



jQuery的核心库的工作稍显枯燥：通过处理浏览器加载HTML时动态创建的模型（即DOM：文档对象模型，后面我会详细介绍），达到动态修改页面内容的目标。既然你会捧起这本书，想必你已经有一些处理DOM的经验，用过别的JavaScript库或者浏览器内置的API（应用编程接口），并渴望找出更好的方法。

jQuery绝非“更好”所能形容。它使DOM操作变成了一件愉快的事，有时候甚至有趣得不得了。jQuery中处理DOM的一些方法相当优雅，那些纯粹的苦活累活变得既简单又容易。一旦开始使用jQuery，你就永远不会想回到过去了。我之所以在项目中使用jQuery，主要原因有以下几个。

- jQuery表达能力很强。我能用更少的代码完成更多的工作。
- jQuery支持一次处理多个元素。传统的DOM API需要先选取再迭代处理每一个元素，而jQuery大大减少了遍历元素的次数，也减少了出错机会。
- jQuery解决了不同浏览器的兼容性问题。我再也不必神经兮兮的，担心IE是否支持某个特性：只管告诉jQuery我想要做什么，它自会替我搞定。
- jQuery是开源的。如果我想知道某些功能是怎么实现的，或者发现哪儿不对劲，可以直接阅读它的源代码，并且可以按需修改它。

jQuery的伟大之处在于把Web开发中那些难以处理的苦差事变成了既简单又快捷的轻松之事。有了jQuery，我别无他求。当然，jQuery并不完美，它也有那么一两处短板，稍后我会提到。但无论如何，瑕不掩瑜，我非常喜欢jQuery，也希望你能逐步发现它的强大之处并乐于使用它。

## 1.1 jQuery UI 和 jQuery Mobile

除了jQuery库，本书也介绍了jQuery UI和jQuery Mobile，它们是建立在jQuery库上的用户界面库。jQuery UI是通用的UIT工具包，旨在支持各种设备，而jQuery Mobile则专为触摸设备（比如智能手机和平板电脑）而生。

## 1.2 jQuery 插件

jQuery插件扩充了基础库的功能。有一些插件非常优秀并得到了广泛的应用，因此我会在本书中介绍这些插件。另外，网上还有很多插件可用（尽管质量参差不齐），如果你不喜欢本书中介绍的插件，完全可以从网上寻找（对你口味的）替代品。

## 1.3 预备知识

阅读本书之前，你最好熟悉Web开发基础知识，知晓HTML和CSS的工作方式，并且最好有一些JavaScript应用知识。如果基础不很扎实，请先阅读本书第2章、第3章和第4章中提供的HTML、CSS和JavaScript相关内容。不过，你可别期望在这几章中全面了解HTML元素和CSS属性，毕竟因为篇幅限制本书无法完全涵盖HTML知识。如果需要完整的HTML和CSS（层叠样式表）参考手册，我建议你阅读我的另一本书：《HTML5权威指南》<sup>①</sup>。

## 1.4 本书组织结构

本书共分六部分，其中分别介绍了一系列的相关主题。

### 1.4.1 第一部分：打好基础

第一部分是阅读本书所需的预备知识，包括本章及HTML、CSS和JavaScript的入门（补习）课程。本章后面将介绍学习本书要用到的软件。

### 1.4.2 第二部分：使用jQuery

第二部分介绍jQuery库的使用，从一个基本示例开始介绍，逐渐过渡到jQuery的各个核心功能：元素选择、DOM操作、事件和特效。

### 1.4.3 第三部分：数据和Ajax

第三部分讲解如何在开发中用jQuery操作本地和远程数据。我会介绍如何利用数据生成HTML内容、验证Web表单中用户输入的数据，以及如何利用jQuery执行异步操作，包括Ajax。

### 1.4.4 第四部分：jQuery UI

jQuery UI是本书重点介绍的两个用户界面库之一，它建立在jQuery库之上并与之结合使用，让我们能够在Web应用中创建多姿多彩的交互界面。

### 1.4.5 第五部分：jQuery Mobile

jQuery Mobile是本书介绍的另一个用户界面库。这个库也依托于jQuery，并整合了jQuery UI的一些基本功能，针对智能手机和平板电脑界面做了许多优化工作。jQuery Mobile中的界面组件比jQuery UI少，但它支持的组件都针对触摸操作和小屏幕做了优化。

### 1.4.6 第六部分：高级功能

本书最后一部分涵盖了jQuery和jQuery UI中一些不常用，但在复杂项目中能够发挥很好作用的功能。

---

<sup>①</sup> 人民邮电出版社2014年1月份出版。——编者注

能。要掌握这些高级功能，你需要对HTML、CSS及jQuery本身有着透彻的理解。阅读第36章时，事先了解一些异步编程的知识会很有帮助。

## 1.5 第2版的新增内容

自第1版出版至今，与当时的情况相比，jQuery、jQuery UI和jQuery Mobile均有大量变化。

### 1.5.1 jQuery核心库有什么变化

jQuery核心库的API相当稳定。几年前，jQuery团队开始公示他们打算改变的部分，在已发布的1.9版中，这些计划已成为现实。其中有些变化相当重要，我会在第二部分的每一章中标明这些变化。

好消息是核心API的变化相当少，在未来几年里，它们仍会继续保持稳定。这并不是说核心库不再添加新功能，而是指你今天编写的代码在未来相当长的时间里都无需做任何修改，仍能正常工作。

还有一个坏消息，jQuery 1.9版本的发布不同寻常。jQuery团队在发布1.9版本的同时，他们还发布了2.0版。这意味着他们将并行开发维护两个分支：jQuery 1.x和jQuery 2.x。两个分支拥有相同的API，只是jQuery 2.x不再支持IE 6、IE 7还有IE 8。

旧版本的IE臭名昭著，因为它们的HTML、CSS和JavaScript实现没有完全遵守Web标准。jQuery 2.x移除了那些为了兼容旧版本IE而不得不增加的种种检测IE怪异行为的判断，从而体积更小，速度更快。

---

**提示** 写作本书时，jQuery的版本分别是2.0.2和1.10.1。在这里要声明一下：jQuery 2.0.2并不是1.10.1的升级版，它们都是最新版本，二者唯一的区别在于jQuery 1.10.1继续支持旧版本IE。

---

如果能保证自己的用户不会使用旧版IE，就应该使用jQuery 2.x版本。如果不能保证（或者不是很确信），就继续用jQuery 1.x版本。旧版IE仍被广泛使用，特别是在一些大公司中。使用jQuery 2.x必须慎之又慎。

理想情况下，编程书籍总是面对喜欢追赶技术前沿的用户。因此，本书基本上都在使用jQuery 2.0.2，不过你可以使用任意一个1.9.x版本替掉我使用的jQuery 2.0.2。两个版本的jQuery总会得到同样的结果，只是1.x版本还支持旧版IE而已。

---

**提示** 我在本书的第四部分（关于jQuery Mobile）使用了jQuery 1.x。与jQuery核心库相比，jQuery Mobile的发布总是慢一拍。当我写作本书时，jQuery Mobile仅支持jQuery 1.x。

---

### 1.5.2 jQuery UI有什么变化

jQuery UI库也更新了。已有用户界面组件的API更加一致，与底层的HTML元素合作更紧密。除此之外，还添加了一些新组件。在本书的第三部分，你会在每一章的起始部分看到那些最重要的变化，没错，我在本书的第二部分（jQuery核心库）中就是这么做的。

### 1.5.3 jQuery Mobile有什么变化

与本书第一版相比，jQuery Mobile变得相当成熟。API更加合理，组件更加丰富，整个开发体验也与jQuery和jQuery UI更加一致。为了回馈jQuery Mobile团队所作的努力，我彻底重写了本书的第四部分，以保证它与本书的其他部分一样，不致落伍。在第四部分中，有更多的例子、参考表以及特色功能演示。

### 1.5.4 其他变化

从第12章起，我使用模板得到HTML元素。这是一项相当重要的技术，我已驾轻就熟。由于本书第一版使用的模板库在今天已经过时，所以我重新选择了一个。新的模板库不再是一个jQuery插件，因此在第12章里我写了一个插件，让我选中的这个模板库能更容易与jQuery一起使用。为使用新的模板库，第12章及之后的所有示例代码都做了更新。

除此之外，用来测试移动应用的工具也有所变化。现在我改用一项基于云技术的测试服务，不再维护自己机器上的各种模拟器。在第27章，我会详细说明为什么值得这么做。

## 1.6 例子多吗

例子非常多。jQuery很棒的一点就是几乎所有的任务都可以用多种方式完成，我们完全可以有自己的jQuery开发风格。为了演示这些方法，本书提供了各式各样的示例，为了容纳这些例子，每章要用的HTML代码只能在某些章节的开头完整地展示一次。每一章的第一个例子都是一个完整的HTML文档，如代码清单1-1所示。

代码清单1-1 一个完整的示例文档

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script type="text/javascript">
    $(document).ready(function() {
      $("img:odd").mouseenter(function (e) {
        $(this).css("opacity", 0.5);
      }).mouseout(function (e) {
        $(this).css("opacity", 1.0);
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
```



```

<div id="row1" class="drow">
  <div class="dcell">
    <label for="aster">aster:</label>
    <input name="aster" value="0" required>
  </div>
  <div class="dcell">
    <label for="daffodil">Daffodil:</label>
    <input name="daffodil" value="0" required >
  </div>
  <div class="dcell">
    <label for="rose">Rose:</label>
    <input name="rose" value="0" required>
  </div>
</div>
<div id="row2" class="drow">
  <div class="dcell">
    <label for="peony">Peony:</label>
    <input name="peony" value="0" required>
  </div>
  <div class="dcell">
    <label for="primula">Primula:</label>
    <input name="primula" value="0" required>
  </div>
  <div class="dcell">
    <label for="snowdrop">Snowdrop:</label>
    <input name="snowdrop" value="0" required>
  </div>
</div>
</div>
<div id="buttonDiv"><button type="submit">Place Order</button></div>
</form>
</body>
</html>

```

这个代码清单取自第5章。不必考虑它究竟做了些什么，在这里你只需要明白，每章的第一个示例都是一个完整的HTML文档，就像上面展示的这个代码清单一样。几乎所有的示例都基于同一个HTML基础文档，这个文档用来显示一个简单的鲜花店。很难说这个示例会十分激动人心，但“麻雀虽小，五脏俱全”，它涉及了我们在使用jQuery过程中感兴趣的所有内容。

除第一个示例外，我只展示那些变化了的元素。这里通常只涉及script元素，也就是我们存放jQuery代码的地方。你一眼就能看出它只是代码清单的一部分，因为它由省略号(...)开始，并由省略号结束，如代码清单1-2所示。

#### 代码清单1-2 代码片段

```

...
<script type="text/javascript">
  jQuery(document).ready(function () {
    jQuery("img:odd").mouseenter(function(e) {
      jQuery(this).css("opacity", 0.5);
    }).mouseout(function(e) {
      jQuery(this).css("opacity", 1.0);
    });
  });
</script>

```

```
        });  
    });  
</script>  
...
```

代码清单1-2是第5章中某代码清单的一个片段。这清单里只有script元素，并且突出了一些语句。这些加粗的代码行，就是要告诉你这儿用到了哪些jQuery功能。在类似的只有部分代码的代码清单中，只会给出那些与每一章开头的HTML文档相比已经发生变化的代码。

在本书中，我尽力让每个示例只关注一个相对独立的功能，以此帮助你掌握jQuery方方面面的知识。当然这种做法也有缺点：你没有机会看到这些功能协同工作。因此，本书每一部分的最后都有简短的一章，其中我会把前面几章中的各个主题的示例融合在一起讲解，从而为你提供jQuery相关功能的一个完整视图。

## 1.7 示例代码下载

本书所有示例代码，以及再现这些示例所必需的所有资源文件（包括图片文件、JavaScript库，还有CSS样式表文件）都可以从Apress.com<sup>①</sup>的源代码/下载区（Source Code/Download）免费下载。你不一定非要下载这些代码，不过下载是体验这些例子的最简单快捷的方法，也便于你复制粘贴，把它们用到自己的项目上。<sup>②</sup>

---

**提示** 为了节省篇幅，书中大量的代码清单只列出了变动的部分，而下载包中所有示例的源代码都是完整的，你可以直接在浏览器里载入它们。

---

## 1.8 所需软件

要跟着书中的例子做练习，需要准备一些软件。

### 1.8.1 jQuery

首先，你需要jQuery库（可从官网<http://jquery.com>下载）。如图1-1所示，jQuery官网的首页有一个Download按钮和一个选项，你可以选择下载稳定但较旧的产品版，也可以下载最新但bug可能较多的开发版。

---

① 也可在图灵社区本书页面<http://www.it-ebooks.com.cn/book/999>免费注册下载。——编者注

② 尽管几乎所有的技术书都提供示例代码下载，我始终建议读者手工敲入示例代码。复制粘贴固然快捷，但复制粘贴那样的浅记忆远远比不上手敲代码形成的记忆深刻。如果你想学得扎实，最好听我的。——译者注

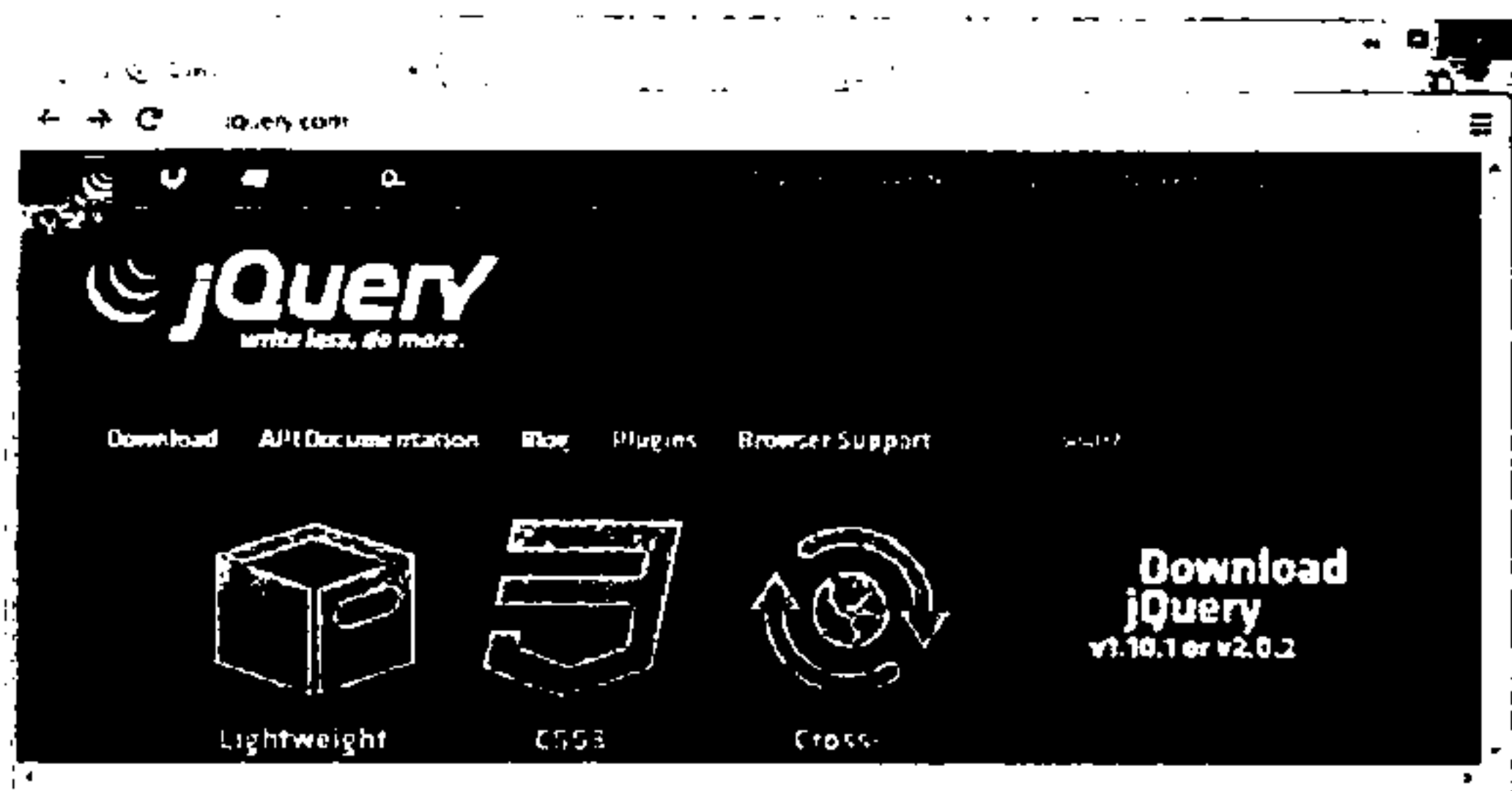


图1-1 下载jQuery库

在第一部分到第四部分和第六部分，你需要下载jQuery 2.x；第五部分则会用到jQuery 1.x。学习本书直接使用开发版就好。我将在第5章解释这两种版本之间的不同之处，并教你使用jQuery。

**提示** 第17章将展示jQuery UI库的下载和安装，第27章展示jQuery Mobile库的下载和安装。

## 1.8.2 HTML编辑器

用来编写HTML文档的编辑器是Web开发最重要的一个工具。HTML就是文本，虽然你可以用非常简单的编辑器，不过一些专门的开发工具能让开发工作更顺畅更简单，而且好多这样的工具都是免费的。

写本书第一版时，我使用的是Active State公司的Komodo Edit编辑器。它免费又简单，对HTML、JavaScript和jQuery的支持非常好。它有3个版本，分别支持Windows、Mac和Linux操作系统。访问<http://activestate.com>可获取更多信息。

然而，最近我转向了微软的Visual Studio。我使用Windows写过好多有关微软Web开发技术的书，我也是在Windows平台上写代码。我发现Visual Studio最近的版本对编辑HTML的支持出奇得好，而且不依赖任何微软Web开发工具。我使用Visual Studio 2012 Express编写本书，它是免费的——请访问<http://www.microsoft.com/visualstudio>。（当然也有收费版本，毕竟是微软的东西嘛，不过对jQuery开发来说，免费的版本已经足够了。）

作为一个备选方案，JsFiddle是一个广受欢迎的支持jQuery库的在线编辑器，虽然我和它有点儿合不来（我不习惯它组织代码的方式），但是它真的非常灵活，非常强大，同样是免费的，网址是<http://jsfiddle.net>。

**注意** 之所以在书里推荐这些产品，仅仅是因为我在使用它们，并且喜欢它们。我与Active State、微软以及其他公司没有任何关系。不过Apress除外，我所有的书都是在Apress出版的。我用到的每一个Web服务或者开发工具，都是花全价购买的；我从未从这些产品的开发团队那里得到特殊支持，或者秘密权限。我唯一的收入，就是来自这本书的版税（所以感谢你购买此书）。

### 1.8.3 Web浏览器

你还需要一个Web浏览器，用它显示HTML文档，并测试jQuery和JavaScript代码。我喜欢Google Chrome：它速度快、用户界面简洁，并且提供了很棒的开发工具。本书中的屏幕截图（还挺多的）都是基于Google Chrome的。

这并不是说你一定要和我使用一样的浏览器，不过确实建议你使用一个提供优质开发工具的浏览器。Mozilla Firefox浏览器通过Firebug扩展提供了出色的JavaScript开发工具，你可以从<http://getfirebug.com>得到这个扩展。

如果你不喜欢Chrome或Firefox，那么最好选择IE浏览器。许多Web开发者都不喜欢IE，但IE 10真的很棒，当Chrome偶尔掉链子的时候，我就用它快速地检查页面完整性。

### 1.8.4 Web服务器

为了再现书中的这些示例，你需要一个Web服务器，这样浏览器才有地方载入示例HTML文档和相关资源文件（如图片和JavaScript文件）。可用的Web服务器很多，并且大都是开源而且免费的，你可以任意选用一种。本书中使用的是微软公司的互联网信息服务（IIS），不过这仅仅是因为我刚好有一台安装好了的Windows服务器。

### 1.8.5 Node.js

从第三部分起，我们使用Node.js作为常规Web服务器的补充。Node.js现在非常流行，不过我使用Node.js的理由很简单，它基于JavaScript，这样你就避免了和一个毫不相关的Web开发框架打交道。你并不需要深入了解Node.js的所有细节，我也只是把它当成一个黑盒子来使用。（不过我确实展示了一些服务器脚本，如果你有兴趣的话，可以看看服务器端究竟发生了些什么。）

我们可以从<http://nodejs.org>下载到Node.js。此处提供了用于Windows的预编译好的可执行文件和源代码，你可以在其他平台上自行编译。本书使用的版本是0.10.13，等你读到本书的时候，它很可能已经有了更新的版本，不过这些服务器端脚本应该仍能正常工作，不会有任何问题。

### 安装和测试Node.js

测试Node.js的最简单方法是使用一个简单的脚本。把代码清单1-3中的内容保存为NodeTest.js文件，并把它放到和Node.js二进制文件相同的目录下。

#### 代码清单1-3 一段Node.js测试脚本

