpassword" name="txtpassword"
12                       value="" placeholder="密码" type="password">
13             <a href="#" data-role="button"
14                data-theme="c" id="btn_login">登录</a>
15           </div>
16       </div>
17        <div data-role="footer" data-position="fixed" >
18           <h2>动态切换实例</h2>
19       </div>
20      <script type="text/javascript">
21         //绑定页面显示之前
22         $("#pagelogin").bind("pagebeforeshow",function(){
23              $("#txtusername").val('');                    //清空用户名文本框
24             $("#txtpassword").val('');                     //清空用户密码文本框
25             $('#btn_login').unbind('click');               //取消绑定按钮单击事件
26              //重新绑定按钮单击事件
27             $("#btn_login").bind('click',function(event){
```

```
28                  var usernameStr=$("#txtusername").val(); //获取用户名字符串
29                  var passwordStr=$("#txtpassword").val();   //获取用户密码字符串
30                  //如果用户名和密码验证通过
31                  if (usernameStr=="admin"&&passwordStr=="123"){
32                          $.mobile.loadPage( "index.php", {    //调用 loadPage()加载首页
33                            type: "post",                      //指定 post 提交方式
34                            data: $("#login_form").serialize()
35                          });
36                    }else{
37                        //加载 id 为 pageloginfailed 的页面，显示登录失败信息
38                      $.mobile.changePage("#pageloginfailed",{transition: "none"});
39                    }
40                  });
41          });
42      </script>
43  </div>
```

在第 22~41 行的 pagebeforeshow 事件中，首先验证用户名和密码，验证成功后，第 32 行调用 loadPage()向服务器端发起一个 AJAX 提交请求，获取 index.php 返回的结果数据。如果验证失败，则调用$.mobile.changePage()切换到一个已经定义好的页面上。

注意：page 页面的 data-title 属性可以指定网页的标题，当网页通过 AJAX 异步加载时，使用 data-title 可以确保页面显示正确的标题。

10.22 在页面切换时显示加载进度框

当页面要执行较长时间的任务时，显示一个加载进度框是非常有必要的。jQuery Mobile 提供了$.mobile.loading()函数，可以用来手动显示一个加载提示。本例在切换页面时显示一个进度框，效果如图 10.21 所示。当单击“单击这里加载第 2 页”按钮时，会显示一个加载进度框，当指定操作完成后，会自动隐藏这个加载进度框。

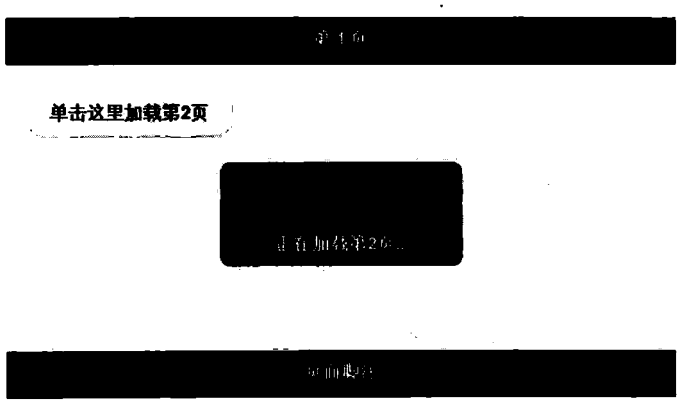

图 10.21 单击按钮时显示加载进度

本例代码如下：

```
/* loadingProgress.html*/
<div data-role="page" id="pageone">
    <div data-role="header">
        <h1>第 1 页</h1>
    </div>
    <div data-role="content">
        <!--放一个按钮，当这个按钮被单击时，显示进度并加载第 2 页-->
        <a href="#" data-role="button" data-inline="true" id="btnloading">
                单击这里加载第 2 页</a>
    </div>
    <div data-role="footer" data-position="fixed">
        <h4>页面脚注</h4>
    </div>

  <script type="text/javascript">
        function showLoading() {              //显示加载进度的函数
            $.mobile.loading('show', {        //使用 loading()函数的参数控制显示
                text: '正在加载第 2 页...',   //显示的文本
                textVisible: true,            //是否显示文本
                theme: 'a',                   //显示的主题
                textonly: false,              //是否仅显示文本
                html: ""                      //HTML 内容
            });
        }
        function hideLoading()
        {
            $.mobile.loading('hide');         //隐藏进度框的显示
        }
       //绑定页面显示之前的事件
       $("#pageone").bind("pagebeforeshow",function(){
            $('#btnloading').unbind('click');   //取消绑定按钮单击事件
            //重新绑定按钮单击事件
            $("#btnloading").bind('click',function(event){
                showLoading();                  //显示加载进度框
                //模拟过了 5000 毫秒之后关闭加载框并切换到 page2 页面
                var tstloading = setTimeout(function(){
                    hideLoading();
                    $.mobile.changePage("#page2",{transition: "slide"});
                },5000);
            });
        });
    </script>
</div>
```

第 16~28 行定义了 showLoading()和 hideLoading()这两个函数，分别调用 $.mobile.loading()来显示进度提示框，通过参数可以配置其显示的外观。在第 30~41 行的 pagebeforeshow 事件中，为按钮 btnloading 关联了 click 事件处理程序，单击时会首先调用 showLoading()显示进度框。第 36~39 行调用了一个定时器函数，在 5000 毫秒之后，调用 hideLoading()隐藏进度显示，并调用 changePage()切换到第 2 页。

10.23 在屏幕旋转时更改显示样式

大多数的移动设备都具有重力感应功能，当移动设备横向放置时，屏幕会自动切换为横向显示，同理，竖向放置时则会竖向显示。jQuery Mobile 提供了 orientationchange 事件，在屏幕方向发生变化时，可以响应该事件来改变屏幕的显示效果。本例就来演示这种功能，代码如下：

```
01  /* screenOrigntation.html*/
02  <script type="text/javascript">
03  $(document).on("pageinit",function(event){          //为页面关联初始化事件
04    $(window).on("orientationchange",function(){      //为窗口关联方向更改事件
05      if(window.orientation == 0)                     //0 表示竖向，更改 CSS 显示效果
06      {
07        $("p").text("方向已经变为竖向！").css({"background-color":"yellow","font-size":"300%"});
08      }
09      else
10      {                                               //否则显示为横向，更改字体大小
11        $("p").text("方向已经变为横向！").css({"background-color":"pink","font-size":"200%"});
12      }
13    });
14  });
15  </script>
16  </head>
17  <body>
18  <!--构建屏幕显示页面-->
19  <div data-role="page">
20    <div data-role="header">
21      <h1>使用 orientationchange 事件</h1>
22    </div>
23     <!--在页面内容区域，简单的放置了几个段落元素-->
24    <div data-role="content">
25      <p>请试着旋转移动设备</p>
26      <p><b>注释：</b>要查看本实例的效果，必须使用移动设备</p>
27    </div>
28    <div data-role="footer">
29      <h1>页脚文本</h1>
```

```
30      </div>
31  </div>
```

第 04~13 行通过响应窗口的 orientationchange 事件，可以捕捉到设备水平/垂直旋转的事件，重点是改变显示的 CSS 样式。

10.24　在列表框中实现加载更多效果

很多的移动应用都有一个“显示更多”的功能，当前仅展示单屏内容，然后给用户一个“显示更多”的按钮，当用户单击这个按钮时，将会通过 AJAX 方式，加载 JSON 数据。比如在一些新闻类 APP 中，一屏只显示 20 条新闻，如果要看更多内容，就点击“显示更多”按钮。本例将模拟一个“加载更多”的功能，用于向 jQuery Mobile 的 listview 添加更多的列表项，效果如图 10.22 所示。

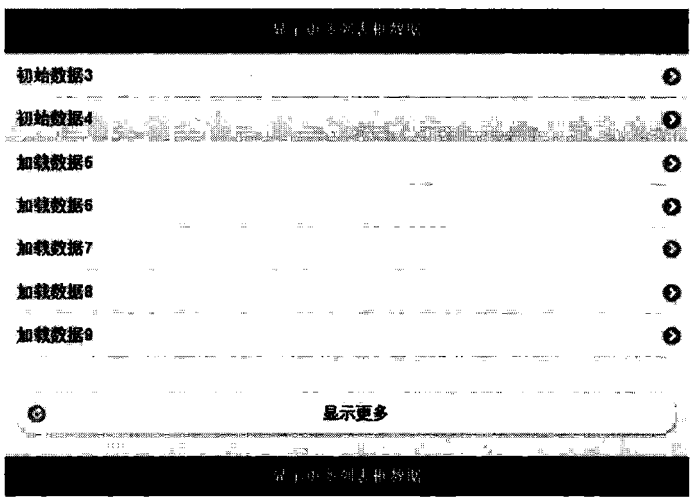

图 10.22　显示更多的效果

本例代码如下：

```
01  /* listviewShowmore.html*/
02  <!--构建一个页面来显示 listview 的初始数据-->
03  <div data-role="page" id="pageone">
04      <div data-role="header" data-position="fixed">
05          <h1>显示更多列表框数据</h1>
06      </div>
07      <div data-role="content">
08        <ul data-role="listview" id="contentList">
09        <li><a href="#">初始数据 1</a></li>
10        <li><a href="#">初始数据 2</a></li>
11        <li><a href="#">初始数据 3</a></li>
12        <li><a href="#">初始数据 4</a></li>
```

```
13          </ul>
14          <p> </p>
15          <div id="morediv">
16          <!--创建一个显示更多的按钮-->
17           <button id="btnmore" data-icon="refresh">显示更多</button>
18          </div>
19        </div>
20        <div data-role="footer" data-position="fixed">
21              <h4>显示更多列表框数据</h4>
22        </div>
23      <script type="text/javascript">
24          var j=5;
25          //绑定页面显示之前的事件
26          $("#pageone").bind("pagebeforeshow",function(){
27              $('#btnmore').unbind('click');                    //取消绑定按钮单击事件
28              //重新绑定按钮单击事件
29              $("#btnmore").bind('click',function(event){
30                      for(i=j;i<=j+5~1;i++){                    //循环向 listview 中添加新的元素
31                          var content = "<li><a href=\"#\">加载数据"+i+"</a></li>";
32                  //调用 append()函数，向 Listview 中添加新元素，并命名用 refresh 刷新控件
33                          $("#contentList").append(content);
34                      }
35                      j+=5;                                     //增加计数器变量
36                      $("#contentList").listview('refresh'); //刷新 listview 的显示
37              });
38          });
39      </script>
40  </div>
```

第 07~19 行在定义的 Listview 中设置一些初始数据，并且有一个“显示更多”的按钮。第 26~38 行为这个按钮关联事件处理程序，可以看到它包含一个全局的计数器变量 j，用来显示序号，然后一次加载 5 个元素，调用 append()将 HTML 内容添加到 Listview 内部，最后刷新 Listview 完成更多内容的加载。

10.25 自定义选择菜单

表单的 select 控件通常用来存放多项选择，比如下拉列表框或多项选择框。jQuery Mobile 提供的表单选择控件对标准 HTML 表单控件进行了一些增强，比如可以多选，可以通过设置 data-native-menu 让其呈现出不一样的显示效果。本例演示了如何使用增强的 select 控件，本例效果如图 10.23 所示。

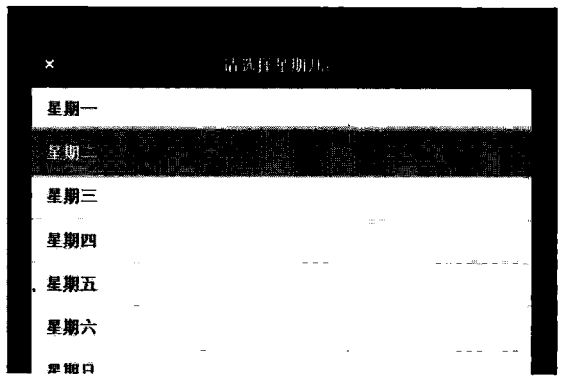

图 10.23 jQuery Mobile 效果的选择框

本例代码如下：