```
var http = require('http');
var url = require('url');

http.createServer(function (req, res) {
  console.log("Request: " + req.method + " to " + req.url);

  res.writeHead(200, "OK");
  res.write("<h1>Hello</h1>Node.js is working");
  res.end();
});
```

```
)).listen(80);
console.log("Ready on port 80");
```

这是一个简单的测试脚本，收到HTTP GET请求时会返回一段HTML片段。

**提示** 如果不是十分理解上文的最后一句，也不必着急。你不需要知道HTTP和Web服务器是如何使用jQuery的，而且第2章会提供一个HTML速成教程。

要测试Node.js，运行Node.js的可执行文件node并把刚刚创建好的文件作为它的命令行参数。针对Windows机器，我在控制台输入以下命令：

```
node NodeTest.js
```

要测试Node.js是否工作正常，请访问运行着Node.js的机器的80端口。如果你看到和图1-2所示内容非常接近的内容，就表明一切正常。

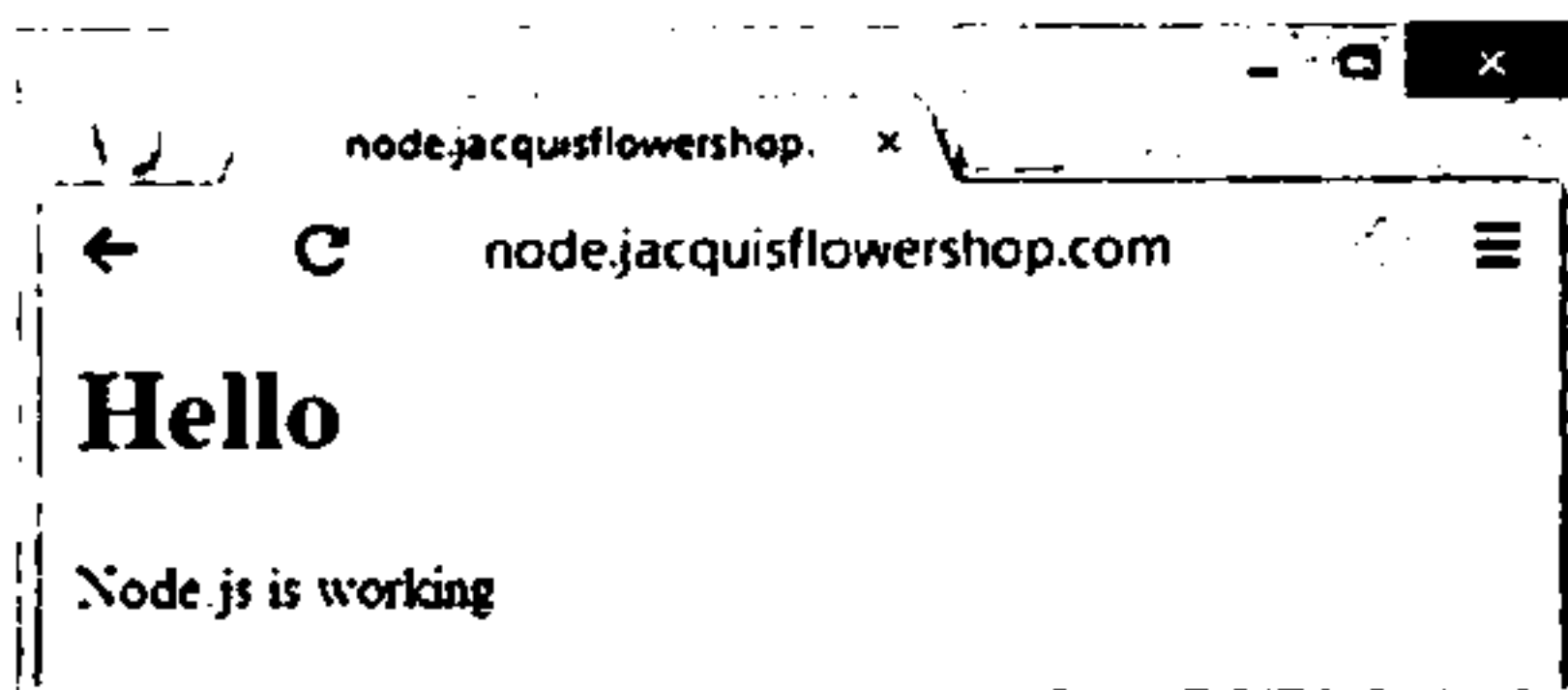


图1-2 测试Node.js

我的Node.js单独运行在一台服务器上，与常规Web服务器并不是同一台机器，所以使用80端口不会造成任何问题。如果你只有一台机器可用，而常规的Web服务器正使用着80端口，可以修改上文中的NodeTest.js，改用另一个端口。我已经在NodeTest.js代码清单中加粗了指定端口的那一行代码。

## 1.9 图片版权

本书示例使用的图片承蒙以下个人和组织惠允使用：Horia Varlan、David Short、Geishaboy500、Tanaka Juuyoh、Mervi Eskelinen、Fancy Speed Queen、Alan “craigie3000” Craigie、notsogood和melalouise。特此致谢。

## 1.10 小结

本章概要介绍了本书的结构和内容，还介绍了jQuery开发必要的软件（这些软件都是免费的）。在接下来的3章中，我们将一起复习HTML、CSS和JavaScript的基础知识。如果你非常熟悉这些基础知识，也可以直接阅读第5章。

本书将花费大量篇幅讲解HTML文档的处理。本章为你准备了理解本书后续内容所必需的知识，但并不是一个完整的HTML教程，由于后面的章节依赖一些HTML基本特性，所以本章简要介绍了这些特性。

HTML的最新版HTML5本身就是一个主题。HTML5有100余种元素，每一种元素都有确切的含义和功能。换句话说，要了解jQuery的工作方式，我们只需要掌握一些HTML基础知识。如果希望深入地了解HTML，建议你阅读我的另一本书《HTML5权威指南》。

## 2.1 基本的 HTML 文档

了解HTML最好的起点，是观察一个HTML文档，通过这样一个文档，我们能看到HTML文件遵循的基本结构和层次关系。代码清单2-1是一个简单的HTML文档，本章会一直使用这个文档介绍HTML的核心概念。

代码清单2-1 一个简单的HTML文档

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <style>
    h1 {
      width: 700px; border: thick double black; margin-left: auto;
      margin-right: auto; text-align: center; font-size: x-large; padding: .5em;
      color: darkgreen; background-image: url("border.png");
      background-size: contain; margin-top: 0;
    }
    .dtable {display: table;}
    .drow {display: table-row;}
    .dcell {display: table-cell; padding: 10px;}
    .dcell > * {vertical-align: middle}
    input {width: 2em; text-align: right; border: thin solid black; padding: 2px;}
    label {width: 5em; padding-left: .5em; display: inline-block;}
    #buttonDiv {text-align: center;}
    #oblock {display: block; margin-left: auto; margin-right: auto; width: 700px;}
  </style>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
```



```
<form method="post">
  <div id="oblock">
    <div class="dtable">
      <div class="drow">
        <div class="dcell">
          <label for="aster">Aster:</label>
          <input name="aster" value="0" required>
        </div>
        <div class="dcell">
          <label for="daffodil">Daffodil:</label>
          <input name="daffodil" value="0" required >
        </div>
        <div class="dcell">
          <label for="rose">Rose:</label>
          <input name="rose" value="0" required>
        </div>
      </div>
      <div class="drow">
        <div class="dcell">
          <label for="peony">Peony:</label>
          <input name="peony" value="0" required>
        </div>
        <div class="dcell">
          <label for="primula">Primula:</label>
          <input name="primula" value="0" required>
        </div>
        <div class="dcell">
          <label for="snowdrop">Snowdrop:</label>
          <input name="snowdrop" value="0" required>
        </div>
      </div>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
</html>
```

这是一个简短的HTML文档，但蕴含着HTML中最重要的特性。图2-1是该文档在浏览器中的显示效果。

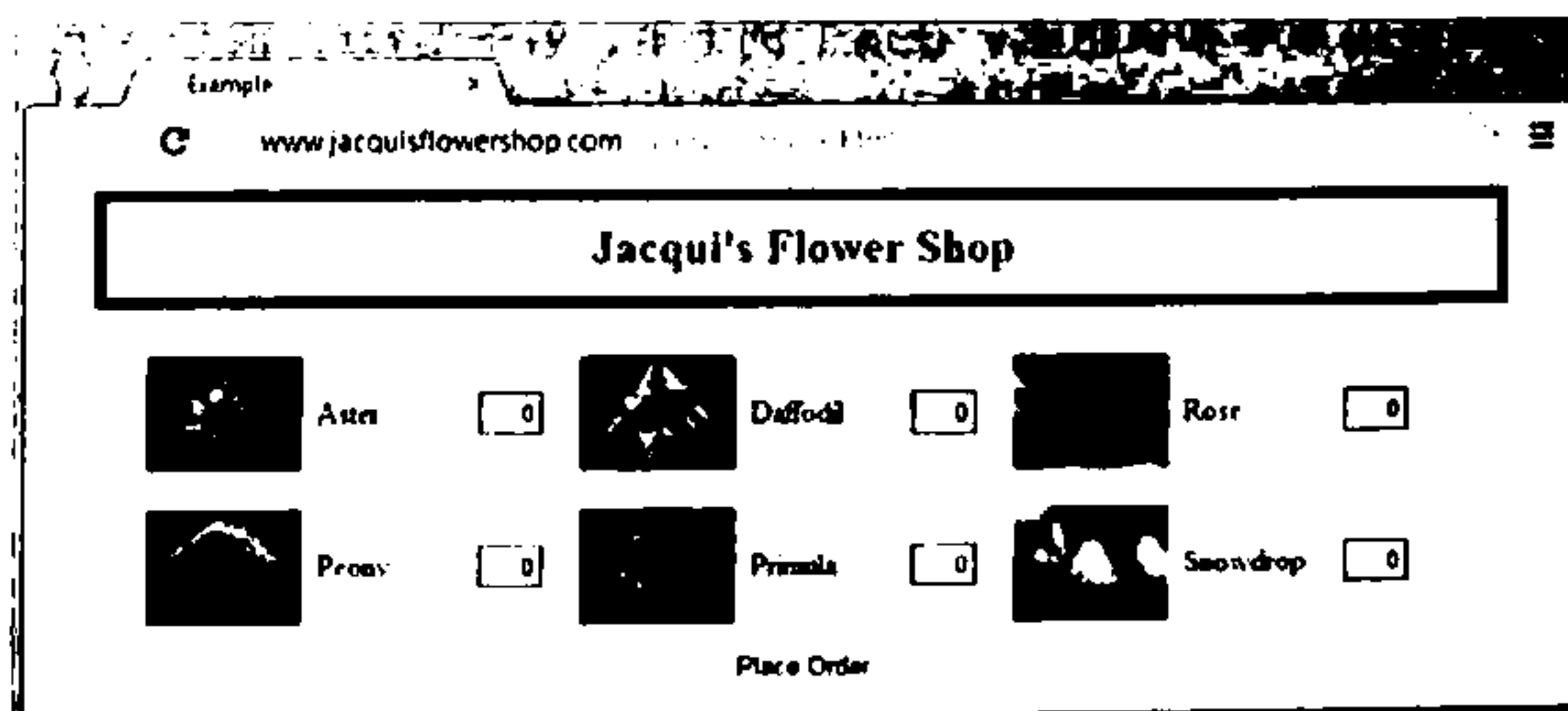


图2-1 示例HTML文档显示在浏览器中

## 2.2 HTML 元素剖析

HTML的核心是元素。元素告诉浏览器HTML文档的每一部分存储什么样的内容。下面是摘自示例文档的一个元素：

```
...  
<h1>Jacqui's Flower Shop</h1>  
...
```

如图2-2所示，这个元素由3部分组成，即开始标签、内容和结束标签。

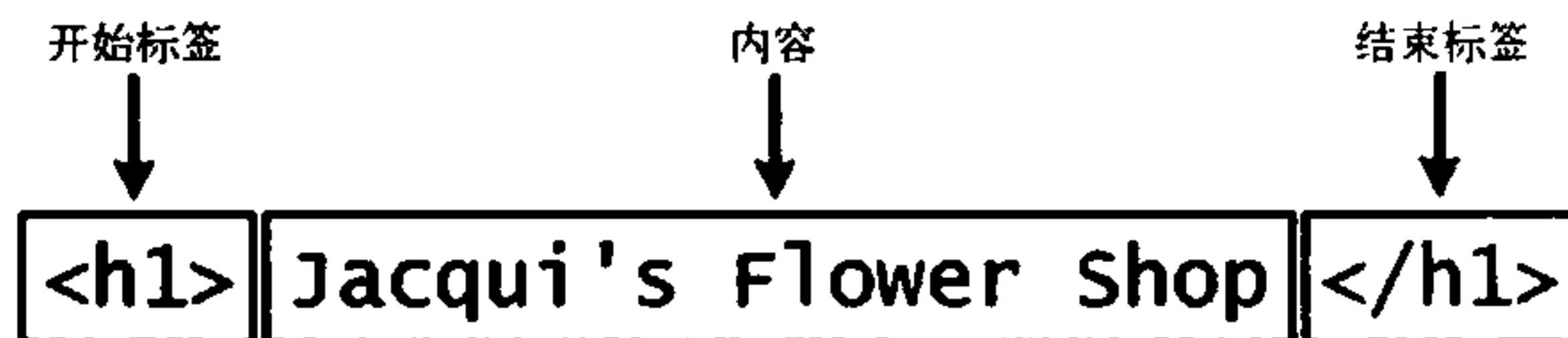


图2-2 一个简单HTML元素的分解图

元素的名称（即标签名称，简称标签）是h1，它告诉浏览器标签之间的内容是最高级标题（1级标题）。把标签名用一对尖括号（<>）括起来，就得到一个开始标签；结束标签的结构与开始标签类似，不过我们要在左尖括号（<）之后加一个/字符。

## 2.3 属性

元素属性告诉浏览器更多的信息。代码清单2-2展示的是来自示例文档的另一个元素，它有一个属性。

### 代码清单2-2 定义属性

```
...  
<label for="aster">Aster:</label>  
...
```

这是一个label元素，它定义了for属性。书中把for属性印刷成粗体，方便大家观察。属性只能是起始标签的一部分。属性由名字和值组成。在这个例子里，名字是for，值是aster。不是所有的属性都需要值，有些属性只需要名字，它是一个标志，用来告诉浏览器我们期望它对这个元素做某种特殊处理。代码清单2-3中就展示了具有这种属性的一个示例元素。

### 代码清单2-3 不需要值的属性

```
...  
<input name="snowdrop" value="0" required>  
...
```

这个元素定义了3个属性，前两个和前面的例子一样，既有名字（name）又有值（value）。（你可能会觉得有点困惑：因为这两个属性的名字分别是name和value。name属性的value是snowdrop，而value属性的value是0。）第三个属性只有一个单词required，表示这是一个不需要值的属性。虽然它不需要

值，我们还是可以把它的值设置为它的名字（`required="required"`），或者把它的值设置为一个空串（`required=""`）。

## id属性和class属性（类属性）

有两个属性特别重要，这就是id属性和class属性。jQuery开发最常见的任务就是在网页中定位一个或多个元素，然后处理这些元素。在HTML文档中定位元素，id和class属性特别有用。

### 1. id属性

id属性用于在HTML页面中定义独一无二的元素（同一页面中不允许出现id值相同的两个元素）。代码清单2-4展示的是一个应用了id属性的简单页面。

代码清单2-4 id属性

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
</head>
<body>
  <h1 id="mainheader">Welcome to Jacqui's Flower Shop</h1>
  <h2 id="openinghours">We are open 10am-6pm, 7 days a week</h2>
  <h3 id="holidays">(closed on national holidays)</h3>
</body>
</html>
```

在上面这个页面中，我为3个元素定义了id属性。h1元素的id属性是mainheader，h2元素的id属性是openinghours，而h3元素的id属性是holidays。利用id属性独一无二的特性，我们可以非常方便地在页面中定位特定元素。

### 2. class属性（类属性）

与id属性相比，我们可以随意添加类属性。如代码清单2-5所示，多个元素可以使用同一个类属性，一个元素可以应用多个类属性。

代码清单2-5 class属性

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
</head>
<body>
  <h1 id="mainheader" class="header">Welcome to Jacqui's Flower Shop</h1>
  <h2 class="header info">We are open 10am-6pm, 7 days a week</h2>
  <h3 class="info">(closed on national holidays)</h3>
</body>
</html>
```

在这个示例中，h1元素具有header类，h2元素则同时拥有header和info两个类，而h3元素则仅拥有info类。从示例可以看出，要为一个元素添加多个类，只需在多个类之间用空格符分隔开。

## 2.4 元素内容

元素可以包含文本，也可以包含其他元素。下面是一个包含其他元素的例子。

```
...
<div class="dcell">
  
  <label for="rose">Rose:</label>
  <input name="rose" value="0" required>
</div>
...
```

上面的例子中，div元素包含3个其他元素，分别是img、label和input。尽管这个例子中只有一层包含关系，实际上HTML元素是支持多层嵌套的。在HTML中，元素嵌套是一个至关重要的概念，因为对于被包含的内容来说，它引入了外部元素的语义（这个话题将在后面专门讲解）。我们也可以像下面这样混合使用文本和其他元素。

```
...
<div class="dcell">
  Here is some text content
  
  Here is some more text!
  <input name="rose" value="0" required>
</div>
...
```

## 2.5 空元素

HTML规范中有些元素可能没有内容。这些元素被称为空元素或自动闭合元素，书写它们时不需要专门的结束标签。下面是空元素的示例：

```
...

...
```

空元素的定义只有一个标签，不过最后的结束尖括号（>）之前要有一个/字符。严格来讲，在标签的最后一个属性和/字符之间应该有一个空格，就像下面这样：

```
...

...
```

不过，浏览器在解释HTML时非常宽容，所以你可以省略掉这个空格字符。空元素的典型用途是引用外部资源文件。在本例中，img元素用来链接一个名为rose.png的外部图片文件。

## 2.6 文档结构

有几个特别关键的元素负责定义HTML文档的基本结构，它们是DOCTYPE、html、head和body。代码清单2-6展示了这些元素的关系（篇幅所限，删除了大部分内容）。

代码清单2-6 HTML文档的基本结构

```
<!DOCTYPE html>
<html>
<head>
    .....头部内容.....
</head>
<body>
    .....主体内容.....
</body>
</html>
```

在一份HTML文档中，这些元素各司其职，每一个都扮演着非常具体的角色。首先是DOCTYPE，它告诉浏览器这是一个HTML文档，并且（更具体的说）是一份HTML5文档。早期版本的HTML需要提供更多的信息。例如，下面示例中的这个DOCTYPE元素表明这是一份HTML4文档：

```
...
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
...
```

html元素定义文档中HTML内容的存放区域，总是包含另两个至关重要的结构元素：head和body。还记得我在本章开始说过的话吧，我不打算讲解具体的HTML元素。由于元素实在太多，我那本HTML5的书里光完整描述HTML5的内容就有1000多页。因此，此处简要描述书中用到的元素，这样你反而更容易搞清文档做了些什么。表2-1简要介绍了代码清单2-1的示例文档中所用到的元素，其中几个元素会在本章后面细讲。

表2-1 示例文档中用到的HTML元素

元 素	描 述
DOCTYPE	表明文档内容的类型
body	定义文档中包含内容元素的区域（2.6.2节）
button	定义按钮，常用于提交表单到服务器
div	通用的（块级）元素，常用于定义文档出于表现目的而划分的结构
form	定义HTML表单，以便收集用户输入的信息并把信息送往服务器处理
h1	定义标题
head	定义文档中包含元数据（metadata）的区域（2.6.1节）
html	定义文档中包含HTML内容的区域（通常是整个文档）
img	定义图片
input	定义输入域以收集来自用户的单一数据项，经常作为HTML表单的一部分
script	定义一段脚本，一般指JavaScript脚本，它会作为文档的一部分执行
style	定义层叠样式表设置区域（参阅第3章）
title	定义文档的标题。浏览器使用它设置窗口标题或者标识页面的标签标题

## 2.6.1 元数据元素

head元素中存放的是文档的元数据,也就是说,这里存放着一些负责描述或处理文档内容的元素,而浏览器不会直接显示这些元素。示例文档的head区域有3个元数据元素,分别是title、script和style。title元素最简单,其内容被浏览器用于设定窗口或标签的标题;每个HTML文档都需要一个title元素。另两个元素对于本书来说更重要,所以我专门在下面详细介绍。

### 1. script元素

多亏有script元素,我们才能在代码中包含JavaScript脚本。等到后面我们深入讲解jQuery时,这个元素的讲解将占用大量篇幅。示例文档里有一个script元素,见代码清单2-7。

代码清单2-7 示例文档中的script元素

```
...  
<script src="jquery-2.0.2.js" type="text/javascript"></script>  
...
```

当定义script元素的src属性时,我们其实是在告诉浏览器需要从另一个文件中载入JavaScript代码。在本例中,这个文件是jQuery核心库,浏览器会在jquery-2.0.2.js文件中读取这些代码。一个HTML文档可以包含多个script元素,如果需要,我们也可以直接把JavaScript代码放到script元素的开始和结束标签之间,如代码清单2-8所示。

代码清单2-8 使用script元素定义内置JavaScript代码

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Example</title>  
  <script src="jquery-2.0.2.js" type="text/javascript"></script>  
  <script type="text/javascript">  
    $(document).ready(function() {  
      $('#mainheader').css("color", "red");  
    });  
  </script>  
</head>  
<body>  
  <h1 id="mainheader" class="header">Welcome to Jacqui's Flower Shop</h1>  
  <h2 class="header info">We are open 10am-6pm, 7 days a week</h2>  
  <h3 class="info">(closed on national holidays)</h3>  
</body>  
</html>
```

上面这个示例有两个script元素,第一个把jQuery库导入到文档中,第二个是一段简单的脚本,使用了一些jQuery基础功能。如果这会儿你搞不懂第二个脚本做了些什么,不必担心,第5章我们开始学习jQuery库。script元素可出现在HTML文档中的head或body元素内。在本书中,我仅仅是出于个人偏好,更喜欢把script标签放置到head元素内。

---

**提示** script元素的顺序很重要,我们必须先导入jQuery库再加载自己编写的代码。

---



## 2. style元素

style元素用来在HTML文档中引入CSS(层叠样式表)。如果非要用一句话来介绍CSS,那就是CSS负责文档在浏览器中的呈现。代码清单2-9展示了示例文档中的style元素及其内容。

代码清单2-9 使用style元素

```
...
<style>
  h1 {
    width: 700px; border: thick double black; margin-left: auto;
    margin-right: auto; text-align: center; font-size: x-large; padding: .5em;
    color: darkgreen; background-image: url("border.png");
    background-size: contain; margin-top: 0;
  }
  .dtable {display: table;}
  .drow {display: table-row;}
  .dcell {display: table-cell; padding: 10px;}
  .dcell > * {vertical-align: middle}
  input {width: 2em; text-align: right; border: thin solid black; padding: 2px;}
  label {width: 5em; padding-left: .5em; display: inline-block;}
  #buttonDiv {text-align: center;}
  #oblock {display: block; margin-left: auto; margin-right: auto; width: 700px;}
</style>
...
```

浏览器维护着一整套属性,利用这些属性的值控制着每个元素的呈现。style元素允许我们选取元素并修改属性的值。我会在第3章详细介绍这些属性。

style元素类似于script元素,可以出现在head和body元素内,不过在本书中,你会发现我总是把style标签放在head区域,就像示例文档中那样。但这只是我的另一个个人爱好,我喜欢把样式和内容分开。

### 2.6.2 内容元素

body元素包裹着HTML文档的内容。这些元素将由浏览器显示给用户,是那些元数据元素(如script和style)的操作对象。

#### 1. 语义与表现分离

HTML5对传统HTML最重要的革新是哲学层面的:让元素的语义和内容表现分离。这是一个明智的决定。我们使用HTML元素赋予内容以结构和语义,然后通过CSS控制元素内容的呈现。由于不是所有的HTML使用者都需要显示页面内容(有些使用者是一些并非浏览器的自动化程序),保持表现层与语义层的分离可以简化自动化处理和文档含义的解析。

这正是HTML的核心理念:使用元素定义要处理的内容。人们非常擅长根据上下文推断真实含义,比如你会因为这一节标题的排版字号比上一个标题小一号,而即时意识到它们之间的从属关系(因为这已成为一个模式,绝大多数非小说书籍都遵循这个模式)。

计算机并不擅长推导上下文,因此每个HTML元素都有具体的含义。比方说,article元素定义了一个独立且适合发布的内容片段,h1元素用来定义内容的标题。代码清单2-10展示了一个使用元素定义结构与语义的示例文档。

## 代码清单2-10 使用HTML元素为内容添加结构和语义

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
</head>
<body>
  <article>
    <header>
      <hgroup>
        <h1>New Delivery Service</h1>
        <h2>Color and Beauty to Your Door</h2>
      </hgroup>
    </header>
    <section>
      We are pleased to announce that we are starting a home delivery service for
      your flower needs. We will deliver within a 20 mile radius of the store
      for free and $1/mile thereafter. All flowers are satisfaction-guaranteed and
      we offer free phone-based consultation.
    </section>
    <section>
      Our new service starts on <b>Wednesday</b> and there is a $10 discount
      for the first 50 customers.
    </section>
    <footer>
      <nav>
        More Information:
        <a href="http://jacquisflowershop.com">Learn More About Fruit</a>
      </nav>
    </footer>
  </article>
</body>
</html>

```

关于什么时候使用section什么时候使用article并没有严格的铁律，不过我建议你在处理自己的内容时保持一致。像section和article这种元素中包含的内容应该如何显示，浏览器毫不知情，而这正是语义/表现分离的核心思想。对于绝大多数HTML元素，浏览器都有默认样式，以便在用户未设置样式时决定这些元素如何显示。关键点在于，我们可以自由根据需要使用CSS改变文档的呈现效果。这种改变表现层的任务可以使用style元素完成，也可以使用script元素引入jQuery轻松完成。

在互联网的“远古”时代，那时还没有内容与表现分离的思想，HTML4添加了一些元素，这些元素让今天的我们非常尴尬。b元素就是一个典型的例子。直到HTML5，b元素告诉浏览器它包含的内容要加粗。在HTML5中，我们并不需要纯粹处理呈现效果的元素，因此我们给b元素下了一个新的定义。

b元素仅仅表示一段文本，它包含的内容并无任何表示强调或者重要的含义，它表现为加粗文本仅仅是缘于排版习惯；比如一份文档摘要中的关键词，或者一次评估中的产品名字。

——HTML: The Markup Language, w3c.org

上面这个冗长的定义只是在告诉我们：b元素就是告诉浏览器让文本加粗。b元素没有语义，它完全是为了呈现而存在。上面这段纯属狡辩之辞的定义只是告诉我们一点有关HTML5的重要信息：现在还处于过渡期。我们渴望一个元素和它的表现完全分离的新世界，现实世界必须兼容那数不尽的使用

老版本HTML编写的页面，我们必须妥协。

## 2. 表单和输入框

示例文档中body元素中的form元素很有意思。form建立了一种收集用户输入数据的机制，之后我们可以把这些数据发送到服务器。在第三部分你会看到，jQuery有几个对表单支持特别出色的功能，有些是由jQuery核心库直接提供的，有些是由一些常用插件提供的。代码清单2-11展示的是示例文档中的body元素及其内容，其中包括form元素。

代码清单2-11 示例文档中的内容元素

```
...
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
        <div class="drow">
          <div class="dcell">
            <label for="aster">Aster:</label>
            <input name="aster" value="0" required>
          </div>
          <div class="dcell">
            <label for="daffodil">Daffodil:</label>
            <input name="daffodil" value="0" required >
          </div>
          <div class="dcell">
            <label for="rose">Rose:</label>
            <input name="rose" value="0" required>
          </div>
        </div>
        <div class="drow">
          <div class="dcell">
            <label for="peony">Peony:</label>
            <input name="peony" value="0" required>
          </div>
          <div class="dcell">
            <label for="primula">Primula:</label>
            <input name="primula" value="0" required>
          </div>
          <div class="dcell">
            <label for="snowdrop">Snowdrop:</label>
            <input name="snowdrop" value="0" required>
          </div>
        </div>
      </div>
      <div id="buttonDiv"><button type="submit">Place Order</button></div>
    </form>
  </body>
...
```

只要你看到form元素，通常就会在附近看到input元素。input元素用来收集用户输入的信息。代码清单2-12展示了一个摘自上面文档的input元素。

### 代码清单2-12 input元素

```
...
<input name="snowdrop" value="0" required>
...
```

这个input元素负责收集用户输入的名为snowdrop的数据项，它有一个初始值0。required属性告诉浏览器：如果用户不填写这个数据项，就不能提交表单。这是一个来自HTML5的称为表单验证的新特性。不过坦白地说，在第三部分，我会教大家使用jQuery实现更好的验证效果。

与表单密切相关的是button元素，我们常常使用它提交表单到服务器（也可以用它清空表单回到初始状态）。代码清单2-13展示了我定义在示例文档中的button元素。

### 代码清单2-13 button元素

```
...
<button type="submit">Place Order</button>
...
```

在上面的button元素中type属性的值被设置为submit，这是告诉浏览器：当这个按钮被按下时，提交表单。在浏览器中，button元素的内容会显示在按钮上，如图2-3所示。

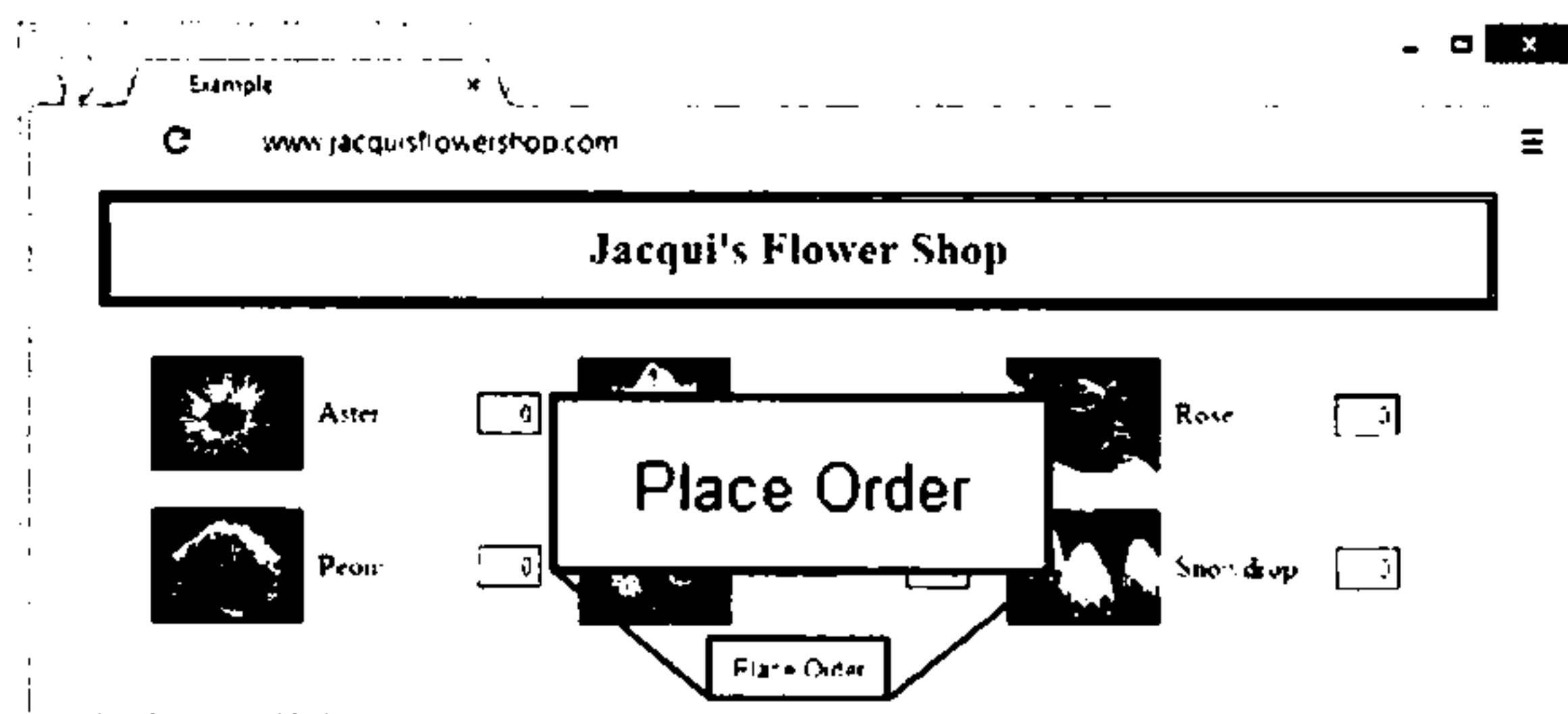


图2-3 button元素的内容

### 3. 表示结构的元素

注意，示例文档中有大量div元素。这是一个没有具体语义的元素，常常用来控制内容的布局。在示例文档中，我使用div元素生成一个表格布局，这样这些div元素包含的元素在用户看来就像一个表格。style元素中有一些针对这些div元素的CSS，控制生成这种布局。CSS的使用会贯穿本书，因此我会在第3章讲解CSS基础。

### 4. 与外部资源有关的元素

有些元素允许我们在文档中引入外部资源。img元素就是一个非常典型的例子，我们可以使用它往文档里添加图片。在示例文档中，我使用了img元素显示那些正在销售的鲜花，如代码清单2-14所示。

### 代码清单2-14 使用img元素引用外部图片

```
...

...
```

src属性用于指定图片。我在这儿用到的图片是snowdrop.png，它是一个相对URL，也就是说浏览器要根据当前页面（包含该元素的文档）的URL计算出它真正的URL。

和相对URL对应的是绝对URL，又称完整URL（full qualified URL）。如图2-4所示，绝对URL包括所有预先定义好的基础部件。图中的URL还包括端口，如果省略端口的话，浏览器就会使用协议（scheme）的默认端口；对于http协议来说，就是80端口。



图2-4 URL的基本结构

由于对每个资源文件都键入完整URL实在太无聊，相对URL才大行其道。当我把这个img元素的src属性指定为snowdrop.png时，其实是在告诉浏览器可以在存放当前页面的地方找到这张图片。表2-2展示了我们可以使用的各种相对URL及其对应的绝对URL。所有这些URL都假定当前页面是http://www.jacquisflowershop.com/jquery/example.html。

表2-2 相对URL格式

相对URL	对应的绝对URL
snowdrop.png	http://www.jacquisflowershop.com/jquery/snowdrop.png
/snowdrop.png	http://www.jacquisflowershop.com/snowdrop.png
/	http://www.jacquisflowershop.com/
//www.mydomain.com/index.html	http://www.mydomain.com/index.html

由于上表中最后一种格式节省不了多少击键，因此很少使用，不过它确实能保证请求的资源文件所用协议与当前页面相同。当页面上一些内容通过加密连接（使用https协议）获取，而另一些内容使用非加密协议（http协议）获取时，这能避免不小心键入错误协议的问题。有些浏览器，特别是IE，不赞同在同一页面混用安全内容和非安全内容，一旦遇到这种情况，它就会向用户发出警告信息。

**警告** 我们可以用两个英文句点（..）表示相对于当前页面的路径。不过我建议不要使用这种技术，因为出于安全考虑一些浏览器会拒绝包含两个句点的URL请求。

## 2.7 元素层次关系

HTML文档中的元素自然而然的形成一种层次关系。html元素包含body元素，body元素又包含内容元素，每个内容元素又可以包含其他元素，如此这般，以至无穷。

当我们需要处理一个页面时，无论是使用CSS设置样式（第3章详细讲解）还是使用jQuery找出页面中的元素（第5章和第6章详细讲解），透彻理解这种层次关系都会非常有帮助。

页面中元素之间的关系是层次关系里最为重要的部分。为了方便讲解，我使用一张图（如图2-5所示）重新描述花店示例页面中某些元素之间的层次关系。

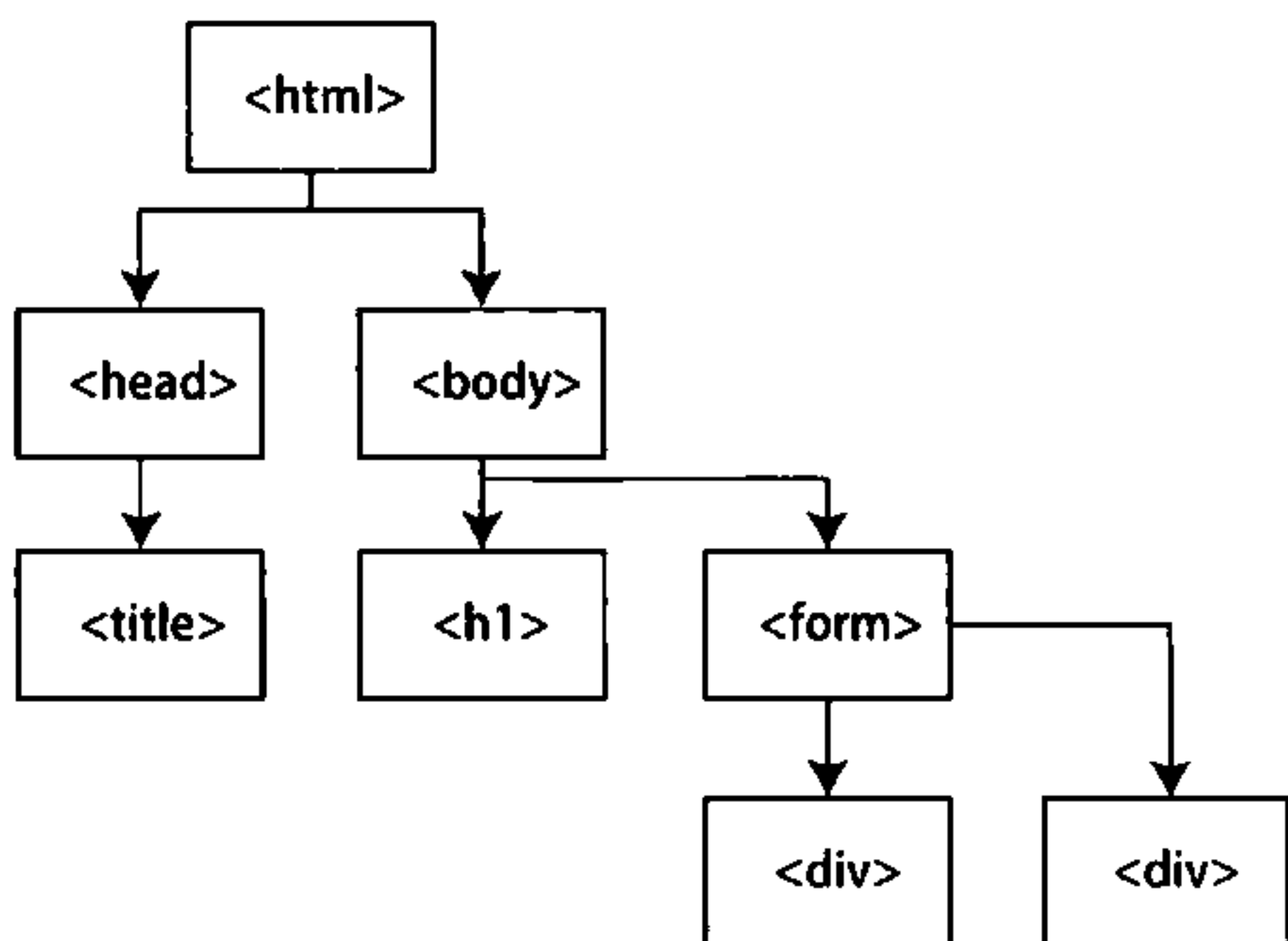


图2-5 页面元素的部分层次关系

这张图只是示例页面元素层次关系图谱的一部分，从中可以看出一个元素可以直接包含另一个元素。元素之间的关系有多种，接下来我会详细介绍。

### 2.7.1 父子关系

举个例子，当一个元素直接包含另一个时，二者之间就是父子关系。如图2-5所示，form元素是body元素的子元素，而body元素是form元素的父元素。一个元素可以有多个子元素，但只能有一个父元素。在上例中，body元素有两个子元素（form和h1），它是这两个元素共同的父元素。