```
/* selectCustomeize.html*/
<!--构建一个表单 select 来选择星期数-->
<div data-role="page" id="pageone">
    <div data-role="header" data-position="fixed">
        <h1>显示更多列表框数据</h1>
    </div>
    <div data-role="content">
      <form id="form1">                                    <!--构建一个表单-->
        <fieldset data-role="fieldcontain">                <!--表单内容分隔容器-->
          <label for="day">请选择星期几：</label>  <!--表单标题-->
          <!--通过指定 data-native-menu，使得表单用一种非原生方式显示-->
          <select name="day" id="day" data-native-menu="false">
           <option value="mon">星期一</option>
           <option value="tue">星期二</option>
           <option value="wed">星期三</option>
           <option value="thu">星期四</option>
           <option value="fri">星期五</option>
           <option value="sat">星期六</option>
           <option value="sun">星期日</option>
          </select>
        </fieldset>
        <!--获取表单选择内容的按钮-->
        <input type="button" data-inline="true" id="btnsend" value="显示您的选择">
      </form>
  </div>
  <div data-role="footer" data-position="fixed">
        <h4>选择框实例</h4>
    </div>
  <script type="text/javascript">
      //在页面初始加载时，对选择框进行初始设置
       $("#pageone").bind("pageinit",function(){
```

```
32              var weekly=new Date().getDay()-1;                    //获取当前星期数
33              var myselect = $("#day");                            //选中 select 控件
34              myselect[0].selectedIndex = weekly;                  //设置选择索引
35              myselect.selectmenu('refresh');                      //刷新 select 控件
36              $('#btnsend').unbind('click');                       //取消绑定按钮单击事件
37              //重新绑定按钮单击事件
38              $("#btnsend").bind('click',function(event){
39                //获取当前选择的文本
40                alert("您当前选择了:  "+myselect.children('option:selected').text());
41              });
42          });
43      </script>
44  </div>
```

在这个实例中，除了 data-native-menu 属性让多项选择以一种更专业化的方式显示外，其他的设置与普通的 select 基本相似。第 34 行通过 selectedIndex 来设置当前选中的索引，在按钮单击时，通过 children('option:selected')这样的语法（第 40 行）来获取当前选中的内容。

10.26 使用多个选择菜单进行组合选择

在实际的应用中，经常需要用到多个选择菜单，比如选择国籍、城市或省份等等。最常见的是日期和时间的选择，比如智能排课系统，经常需要选择排课的日期和排课的时间，则可能需要用到两个选择菜单。本例演示了如何通过两个选择菜单来选择日期，效果如图 10.24 所示。

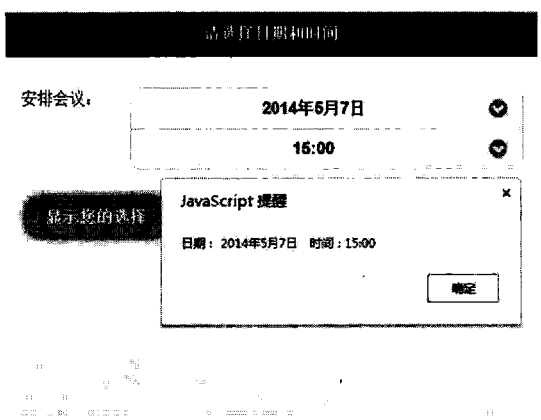

图 10.24 使用多个 select 进行日期和时间选择

本例代码如下：

```
01  /* multiSelectGroup.html*/
02      <div data-role="content">
03          <form id="myform">
04              <div data-role="fieldcontain">
```

```
        <fieldset data-role="controlgroup">
          <legend>安排会议：</legend>
            <label for="day">选择日期：</label>
            <select name="day" id="day">          <!--动态添加日期-->
            </select>
            <!--定义一个选择时间的菜单-->
            <label for="time">选择时间：</label>
            <select name="time" id="time">
              <option value="08">08:00</option>
              <option value="09">09:00</option>
              <option value="10">10:00</option>
              <option value="11">11:00</option>
              <option value="12">12:00</option>
              <option value="13">13:00</option>
              <option value="14">14:00</option>
              <option value="15">15:00</option>
              <option value="16">16:00</option>
            </select>
        </fieldset>
      </div>
      <!--定义一个显示选择项的按钮-->
      <input type="button" data-inline="true" id="btnsend" value="显示您的选择">
    </form>
  </div>
  <script type="text/javascript">
      //在页面初始加载时，对选择框进行初始设置
      $("#pageone").bind("pagebeforeshow",function(){
          var selOpt = $("#day option");          //获取 option 列表
          selOpt.remove();                        //移除现有的 option 数据
          var mydate=new Date();                  //构建一个新的日期
          var myselect = $("#day");               //得到日期选择列表对象
          var mytime = $("#time");                //获取时间选择列表对象
          var content="";
          for(var i=0;i<=10;i++){                 //循环在日期选择列表中加入 10 个日期
          content=mydate.toLocaleDateString();
          myselect.append("<option value='"+content+"'>"+content+"</option>");
          mydate.setDate(mydate.getDate()+1);
          }
          myselect[0].selectedIndex = 0;          //指定日期选择列表的选择项为 0
          mytime[0].selectedIndex=0;
          myselect.selectmenu('refresh');         //刷新日期选择列表
          $('#btnsend').unbind('click');          //取消绑定按钮单击事件
          //重新绑定按钮单击事件
          $("#btnsend").bind('click',function(event){
              //获取当前选择的文本
```

```
50                  alert("日期 :  "+myselect.children('option:selected').text() +
51                    "    时间 : "+mytime.children('option:selected').text());
52                });
53            });
54        </script>
55    </div>
```

第 05 行通过 data-role="controlgroup"将两个 select 控件组合为一个控件组，日期型的 select 里面的日期将在页面显示之前，动态地加载。时间型的 select 控件固定显示。在第 30~53 行的 pagebeforeshow 事件中，通过 JavaScript 的日期类型来获取从今日起后 10 天的日期，然后在第 48~52 行的单击按钮事件中，获取当前选中的日期和时间。

注意：通过编程方式对 select 进行添加、修改或删除时，记得要调用 selectmenu('refresh') 进行刷新，否则最新修改的内容可能不会立即显示到屏幕上。

10.27 使用多选菜单选择多个值

jQuery Mobile 的选择列表支持选择多个值，它以数组的形式返回用户的多项选择，使用这个功能可以让用户轻松地实现多选功能。本例不仅演示了多选功能，还演示了省份和城市的级联选择功能，当选择不同的省份时，可以动态地更新省份下面的城市列表，实现了城市列表的多选功能。本例效果如图 10.25 所示。

图 10.25 列表框的多选

本例代码如下：

```
01  /* multiSelectmultiValue.html*/
02      <div data-role="content">
03          <form id="myform">
04              <div data-role="fieldcontain">
05                  <fieldset data-role="controlgroup">
06                      <legend>选择省份</legend>
07                          <label for="province">省份</label>
```

```
            <!--定义一个选择省份的菜单-->
            <select name="province" id="province">
              <option value="广东">广东</option>
              <option value="广西">广西</option>
              <option value="湖南">湖南</option>
            </select>
            <!--定义一个选择城市的菜单-->
            <label for="city">选择城市</label>
            <!--允许多选，不使用原生的菜单样式-->
            <select name="city" id="city" multiple data-native-menu="false">
            </select>
          </fieldset>
        </div>
        <!--定义一个显示选择项的按钮-->
        <input type="button" data-inline="true" id="btnsend" value="显示您的选择">
      </form>
    </div>
     <script type="text/javascript">
         var area=[              //定义一个城市数组，每一个省份对应一个城市子数组
         {"省份":"湖南","城市":["长沙","常德","益阳","株洲","湘潭"]},
         {"省份":"广东","城市":["广州","深圳","珠海","中山","东莞"]},
         {"省份":"广西","城市":["南宁","桂林","柳州","漓江","北海"]}
         ];
         //在页面初始加载时，对选择框进行初始设置
         $("#pageone").bind("pagebeforeshow",function(){
             var myselect = $("#province");                    //得到省份选择列表对象
             var mycity = $("#city");                          //得到城市选择列表对象
             $('#province').unbind('change');                  //取消绑定选择改变事件
              //重新绑定选择绑定事件
             $("#province").bind('change',function(event){
                 //获取当前选择的文本
               var str=myselect.children('option:selected').text();
               var selOpt = $("#city option");                 //获取 option 列表
                   selOpt.remove();                            //移除现有的 option 数据
                    //循环数——用来查找当前选中的省份
                  for(var i=0, len=area.length; i<len; i++) {;
                    var provstr=area[i]["省份"];
                    if (provstr==str){                         //如果省份匹配
                        cities=area[i]["城市"];                //获取城市数组
                        //循环城市数组，并添加到城市选择列表
                        for (var j=0,clen=cities.length;j<clen;j++){
                    mycity.append("<option value='"+cities[j]+"'>"+cities[j]+"</option>");
                            }
                        }
                    mycity[0].selectedIndex=0;                 //选中第 1 个城市
```

```
53                      myselect.selectmenu('refresh');      //刷新 2 个列表框
54                      mycity.selectmenu('refresh');
55                  }
56              });
57              $('#province').trigger( "change" );          //强制触发省份列表框的 change 事件
58              $('#btnsend').unbind('click');               //取消绑定按钮单击事件
59              //重新绑定按钮单击事件
60              $("#btnsend").bind('click',function(event){
61                  //获取当前选择的省份文本
62                  alert("省份 : "+myselect.children('option:selected').text() +
63                      "   城市 :"+mycity.val());           //获取当前多选的城市数组
64              });
65          });
66      </script>
67  </div>
```

第 17 行为 city 这个 select 列表框定义了 multiple 属性，表示允许多选。city 城市列表框并未静态地添加城市列表，而是在第 37~56 行的省份列表框的 change 事件中，从 area 数组中动态地过滤省份，然后动态地添加到列表框中。在程序中，这种动态级联一般是向服务器端发送一个 AJAX 请求，由服务器端返回 JSON 数组进行添加。在动态添加了城市列表后，调用 refresh 对两个列表框进行刷新，并且使用 trigger()函数强制触发 change 事件，最后为 btnsend 按钮关联事件处理程序，以返回多选的结果。

注意：第 63 行的 mycity.val()将返回一个城市列表数组，通过对这个数组进行循环，就可以分别取出多选的每个数组元素。

10.28 创建双重范围的滑块

在移动开发中，滑块是非常有用的一个控件，它可以让用户通过手指拖动的方式选择一定的范围，特别是一些商品评价或者是价格范围等等。jQuery Mobile 的表单控件提供了一个很有用的滑块控件，只需要指定 input 的 type 为 range 即可。本例将演示如何通过两个滑块控件来实现最大和最小值的双滑块选择功能，效果如图 10.26 所示。

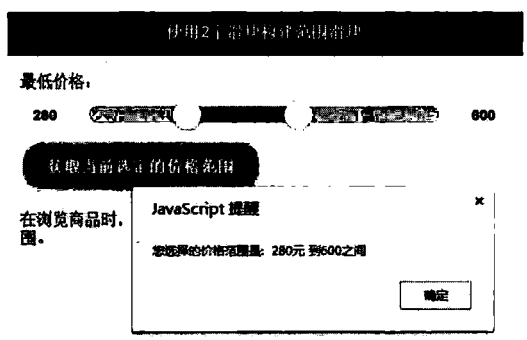

图 10.26 双滑块选择效果

本例代码如下：