只有当一个元素直接包含另一个（或多个）元素时，它们之间才是父子关系。比如那些div元素是form的子元素，但不是body的子元素。

俗话说“龙生九子，各不相同”，在页面代码中第一个出现的子元素是父元素的第一个“孩子”。上例中body元素的第一个“孩子”是h1元素。在页面代码中最后出现的子元素是父元素的最后一个“孩子”。如此说来，body元素的最后一个“孩子”是form元素。我们还可以根据顺序号引用第 $n$ 个“孩子”，从第一个“孩子”开始数起（第一个“孩子”的序号为1），直到第 $n$ 个。

### 2.7.2 祖先—后代关系

一个元素的后代包括它的“孩子”、“孩子的孩子”，等等。实际上，任何元素直接或间接包含的元素都是它的后代元素。上面的例子中，body元素的后代有h1、form和所有的div元素，并且图中显示的所有元素（html元素除外）都是html元素的后代元素。

与后代元素相对应的是一个元素的祖先元素，包括它的父元素、父元素的父元素，等等。举个例子，form元素是body元素和html元素的后代元素。两个div元素拥有共同的祖先：form、body和html元素。

### 2.7.3 兄弟关系

拥有同一个父元素的元素是兄弟元素。在图2-5中，因为h1元素和form元素拥有共同的父元素body，所以它们是兄弟元素。在处理兄弟元素时，我们习惯上使用“下一个兄弟元素”和“上一个兄弟元素”。



这种字眼。它们分别对应着当前元素之前和之后的兄弟元素。不是所有的元素都既有上一个兄弟元素又有下一个兄弟元素，第一个子元素和最后一个子元素都只有其中的一个。<sup>①</sup>

## 2.8 文档对象模型

当浏览器载入并处理HTML文档时，它会根据文档生成具体的DOM（Document Object Model，文档对象模型）。DOM是用来表示文档中所有元素的JavaScript对象模型，也是一种使我们能够与HTML文档交互的机制。

**注意** 从本质上说，浏览器支持的任何语言实现都能够使用DOM。事实上所有主流浏览器都支持JavaScript，因此我也就不再区分作为抽象概念的DOM和作为JavaScript对象集合的DOM了。

我们之所以关心前面讲过的那些元素之间的关系，一个原因就是这些关系保存在DOM内。因为这些关系保存在DOM中，我们才能使用JavaScript遍历DOM对象网络，了解这些对象的本质和结构。

**提示** 使用DOM就意味着使用JavaScript。如果需要复习JavaScript基础知识，请翻阅第4章。

在本章的这一部分，我会演示DOM的一些基本特性。在本书后面的部分，我会重点讲解如何使用jQuery访问DOM，不过这里先教你使用内建方法访问DOM，以便将来和更优雅的jQuery解决方法进行对比。

### 2.8.1 使用DOM

DOM中有一个基础JavaScript对象HTML`Element`，它定义了一些基本的属性和方法，这些属性和方法对所有类型的元素都有效。表2-3中列出了部分常用属性。

表2-3 HTML`Element`对象的一些基本属性

属 性	描 述	返回值（类型）
<code>className</code>	获取或设置一个元素的class属性列表	string
<code>id</code>	获取或设置一个元素的id属性值	string
<code>lang</code>	获取或设置一个元素的lang属性值	string
<code>tagName</code>	返回（标识元素类型的）标签名	string

除了表中列出的这几个，HTML`Element`对象还有很多属性。具体的属性集合与使用的HTML版本有关。不过对于我来说，用上面这4个属性来演示DOM如何工作已经足够了。

DOM使用从HTML`Element`对象派生来的各种对象表示具体的元素。举个例子，HTML`ImageElement`

<sup>①</sup> 这种说法不够严谨。当一个父元素只有一个子元素时，它既没有上一个兄弟元素，也没有下一个兄弟元素。

——译者注

对象用来表示DOM中的img元素，这个对象有一个src属性，对应着img元素的src属性。我不会深入介绍任何一种具体元素对象，不过作为一条规则，我们可以认为：一个元素有哪些属性，与它相对应的元素对象就具有哪些属性。

我们使用全局变量document来访问DOM，它是Document对象的实例。Document对象用来表示浏览器窗口中正在显示的HTML文档，而且它定义了很多方法（如表2-4所示），我们可以利用这些方法在DOM中查找具体的元素对象。

表2-4 Document对象提供的用于查找元素的方法

方 法	描 述	返 回 值
getElementById(<id>)	返回具有指定id值的元素	HTMLElement
getElementsByClassName(<class>)	返回具有指定class值的元素集合	HTMLElement[]
getElementsByTagName(<tag>)	返回参数指定标签名字对应的元素集合	HTMLElement[]
querySelector(<selector>)	返回匹配指定CSS选择器的第一个元素	HTMLElement
querySelectorAll(<selector>)	返回匹配指定CSS选择器的元素集合	HTMLElement[]

再强调一次，之所以选择这些方法，这是因为本书中用到了这些方法。表2-4中最后两个方法提到的CSS选择器将在第3章讲解。代码清单2-15演示怎样使用Document对象查找文档中某种具体类型的元素。

#### 代码清单2-15 在DOM中查找元素

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <style>
    h1 {
      width: 700px; border: thick double black; margin-left: auto;
      margin-right: auto; text-align: center; font-size: x-large; padding: .5em;
      color: darkgreen; background-image: url("border.png");
      background-size: contain; margin-top: 0;
    }
    .dtable {display: table;}
    .drow {display: table-row;}
    .dcell {display: table-cell; padding: 10px;}
    .dcell > * {vertical-align: middle}
    input {width: 2em; text-align: right; border: thin solid black; padding: 2px;}
    label {width: 5em; padding-left: .5em; display: inline-block;}
    #buttonDiv {text-align: center;}
    #oblock {display: block; margin-left: auto; margin-right: auto; width: 700px;}
  </style>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
```

```

    <div class="drow">
      <div class="dcell">
        <label for="aster">Aster:</label>
        <input name="aster" value="0" required>
      </div>
      <div class="dcell">
        <label for="daffodil">Daffodil:</label>
        <input name="daffodil" value="0" required >
      </div>
      <div class="dcell">
        <label for="rose">Rose:</label>
        <input name="rose" value="0" required>
      </div>
    </div>
    <div class="drow">
      <div class="dcell">
        <label for="peony">Peony:</label>
        <input name="peony" value="0" required>
      </div>
      <div class="dcell">
        <label for="primula">Primula:</label>
        <input name="primula" value="0" required>
      </div>
      <div class="dcell">
        <label for="snowdrop">Snowdrop:</label>
        <input name="snowdrop" value="0" required>
      </div>
    </div>
  </div>
</div>
<div id="buttonDiv"><button type="submit">Place Order</button></div>
</form>
<script>
  var elements = document.getElementsByTagName("img");
  for (var i = 0; i < elements.length; i++) {
    console.log("Element: " + elements[i].tagName + " " + elements[i].src);
  }
</script>
</body>
</html>

```

在上面这个例子中，我在body元素的末尾添加了一个script元素。当浏览器在一个页面中发现一个script元素，它就会立刻执行其中的JavaScript语句，即使后面还有内容没有加载或处理完毕。这为我们使用DOM带来了麻烦，因为这意味着通过Document对象查找DOM元素时，（很可能）要查找的内容还没在DOM中创建出来（因为页面还没有加载完）。为了避免这个问题，我把script元素放到了页面的末尾。jQuery提供了一个更好的方法来处理这个问题，详见第5章。

在这个脚本中，我使用getElementsByTagName方法查找文档中所有的img元素，该方法返回由对象组成的一个数组。我随后遍历这个数组，在控制台打印出tagName属性和src属性的值。控制台中的输出如下：

```

Element: IMG http://www.jacquisflowershop.com/jquery/aster.png
Element: IMG http://www.jacquisflowershop.com/jquery/daffodil.png
Element: IMG http://www.jacquisflowershop.com/jquery/rose.png
Element: IMG http://www.jacquisflowershop.com/jquery/peony.png
Element: IMG http://www.jacquisflowershop.com/jquery/primula.png
Element: IMG http://www.jacquisflowershop.com/jquery/snowdrop.png

```

## 2.8.2 修改DOM

DOM中的对象都是“实时”的，这意味着对DOM对象的属性进行修改会立即影响浏览器中页面的显示。代码清单2-16展示了一段具有此类效果的脚本（为节省篇幅，我只列出了script元素部分）。页面其他部分与代码清单2-15完全相同。

代码清单2-16 修改DOM对象的属性

```

...
<script>
    var elements = document.getElementsByTagName("img");
    for (var i = 0; i < elements.length; i++) {
        elements[i].src = "snowdrop.png";
    }
</script>
...

```

在这个脚本中，我把所有img元素的src属性都设置成了snowdrop.png。图2-6展示了该脚本产生的效果。

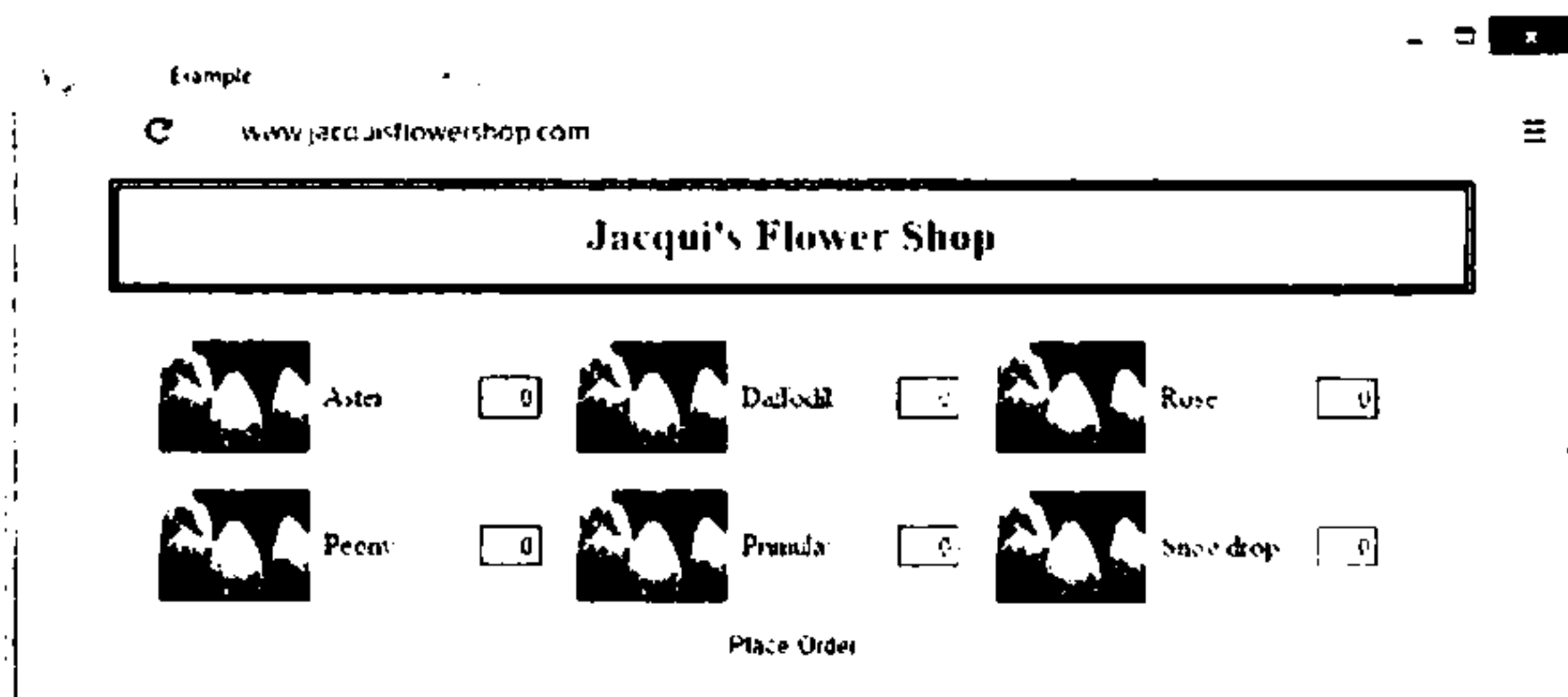


图2-6 使用DOM修改HTML文档

## 2.8.3 修改样式

我们能够使用DOM修改CSS属性的值。（如果你需要，第3章专门介绍了CSS基础知识。）DOM提供了极其丰富的API来处理CSS，不过处理CSS最简单的方法是使用HTMLElement对象中的style属性。style属性返回的对象定义了该元素style属性中定义的各种CSS属性。（抱歉，我不得不在这一句话使用3个“属性”来说清楚情况。）

CSS中样式属性的命名与style对象中相应属性的命名稍有不同。比如说，CSS属性background-color在style对象中就成了style.backgroundColor属性。代码清单2-17演示如何使用DOM处理样式。

代码清单2-17 使用DOM修改元素样式

```
...
<script>
    var elements = document.getElementsByTagName("img");
    for (var i = 0; i < elements.length; i++) {
        if (i > 0) {
            elements[i].style.opacity = 0.5;
        }
    }
</script>
...
```

在这个脚本中，我修改了除第一个img元素外所有其他img元素的opacity属性。之所以留下一个元素不做修改，这是为了方便你进行对比（见图2-7）。

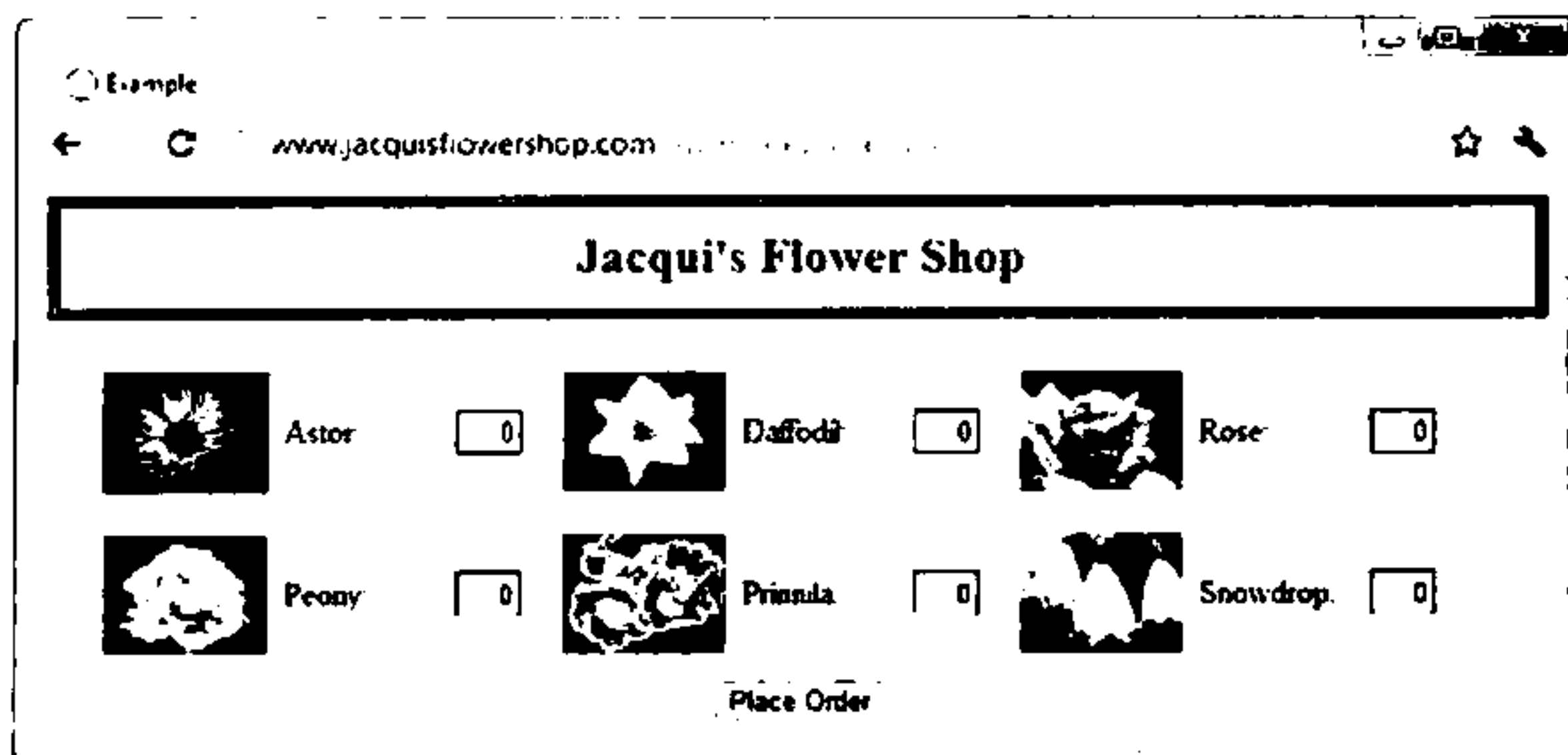


图2-7 使用JavaScript修改CSS属性值

## 2.8.4 处理事件

事件是浏览器发送的信号，表示DOM中一个或多个元素的状态发生了变化。状态变化有很多种，不同的变化对应着不同的事件。举例来说，当用户点击页面中的元素时，就会触发click事件；当用户提交表单时，就会触发表单的submit（提交）事件。有些事件是相关的，比如用户鼠标挪到某个元素之上时会触发mouseover事件，用户把鼠标从元素上移开时会触发mouseout事件。为DOM事件关联相应的JavaScript函数（即事件处理函数），我们就可以响应这个事件。事件每发生一次，事件处理函数中的语句就执行一次。代码清单2-18给出了一个处理事件的示例。

代码清单2-18 处理一个事件

```
...
<script>
    var elements = document.getElementsByTagName("img");
```

```

    for (var i = 0; i < elements.length; i++) {
        elements[i].onmouseover = handleMouseOver;
        elements[i].onmouseout = handleMouseOut;
    }
    function handleMouseOver(e) {
        e.target.style.opacity = 0.5;
    }
    function handleMouseOut(e) {
        e.target.style.opacity = 1;
    }
</script>
...

```

这段脚本定义了两个处理函数，我分别把它们分配给了img元素对应DOM对象的onmouseover属性和onmouseout属性。这个脚本的效果是：当鼠标放到图片上时，图片就会变成半透明（透明度0.5），当鼠标离开时，图片又恢复正常（透明度1）。由于本书第二部分会专门讲解jQuery对DOM事件的支持，在这儿我就不深入讲解DOM API的事件处理机制了。不过，你需要观察一下传递给事件处理函数的那个对象：对，就是Event对象。表2-5列出了Event对象中定义的几个最重要的成员。

表2-5 Event对象的方法和属性

名 称	描 述	返 回 值
type	事件名，比如mouseover	string
target	事件对应的目标元素	HTMLElement
currentTarget	正在调用的事件侦听器对应的元素	HTMLElement
eventPhase	事件生命周期的阶段数	数字
bubbles	如果事件通过冒泡方式传递，返回true，否则返回false	boolean
cancelable	如果事件有一个可以取消的默认行为，返回true，否则返回false	boolean
stopPropagation()	执行完当前元素的事件处理函数后终止事件在元素树中的继续传播	void
stopImmediatePropagation()	立刻终止事件在元素树中的传播；当前元素尚未执行的事件处理函数会被忽略	void
preventDefault()	阻止浏览器执行事件的默认行为	void
defaultPrevented	如果preventDefault()被调用过则返回true	boolean

在前面的例子中，为了得到触发事件的元素，我使用event对象的target属性。几个与事件流和默认行为有关的其他属性将在下一节简单介绍。这一章的内容主要是为后面各章奠定基础。

### 1. 了解事件流

一个事件在它的生命周期中要经历3个阶段：捕获、处理目标元素和冒泡。当事件发生时，浏览器首先要找出是哪个元素触发了这个事件，这个元素又称为事件的目标元素（target）。浏览器会询问body元素与目标元素之间的所有元素，挨个检查它们（它们又去问自己的后代是否接到事件通知）是否定义有事件处理函数。在执行目标元素的事件处理函数之前，浏览器会先执行祖先元素绑定的事件处理函数。（第二部分介绍如何请求后代事件的通知。）

一旦捕获阶段完成，接下来进入处理目标元素阶段，这也是3个阶段中最简单的一个。当捕获阶段完成，浏览器会触发执行绑定在目标元素该事件上的所有侦听函数（事件处理函数）。



处理目标元素完成之后，浏览器开始向着body元素方向上行检查每个祖先元素（冒泡）。浏览器会一个个询问这些元素是否绑定有非捕获型事件（我会在第二部分讲解这是怎么回事），因为不是所有的事件都支持冒泡。我们可以检查事件的bubbles属性来查看事件是否支持冒泡。如果该属性的值为true，则表示该事件支持冒泡，否则就是不支持。

## 2. 了解默认行为

有些事件具有预定义行为，当事件发生时会自动触发这种行为。举例来说，a元素click事件的默认行为是：载入该元素href属性中的URL对应的内容。如果一个事件拥有默认行为，那么它的cancelable属性值就是true。调用preventDefault()方法可以终止事件的默认行为。值得注意的是，调用preventDefault()方法不会阻止事件流继续走完捕获、处理目标元素和冒泡三阶段。这3个阶段仍会按部就班地走完，只是在最后的冒泡阶段浏览器不再执行事件的默认行为。我们可以通过在早一点的事件处理函数中读取defaultPrevented属性值来测试preventDefault()方法是否已经执行过。如果这个属性的值为true，说明preventDefault()方法已经执行过了。

## 2.9 小结

本章虽然没有具体讲解那100多个元素，但我们一起了解了HTML工作原理。相信大家了解了怎样创建基础HTML文档的结构、元素怎样混合包含文本和其他元素，以及如何推导出元素层次的具体关系。在本章我们还一起了解了DOM API的基本用法，以及如何选择元素和处理事件——在后面你会发现，使用jQuery一个最重要的理由，就是它隐藏了DOM API的细节，让我们能够极其简单地处理HTML元素和表示HTML元素的JavaScript对象。接下来的第3章，是层叠样式表（CSS）的快速入门教程，主要讲解如何利用CSS控制HTML元素的外观。

层叠样式表（CSS）是用来控制HTML元素呈现的方式。从两个方面看，CSS对于jQuery有着分外重要的意义。首先，我们使用CSS选择器（本章讲解）告诉jQuery如何在页面中查找元素；其次，使用jQuery修改元素CSS样式是最常见的任务。

全部的CSS属性超过130种，每种属性控制着元素表现的某个侧面。与HTML元素类似，因为CSS属性如此之多，本章无法做到面面俱到，因此只能着重讲解CSS的工作原理，以及如何为元素设置样式。如果希望深入了解CSS各方面的知识，建议你阅读我的另一本书：《HTML5权威指南》。

## 3.1 上手 CSS

要在屏幕上显示某个元素，浏览器首先要计算该元素的各个CSS属性，然后才能知道将该元素显示成什么样子。代码清单3-1展示了一个非常简单的HTML文档。

代码清单3-1 一个简单的HTML文档

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
</head>
<body>
  <h1>New Delivery Service</h1>
  <h2>Color and Beauty to Your Door</h2>
  <h2>(with special introductory offer)</h2>
  <p>We are pleased to announce that you are starting a home delivery service for
  your flower needs. You will deliver within a 20 mile radius of the store
  for free and $1/mile thereafter.</p>
</body>
</html>
```

在浏览器中，这个页面会是什么样子的？请看图3-1。

虽然本章无法做到事无巨细地详细讲解所有CSS属性，但是我们仍然可以通过研究较少一部分属性（见表3-1）学习CSS的工作原理。



图3-1 显示在浏览器中的一个简单页面

表3-1 一些CSS属性

属 性	描 述
color	设置元素的前景色（通常是设置文本的颜色）
background-color	设置元素的背景色
font-size	设置元素包含文本的字号
border	设置元素的边框

在示例页面中，我并没有定义这些属性，然而浏览器仍然按照一定的规则显示出这些元素的内容。如图3-1所示，每种元素的显示方式各不相同。即使我们并未对这些属性做任何设置，浏览器仍然会显示这些元素。每一种元素都有默认样式，也就是说，浏览器为这些元素内建了样式。如果我们不做任何设置，浏览器就会使用默认值。尽管HTML标准中对元素的默认外观有一些规定，但是各个浏览器却有“微创新”的自由。因此，同一元素在不同浏览器（如Chrome和IE浏览器）中的默认样式常常稍有不同。表3-2展示的是Google Chrome浏览器定义的一些默认值。

表3-2 一些CSS属性默认值

属 性	h1	h2	p
color	black	black	black
background-color	transparent	transparent	transparent
font-size	32px	24px	16px
border	none	none	none

从表中可以看出，3种元素的color、background-color和border属性都相同，只有font-size属性不同。稍后我会在本章讲解属性值的单位。现在我们只关心如何设置属性值，先不理睬这些单位的具体含义。

## 3.2 行内样式

为元素设置style属性是设置样式最直接的方式。代码清单3-2展示设置行内样式的方法。

## 代码清单3-2 使用style属性设置一个元素的CSS属性

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
</head>
<body>
  <h1>New Delivery Service</h1>
  <h2 style="background-color: grey; color: white">Color and Beauty to Your Door</h2>
  <h2>(with special introductory offer)</h2>
  <p>We are pleased to announce that we are starting a home delivery service for
your flower needs. We will deliver within a 20 mile radius of the store
for free and $1/mile thereafter.</p>
</body>
</html>

```

在上面这个示例中，我们使用style属性设置了两个CSS属性（样式声明）。图3-2是属性值剖析图。

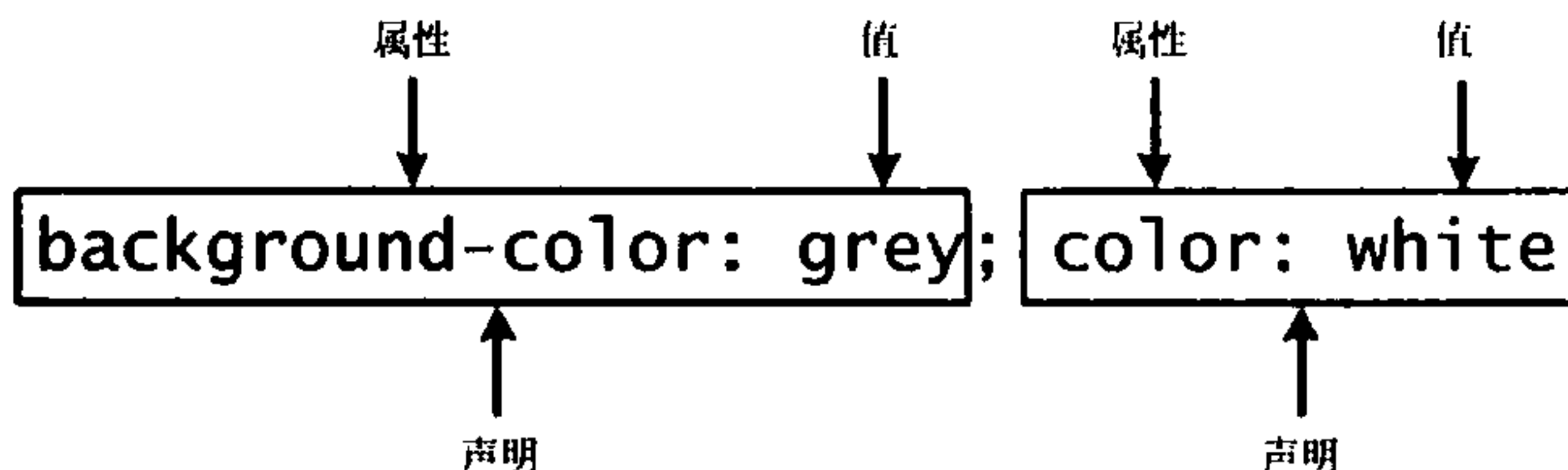


图3-2 样式属性及其值的剖析图

一个样式声明包含你打算设置的属性的名称和值，名称和值之间以冒号分隔。一次可以声明多个属性，多个属性之间用分号隔开。在上图中，我把background-color属性设置为grey，color属性设置为white。由于这些样式是通过h2元素的style属性设置的，因此只影响这一个h2元素，即页面中的其他元素不受任何影响，都会保持原样，即便它们也是h2元素。图3-3展示了第一个h2元素设置样式之后的效果。



图3-3 通过style属性修改第一个h2元素样式之后的显示效果

## 3.3 内嵌样式

用style属性设置样式非常简单，然而它设置的样式只能作用于一个元素。我们可以通过这种方式为元素设置样式，但是这样设置的样式非常难以管理和维护，特别是过上一段时间之后需要修订样式的时候。有一种更强大的技术，那就是使用style元素(而非style属性)定义内嵌样式(embedded style)，用内嵌样式告诉浏览器应该把什么样式应用到什么元素上。代码清单3-3展示了内嵌样式的用法。

代码清单3-3 定义内嵌样式

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <style>
    h2 { background-color: grey; color: white;}
  </style>
</head>
<body>
  <h1>New Delivery Service</h1>
  <h2>Color and Beauty to Your Door</h2>
  <h2>(with special introductory offer)</h2>
  <p>We are pleased to announce that we are starting a home delivery service for
  your flower needs. We will deliver within a 20 mile radius of the store
  for free and $1/mile thereafter.</p>
</body>
</html>
```

我们在内嵌样式中仍然使用声明，不过要注意一点：声明要放在大括号内（{和}），并且大括号前要有元素选择器。内嵌样式的剖析图见图3-4。

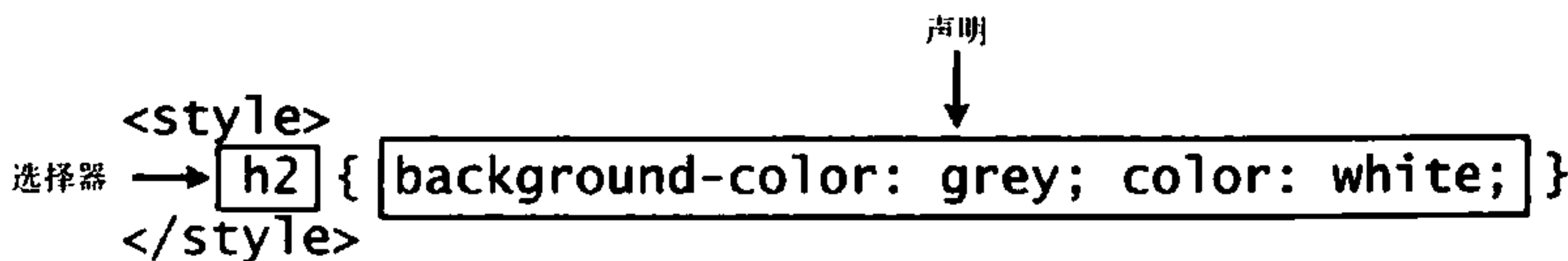


图3-4 内嵌样式剖析图

**提示** 上面的例子中，我将style元素放在head元素中，但其实也可以把它放到body元素中，显示效果完全相同。但我喜欢把内嵌样式放到head元素之中，因为这样符合内容与控制内容呈现的CSS相分离的原则。

在jQuery中CSS选择器非常重要，因为它们是在DOM中匹配元素的基础。毕竟我们要对DOM中的某些元素进行某种操作，就总得先找到它。在上面的例子中我使用的选择器是h2，这表明后面大括号内的样式声名将作用于页面中的所有h2元素。图3-5展示了所有h2元素的显示效果。

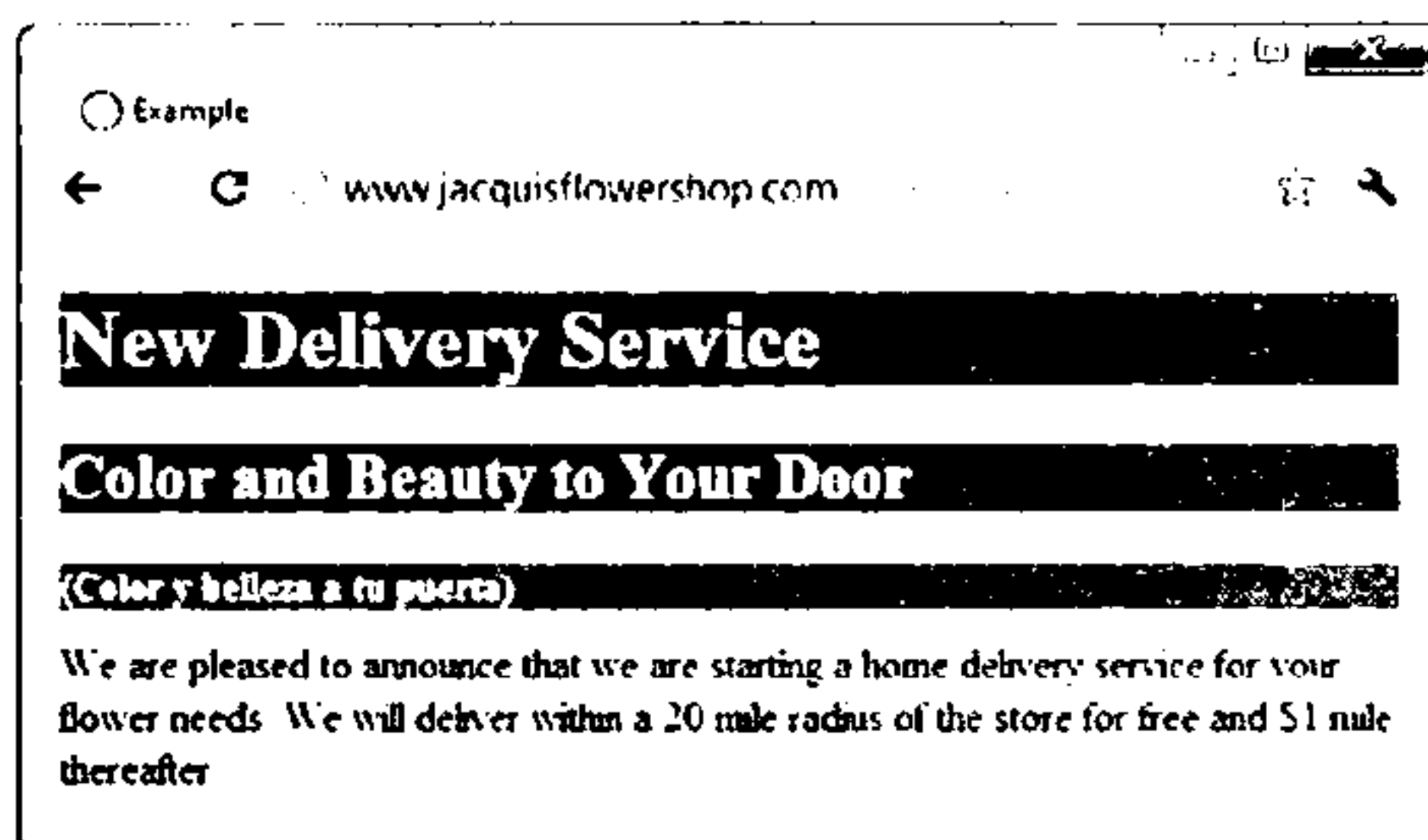


图3-5 内嵌样式显示效果

与内嵌样式相比，`style`元素能包含更多的样式。代码清单3-4展示的仍然是我们在第2章一开始看到的花店页面，只是现在多了一些相对复杂的样式。

#### 代码清单3-4 HTML页面中相对比较复杂的样式

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <style>
    h1 {
      min-width: 700px; border: thick double black; margin-left: auto;
      margin-right: auto; text-align: center; font-size: x-large; padding: .5em;
      color: darkgreen; background-image: url("border.png");
      background-size: contain; margin-top: 0;
    }
    .dtable {display: table;}
    .drow {display: table-row;}
    .dcell {display: table-cell; padding: 10px;}
    .dcell > * {vertical-align: middle}
    input {width: 2em; text-align: right; border: thin solid black; padding: 2px;}
    label {width: 5em; padding-left: .5em; display: inline-block;}
    #buttonDiv {text-align: center;}
    #oblock {display: block; margin-left: auto; margin-right: auto; min-width: 700px;}
  </style>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
        <div class="drow">
          <div class="dcell">
            <label for="aster">Aster:</label>
            <input name="aster" value="0" required>
          </div>
          <div class="dcell">
```

```

        <label for="daffodil">Daffodil:</label>
        <input name="daffodil" value="0" required >
    </div>
    <div class="dcell">
        <label for="rose">Rose:</label>
        <input name="rose" value="0" required>
    </div>
</div>
<div class="drow">
    <div class="dcell">
        <label for="peony">Peony:</label>
        <input name="peony" value="0" required>
    </div>
    <div class="dcell">
        <label for="primula">Primula:</label>
        <input name="primula" value="0" required>
    </div>
    <div class="dcell">
        <label for="snowdrop">Snowdrop:</label>
        <input name="snowdrop" value="0" required>
    </div>
</div>
</div>
</div>
<div id="buttonDiv"><button type="submit">Place Order</button></div>
</form>
</body>
</html>

```

本例中我使用style元素定义了一些内嵌样式，其中有几个（特别是h1选择器那个）定义了多个属性的样式。

## 3.4 外部样式表

与其在每个HTML文档中重复定义样式，不如把这些样式保存到一个样式表文件中。样式表文件是一个独立的文件，通常使用.css扩展名，里面存放着我们定义的样式。代码清单3-5展示的是style.css文件的内容，里面存放着花店页面里定义的那些样式。

代码清单3-5 styles.css 文件

```

h1 {
    min-width: 700px; border: thick double black; margin-left: auto;
    margin-right: auto; text-align: center; font-size: x-large; padding: .5em;
    color: darkgreen; background-image: url("border.png");
    background-size: contain; margin-top: 0;
}
.dtable {display: table;}
.drow {display: table-row;}
.dcell {display: table-cell; padding: 10px;}
.dcell > * {vertical-align: middle}
input {width: 2em; text-align: right; border: thin solid black; padding: 2px;}
label {width: 5em; padding-left: .5em; display: inline-block;}

```