```
/* doubleSlip.html*/
<div data-role="page" id="pageone">
  <div data-role="header" data-position="fixed">
  <h1>使用2个滑块构建范围滑块</h1>
  </div>
  <div data-role="content">
    <form id="formprice">
      <div data-role="rangeslider">
        <label for="price-min">最低价格：</label>
        <!--创建一个显示最低价格的滑块-->
      <input type="range" name="price-min"
                id="price-min" value="200" min="0" max="1000">
        <label for="price-max">最高价格：</label>
        <!--创建一个显示最高价格的滑块-->
        <input type="range" name="price-max"
                id="price-max" value="800" min="0" max="1000">
      </div>
       <input type="button" id="btnget" data-inline="true" value="获取当前选定的价格范围">
        <p>在浏览商品时，范围滑块会很有用，它允许用户选择具体的价格范围。</p>
      </form>
  </div>
   <script type="text/javascript">
      //在页面初始加载时，为按钮关联事件处理代码，获取滑块范围
       $("#pageone").bind("pageinit",function(){
           $('#btnget').unbind('click');                    //取消绑定按钮单击事件
           //重新绑定按钮单击事件
           $("#btnget").bind('click',function(event){
             //获取当前的价格范围信息
             alert("您选择的价格范围是:  "+$("#price-min").val()+"元 到
                          "+$("#price-max").val()+"之间");
           });
       });
   </script>
</div>
```

第08~17行创建了两个type为range的范围滑块，每一个滑块的min到max都为0~1000之间，以表示这两个滑块的取值范围，其中 price-min 将用来显示起始价格，price-max 用来显示结束价格，这样就形成了选择区间。第23~32行通过为按钮关联单击事件处理程序，获取选择的价格范围，从而实现了范围选择的编程效果。

10.29 实现开关效果的选择功能

在移动应用中，经常可以看到具有开关选择效果的应用，比如打开蓝牙、开启 WIFI 等等，这类应用随着 iPhone 的流行而频繁地出现在我们生活中。本例演示了如何利用 jQuery Mobile 创建一个模拟蓝牙开启功能的开关，效果如图 10.27 所示。

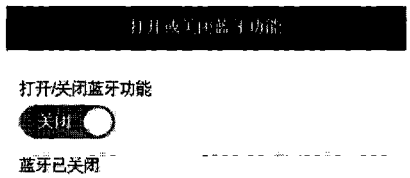

图 10.27 使用开关滑块模拟蓝牙的打开或关闭

本例代码如下：

```
/* switchSlider.html*/
<div data-role="page" id="pageone">
  <div data-role="header" data-position="fixed">
  <h1>打开或关闭蓝牙功能</h1>
  </div>
  <div data-role="content">
    <form id="form1">
      <div data-role="fieldcontain">
        <label for="switch">打开/关闭蓝牙功能</label>
        <select name="switch" id="switch" data-role="slider">
          <option value="on">打开</option>
          <option value="off" selected>关闭</option>
        </select>
      </div>
      <div id="result">蓝牙已关闭</div>
    </form>
  </div>
  <script type="text/javascript">
      //在页面初始加载时，为滑块关联 change 事件
      $("#pageone").bind("pageinit",function(){
          $('#switch').unbind('change');                    //取消绑定 change 事件
          $("#switch").bind('change',function(event){       //重新绑定 change 事件
              //获取滑块当前的值
              var onoff=$("#switch").children('option:selected').val();
              if (onoff=="on"){          //判断当前的值，然后显示打开/关闭消息
                  $("#result").text("蓝牙已打开");
                  }
                  else
```

```
29                          {
30                          $("#result").text("蓝牙已关闭");
31                          }
32                  });
33          });
34      </script>
35  </div>
```

从代码可以看出，滑块实际上是一个 select 控件，不同的是它指定了 data-role="slider" 属性，表示这是一个开关滑块。第 10~13 行在 select 里面包含了两个 option 子元素，用来指示开或关的选项。第 20~33 行是 pageinit 事件处理程序，其中第 22 行为滑块关联了 change 事件，当滑块移动时，便会显示蓝牙打开或关闭的信息。

10.30 创建主题化的表单效果

jQuery Mobile基于主题的外观呈现方式，让开发人员可以创建出非常漂亮的APP外观，而且 jQuery Mobile 的官方网站上还提供了一个主题设计器，让开发人员可以用可视化的方式来设置主题。设置好的主题保存为 CSS 文件后，就可以通过 data-theme 属性来应用新设置的主题。

注意：jQuery Mobile 具有从 a-e 之间的 5 种不同属性，通过使用 data-theme= "a|b|c|d|e" 来指定这 5 种不同的主题，默认情况下页眉和页脚使用 a 主题，页眉内容使用 c 主题。

本例演示了如何使用 data-theme 属性为表单应用各种内置的主题，从而达到吸引用户的效果。本例代码如下：

```
01  /* themedForm.html*/
02  <div data-role="page" id="pageone">
03      <!--页眉使用 e 主题-->
04      <div data-role="header" data-position="fixed" data-theme="e">
05      <h1>创建表单的主题</h1>
06      </div>
07      <!--内容默认使用 e 主题-->
08      <div data-role="content" data-theme="e">
09      <form id="form1">
10      <div data-role="fieldcontain">
11        <label for="name">请输入您的姓名：</label>
12        <!--输入框使用 a 主题-->
13        <input type="text" name="text" id="name"
14                    placeholder="您的姓名..." data-theme="a">
15        <br><br>
16        <label for="search">请输入您要搜索的内容？</label>
17        <!--搜索框使用 d 主题-->
18        <input type="search" name="search" id="search"
19                    placeholder="需要搜索的内容..." data-theme="d">
```

```
20        <br><br>
21        <!--日期选择框使用 c 主题-->
22        <label for="date">请选择您的出生日期：</label>
23        <input type="date" name="date" id="date" date-theme="c">
24        <br><br>
25        <!--下拉列表框使用 b 主题-->
26        <label for="colors">选择喜欢的网站：</label>
27        <select id="colors" name="colors" data-theme="b">
28          <option value="ifeng">凤凰</option>
29          <option value="douban">豆瓣</option>
30          <option value="sina">新浪</option>
31        </select>
32        <br><br>
33        <!--滑块使用 a 主题-->
34        <label for="switch">是否打开蓝牙：</label>
35        <select name="switch" id="switch" data-role="slider" data-theme="a">
36          <option value="on">开</option>
37          <option value="off">关</option>
38        </select>
39        <br><br>
40        <!--复选按钮分别使用 a、b、c 主题-->
41      <div data-role="controlgroup">
42      <legend>请选择喜爱的电影：</legend>
43      <label for="mov1">美国队长 2</label>
44      <input type="checkbox" name="mov1" id="mov1" data-theme="a">
45      <label for="mov2">救火英雄</label>
46      <input type="checkbox" name="mov2" id="mov2" data-theme="b">
47      <label for="mov3">催眠大师</label>
48      <input type="checkbox" name="mov3" id="mov3" data-theme="c">
49      </div>
50    </div>
51    <!--按钮使用 a 主题-->
52    <input type="button" data-inline="true" value="提交表单" data-theme="a">
53    </form>
54    </div>
55  </div>
```

从代码可以看出，本例为不同的表单元素分别指定了 a、b、c、d、e 等几种主题，使得表单的显示效果更加丰富。

jQuery Mobile 的 themeroller 可以创建更多的主题，比如 f、h、i 等等这样的主题，进入网址是：

http://themeroller.jquerymobile.com/

例如可以创建一个粉红色的主题 r，这样就可以使用 data-theme="r"来应用粉红色的效果了。

第 11 章　其他常用代码段

本章将介绍一些使用 jQuery 进行网站开发时常用的代码段，这些代码段结构精炼但能解决很多常见的问题，比如检测 jQuery 插件库是否已经加载、jQuery 版本以及浏览器类型的检测等等。

本章主要涉及的知识点有：

- 检查 jQuery 库是否加载。
- 解决 jQuery 的版本冲突。
- 使用 CDN 加载 jQuery 库。
- jQuery 的 Cookies 管理。
- jQuery 选择器和 DOM 操作的性能优化。

11.1　判断 jQuery 库是否被加载

如果没有加载 jQuery 库，则所有与 jQuery 相关的操作都无法正确进行，对于一些动态生成的页面来说，尤其需要判断 jQuery 库是否加载。

注意：在引用了 jQuery 库之后，会为 window 对象声明一个 jQuery 的全局变量，这样就可以在任何地方调用了。

判断 jQuery 库是否被加载，需要判断页面中是否存在一个 jQuery 对象（即 window.jQuery 对象）。使用 window.jQuery=window.$ 就可以判断是否加载了 jQuery 库，代码如下：

```
01  /* checkjQueryLoaded.html */
02  <title>检测 jQuery 库是否已经加载</title>
03  <script type="text/javascript">
04      //判断是否加载 jQuery，如果未加载则手动写入 jQuery 加载代码
05      window.jQuery||document.write('<script type="text/javascript"
06                                          src="jquery/jquery-1.9.1.min.js"> </script\>');
07  </script>
08  </head>
09  <body>
10  <script type="text/javascript">
11      if (window.jQuery) {      //在页面中判断 jQuery 是否已经加载
```

```
12          document.write("jQuery 已经加载");
13      } else {
14          document.write("jQuery 没有加载");
15      }
16  </script>
```

第 05~06 行判断 window.jQuery，如果未定义的 window.jQuery 为真，使用||运算符返回右侧的值，即动态加载 JavaScript 脚本。

11.2 解决 jQuery 库冲突的方法

在使用 jQuery 开发网页应用时，习惯性的会使用“$”符号来指代 jQuery 的全局变量，比如$("#result")表示引用页面上 id 等于 result 的元素。但是如果在一个页面上引入了多个 jQuery 库时，而在另外的 js 库中也定义了$符号，那么就会引发冲突。

如果在其他库中出现了相同的“$”定义，可以使用 jQuery 库提供的 noConflict()函数。该函数会让 jQuery 放弃对$的所有权，将之交给其他的 js 库。

注意：在使用 noConflict 将$的所有权交给其他库之后，可以使用 jQuery 全局变量来引用 jQuery。

本例构建一个新的$变量，它保存的是一个函数，用来弹出一个消息，代码如下：

```
$=function(msg){
    alert("这是另一个$的用法 :"+msg);
}
```

页面重新定义了$变量，它指向一个匿名函数，下面演示如何使用 jQuery 库解决命名的冲突：

```
01  /* checkjQuerynoConflict .html */
02  <!--注意这里的顺序，将对$的重写放在了第 1 个，这样 jQuery 库的$会覆盖第 1 个-->
03  <script type="text/javascript" src="11_2_1.js"></script>
04  <script type="text/javascript" src="jquery/jquery-1.9.1.min.js"></script>
05  <title>解决 jQuery 冲突</title>
06  </head>
07
08  <body>
09  <div id="result">这是一个 div</div>
10  <script type="text/javascript">
11      jQuery.noConflict();                    //解决命名冲突的函数
12      $("#result");                           //调用的是自定义的$
13      $j=jQuery;                              //重命名 jQuery 库
14      alert($j("#result").text());            //调用 jQuery 库的函数
15  </script>
16  </body>
```

必须注意 jQuery 库与其他库的引用顺序，实例中 jQuery 库中的全局变量会覆盖在

11_2_1.js 中定义的$变量，因此在 body 区中，如果不指定 noConflict()函数，将调用的是 jQuery 的$。

注意：为 jQuery 全局变量重命名也是常见的避免冲突的一个方法。

11.3 jQuery 版本检查方法

在页面运行时，可以判断当前引用的 jQuery 库的版本，对版本进行比较，当 jQuery 库的版本过低时，则显示错误消息。这个功能在一些动态生成的网页程序中非常有用，避免了因不同的版本导致页面不兼容的问题。本例演示了如何完成 jQuery 的版本检查，代码如下：