```
#buttonDiv {text-align: center;}
#oblock {display: block; margin-left: auto; margin-right: auto; min-width: 700px;}
```

在样式表文件中不需要使用style元素，我们直接定义选择器及样式即可。代码清单3-6展示了使用link元素在页面中导入样式的技术。

#### 代码清单3-6 导入外部样式

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
        <div class="drow">
          <div class="dcell">
            <label for="aster">Aster:</label>
            <input name="aster" value="0" required>
          </div>
          <div class="dcell">
            <label for="daffodil">Daffodil:</label>
            <input name="daffodil" value="0" required >
          </div>
          <div class="dcell">
            <label for="rose">Rose:</label>
            <input name="rose" value="0" required>
          </div>
        </div>
        <div class="drow">
          <div class="dcell">
            <label for="peony">Peony:</label>
            <input name="peony" value="0" required>
          </div>
          <div class="dcell">
            <label for="primula">Primula:</label>
            <input name="primula" value="0" required>
          </div>
          <div class="dcell">
            <label for="snowdrop">Snowdrop:</label>
            <input name="snowdrop" value="0" required>
          </div>
        </div>
      </div>
      <div id="buttonDiv"><button type="submit">Place Order</button></div>
    </form>
  </body>
</html>
```

如果需要，我们可以在一个页面中导入多个样式表文件，其中每个样式表文件使用一个link元素。如果在不同样式表文件中为同一个选择器定义了不同的样式，这些样式表文件的导入顺序就会非常重要。浏览器总是使用最后导入的样式。

### 3.5 理解 CSS 选择器

注意看花店样式表中的选择器，它们有很多种形式。有些是元素的名字（比如h1和input），有些由一个小数点开头（比如.dtable和.row），还有一些由井号开头，比如#buttonDiv和#oblock。如果你观察得比较仔细，就会发现其中的选择器.dcell > \*比较特别，它由多个部分组成。顾名思义，选择器就是用来在页面中选择元素的。浏览器对于不同类型的选择器使用不同的方法查找元素。我们先从核心选择器学起，这些选择器见表3-3。

表3-3 核心选择器

选 择 器	描 述
*	选择所有的元素
<type>	选择特定类型的元素
.<class>	选择具有特定class的元素（与元素类型无关）
<type>.<class>	选择具有特定class的某类元素
#<id>	选择具有特定id属性值的元素

这些选择器是一些应用最为广泛的选择器（这些选择器在示例文档的样式中均有使用）。

#### 3.5.1 属性选择器

我在第2章中提到过，简单选择器基于id和class属性选取元素。除了简单选择器，还有基于属性选取元素的选择器，见表3-4。

表3-4 属性选择器

选 择 器	描 述
[attr]	选取定义了attr属性的元素，即使这个属性没有值
[attr="val"]	选取attr属性值等于字符串val的元素
[attr^="val"]	选取attr属性值以字符串val开头的元素
[attr\$="val"]	选取attr属性值以字符串val结尾的元素
[attr*="val"]	选取attr属性值包含字符串val的元素
[attr~="val"]	选取attr属性包含空格分隔的多个值，且其中一个值是字符串val的元素
[attr =“val”]	选取attr属性值等于字符串val，或属性值为连字符分隔的值列表且第一个值是字符串val的元素

代码清单3-7展示了一个具有内嵌样式的简单页面，其中使用了属性选择器。

## 代码清单3-7 属性选择器

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <style>
    [lang] { background-color: grey; color: white;}
    [lang="es"] {font-size: 14px;}
  </style>
</head>
<body>
  <h1 lang="en">New Delivery Service</h1>
  <h2 lang="en">Color and Beauty to Your Door</h2>
  <h2 lang="es">(Color y belleza a tu puerta)</h2>
  <p>We are pleased to announce that we are starting a home delivery service for
  your flower needs. We will deliver within a 20 mile radius of the store
  for free and $1/mile thereafter.</p>
</body>
</html>
```

第一个选择器匹配具有lang属性的任意元素，第二个选择器则匹配具有lang属性且属性值为es的元素。图3-6展示了应用这两条样式规则的显示效果。

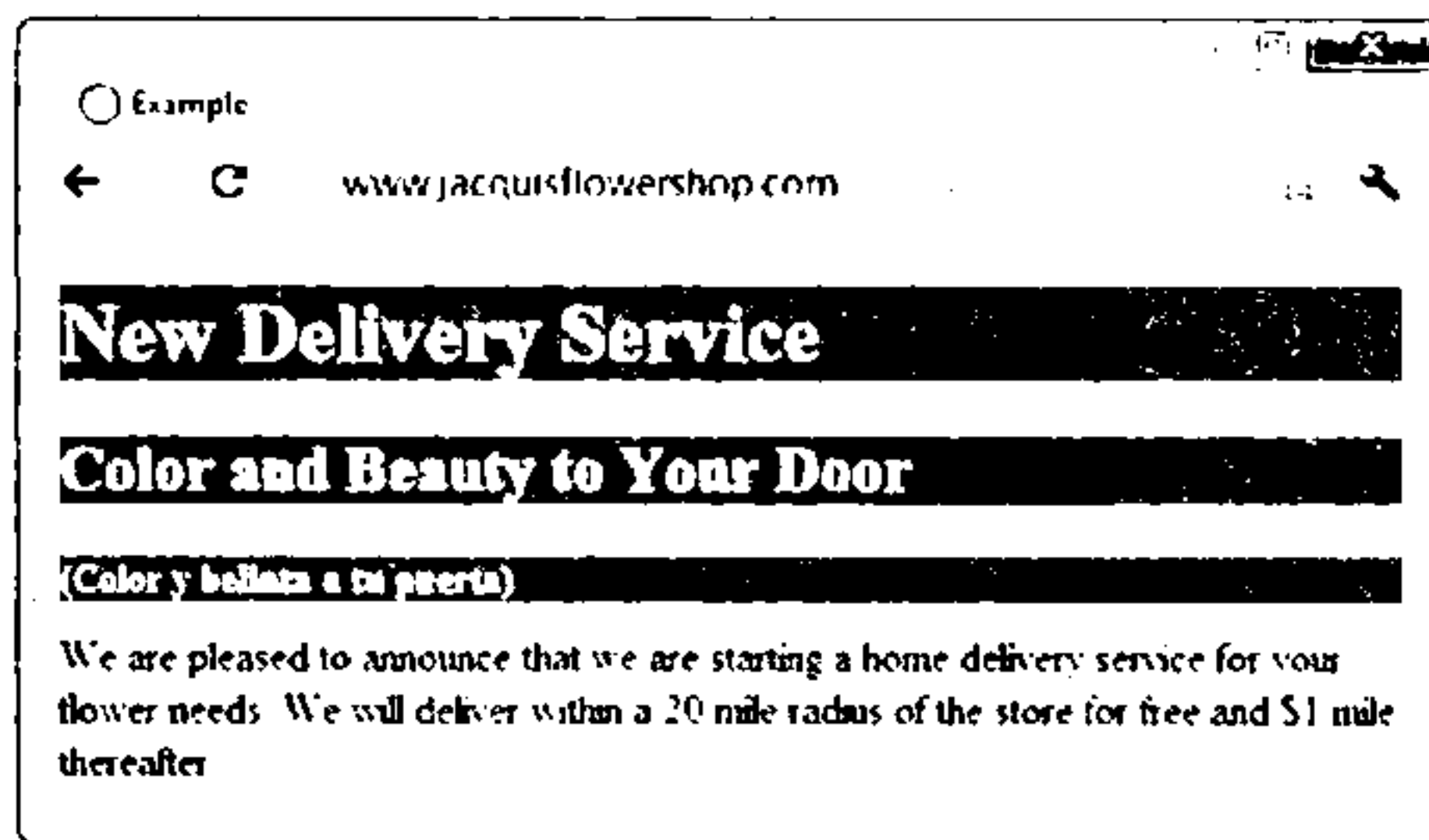


图3-6 使用属性选择器设置样式后的显示效果

**注意** 图3-6中有一处很重要。注意观察h2元素，它受到两个样式规则的影响。第一个样式对页面上所有具有lang属性的元素起作用，第二个样式对所有lang属性等于es的元素起作用。页面中的第二个h2元素同时满足两个选择器条件，因此它的background-color、color以及font-size都发生了变化。3.6节会详细讲解样式的层叠原理。

### 3.5.2 关系选择器

第2章已经介绍过HTML元素及其对应的DOM对象的层级关系。有一些选择器基于层级关系选取元素，这些关系选择器见表3-5。

表3-5 关系选择器

选 择 器	描 述
<selector> <selector>	选择第一个选择器匹配元素内匹配第二个选择器的后代元素
<selector> > <selector>	选择第一个选择器匹配元素内匹配第二个选择器的直接子元素
<selector> + <selector>	选择第一个选择器匹配元素的匹配第二个选择器的下一个兄弟元素
<selector> ~ <selector>	选择第一个选择器匹配元素之后的匹配第二个选择器的所有兄弟元素

我在花店示例文档中使用了其中的一种选择器，如下：

```
.dcell > * {vertical-align: middle}
```

这个选择器匹配dcell元素的所有子元素，将它们的vertical-align属性设置为middle。代码清单3-8演示了其他关系选择器的用法。

#### 代码清单3-8 关系选择器

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <style>
    h1 ~ [lang] { background-color: grey; color: white;}
    h1 + [lang] {font-size: 12px;}
  </style>
</head>
<body>
  <h1 lang="en">New Delivery Service</h1>
  <h2 lang="en">Color and Beauty to Your Door</h2>
  <h2 lang="es">(Color y belleza a tu puerta)</h2>
  <p>We are pleased to announce that we are starting a home delivery service for
  your flower needs. We will deliver within a 20 mile radius of the store
  for free and $1/mile thereafter.</p>
</body>
</html>
```

我在上例中使用了两种兄弟元素选择器。第一种中使用了波浪字符（~），匹配那些位于h1元素之后的，具有lang属性的兄弟元素。也就是说，示例文档中的两个h2元素都匹配（它们都是h1元素的兄弟元素，都有lang属性且都在h1元素之后）。第二种选择器中使用了加号（+），与前一种选择器类似，但只匹配紧跟在h1元素之后的第一个兄弟元素，也就是说只有第一个h2元素匹配。图3-7展示了应用这两条样式规则后的显示效果。

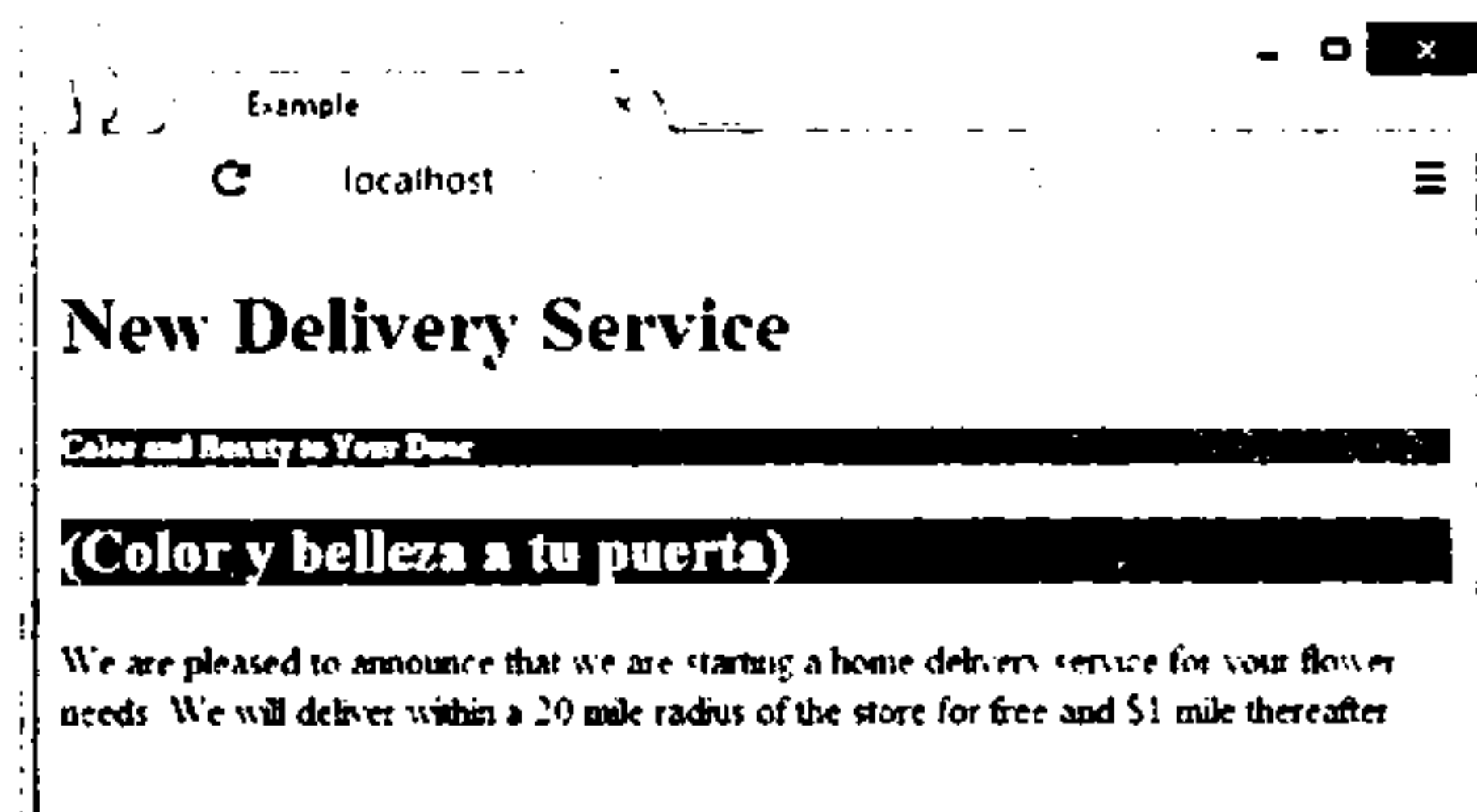


图3-7 使用兄弟关系选择器

### 3.5.3 伪元素和伪类选择器

CSS支持伪元素和伪类选择器。伪元素和伪类选择器为我们提供了许多便利，但你在页面中无法直接找到它们。这些伪选择器见表3-6。

表3-6 伪选择器

选 择 器	描 述
:active	选取用户激活的当前元素，一般是当前鼠标点击的元素
:checked	选取处于选中状态的元素
:default	选取默认元素
:disabled	选取处于禁用状态的元素
:empty	选取没有任何内容的元素
:enabled	选取处于可用状态的元素
:first-child	选取元素的第一个子元素
:first-letter	选取文本的第一个字母
:first-line	选取文本的第一行
:focus	选取得到焦点的元素
:hover	选取鼠标悬停位置的元素
:in-range:out-of-range	选取指定范围之内/之外的input元素
:lang(<language>)	选取lang属性为<language> 的元素
:last-child	选取元素的最后一个子元素
:link	选取链接元素
:nth-child(n)	选取元素的第n个子元素 <sup>①</sup>
:nth-last-child(n)	选取元素的倒数第n个子元素
:nth-last-of-type(n)	选取元素中同类型子元素的倒数第n个元素

① nth开头的系列伪选择器从1而不是从0开始计数。——译者注

(续)

选 择 器	描 述
:nth-of-type(n)	选取元素中同类型子元素的第n个元素
:only-child	选取元素中唯一的子元素
:only-of-type	选取元素中唯一且与指定类型相同的那个子元素
:required	选取具有required属性的元素
:optional	选取缺少required属性的元素
:root	选取文档的根元素
:target	选取URL中锚点引用的元素
:valid	选取表单中通过验证的input元素
:invalid	选取表单中未通过验证的input元素
:visited	选取用户已经访问过的链接元素

代码清单3-9展示了一些伪选择器的用法。

代码清单3-9 伪选择器的使用

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <style>
    :nth-of-type(2) { background-color: grey; color: white;}
    p:first-letter {font-size: 40px;}
  </style>
</head>
<body>
  <h1 lang="en">New Delivery Service</h1>
  <h2 lang="en">Color and Beauty to Your Door</h2>
  <h2 lang="es">(Color y belleza a tu puerta)</h2>
  <p>We are pleased to announce that we are starting a home delivery service for
  your flower needs. We will deliver within a 20 mile radius of the store
  for free and $1/mile thereafter.</p>
</body>
</html>
```

伪选择器可以单独使用，也可以与其他选择器一起使用，这样伪选择器就相当于过滤器。在上面的例子里，这两种用法我都有使用。第一个选择器匹配同类型子元素中的第二个元素（第二个h2）。第二个选择器匹配任意p元素的第一个字母。图3-8展示了应用这两条样式规则后的显示效果。

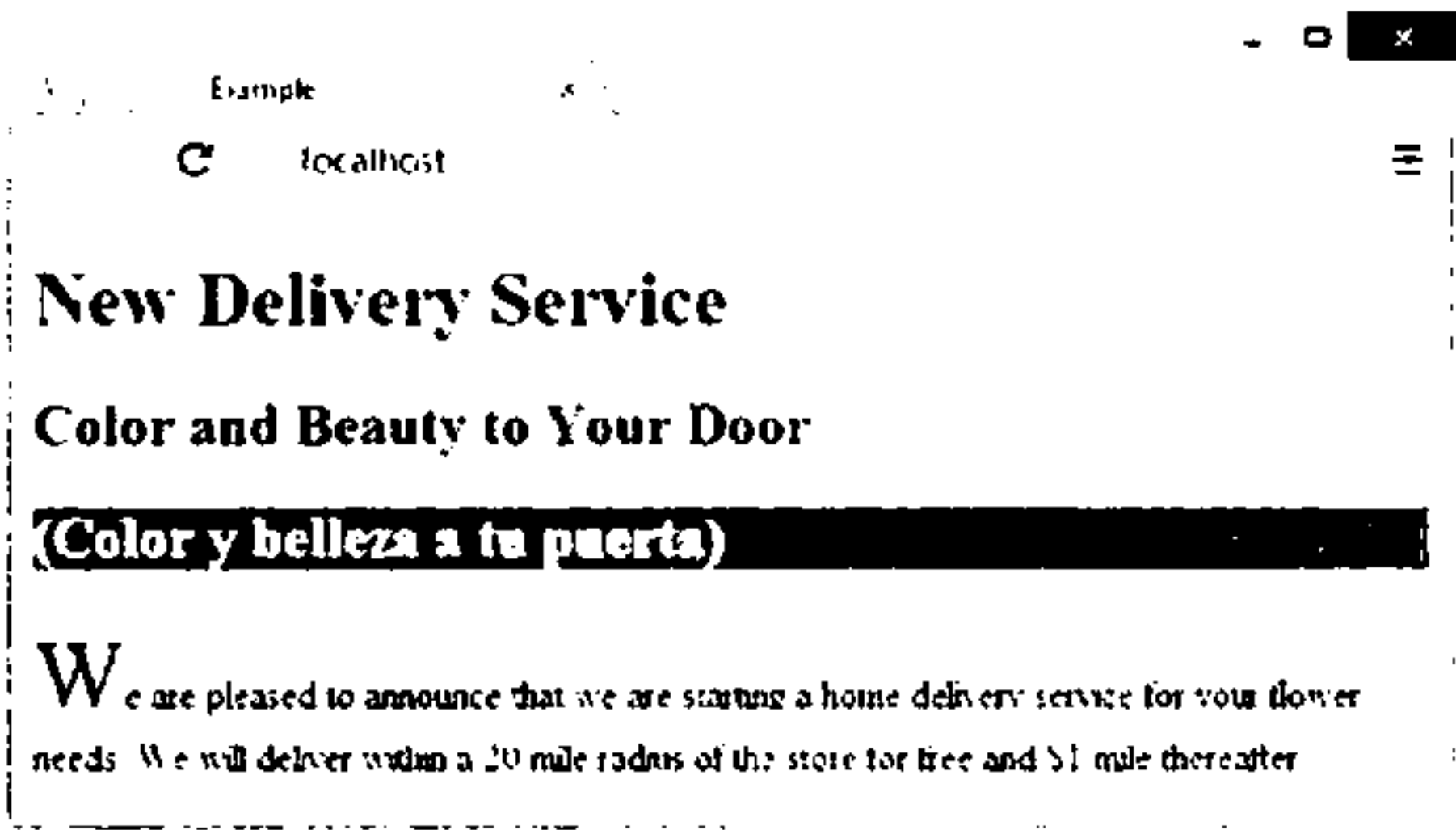


图3-8 使用伪选择器设置样式

3.5.4 联合选择器和反选择器

通过对选择器的合理安排，我们几乎能达成任何目标。特别值得一提的是，我们可以利用选择器组合实现联合选择和反选择。表3-7列出了这两种用法。

表3-7 灵活使用选择器

选 择 器	描 述
<selector>, <selector>	选取匹配第一个选择器的元素和匹配第二个选择器的元素
:not(<selector>)	选取不匹配指定选择器的元素

代码清单3-10演示了联合选择器和反选择器。

代码清单3-10 联合选择器和反选择器

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <style>
    h1, h2 { background-color: grey; color: white;}
    :not(html):not(body):not(:first-child) {border: medium double black;}
  </style>
</head>
<body>
  <h1 lang="en">New Delivery Service</h1>
  <h2 lang="en">Color and Beauty to Your Door</h2>
  <p>We are pleased to announce that we are starting a home delivery service for
  your flower needs. We will deliver within a 20 mile radius of the store
  for free and $1/mile thereafter.</p>
</body>
</html>
```

上面例子中第一个选择器是h1选择器和h2选择器联合实现的。和你想的一样，它会匹配页面中所有的h1和h2元素。第二个选择器稍微有点难懂，因为我想演示一下如何将伪选择器用作另一个伪选择器的过滤器，包括反选择器：



```
...
:not(html):not(body):not(:first-child) {border: medium double black;}
...
```

这个选择器匹配所有不是html元素，也不是body元素，并且不是第一个子元素的元素。图3-9展示了应用这两条样式规则后的显示效果。



图3-9 联合选择器和反选择器

## 3.6 理解样式层叠

学习样式表的关键在于理解样式规则的层叠和继承。浏览器中显示的页面有多个样式来源，而浏览器正是通过层叠和继承来决定元素的最终显示效果。我们已经知道3种样式定义方式（即行内、内嵌和外部样式表），但还应该知道，页面样式除了这个来源还有两个来源，即浏览器样式和用户样式。

浏览器样式，其实更合适的叫法是用户代理样式，是指未指定其他样式时浏览器内定的样式。在本章的开头，我们已经看过一个浏览器样式的例子。

除了浏览器样式，绝大多数浏览器还允许用户定义自己的样式表。这些由用户定义的样式称为用户样式。用户定义样式并不是一个被广泛使用的功能，但那些用到了这一功能的用户往往特别看重这一功能，因为它至少提供了一种让页面可访问性更好的方法。

不同的浏览器用户样式机制各不相同。对于Google Chrome的Windows版本来说，我们可以在用户配置文件目录下创建名为Default\User StyleSheets\Custom.css的文件来定义用户样式。这个文件中的每一条样式规则都会应用在用户访问的每一个页面上，并遵守下面几节中讲到的层叠规则。

### 3.6.1 样式层叠原理

现在我们已经看到显示一个页面时浏览器必须考虑的所有样式来源，接下来关注浏览器显示元素时查找样式属性值的顺序。这个顺序非常明确：

- (1) 行内样式（定义在元素style属性中的样式）；
- (2) 内嵌样式（定义在style元素中的样式）；
- (3) 外部样式（使用link元素导入的样式）；
- (4) 用户样式（用户自己定义的样式）；
- (5) 浏览器样式（浏览器提供的默认样式）。

假设要显示某个元素p，浏览器需要知道将文本显示成什么颜色。那么，它需要查找CSS的color属性值。首先，它检查这个元素的style属性，看看是否有像下面这样定义了color属性的行内样式：

```
...  
<p style="color: red">We are pleased to announce that we are starting a home delivery  
  service for your flower needs. We will deliver within a 20 mile radius of the store  
  for free and $1/mile thereafter.</p>  
...
```

如果没有行内样式，浏览器就接着检查页面的style元素中是否有类似下面这样匹配该元素的样式：

```
...  
<style>  
  p {color: red};  
</style>  
...
```

如果根本没有这样的style元素，浏览器接着查找通过link元素导入的样式表，以此类推，直到找到color属性的值。也就是说，如果没有找到任何定义，它就使用浏览器默认样式。

---

**提示** 我们把行内样式、内嵌样式和样式表又合称为作者样式，把用户样式表中定义的样式称为用户样式，把浏览器定义的默认样式称为浏览器样式。

---

### 3.6.2 使用important规则微调样式应用的顺序

如代码清单3-11所示，我们标识一条样式规则为important（重要）样式，从而覆盖正常的层叠顺序。

代码清单3-11 标识样式属性为重要样式

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Example</title>  
<style>  
  p {color: black !important};  
</style>  
</head>  
<body>  
  <h1 lang="en">New Delivery Service</h1>  
  <h2 lang="en">Color and Beauty to Your Door</h2>  
  <p style="color: red">We are pleased to announce that we are starting a home delivery  
    service for your flower needs. We will deliver within a 20 mile radius of the store  
    for free and $1/mile thereafter.</p>  
</body>  
</html>
```

在样式声明的末尾追加!important可以把该值标识为重要样式。浏览器会给重要样式以优先权，而不再考虑这个样式的定义位置。图3-10展示了重要样式的显示效果，显然内嵌样式定义的color属性覆盖了行内样式的color值。当然在书上看这个效果可能会有点费劲，因为所有的文本印出来都是黑的。



图3-10 重要样式的值覆盖了行内样式的值

**提示** 唯一能优先于作者样式中重要样式的样式是定义在用户样式表中的重要样式。对于普通样式来说，作者样式优先于用户样式，但对于重要样式，浏览器的做法恰恰相反。

### 3.6.3 通过“专一程度”和顺序评估决定样式优先级

假设针对同一个元素有两个同样优先级的样式规则，它们都定义了浏览器希望使用的值，那么应该如何决策？为了决定使用哪个值，浏览器要评估每条样式规则的“专一程度”，然后选择最“专一”的那个。浏览器根据以下3个特征计算样式的“专一程度”：

- ❑ 选择器中出现的id值个数；
- ❑ 选择器中出现的其他属性和伪类个数；
- ❑ 选择器中出现的元素名字和伪元素名字个数。

浏览器合并各项评估值并据此应用最“专一”规则的样式。代码清单3-12是一个最简单的能够解释“专一程度”的例子。

#### 代码清单3-12 样式的专一程度

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <style>
    p {background-color: grey; color: white;}
    p.details {color:red;}
  </style>
</head>
<body>
  <h1 lang="en">New Delivery Service</h1>
  <h2 lang="en">Color and Beauty to Your Door</h2>
  <p class="details">We are pleased to announce that we are starting a home delivery
  service for your flower needs. We will deliver within a 20 mile radius of the store
  for free and $1/mile thereafter.</p>
</body>
</html>
```

我们以a-b-c这种形式评估样式规则的“专一”程度，每个字母依次代表前面提到的3个特征的数量，它并不是一个三位数。数字越大表示越专一。只有a的值相等时，浏览器才去比较b的值，这时b值比较大的规则更专一。如果a与b的值都相同，浏览器才去考察c的值。这意味着专一程度1-0-0比0-5-5更专一。

在本例中，选择器p.details包含1个class属性，这意味着它的专一程度为0-1-1（0个id、1个其他属性、1个元素名）。另一个选择器的专一程度是0-0-1（0个id、0个其他属性、1个元素名字）。

在渲染p元素时，浏览器要查找p元素color属性的值。如果p元素具有details类，那么p.details选择器就最专一，即浏览器会使用颜色值red，而没有details类的其他p元素则使用颜色值white。图3-11展示了这个例子在浏览器中显示的最终效果。

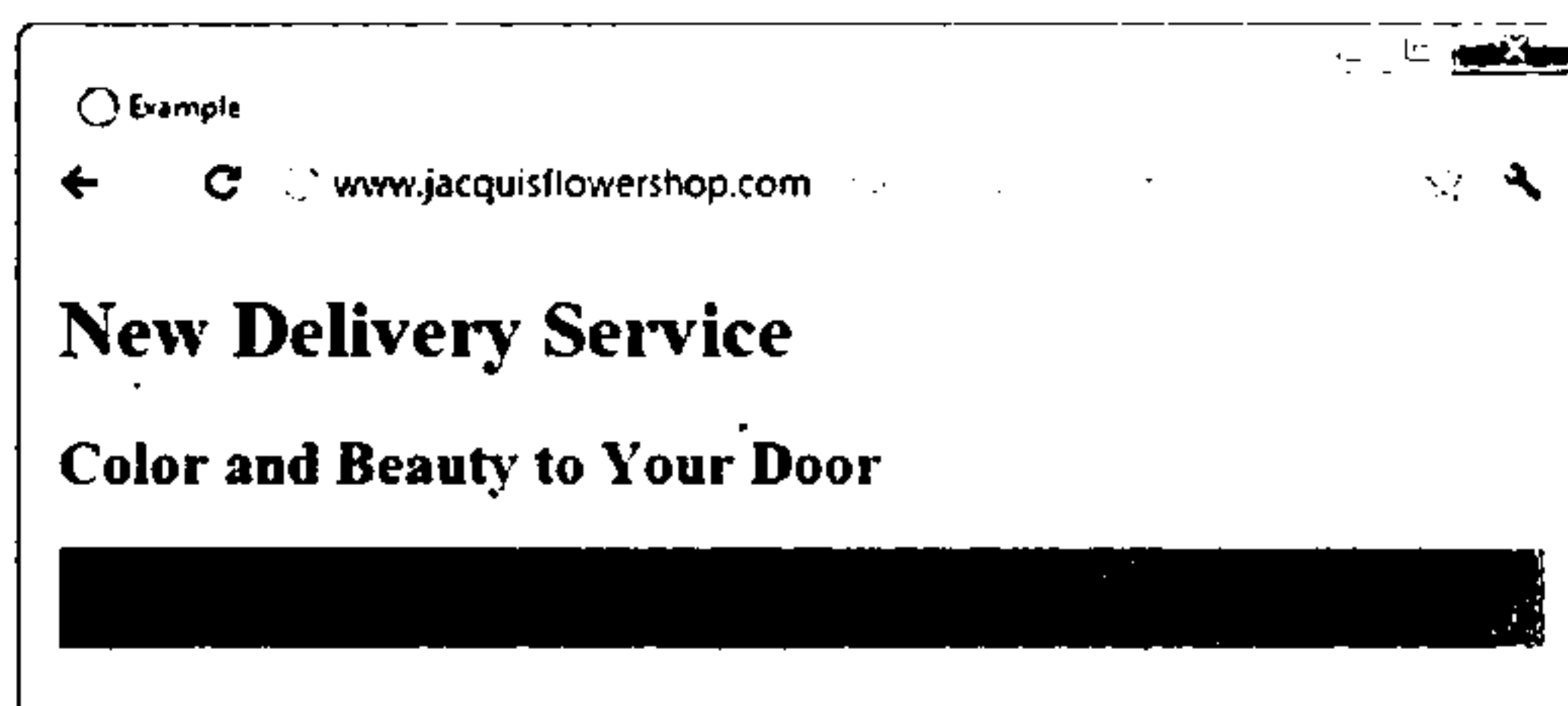


图3-11 基于专一程度应用样式值

如果有多条样式规则的专一程度相同，那么浏览器会选择最后定义的那条规则。代码清单3-13展示了同一页面中的两条样式规则专一程度相同的情况。

### 代码清单3-13 专一程度相同的样式

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <style>
    p.details {color:red;}
    p.information {color: blue;}
  </style>
</head>
<body>
  <h1 lang="en">New Delivery Service</h1>
  <h2 lang="en">Color and Beauty to Your Door</h2>
  <p class="details information">We are pleased to announce that we are starting a home
  Delivery service for your flower needs. We will deliver within a 20 mile radius of
  the store for free and $1/mile thereafter.</p>
</body>
</html>
```

style元素内的两条样式规则专一程度相同，并且都匹配p元素。浏览器在显示这个页面中的p元素时会选择blue颜色，这是因为blue值定义得最晚。图3-12展示了这个页面的显示效果。

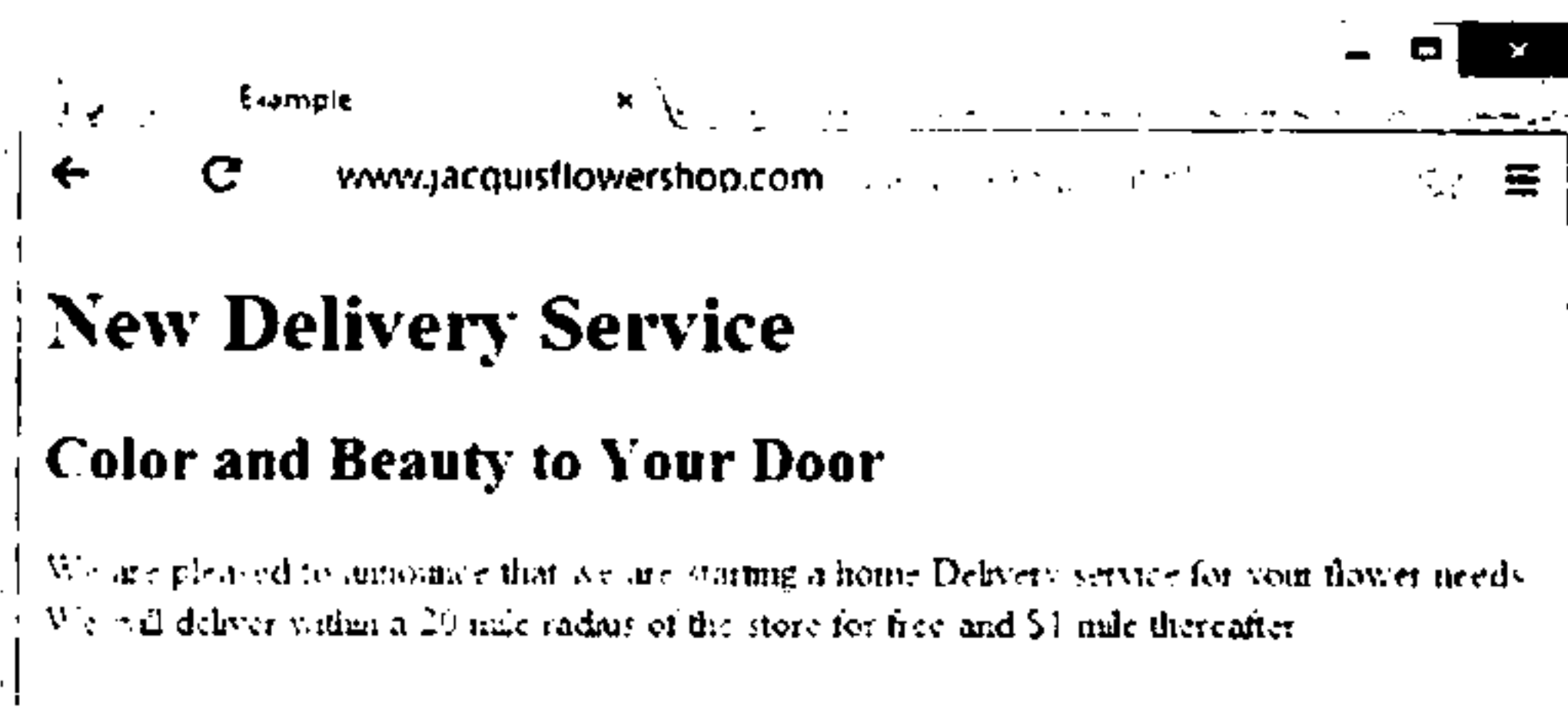


图3-12 基于样式定义的顺序选择属性值

**提示** 专一程度规则仅在同一层叠级别有效。换句话说，定义在元素style属性上的样式总是优先于定义在style元素内的样式。

## 3.7 样式的单位

在本章的开头部分，我列出了几个浏览器内建样式及其默认值，也就是例子中出现的那些元素的默认样式。表3-8再次展示了这些信息。

表3-8 一些CSS属性及其默认值

属 性	h1	h2	p
color	black	black	black
background-color	transparent	transparent	transparent
font-size	32px	24px	16px
border	none	none	none

CSS定义了许多计量单位，接下来我们一起了解一些比较常用的单位，包括本书中用到的那些。

### 3.7.1 颜色

很多CSS属性都用到颜色，包括我在本章示例中用到的color和background-color属性。指定颜色最简便的方式是使用预定义颜色名，也可以使用分别表示RGB浓度的十进制、十六进制颜色值。十进制值RGB各部分由逗号分隔，十六制值则依照惯例要在值前面加一个井号(#)，比如#ffffff表示白色。表3-9列出了一些预定义颜色的名字及它们的十进制、十六进制表示。

表3-9 常见颜色

颜 色 名	十六进制值	十 进 制
black	#000000	0,0,0
silver	#C0C0C0	192,192,192
grey	#808080	128,128,128

(续)

颜色名	十六进制值	十进制
white	#FFFFFF	255,255,255
maroon	#800000	128,0,0
red	#FF0000	255,0,0
purple	#800080	128,0,128
fuchsia	#FF00FF	255,0,255
green	#00FF00	0,255,0
lime	#008000	0,128,0
olive	#808000	128,128,0
yellow	#FFFF00	255,255,0
navy	#000080	0,0,128
blue	#0000FF	0,0,255
teal	#008080	0,128,128
aqua	#00FFFF	0,255,255

这都是些基础色。CSS还定义了更多有名字的颜色，要是都列在这里表3-9就有点儿太长了，完整的列表参见[www.w3.org/TR/css3-color](http://www.w3.org/TR/css3-color)，如果需要你可以去那里查看。作为对基础色的补充，还有很多基础色的微小变化颜色品种。表3-10展示了更多的灰色系颜色。

表3-10 灰色系颜色

颜色名	十六进制值	十进制
darkgrey	#a9a9a9	169,169,169
darkslategrey	#2f4f4f	47,79,79
dimgrey	#696969	105,105,105
grey	#808080	128,128,128
lightgrey	#d3d3d3	211,211,211
lightslategrey	#778899	119,136,153
slategrey	#708090	112,128,144

### 更复杂的颜色

我们不仅可以使使用颜色名和十六进制值表示颜色，还可以用颜色函数选择颜色。表3-11列出了这些颜色函数。

表3-11 CSS颜色函数

函数	描述	例子
rgb(r, g, b)	使用RGB模型指定颜色	color: rgb(112, 128, 144)
rgba(r, g, b, a)	使用RGB模型和指定透明度的alpha通道值(0表示全透明, 1表示完全不透明)指定颜色	color: rgba(112, 128, 144, 0.4)
hsl(h, s, l)	使用HSL (hue: 色相; saturation: 饱和度; lightness: 亮度)模型指定颜色	color: hsl(120, 100%, 22%)
hsla(h, s, l, a)	类似HSL, 增加了指定透明度的alpha通道值	color: hsla(120, 100%, 22%, 0.4)

我们可以使用rgba函数设置半透明颜色，如果要把元素设置得完全透明，可以使用专门的颜色值transparent。

### 3.7.2 长度

许多CSS属性要求我们指定长度,比如font-size属性就用于指定字体的渲染尺寸。在指定长度时,需要同时指定数值和单位,并且在数值和单位之间不能有任何空白。举个例子,假设20pt是某个font-size属性的值,它表示20个单位的pt(即point,稍后我会解释)。单位共有两大类:一类是绝对单位,另一类是相对单位。下面我来详细地介绍这两大类单位。

#### 1. 绝对长度单位

绝对长度单位是真实世界中的长度度量单位。CSS支持5种绝对长度单位,见表3-12。

表3-12 绝对长度单位

单位符号	描 述
in	英寸
cm	厘米
mm	毫米
pt	磅(1磅等于1/72英寸)
pc	皮卡(1皮卡等于12磅)

在样式声明中,我们可以混用长度单位,包括绝对单位和相对单位。如果你以前对页面渲染机制有所了解,比如正在为打印而设计,绝对单位非常有用。不过,由于相对单位更灵活又便于维护,我很少使用绝对单位,极少创作固定尺寸的页面内容。

**提示** 你一定会奇怪在上面的绝对单位表中为什么没有像素。事实上,CSS把像素视为一种相对度量单位,遗憾的是,CSS规范在像素单位的定义上非常蹩脚。我们会在后面的使用像素部分详细介绍像素。

#### 2. 相对长度单位

与绝对长度单位相比,相对长度单位复杂得多,必须使用紧凑和简明的语言才能明确它们的含义。相对单位所度量的是别的单位。遗憾的是,CSS规范中使用的用词不够准确(老毛病了,多年来一直如此)。虽然CSS定义了如此多有趣有用的相对单位,然而由于不够流行,或者缺乏浏览器的一致支持,其中的好多单位我们都不能用。表3-13列出了那些已得到主流浏览器可靠支持的相对单位。

表3-13 相对长度单位

单位符号	描 述
em	相对于元素字号的高度
ex	相对于元素字体中小写字母x的高度
rem	相对于根元素的高度
px	CSS像素(假定位于一个96dpi的显示设备上)
%	另一属性值的百分比



使用相对单位时，我们实际上指定的是另一个尺寸的倍数。

代码清单3-14演示了如何使用相对单位设置font-size。

#### 代码清单3-14 相对长度单位

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <style>
    p.details {
      font-size: 15pt;
      height: 3em;
      border: thin solid black;
    }
  </style>
</head>
<body>
  <h1 lang="en">New Delivery Service</h1>
  <h2 lang="en">Color and Beauty to Your Door</h2>
  <p class="details information">We are pleased to announce that we are starting a home
  delivery service for your flower needs. We will deliver within a 20 mile radius of
  the store for free and $1/mile thereafter.</p>
</body>
</html>
```