```
01  /* checkjQueryVersion .html */
02  <title>jQuery 版本检查方法</title>
03  <script type="text/javascript" src="jquery/jquery-1.9.1.js">  </script>
04  </head>
05  <body>
06  <script type="text/javascript">
07  jQuery(function(){
08      if (jQuery.fn.jquery>"1.9.0"){          //使用 jQuery.fn.jquery 可以获取 jQuery 版本
09          alert("jQuery 版本过高");           //如果版本大于 1.9.0，则显示提示消息
10      }
11      else{
12          alert(jQuery.fn.jquery);            //否则仅显示版本
13      }
14  });
15  </script>
```

通过 jQuery.fn.jquery 可以获取到当前运行的 jQuery 版本信息，然后与指定的版本号进行比较，以此来判断当前的 jQuery 版本是否为所期望的版本，如果不是则跳出提示消息以提醒开发人员的注意。

11.4 解决 jQuery 版本冲突问题

如果要在一个页面上引用不同版本的 jQuery 库，特别是一些维护时间比较长，又需要用到 jQuery 库新功能的网页来说，这样的需求很常见，但是由于引入了两个不同版本的库，有可能会导致兼容性问题，此时可以使用 jQuery 的 noConflict()函数将 jQuery 移到一个新的命名空间。

本例演示了当使用不同版本的 jQuery 库时，如何在代码中引用各自版本的库来完成一系列功能，代码如下：

```
01  /* versionConflict.html */
```

```
<!--引用 jQuery 1.7.2 库-->
<script type="text/javascript" src="jquery/jquery-1.7.2.min.js"></script>
<!--创建 jQuery 1.7.2 的全局命名空间，即 jq172 表示 jQuery 全局变量-->
<script> var jq172 = jQuery.noConflict(true); </script>
<!--引用 jQuery 1.9.1 库-->
<script type="text/javascript" src="jquery/jquery-1.9.1.min.js"></script>
<!--创建 jQuery 1.9.1 的全局命名空间，即 jq191 表示 jQuery 全局变量-->
<script> var jq191 = jQuery.noConflict(true); </script>
<script type="text/javascript">
  jq172("document").ready(function(e) {          //使用 jq172 关联页加载事件
    jq172("#jq172").click(function(){            //为页面上的 jq172 链接关联事件处理代码
        alert(jq172.fn.jquery);                  //显示 jq172 的版本
    });
    jq191("#jq191").click(function(){            //为页面上的 jq191 链接关联事件处理代码
        alert(jq191.fn.jquery);                  //显示 jq191 的版本
    });
});
</script>
<title>解决 jQuery 版本冲突的办法</title>
</head>
<body>
<!--当用户单击这 2 个链接时，显示各自不同的调用版本-->
<a href="#" id="jq191">jQuery 1.91 调用</a>
<a href="#" id="jq172">jQuery 1.72 调用</a>
</body>
</html>
```

第 03~09 行分别引用了 jQuery 的 1.7.2 和 1.9.1 这两个版本。可以看到每引用一个不同版本的 jQuery 库后，都会调用 noConflict()函数，传递的参数 true 指示是否允许彻底将 jQuery 变量还原，这个函数返回为 jQuery 变量规定的新名称，从而可以使用 jq172 或 jq191 来引用不同的 jQuery 库。第 24~26 行设计了两个链接，在单击不同的链接时，将会显示使用不同的 jQuery 库的效果，通过 fn.jquery 显示当前所用的 jQuery 库版本。使用 noConflict()这个函数就可以在同一个页面上，使用不同版本的 jQuery 来完成操作。

11.5 如何设置 IE 特有的功能

在编写网页时，不同浏览器的兼容性往往是开发者需要特别注意的事情。本例将演示如何为 IE 浏览器添加特别的功能。比如当客户端的浏览器是 IE 7 时，为 body 添加一个名为 IE 7 的 CSS 样式。下面演示使用 JavaScript、jQuery 和条件注释来为 IE 7 浏览器添加特别的功能，代码如下：

```
/* ieSpecFunction.html */
<title>为 IE 添加特定的功能</title>
```

```
<script type="text/javascript" src="jquery/jquery-1.7.2.js"></script>
<script type="text/javascript">
    <!-- 使用 Javascript 脚本 -->
 var IE7 = window.navigator.userAgent.match(/msie 7.0/gi);      //如果是 IE 7
 if(IE7 != null){
window.onload = function(){document.body.className="IE7";}; //在页面加载时添加样式类
  }

    <!--使用 jQuery 库 -->
    $(document).ready(function(){
        var browser = $.browser.msie;                          //判断是否为 IE
        var version = $.browser.version;                       //获取 IE 版本
        if(browser != null && version == "7.0"){               //如果为 IE 7
          $("body").addClass("IE7");                           //添加样式类
        }
    });
</script>
<!--使用 HTML 条件注释 -->
<!--[if IE 7]>
<script type='text/javascript'>
      window.onload = function(){document.body.className="IE7";};
</script>
<![endif]-->
 </head>
```

本例演示了检测 IE 浏览器的 3 种方法。JavaScript 方法是传统的方法，jQuery 方法具有更多的灵活性，HTML 条件注释方法在过去经常使用，不过已经越来越少见了。

11.6 判断浏览器类型并设置 HTML 元素内容

很多网站都会对不同的浏览器进行优化，以弥补浏览器由于兼容问题所呈现的差异。为此可能需要判断浏览器的类型并设置相应的内容，本例演示了如何使用 jQuery 的 $.browser 来判断浏览器的类型。

注意：在 jQuery 1.9.1 的版本中，已经去掉了$.browser 功能，因此要实现浏览器检测，可以使用 JavaScript 方式，或者是使用 jQuery.migrate 插件来实现浏览器的检测功能。

为了在 jQuery 1.9.1 中使用$.browser 的功能，可以从如下的网址下载 jQuery.migrate 插件：

http://plugins.jquery.com/migrate/

下面的代码演示了如何使用 jQuery.migrate 插件在 jQuery 1.9.1 中检测浏览器：

```
/* checkBrowsersetElement.html */
<title>检测浏览器</title>
```

```
<script type="text/javascript" src="jquery/jquery-1.9.1.js"></script>
<!--使用 jQuery migrate 插件使得 jQuery 1.9.1 支持$.browser-->
<script type="text/javascript" src="jquery/jquery-migrate-1.2.1.js"></script>
<script type="text/javascript">
 $(document).ready(function(e) {
 if($.browser.msie) {                    //判断 IE
    $("#browsetype").append("这是 IE "+$.browser.version);
 }else if($.browser.opera) {             //判断 Opera
    $("#browsetype").append("这是 opera "+$.browser.version);
 }else if($.browser.mozilla){            //判断 Firefox
    $("#browsetype").append("这是 mozilla "+$.browser.version);
 }else if($.browser.safa){               //判断 Safari
    $("#browsetype").append("这是 safa "+$.browser.version);
 }else{                                  //判断 Chrome 或其他
    $("#browsetype").append("这是谷歌或者其他浏览器")
 };
});
</script>
</head>
<body>
<!--显示当前的浏览器类型-->
<div id="browsetype">当前的浏览器类型：</div>
</body>
</html>
```

由于 jQuery 1.9.1 已经移除了对$.browser 的支持，因此第 05 行使用了 jquery-migrate-1.2.1.js 插件为 jQuery 1.9.1 添加这个功能。这样就可以使用$.browser 以及 $.browser.version 来获取浏览器的类型和版本号了，本例在 Firefox 中的运行效果如下：

当前的浏览器类型：这是 mozilla 28.0

11.7 加载 Google CDN 的 jQuery 库

很多网站将 jQuery 库放到支持 CDN 的服务器上调用，以便加速网站的载入速度。这些公共的 CDN 服务器常常又称为 CDN 节点，它具有稳定、高速等特点。

注意：CDN 的全称是 Content Delivery Network，中文全称是内容分发网络，它可以提供更快更稳定的传输速度。CDN 可以实时根据网络流量、节点的连接和负载状况以及用户的距离和响应时间等信息将用户的请求重定向到最快的服务节点上。

不过有时候为了避免因为 CDN 服务器的不可访问而造成网站故障，也会增加一个本地的容错手段。本例演示了如何使用谷歌的 CDN 库，同时又添加本地的容错引用，代码如下：

```
/* googleCDN.html */
<title>加载 Google CDN 库</title>
```

```
<!--先使用 google 的 CDN 库-->
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
<script type="text/javascript">
    //如果 Google CDN 无法使用时，则可以使用本地的 jQuery 库
    if (typeof jQuery === "undefined") {
        var script = document.createElement('script');          //创建 script 标签
        var attr = document.createAttribute('type');            //创建标签的属性
        attr.nodeValue = 'text/javascript';                     //指定属性值
        script.setAttributeNode(attr);
        attr = document.createAttribute('src');                 //指定值
        attr.nodeValue = 'jquery/jquery-1.9.1.min.js';          //指定本地的 jQuery 库
        script.setAttributeNode(attr);
        //将其添加到 head 区
        document.getElementsByTagName('head')[0].appendChild(script);
    }
</script>
</head>
```

第 05 行首先引用了谷歌 CDN 服务器中的 jQuery 库，第 08 行通过判断 jQuery 全局变量是否已定义来确认是否成功引用了 CDN 中的 jQuery 库。如果引用失败，第 10~17 行使用 JavaScript 的 CreateElement()、CreateAttribute()等函数，创建一个到本地 jQuery 库的链接，从而避免了因为 CDN 服务器无法连接导致的网页呈现问题。

11.8 Cookies 应用方法

Cookie 俗称“小甜饼”，是由服务器端发送给浏览器的一些服务器端信息，主要用来辨别用户身份或存储一些用户偏好的信息。

注意：Cookie 以键/值对的方式存储在客户端机器的某个目录下的文本文件内，下次请求同一网站时就发送该 Cookie 给服务器。

JavaScript 的 document 对象提供了 cookie 函数可以用来设置、获取或删除 Cookie。本例演示如何通过 document 对象的 cookie 函数来实现对 Cookie 的管理，代码如下：

```
/* useCookies.html */
<script type="text/javascript" src="jquery/jquery-1.9.1.js"></script>
<script type="text/javascript">
$(document).ready(function(e) {
    $("#btnsetcookie").click(function(){
        setCookie("Demo","我的实例 Cookie 数据",2);          //设置 cookie
    });
    $("#btngetcookie").click(function(){
        alert(getCookie("Demo"));                           //获取 cookie
```

```
10        });
11        $("#btndelcookie").click(function(){
12              deleteCookie("Demo");                              //删除 cookie
13        });
14  });
15  //获取 cookie，参数 name 指定要获取的 cookie 名称
16  function getCookie( name ) {
17        var start = document.cookie.indexOf( name + "=" );        //得到 cookie 字符串中的名称
18        var len = start + name.length + 1;    //得到从起始位置到结束 cookie 位置的长度
19         //如果起始没有值且 name 不存在于 cookie 字符串中，则返回 null
20        if ( ( !start ) && ( name != document.cookie.substring( 0, name.length ) ) ) {
21              return null;
22        }
23        if ( start == -1 ) return null;                          //如果起始位置为-1 也为 null
24        var end = document.cookie.indexOf( ';', len );   //获取 cookie 尾部位置
25        if ( end == -1 ) end = document.cookie.length;  //计算 cookie 尾部长度
26        return unescape( document.cookie.substring( len, end ) );       //获取 cookie 值
27  }
28  //设置 cookie，name 为名称，value 为值
29  //expires 为过期日，path 为路径，domain 为域名，secure 为加密
30  function setCookie( name, value, expires, path, domain, secure ) {
31        var today = new Date();
32        today.setTime( today.getTime() );
33        if ( expires ) {
34              expires = expires * 1000 * 60 * 60 * 24;                //计算 cookie 的过期毫秒数
35        }
36        //计算 cookie 的过期日期
37        var expires_date = new Date( today.getTime() + (expires) );
38        //构造并保存 cookie 字符串
39        document.cookie = name+'='+escape( value ) +
40              ( ( expires ) ? ';expires='+expires_date.toGMTString() : '' ) + //expires.toGMTString()
41              ( ( path ) ? ';path=' + path : '' ) +
42              ( ( domain ) ? ';domain=' + domain : '' ) +
43              ( ( secure ) ? ';secure' : '' );
44  }
45  //删除 cookie，必须先获取指定名称的 cookie，然后让 cookie 过期
46  function deleteCookie( name, path, domain ) {
47        if ( getCookie( name ) ) document.cookie = name + '=' +
48                    ( ( path ) ? ';path=' + path : '') +
49                    ( ( domain ) ? ';domain=' + domain : '' ) +
50                    ';expires=Thu, 01-Jan-1970 00:00:01 GMT';
51  }
52  </script>
53  </head>
```

上述代码分别定义了 setCookie()、getCookie()、deleteCookie() 3 个函数，只需要传入相应的参数，这些函数就会自动地获取或设置 document.cookie 值，从而完成对 Cookie 的添加、删除和修改操作。

11.9 使用 cookie.js 管理 Cookies

cookie.js 是 jQuery 插件库中的一个插件，可以非常方便地设置和管理 Cookie，下载地址是：

http://plugins.jquery.com/cookie/

cookie.js 简化了管理 Cookies 的复杂性，它允许用户使用简单的类似$.cookie 这样的语法来设置 Cookie。本例演示了如何使用 cookie.js 来获取和设置用户登录信息，代码如下：

```
01  /* cookiJsCookies.html */
02      <title>cookie 保存用户名密码</title>
03      <script type="text/javascript" src="jquery /jquery-1.9.1.js"></script>
04      <!--插件必须放在 jquery 下面，以便引用 cookie.js 中的功能-->
05      <script type="text/javascript" src="jquery/ jquery.cookie.js"></script>
06      <script type="text/javascript">
07        $(document).ready(function () {
08              $("#button").click(function() {
09                  if ($("#rember_me").attr("checked") == "checked") { //如果指定“记住我”功能
10                      var username = $("#username").val();          //获取用户名
11                      var password = $("#password").val();          //获取密码
12                      //设置 Cookie，指定过期日为 7 天
13                      $.cookie("username", username, { expires: 7 });
14                      $.cookie("password", password, { expires: 7 });
15                  }
16              });
17              //启动时获取 Cookie 中保存的用户名和密码信息
18              var username = $.cookie("username");          //获取用户名
19              var password = $.cookie("password");          //获取密码
20              if (username && password) {                   //如果存在用户名和密码
21              alert("您已经登录");
22              } else {
23              alert("还没有登录，请输入用户名和密码");
24              }
25            });
26      </script>
```

第 05 行引用了 jquery.cookie.js 插件来获取和设置 Cookie。在“登录”按钮被单击时，第 09 行检查是否勾选了“记住我”复选框，如果勾选，第 10~14 行调用$.cookie 设置用户名和密码，并指定过期日期为 7 天。以后在每次页面加载时，都会自动从 Cookie 中获取用户名和密码，并通过这两个值在 Cookie 中是否存在来判断用户是否已经登录。

11.10 让 Cookies 在 N 分钟后过期

Cookie 的有效期默认是从创建 Cookie 的日期开始，如果没有指定$.cookie 的参数，所创建的 Cookie 有效期默认到用户关闭浏览器为止，因而被称为是会话 Cookie。如果指定了 expires 参数，比如{expires:7}表示 7 天后过期，也就是说，Cookie 默认是以天为单位设置过期日期的。也可以使用如下的方式设置以分钟为过期单位，代码如下：

```
01  /* cookieExpires.html */
02  <title>设置 Cookie 在 N 分钟后过期</title>
03  <script type="text/javascript" src="jquery/jquery-1.9.1.js"></script>
04  <!--插件必须放在 jquery 下面，以便引用 cookie.js 中的功能-->
05  <script type="text/javascript" src="jquery/jquery.cookie.js"></script>
06  <script type="text/javascript">
07      //设置以分钟为单位的 Cookie 过期时间
08      function setCookieExpires(name,value,seconds){
09              var mydate =new Date()
10              //设置当前的过期时间
11              mydate.setTime(mydate.getTime() + (seconds * 60 * 1000));
12              $.cookie(name, value, { expires: mydate }); //设置 Cookie 过期时间以分钟为单位
13      }
14      setCookieExpires("Demo","一个实例",20);          //调用实例
15  </script>
16  </head>
```

本例的功能主要是通过 JavaScript 的 setTime()来实现，它以毫秒为单位将时间值相加，从而得到了想要的过期日期值，然后将这个日期值传给 expires 参数，从而实现了 N 分钟后过期的 Cookie。第 12 行将$.cookie 的过期日期设置为分钟。

11.11 如何删除 Cookies

通过为 cookies.js 的 expires 参数设置为-1，可以删除一个 Cookie，代码如下：

```
01  /* cookieExpires.html */
02  <script   type="text/javascript">
03      $.cookie('my_cookie', '', { expires: -1 });      //删除 my_cookie
04      $.cookie('the_cookie', null);                    //删除 the_cookie
05  </script>
```

本例用了两种方法来删除 Cookies：一种是设置其过期参数为-1，这会立刻删除 my_cookie；另一种是设置 the_cookie 的值为 null，这就使得这个 Cookie 的内容被清空，从而被删除。

11.12 获取当前页面的URL并添加样式

在创建网页时，导航栏需要感知当前的页面，从而对当前页面的导航链接进行特殊显示，以表示用户正处于当前的位置。要实现这样的效果，需要使用 window.location.href 来获取当前的 URL。

注意：使用 window.location 可以获取到当前 URL 的地址、端口、协议、哈希值以及主机信息。

```
01  /* getUrlandSetCSS.html */
02  <script type="text/javascript">
03  $(document).ready(function () {
04      //定义 url，一般是 li 下 a 元素的 href 值
05      var domainUrl = ['a.html','b.html','c.html'];
06      $("#menu li").each(function (i) {
07          //判断当前的网页是否为链接所在的页面
08          if (window.location.href.toLowerCase().indexOf(domainUrl[i]) > 0) {
09              $(this).addClass("onhover");    //为当前的页面应用样式
10          }
11      });
12  });
13  </script>
14
15  <style type="text/css">
16      .onhover{       /*网页处于当前页面时的样式，可以定制这个样式*/
17          font-size:14px;
18      }
19  </style>
20  </head>
21
22  <body>
23  <!--网页的导航链接-->
24  <div id="menu">
25  <ul>
26  <li><a href="a.html">工作计划</a></li>
27  <li><a href="b.html">学习计划</a></li>
28  <li><a href="c.html">运动计划</a></li>
29  </ul>
30  </div>
```

第 26~28 行的导航菜单具有 3 个链接，第 05 行定义一个 domainUrl 数组包含了这几个链接，第 06~11 行对导航链接所在的 li 元素进行循环，将当前网页的页面 URL 与导航链接中的页面进行比较。如果匹配当前页，第 08~09 行为链接添加 onhover 样式，实现了当前页面的突出显示效果。

11.13 向表格追加一行数据

使用 JavaScript 向表格动态添加数据比较复杂，需要编写较多代码，有了 jQuery 的辅助，这个工作就轻松多了，还可以使用 jQuery 的 live 语句为动态添加的表格行指定事件，这样就可以用极少的代码量来完成整个工作。本例演示了如何使用 jQuery 来向表格尾部添加一行数据，并且可以使用提供的按钮删除表格行数据，效果如图 11.1 所示。

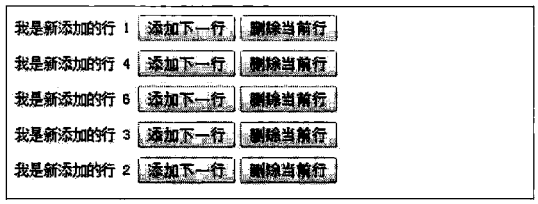

图 11.1 向表格添加行

本例代码如下：

```
/* addRowToTable.html */
<script  type="text/javascript" src="jquery/jquery-1.9.1.min.js"></script>
<!--jQuery 1.9.1 取消了 live，因此使用 migrate 插件使其可以使用 live-->
<script type="text/javascript" src="jquery/jquery-migrate-1.2.1.js"></script>
<script type="text/javascript">
var i=1;
$(document).ready(function () {
    $(".j_del").live("click",function(){                    //关联删除功能
        $(this).parent().parent().eq(0).remove();           //删除按钮当前行
    });
    $(".j_add").live("click",function(){                    //关联添加按钮功能
        //在当前行的下面添加一个新行
        $(this).parent().parent().after("<tr><td>我是新添加的行 "+i+"</td><td><input
        class=\"j_add\" type=button value=\"添加下一行\" />
        <input class=\"j_del\" type=button value=\"删除
        当前行\" /></td></tr>");
        i++;                                                //递增计数器
    });
});
</script>
</head>
<body>
<div>
<table>
<tr>
    <td>请单击右边的按钮添加行</td>
    <td>
        <input type="button" class="j_add" value="添加下一行" />
```

```
29              <input type="button"   class="j_del"   value="删除当前行" />
30          </td>
31      </tr>
32      </table>
```

从代码中可以看出，table 元素中的每个按钮都具有 class 等于 j_add 或 j_del 的标识，jQuery 通过这两个 CSS 样式类来进行选择。

注意：jQuery 1.9.1 取消了 live 功能，为了使用 live()函数，通过 jquery.migrate.js 来实现向下兼容。

对于动态添加的表格行，第 08 行通过 live()函数为其关联单击事件，这个实例最重要的是对 jQuery 父项选择器的应用，第 13~16 行通过两个 parent()获取表格行对象。

11.14 获取客户端 IP

JavaScript 自身并没有附带如何获取当前客户端 IP 地址的功能，不过借助于网络上的一些服务器端工具，也可以轻松地获取本地 IP。本例将使用 jQuery 的 AJAX 来请求 chaxun.1616.net 中的 IP 地址服务以获取本例 IP 信息，并显示到客户端，如图 11.2 所示。

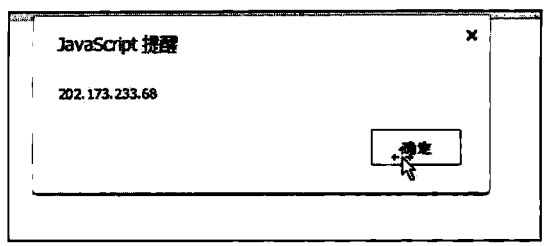

图 11.2 显示客户端的 IP 地址

本例代码如下：

```
01  /* getCientIP.html */
02  <script language="javascript">
03  jQuery(function($){
04      //设定要请求的目标地址的 url
05      var url = 'http://chaxun.1616.net/s.php?type=ip&
06      output=json&callback=?&_='+Math.random();
07      $.getJSON(url, function(data){        //请求 json 数据
08          alert(data.ip);                   //显示客户端的 IP
09      });
10  });
11  </script>
```

第 07 行通过异步的方法请求 chaxun.1616.net 网站上的 s.php 来获取客户端 IP 地址，从返回的 JSON 数组中提取 IP 属性，从而显示了最终的 IP。

11.15 向 Firebug 的控制面板发送消息

说起 Firefox 中的插件，Firebug 无疑是最好的插件之一。Firebug 的控制台窗口允许开发人员查看当前网页的详细信息，比如网页的 HTML 代码、布局模型、JavaScript 脚本等等，还可以对 JavaScript 代码进行单步跟踪，大大方便了网站的开发人员。

在调试 JavaScript 代码时，可以使用 Firebug 的控制台功能。安装了 Firebug 之后，可以在 JavaScript 代码中使用 console.log 在控制台中显示信息，如本例使用了 console.log 来输出程序运行时的相关信息：

```
/* logConsole.html */
<script type="text/javascript">
$(document).ready(function() {
$.ajax({                                    //发起异步请求
    url: 'ParseXML.xml',                    //请求指定的 XML 文件
    dataType: 'xml',                        //指定格式为 xml
    success: function(data){                //如果请求成功
        $(data).find("people").each(function() {                    //循环提取节点
            var field = $(this);
            var fNumber = field.attr("number");                     //读取节点属性
            var fFullname = field.find("fullname").text();          //读取子节点的值
            var fAge=field.find("age").text();                      //读取年龄值
          console.log(fNumber+" "+fFullname+" "+fAge);              //在 firebug 上显示日志消息
          console.info(fNumber+" "+fFullname+" "+fAge);             //在消息面板显示信息
          console.debug(fNumber+" "+fFullname+" "+fAge);            //在调试面板显示信息
          console.warn(fNumber+" "+fFullname+" "+fAge);             //在警告面板显示信息
          console.error(fNumber+" "+fFullname+" "+fAge);            //在错误面板显示信息
            });
        }
    });
})
</script>
```

本例将 XML 解析后的结果使用 console.log 输出到 Firebug 的控制台上，同时也使用了 info()、debug()、warn()、error()等控制台函数显示消息。本例运行时在 Firebug 控制台上的显示界面如图 11.3 所示。

通过切换“错误”、“警告”、“消息”等面板，可以看到使用不同的 console 语句输出的消息类型，灵活地使用 Firebug 能够大大方便 JavaScript 代码的编写和调试。

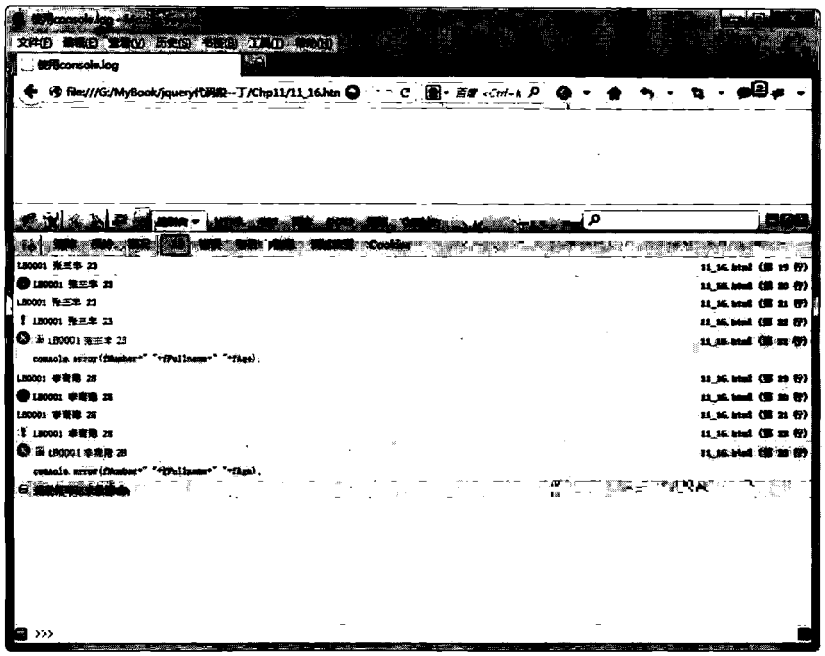

图 11.3 在 firebug 控制台上显示消息

11.16 根据不同的屏幕大小显示不同的网页

当我们创建了一个设计精美的网页后，总是希望用户无论在任何屏幕、任何浏览器中都能呈现相同的效果，尽管有很多自适应的办法可以实现同一网页在不同屏幕下的自适应，但有时网页必须量身定做才最适合需求。本例演示了如何根据不同的屏幕分辨率来显示不同的页面，代码如下：

```
/* fittoScreenSize.html */
<script type="text/javascript">
    //下面的常量定义了不同的分辨率所显示的页面
    pageFor640 = "pageFor640.html";
    pageFor800 = "pageFor800.html";
    pageFor1024 = "pageFor1024.html";
    pageForLarger = "pageForLarger.html";
    showAlert = "yes";
    sayOnAlert = "重定向最大宽度的页面.....";
    var Wide = screen.width;                    //获取屏幕宽度
    if (Wide <= 640){                           //如果是 640x480 分辨率
    if (showAlert == "yes"){
    alert(sayOnAlert);
    }
    window.location = pageFor640;               //重定向到 640 宽的页面
    }
```

```
        else if (Wide <= 800){                  //如果是 800x600 分辨率
        if (showAlert == "yes"){
        alert(sayOnAlert);
        }
        window.location = pageFor800;           //重定向到 800 宽的页面
        }
        else if (Wide <= 1024){                 //如果是 1024x768 分辨率
        if (showAlert == "yes"){
        alert(sayOnAlert);
        }
        window.location = pageFor1024;          //重定向到 1024 宽的密度面
        }
        else {                                  //如果比 1024 还要宽
        if (showAlert == "yes"){
        alert(sayOnAlert);
        }
        window.location = pageForLarger;        //重定向到最大宽度的页面
        }
//-->
</script>
```

本例使用 screen.width 获取屏幕的宽度，然后根据这个宽度来选择合适的页面，如果指定的 3 种分辨率不能满足要求，则会显示 pageForLarger.html 页面，这是为所有更大尺寸的屏幕所准备的标准页面。

11.17 jQuery 遍历对象的属性

在 JavaScript 中，对象的属性可以看作是具有键/值对的数组，因此使用 jQuery 的$.each()可以很容易地遍历一个对象的所有属性。本例演示了如何使用 jQuery 遍历一个自定义对象的属性，代码如下：

```
/* forLoopObjects.html */
<script type="text/javascript">
    var object =                                //定义一个对象
    {
        姓名:"丁哥",
        性别:"男",
        级别:'XXL 级'
    };
    $.each(object, function(name, value)        //使用$.each()遍历对象
    {
        console.log(name+':'+value);            //其中 name 表示键，value 表示值
    });
</script>
```

第 09~12 行的$.each()要遍历的并不是一个数组或一个列表，而是一个对象，于是对象的属性和值就作为回调函数的 name 和 value 参数分别被提取出来。第 11 行使用了 console.log 将结果写到 Firebug 的控制台上，如图 11.4 所示。

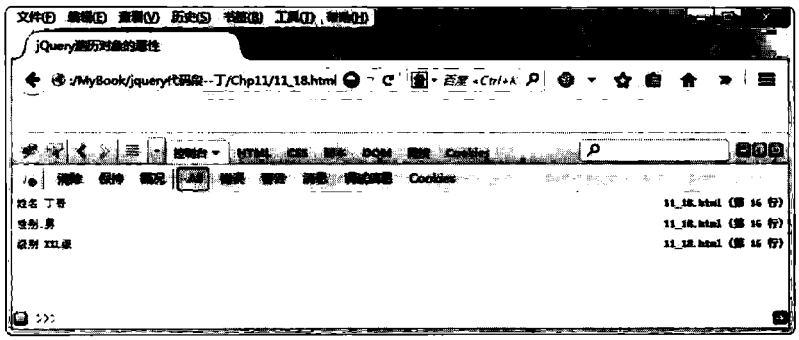

图 11.4 在控制台中显示对象的遍历结果

11.18 最优化的循环语句写法

在编写循环代码时，应尽可能地将循环所需要的函数或子句结果缓存起来，以避免因为多次调用而导致的性能损失。

注意：jQuery 选择器如果需要多次使用，也要将 jQuery 的选择结果缓存为一个变量，避免重复多次的使用。

本例列举了一些常见的循环语句，对其性能影响也进行了详细的标识，希望能够引起读者的重视，代码如下：

```
01  /* betterLoop.html */
02  <title>最优化的循环语句的写法</title>
03  <script type="text/javascript">
04  //慢，因为每次循环都要计算数组的长度
05  for(var i = 0; i < my_array.length; i++){
06  }
07  //也很慢，因为每次循环都要调用函数
08  for(var i = 0; i < myMethod(); i++){
09  }
10  //快，先将数组的长度缓存到变量中，不需要多次调用
11  var length = my_array.length;
12  for(var i = 0; i < length; i++){
13  }
14  //也很快，函数的结果被缓存到了变量中，会很快
15  var length = myMethod();
16  for(var i = 0; i < length; i++){
17  }
```

```
18  //如果非要把代码都写到一行上，这样写也很快，虽然使用了表达式，但这样写也只计算一次
19  for(var i = 0, length = my_array.length; i < length; i++){
20  }
21  </script>
```

从代码中可以看出，通过将数组的长度或函数调用的结果缓存起来，可以得到较快的执行速度。如果代码写到一行上，可以先将赋值语句写在条件语句的前面，这样也能得到较好的执行效率。

11.19 如何构建最优化的字符串

在编写 JavaScript 或 jQuery 代码时，很多读者都会习惯性地使用“+”号来连接多个字符串以形成一个字符串。JavaScript 中的字符串都是 String 对象，当使用“+”运算符合并两个字符串时，实际上 JavaScript 会在原字符串基础之上产生一个新的字符串。如果网页上存在大量的字符串连接，将会非常消耗性能。因此在编程时要尽量避免直接用加号连接字符串，最好的办法是使用堆栈数组。本例演示了如何将加号连接的字符串转化为堆栈数组，代码如下：

```
01  /* betterString.html */
02  <title>使用堆栈数组连接字符串</title>
03  <script type="text/javascript">
04  //不推荐使用如下用“+”号连接的方式
05  var string    = '张三丰';
06        string += '李四友';
07        string += '刘五七';
08        string += '张五哥';
09        string += '李世民';
10  alert(string);                    //显示连接后的字符串
11  //本例先构建了一个 Array 对象
12  var string = Array();
13        string.push('张三丰');      //使用 push()添加数组元素
14        string.push('李四友');
15        string.push('刘五七');
16        string.push('张五哥');
17        string.push('李世民');
18  alert(string.join(""));           //用 join()合并数组元素为字符串
19  </script>
```

第 05~10 行使用+=连接了多个字符串，每一次连接，实际上都是创建一个新的 String 对象，这样极其占用内存资源。第 12~18 行的第 2 个实例先新建了 Array 数组对象，然后调用其 push()函数压入数组元素，最后调用数组的 join()函数，将数组元素转换成了字符串。

同样地，当要连接较多的字符串时，使用堆栈数组的方法，往往会获得较好的性能。

11.20　使用 jQuery 产生 GUID 值

GUID 是指全球唯一标识（Globally Unique Identifier），是由一个特定的算法创建的一个唯一的标识，GUID 值就是这个唯一的标识码。

注意：GUID 的格式为“xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx”，其中每个 x 是 0~9 或 a-f 范围内的一个 4 位十六进制数。例如：6F9619FF-8B86-D011-B42D-00C04FC964FF 即为有效的 GUID 值。

GUID 值常被用来作为一些表的主键 ID 或是一些标识符使用，在网站开发中，也会经常在 URL 中看到 GUID 的身影。使用 jQuery 也可以创建 GUID，这样在创建数据库应用时，就可以使用这个功能在客户端动态地生成 GUID，而不用借助数据库的功能了。本例演示了如何使用 jQuery 创建 GUID 值，代码如下：

```
01  /* createGUID.html */
02  <script type="text/javascript">
03  $(document).ready(function(e) {
04      alert($("body").newguid().val());    //显示随机生成的 GUID 值
05  });
06   (function($){                          //在匿名函数中扩展 jQuery 库的实例函数
07       $.fn.newguid = function () {       //newguid 用来产生一个新的 GUID 值
08           this.val('xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function (c)
09  { var r = Math.random() * 16 | 0, v = c == 'x' ? r : r & 0x3 | 0x8; return v.toString(16); }));
10           return this;
11       };
12  }(jQuery));
13  </script>
```

第 07~11 行为 jQuery 实例创建了一个新的函数 newguid，这个函数具有一个属性 val，这个属性使用了数机函数产生值，然后用 replace()进行替换的方式，产生出一个 GUID 值。网页运行时，实例会调用 newguid 的 val 属性返回一个随机值，效果如图 11.5 所示。

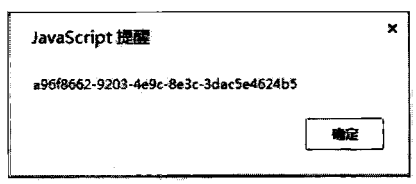

图 11.5　产生 GUID 值

11.21　使用 jQuery 实现聚合函数

聚合函数在数据库 SQL 语句中应用得比较多，比如可以获取元素个数、最大的元素、最小的元素、平均元素等等。它对一组数据中的某列执行计算，并返回单个值。由于 jQuery

的选择器也是以集合为单位，因此可以构造一些自定义的聚合函数来实现特殊的功能。本例演示了如何用 jQuery 创建一个用于统计的 count()函数，代码如下：

```
01  /* jQueryAggregate.html */
02  <script type="text/javascript">
03  $(document).ready(function(e) {
04     var i= $("a").count();                    //调用自定义的聚合函数
05     alert(i);                                 //显示聚合函数
06  });
07  (function($) {                               //定义一个 jQuery 插件实例函数
08        $.fn.aggregate = function(seed, func) {
09              var _r = seed==null?"":seed;     //获取传入的种子值
10               //循环选中的元素，调用 func 参数执行代码
11              this.each(function(index, element) {
12                    _r = func(_r, element);
13              });
14              return _r;
15        };
16        $.fn.count = function() {                //定义 count 聚合函数
17               //调用 aggregate 进行选中的元素个数统计
18              var mycount=this.aggregate(0, function(result, _) { // func
19                        return   result + 1;
20                    });
21              return mycount;                   //返回统计的个数
22        };
23  })(jQuery);
24  </script>
```

第 08~15 行定义了 aggregate jQuery 插件函数，这个函数会对选中的元素进行循环计算，但是计算所用的 func 参数是由 count()函数传入的，count()用来对选中的元素进行计数。同样的道理，可以编写自己的 min()、max()、avg()、sum()等函数来实现自定义的聚合函数。

11.22 用 jQuery 打印网页的特定区域

在编写网页版应用系统时，经常会需要打印功能，特别是对于一些单据的打印，有时只需要打印特定的网页部分。浏览器的打印功能默认情况下会打印 body 区中所有的 HTML 内容，但是通过动态地为 body 区的 HTML 赋不同的值，再调用打印功能就能实现打印特定的区域了。本例演示了如何打印页面上的一个 div 区域，代码如下：

```
01  /* printArea.html */
02  <script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>
03  <script type="text/javascript">
04  function printHtml(html) {                          //定义一个函数用来打印 HTML
05        var bodyHtml = document.body.innerHTML;       //保存当前 body 区的 HTML
```

```
06        document.body.innerHTML = html;              //body 区赋选中的 HTML
07        window.print();                              //调用打印功能
08        document.body.innerHTML = bodyHtml;          //恢复 body 区原来的 HTML
09    }
10    function onprint() {                             //打印按钮所执行的代码
11        var html = $("#printArea").html();           //获取打印区域的 HTML
12        printHtml(html);                             //打印 HTML 内容
13    }
14    </script>
15    </head>
16    <body>
17    <div>
18        <!--定义所要打印的 div 区域-->
19        <div id="printArea" style="width: 500px; text-align: left;">
20            这个区域是定义的打印区域
21        </div>
22        <br />
23        <div>
24        <!--定义打印所要的按钮-->
25        <input type="button" id="btnPrint" onclick="onprint()" value="print" />
26        </div>
27    </div>
```

第 04~09 行定义了 printHtml()函数，它先将 body 部分的 HTML 代码保存到一个变量中，然后将选中的 div 中的 HTML 代码赋给 body 的 HTML，再调用 window.print()打印这个 HTML，打印之后恢复 body 的 HTML，从而实现特定区域的打印效果。

11.23 禁止表单被提交

如果表单具有一个提交按钮，即 type="submit"，当用户单击这个按钮时，会自动将表单内容提交到 action 指定的服务器端地址，有时需要禁用表单的默认提交动作，比如要编写一段 AJAX 异步提交代码，就要禁用表单的默认提交。此时可以使用 preventDefault()函数禁用表单的默认提交，本例演示了如何实现这样的功能，代码如下：

```
01    /* disableSubmit.html */
02    <script type="text/javascript" src="jquery/jquery-1.7.2.min.js"></script>
03    <script type="text/javascript">
04    $(document).ready(function(){
05      $("form").submit(function(e){
06        e.preventDefault();                    //禁用表单的默认提交动作
07        alert("已经禁用表单的自动提交");       //弹出提示消息
08      });
09    });
10    </script>
```

```
11  </head>
12  <body>
13  <!--定义一个表单-->
14  <form name="input" action="" method="get">
15  姓名:
16  <input type="text" name="nickname" value="" size="20">
17  <br />年龄:
18  <input type="text" name="age" value="" size="20">
19  <br />
20  <!--这个按钮会引发表单的默认提交动作-->
21  <input type="submit" value="提交表单">
22  </form>
23  </body>
```

第 05~08 行为表单关联 submit 事件，事件回调函数中的 e 参数具有一个名为 preventDefault()的函数，调用这个函数，就可以禁止表单提交，从而使得用户可以编写 AJAX 异步提交的代码。

11.24 使用 delay()延迟执行动画

JavaScript 是一种异步执行的语言，也就是说前一行语句的执行并不会阻塞后面的语句，有时可能需要延迟执行一些代码，JavaScript 本身提供了 setTimeout()函数，可以延迟一段时间来执行某个脚本，如下面的代码：

```
01  /* delayAnimate.html */
02      setTimeout(function() {
03          $('#test').hide(1);          //在指定的时间内，隐藏 div 的显示
04      }, 5000);
```

setTimeout()接收一个要延迟执行的回调函数和要延迟执行的毫秒数，本例在 5000 毫秒后执行隐藏 div 显示的代码，实现了延迟的隐藏功能。

jQuery 1.4 以后的版本中，新增加了一个 delay()函数，与 setTimeout()不同的是，这个函数设置一个延时来推迟执行队列中的项目，它既可以推迟动画队列中函数的执行，也可用于自定义队列。只有在队列中的连续事件才可以被延时，因此不带参数的.show() 和.hide()不会有延时，因为它们没有使用动画队列。也就是说，只有 show()或 hide()带参数的时候才能被插入到执行队列中。

举个例子，假如有一个 display 为 none 的 div 元素 test，使用如下代码来调用 delay()延迟执行 div 元素的动画显示：

```
01  /* delayAnimate.html */
02      $( "#test" ).slideUp( 300 ).delay( 800 ).fadeIn( 400 );
```

这个语句会以 300 毫秒的速度显示 div，然后暂停 800 毫秒，最后又以 400 毫秒的速度淡出显示。

注意：delay()函数最佳用途是队列化的 jQuery 特效，它不是原生的 setTimeout()函数的替代。对于需要延迟执行一些特定的函数或功能，仍然强烈建议使用 setTimeout()。

11.25　在网页上运行本地程序的方法

在网页上运行本地程序是一个危险的做法，不过有一些基于网页的程序出于需要也可能必须得有这个功能。本例演示了如何在网页上运行本地的记事本和命令行工具。

注意：要运行本例，浏览器必须支持 ActiveX。一般 IE 会支持 ActiveX，但也可能因为安全性原因禁用了 ActiveX，请确保指定的浏览器的 ActiveX 功能已经打开。

本例代码如下：

```
/* executeLocalProgram.html */
<script type="text/javascript">
    function exec(command)
    {
            window.oldOnError = window.onerror;        //保存 window 错误信息
            window._command = command;                 //保存传入的 command 命令
            window.onerror = function (err)            //定义错误函数
            {
              if (err.indexOf('utomation') != -1)
              {
                  //显示错误消息
                 alert('命令' + window._command + ' 已经被用户禁止！');
                 return true;
              }
              else
                return false;
      };
       //创建 WScript.Shell 脚本执行环境
       var wsh = new ActiveXObject('WScript.Shell');
       if (wsh)   wsh.Run(command);                    //运行传入的命令
       wsh = null;
       window.onerror = window.oldOnError;             //恢复错误处理
 }
</script>
</head>
<body>
<!--下面的两个按钮在单击时，将会调用 exec()函数执行 notepad.exe 和 cmd.exe -->
<input onclick="exec('notepad.exe')" value="执行 noteppad.exe" type="button">
<input onclick="exec('cmd.exe')" value="执行 cmd.exe" type="button">
</body>
```

第 03~17 行创建了一个 exec()函数，它接收要执行的可执行程序的名称。在函数内部

会新建一个 WScript.Shell 的 ActiveX 对象，这个对象是 Windows 脚本执行环境，通过调用 wsh.Run 命令，就可以执行本地程序。本例在 IE 中的效果如图 11.6 所示。

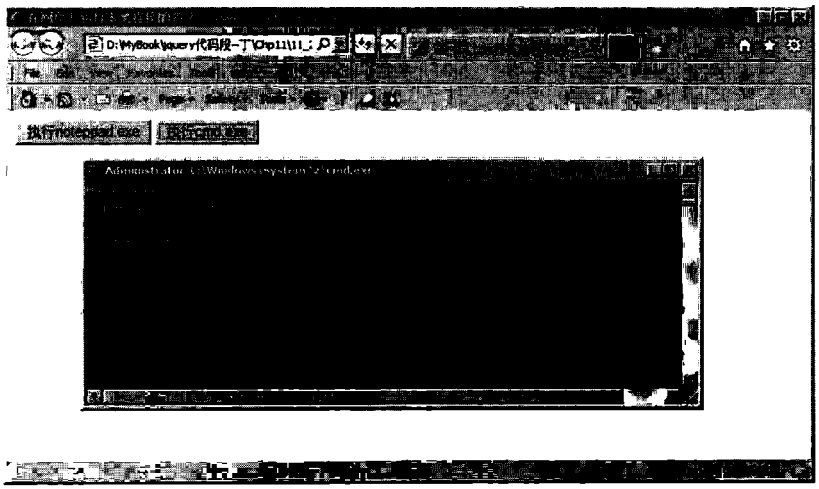

图 11.6　在 IE 中执行本地程序的效果

提示：有可能 IE 会弹出一些警告提示，提示用户将执行 ActiveX 控件，确认后就能正确地执行本地的程序了。

11.26　动态过滤 HTML 表格中的内容

很多数据在呈现时都会使用 HTML 的 table 元素构建表格对象，这种行列式的呈现方式简单易用、易控制。但有时可能需要对 HTML 表格中的数据进行动态过滤，这种过滤不属于服务器端的查询过滤，而是在客户端浏览器上，根据指定的条件来过滤数据，属于客户端的过滤。本例将根据表格中的列数，动态地构建 select 控件，并添加表格中唯一的记录到 select 的选择列表中，然后允许用户选择指定的记录。

本例效果如图 11.7 所示。当变更 table 表格中的内容时，新内容会自动地添加到相应列的 select 控件中，选中 select 列表框中的项时，会显示相同名称的表格行，从而实现了过滤的效果。

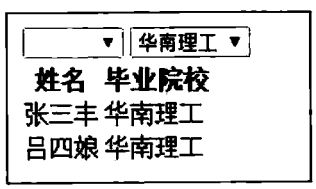

图 11.7　动态生成 select 来过滤 HTML 表格中的内容

HTML 表格代码如下：

```
01  /* filterHTML.html */
02  <!--放置动态添加下拉列表框的 div 控件-->
```

```
<div id="filterdiv"></div>
<!--定义一个两列表格-->
<table id="filedemo">
    <thead>
        <tr>
            <th>姓名</th>
            <th>毕业院校</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>张三丰</td>
            <td>华南理工</td>
        </tr>
        <tr>
            <td>吕四娘</td>
            <td>华南理工</td>
        </tr>
        ……
    </tbody>
</table>
```

接下来使用 jQuery 代码来动态创建 select 列表框，并向列表框中插入数据，代码如下：

```
/* filterHTML.html */
<script type="text/javascript">
  $(document).ready(function(e) {
        $("#filedemo > thead th").each(function(i) {      //循环表格中表头的 th 行
         //构建一个 select 对象，指定 data-index 为 i，并为其关联 change 事件
        $("<select />").attr("data-index", i).html($("<option />")).change(function() {
        //过滤 tbody 中的 tr 行，过滤条件是查找表格中与 select 中选中的值相同的元素
                $("#filedemo > tbody > tr").show().filter(function() {
                    //得到 tr 行的所有子元素
                    var comb = [], children = $(this).children();
                    children.each(function(i) {              //循环所有子元素
                        var value = $("select[data-index='" + i + "']").val();
                        //向数组中添加 value 与选中表格中的子元素匹配的值
                        if (value == $(this).text() || value == "") comb.push(1);
                    });
                    return comb.length != children.length;
                }).hide();                                   //隐藏显示不匹配的表格行
            }).appendTo($("#filterdiv"));                    //将 select 清加到 div 中
        });
        //通过循环表格行，为 select 控件的 option 添加值
        $("#filedemo > tbody tr").each(function() {
            $(this).children().each(function(i) {            //循环表格单元格
```

```
23                      var that = $(this);                          //得到当前循环的单元格
24              //获取 select 对象
25              var select = $("select[data-index='" + i + "']");
26              if (!select.children().filter(function() {           //调用 filter 过滤掉文本相同的记录
27                      return $(this).text() == that.text();
28                      }).length) {                                 //如果长度小于 1
29                          //向 select 中追加 option
30                           select.append($("<option />").text($(this).text()));
31                      }
32                  });
33              });
34  });
35 </script>
```

可以将这个实例看作 filter()函数的灵活运用，第 04~19 行循环 HTML 表格中的表头元素，即 th 元素，这个元素是表格的标题栏，通过循环 th 中的单元格，来构建 select 元素，本例为 select 赋了一个空的 option 元素。第 06 行关联了 change 事件，第 08 和第 26 行在这个事件内部调用 filter()，对当前 select 中的选择列表和表格行 tr 中的单元格进行比较，得到比较结果。最后通过循环 tr 行来为表格中的 select 添加选择项。

11.27 使用递归函数创建文字闪烁特效

闪烁的文字动画效果有时候特别能吸引人，就像 Flash 中的文字动画一样。第 1 章曾介绍过几种实现文本闪烁的方法，本例将创建一个递归调用的函数，在函数内部通过淡入和淡出文字来实现文字的闪烁效果，同样的原理也可以用在很多其他的动画显示方面。

本例代码如下：

```
01  /* blinkWords.html */
02  <script type="text/javascript">
03      function blink(selector){
04          $(selector).fadeOut('slow', function(){        //缓慢的淡出显示
05              $(this).fadeIn('slow', function(){         //缓慢的淡入显示
06                  blink(this);                           //递归调用自身
07              });
08          });
09      }
10      $(document).ready(function(){                      //在页面就绪时
11          blink('#title');                               //调用递归的文字闪烁方法
12      });
13  </script>
14  </head>
15  <body>
16  <h2 id="title">我是闪烁的文字</h2>
17  </body>
```

第 03~09 的 blink()函数内部，对选定的元素进行缓慢地淡入和淡出操作，在淡入后又重新调用了 blink()，并传入 this 作为参数，表示连续不断地调用自身来实现淡入和淡出的效果。

11.28 使用 ID 选择器加快选择的速度

jQuery 的强大特色是选择器，比如可以一次性选中页面上所有元素的元素选择器、选中具有某特定属性的属性选择器，以及表单选择器等等。在 jQuery 中最快的选择器是 ID 选择器，它来自 JavaScript 的 getElementById()函数。

举例来说，假定 HTML 页面上有如下一段代码：

```
/* idSelector.html */
<div id="div1">
  <p>
    <div id="div2">
      <h3>这是一个实例</h3>
    </div>
  </p>
  <p>演示了在一个 div 中具有多个元素的例子</p>
</div>
```

要选择 div 元素，使用 ID 选择器比标签选择器或类选择器就要快得多：

```
/* idSelector.html */
  $(document).ready(function(e) {
    alert($("div h3").html());              //使用标签选择器
    alert($("#div1 h3").html());            //使用 ID 选择器
});
```

对于层次化的选择器，应该总是使用 ID 选择器，并且为了提高性能，最好从就近的 ID 开始进行选择。比如要选中 h3，可以使用如下的代码：

```
alert($("#div2 h3").html());
```

如果让 jQuery 选择器在大量的元素中循环查找，会非常消耗系统资源，因此从就近的 ID 开始选择，能得到较好的性能。

11.29 在类选择器前用标签选择器加快速度

标签选择器又称为 Tag 选择器，它可以一次选中网页中的多个标签，它是 jQuery 中第二快的选择器，因为它来自原生的 getElementsByTagName()函数。

本书以交通信号灯为例。HTML 代码如下：

```
/* lableSelectorbeforeClass.html */
<div id="content">
  <form method="post" action="/">
```

```
04      <h2>交通信号灯</h2>
05      <ul id="tagdemo">
06        <li><input type="radio" class="on" name="light" value="red" /> 红灯</li>
07        <li><input type="radio" class="off" name="light" value="yellow" /> 黄灯</li>
08        <li><input type="radio" class="off" name="light" value="green" /> 绿灯</li>
09      </ul>
10     <input class="button" id="traffic_button" type="submit" value="选中" />
11    </form>
12    </div>
```

如果要选中所有的灯光单选框（radio），则可以使用如下语句：

```
var light=$("input[name='light']");
```

这里也使用了属性选择器选中 input 中 name 等于 light 的 input 集合。最常见的用法是使用标签选择器配合类选择器，比如要选中已经关灯（即 class 为 off）的单选框，可以使用如下语句：

```
var offlight=$("#content input.off");          //在 ID 选择器后，使用标签选择器配合类选择器
```

这里配合了 ID 选择器来选择元素，以缩小搜索的范围。

注意：在 jQuery 中类选择器是最慢的选择器，在 IE 浏览器下会遍历所有的 DOM 节点，无论其在哪个位置使用。

不能使用标签选择器作为 ID 选择器的前缀，这会导致更多的搜索周期，如下代码：

```
var content = $('div#content');
```

这会搜索页面上所有的 div，然后在 div 里面搜索 ID 为 content 的元素，会造成较多的搜索资源，同样的原理，用 ID 选择器作为 ID 选择器的前缀，也造成较多的资源消耗：

```
var lightbutton = $('#content #lightbutton');      //在 ID 选择器中用 ID 选择器也会很慢
```

因此 ID 选择器应该总是搭配其他的选择器来使用，如可以和类选择器等搭配使用，从而获得较好的效能。

11.30 缓存 jQuery 对象以提升性能

当需要多次使用一个选择器选中的结果时，应该把这个选择器保存到一个变量中，像以下形式的调用将会损失性能：

```
01  /* cacheObjects.html */
02      $('#tagdemo input.off').bind('click', function(){});          //为按钮关联单击事件
03      $('#tagdemo input.off').css('border', '3px dashed yellow');   //更改按钮的边框
04      $('#tagdemo input.off').css('background-color', 'orange');    //更改按钮的背景色
05      $('#tagdemo input.off').fadeIn('slow');                       //淡出显示按钮
```

可以看到，实例中多次使用了$('#tagdemo input.off')这个选择器，这样每一次都会让 jQuery 来进行选择，很明显不是高效的写法，应该尽可能地将 jQuery 对象缓存起来，如下所示：

```
01  /* cacheObjects.html */
```

```
02      var offbutton=$('#tagdemo input.off');          //将 jQuery 对象缓存起来
03      offbutton.bind('click', function(){});          //为按钮关联单击事件
04      offbutton.css('border', '3px dashed yellow');   //更改按钮的边框
05      offbutton.css('background-color', 'orange');    //更改按钮的背景色
06      offbutton.fadeIn('slow');                       //淡出显示按钮
```

上述代码将 jQuery 的选择器放在了一个本地变量中，这样多次使用选择器时，实际上使用的是已经缓存的 jQuery 选择结果，因而会提升程序的性能。

11.31 使用 find()函数提升子查询的性能

jQuery 1.3 版本以后的命名用了 Sizzle 库，其选择器引擎使用了从左至右的选择模型，因此对于级联的选择器，比如：

```
var linkContacts = $('.contact-links div.side-wrapper');
```

尽量要使左边的选择器选择范围大一些，否则会降低其性能，如果有很多节点需要来回查找，最好是使用 find()函数而不是用层次化的选择器。

举个例子，有如下一段 HTML 代码：

```
01  /* findFunction.html */
02  <ul class="level-1">
03    <li class="item-i">I</li>
04    <li class="item-ii">II
05      <ul class="level-2">
06        <li class="item-a">A</li>
07        <li class="item-b">B
08          <ul class="level-3">
09            <li class="item-1">1</li>
10            <li class="item-2">2</li>
11            <li class="item-3">3</li>
12          </ul>
13        </li>
14        <li class="item-c">C</li>
15      </ul>
16    </li>
17    <li class="item-iii">III</li>
18  </ul>
```

要取出 class 为 level-3 类下面的所有 li 元素，可以使用如下语句：

```
$('li.level-3').find('li').css('background-color', 'red');
```

find()的参数是一个选择器，这个选择器可以是任何的 jQuery 选择器，下面的代码演示了其他的 find()用法：

```
01  /* findFunction.html */
02  <script type="text/javascript">
03    $(document).ready(function(e) {
```

```
04        $('li.item-ii').find('li').css('background-color', 'red');          //查找 item—ii 相关的 li 子元素
05        var item1 = $('li.item-1')[0];                                      //取得类名为 item-1 的子元素
06        $('li.item-ii').find( item1 ).css('background-color', 'blue');      //按变量进行查找
07    });
08    </script>
```

可以看到，find()选择器可以灵活地使用选择器进行查找，并且相对于 jQuery 层次选择器的查找方法，使用 find()能提供更快的性能。

注意：find()与 children()都可以取得选定的子元素，不同之处在于 find()会层层查找直到叶子节点，而 children()只会查找下一层的节点。

11.32 使用 jQuery 操作 DOM 的限制

使用 jQuery 操作 DOM 是比较慢的，因此要尽可能少地操作 DOM。如果必须要对 DOM 进行处理，可以先在内存中构建好 DOM 代码，然后一次性地加入到 DOM 树中。如果对 DOM 进行多次操作，将影响程序的性能。

本例演示了不正确的做法，这个实例会对 DOM 进行 100 次操作，代码如下：

```
01  /* DOMConstraints.html */
02    $(document).ready(function(e) {
03        $('#header').prepend('<ul id="menu"></ul>');          //在被选元素的开头插入内容
04        for (var i = 1; i <= 100; i++) {                      //对 DOM 进行 100 次操作
05          $('#menu').append('<li>' + i + '</li>');            //追加元素
06        }
07    });
```

应该尽可能减少直接对 DOM 操作的次数，下面的代码将改进上面的实例：

```
01  /* DOMConstraints.html */
02    $(document).ready(function(e) {
03        var menu = '<ul id="menu">';                          //构建 DOM 字符串
04        for (var i = 1; i <=100; i++) {                       //循环对字符串操作
05        menu += '<li>' + i + '</li>';
06        }
07        menu += '</ul>';                                      //在内存中构建 DOM 结构
08        $('#header').prepend(menu);                           //一次将其添加到 DOM 树中
09    });
```

上述代码并没有对 DOM 立即进行操作，而是先在变量中构建了 DOM 的 HTML 结构，最后调用 prepend()一次性地将整个结构添加到 DOM 树中，由于对 DOM 树的操作仅发生一次，因此具有较好的性能。