在这个例子中，我把p.details元素的height属性（height属性用来设置元素的高度）指定为3em，即应该将此p元素渲染成默认字体高度的3倍。图3-13展示的是这个页面在浏览器中的实际显示效果。我使用border属性为p元素添加了边框，这样大家就能更清楚地观察字体的尺寸。



图3-13 相对长度单位的实际效果

### 3. 像素

CSS中的像素与你想像的可能有所不同。像素通常定义为屏幕上可测量的最小的点，即图像元素。而CSS却尝试着重新定义像素，即：

隔一臂远的距离看一个96dpi的显示设备上一个真实像素的视角。

没错，CSS标准就是这么定义的。这是一个含糊不清、极其糟糕的定义。我真的不想抱怨，可规范居然依赖用户胳膊的长短，这实在太成问题了！好在主流的浏览器都忽略了真实像素和CSS像素之间的不同，把它们都看做1/96英寸：Windows的标准像素密度。不同平台上的显示设备通常有着不同

的像素密度，浏览器会主动做一些转换，尽量保证1像素仍然差不多等于1/96英寸。

**提示** 虽然没有什么实际价值，然而如果你想看，可以在[www.w3.org/TR/CSS21/syndata.html#length-units](http://www.w3.org/TR/CSS21/syndata.html#length-units)上读到CSS像素的完整定义。

结果如何？根据标准，CSS像素是一个相对度量单位，然而所有的浏览器都把它视为绝对度量单位。代码清单3-15展示了以像素为单位定义的样式。

#### 代码清单3-15 在样式中使用像素

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <style>
    p.details {
      font-size: 20px;
      width: 400px;
      border: thin solid black;
    }
  </style>
</head>
<body>
  <h1 lang="en">New Delivery Service</h1>
  <h2 lang="en">Color and Beauty to Your Door</h2>
  <p class="details information">We are pleased to announce that we are starting a home
  delivery service for your flower needs. We will deliver within a 20 mile radius of
  the store for free and $1/mile thereafter.</p>
</body>
</html>
```

在上面的例子中，我使用像素设置了font-size和width属性（width属性是height属性的补充，用来设置元素的宽度）。图3-14展示的是这个例子在浏览器中的显示效果。

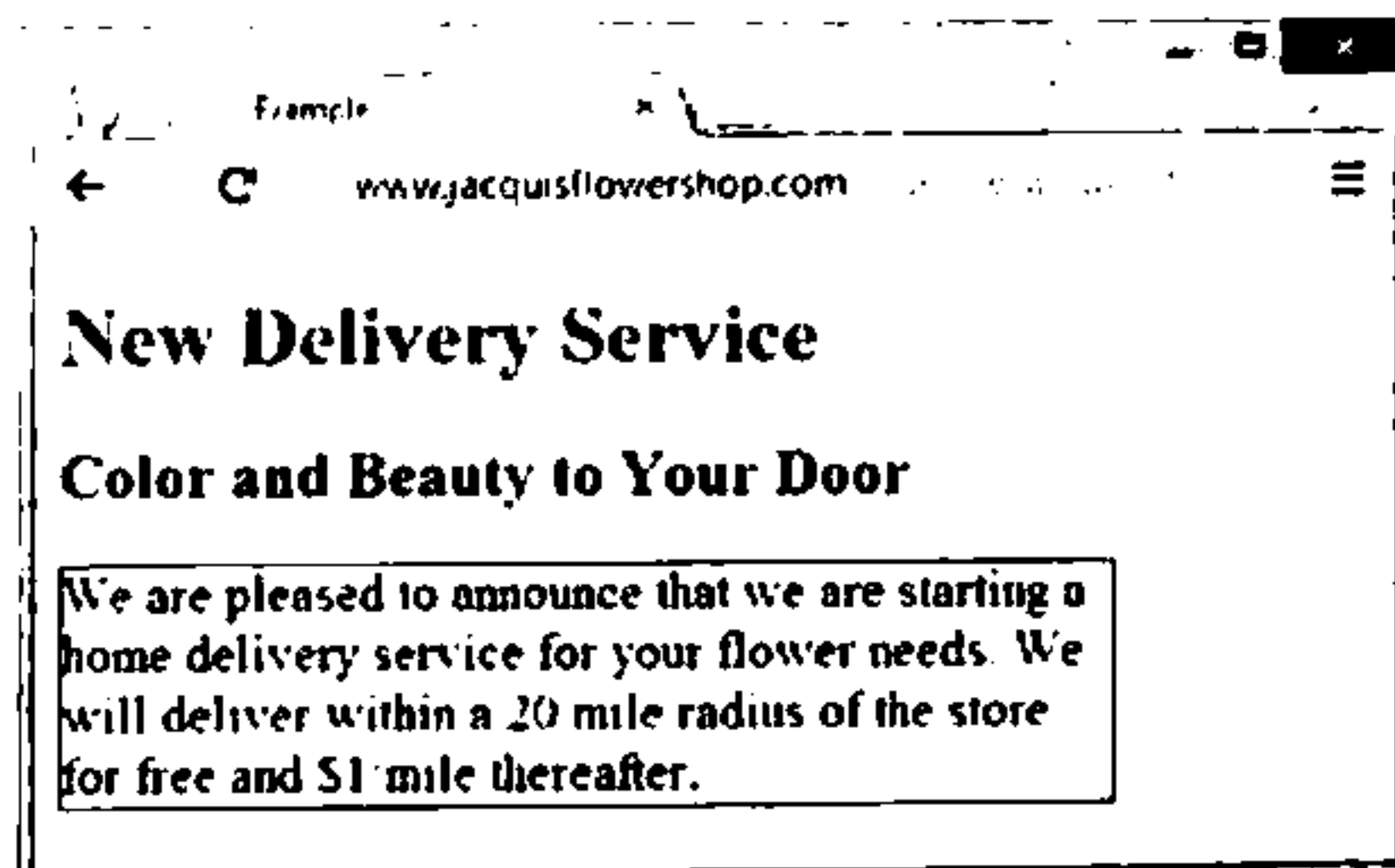


图3-14 使用像素设置样式

**提示** 我在样式表中经常使用像素，个人习惯而已。其实，我发现em单位更灵活。如果使用em，当我需要做些什么变动时，唯一必须修改的地方就是基准字号，其他地方的样式会自动平滑变更。明白这一点非常重要：尽管CSS像素被视为相对单位，但它实际是绝对单位，不会改变大小。

#### 4. 百分比

我们可以用（参照属性值的）百分比表示一个尺寸。代码清单3-16演示了一个使用%（百分比）单位设置尺寸的例子。

代码清单3-16 以百分比表示尺寸

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <style>
    p.details {
      font-size: 200%;
      width: 50%;
      border: thin solid black;
    }
  </style>
</head>
<body>
  <h1 lang="en">New Delivery Service</h1>
  <h2 lang="en">Color and Beauty to Your Door</h2>
  <p class="details information">We are pleased to announce that we are starting a home
  delivery service for your flower needs. We will deliver within a 20 mile radius of
  the store for free and $1/mile thereafter.</p>
</body>
</html>
```

使用百分比单位有两个问题：首先，不是所有的属性都支持百分比单位；其次，支持百分比单位的属性所参照的属性各不相同。举例来说，font-size属性参照的是继承而来的父元素的font-size值，而width属性参照的则是包含元素的宽度。

### 3.8 属性速记法和自定义值

不是所有的属性都需要单位和颜色。一些属性有着特殊的值，每个值控制着元素的一个行为。典型的例子是border属性，我在一些例子里会用它给一些元素设置边框。设置border属性需要3个值。例如：

```
...
border: thin solid black;
...
```

第一个值设置边框的厚度（粗细），第二个值设置边框的风格，最后一个值设置边框的颜色。表3-14列出了可以使用的边框厚度值。

表3-14 边框粗细值一览表

值	描 述
<length>	用CSS长度单位（如em、px或cm表示的边框粗细）
<perc>%	以当前元素宽度的百分比表示的边框粗细
Thin、medium、thick	预定义宽度，具体宽度由浏览器决定，从左至右越来越粗

表3-15列出了所有可用的边框风格。

表3-15 边框风格一览表

值	描 述
none	没有边框
dashed	边框由一系列矩形线段组成
dotted	边框由一个个圆点组成
double	双实线边框，线之间有小空隙
groove	有着凹陷效果的边框
inset	使内容有凹陷效果的边框
outset	使内容有凸出来效果的边框
ridge	有着凸出来效果的边框
solid	单实线边框

结合使用上述表中的值与边框颜色，我们可得到各式各样的边框效果，参见图3-15。

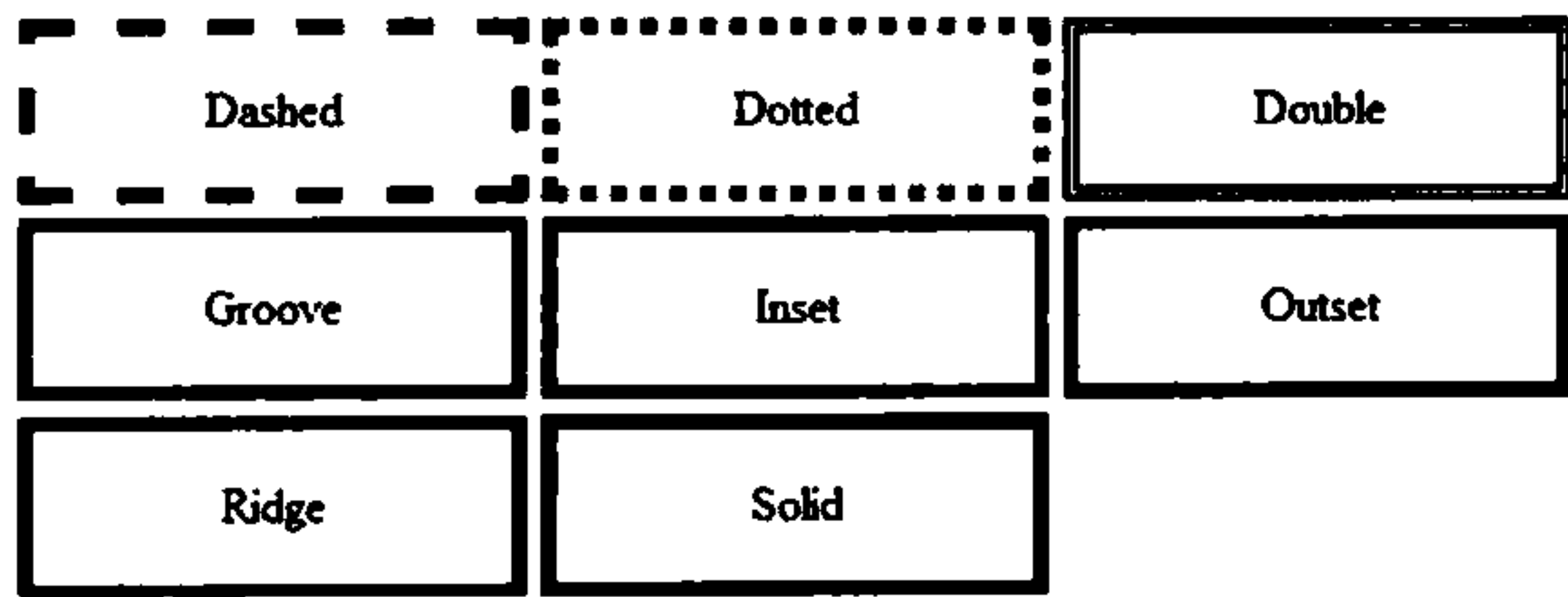


图3-15 边框风格一览

border属性特别适合用来说明什么是简写属性。简写属性用于在一个声明中设置多个相关属性的值。也就是说，刚才那个边框声明（即border: thin solid black;）等价于代码清单3-17列出来的12个属性声明。

代码清单3-17 边框属性一览

```
border-top-color: black;
border-top-style: solid;
border-top-width: thin;
border-bottom-color: black;
```

```
border-bottom-style: solid;  
border-bottom-width: thin;  
border-left-color: black;  
border-left-style: solid;  
border-left-width: thin;  
border-right-color: black;  
border-right-style: solid;  
border-right-width: thin;
```

CSS支持我们单独设置每个属性的值，以便精确控制元素的呈现，并支持在有关设置值完全一样时使用简写属性。

## 3.9 小结

本章是一个短平快的CSS教程，介绍了如何使用style属性和style元素设置元素样式，还讲解了各种选择器的用法，以及浏览器在显示元素时如何层叠计算属性的最终值。最后介绍了CSS计量单位、自定义值及属性规则的快捷写法。有好多种方法表达属性的值，这为设置样式增添了灵活性，也给我们带来了一点点烦恼。

在第4章，我将介绍JavaScript基础知识，jQuery的功能正是使用JavaScript定义并应用于HTML内容的。

jQuery是一个JavaScript库，要使用jQuery，我们必须先把它添加到页面中，它的代码由浏览器执行。依托jQuery库的基础，我们在页面中编写脚本以实现各种功能，这意味着我们必须了解如何使用JavaScript编程。本章是一个JavaScript入门教程，重点讲解使用jQuery所必需的语言特性。

在程序语言的世界里，JavaScript毁誉参半。在它成为标准，变得足够成熟之前，JavaScript经历过一段困难时光，时至今日，它的一些行为仍然相当怪异。来自开发者的绝大多数抱怨，都是说这门语言与开发者所喜爱的后端语言（如C#、Java或者Python）大不相同。

当人们抛弃成见，接受JavaScript是一门独立语言的现实之后，就会发现这是一门相当灵活的动态语言，用起来相当舒服。当然，确实有一些怪异之处，然而整体使用体验不俗。在付出一些努力之后，你会发现JavaScript是一门表达力很强、非常值得学习的语言。

---

**提示** 如果你是一个新手，可以参考深受欢迎的Lifehacker.com上的一系列编程入门文章，它们是非常好的编程启蒙。阅读这些文章无需任何编程知识，所有示例都使用了方便练习的JavaScript语言。这套编程指南位于：<http://lifehacker.com/5744113/learn-to-code-the-full-beginners-guide>。

---

## 4.1 上手 JavaScript

JavaScript以脚本形式添加进HTML文档中并被浏览器执行，添加的方法有不同的几种。一种定义内嵌脚本，即脚本内容本身就是HTML的一部分。另一种则定义外部脚本，即把脚本单独存放在一个文件中，然后在HTML中使用脚本文件的URL来引用脚本（在本书的第二部分，访问jQuery库时就是使用的这种方式）。两种方式都使用script标签。在本章中，我使用内嵌脚本。我们先看一下代码清单4-1，这是一个很简单的例子。

**代码清单4-1 简单的内嵌脚本**

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    console.log("Hello");
  </script>
```

```

</head>
<body>
  This is a simple example
</body>
</html>

```

这个脚本普普通通，它输出一条消息到控制台。控制台是由浏览器提供的一个虽然简单但非常有用的工具，在程序运行的时候我们可以用它输出调试信息。每种浏览器的控制台使用方法不同，对于Google Chrome来说，我们必须在Tool（工具）菜单里选择JavaScript console（JavaScript控制台）子菜单。图4-1展示的是Chrome中的控制台，其他浏览器的功能与此类似。

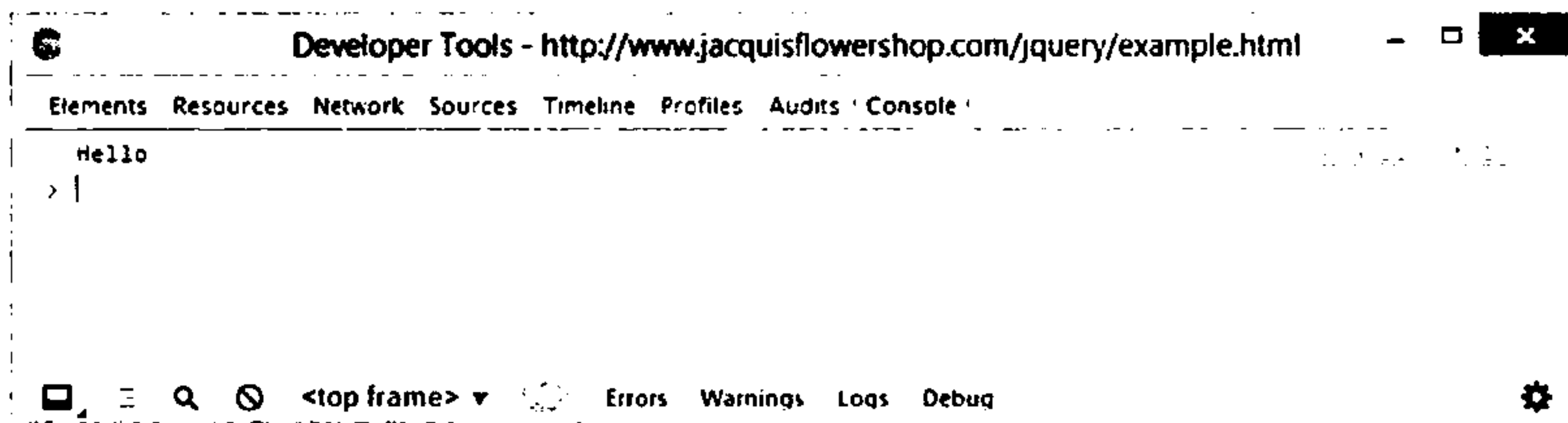


图4-1 Google Chrome浏览器的JavaScript控制台

在控制台窗口我们可以看到调用`console.log`方法输出的信息，以及信息的详细来源信息（在本例中是`example.html`文件的第6行）。本章后面不再使用屏幕截图，而是直接引用例子的运行结果，比如代码清单4-1的输出如下：

```
Hello
```

为了方便阅读，我对输出结果的格式做了一定处理。在接下来的各节中，我们将一起学习JavaScript语言的核心特性。如果你以前使用其他现代编程语言写过程序，就会非常熟悉JavaScript的语法和编程风格——尽管，我在本章开头已经说过，有些地方它确实有点怪。

## 4.2 语句

JavaScript语言的基础构造是语句。每一条语句表示一个命令，通常以分号（`;`）结尾。语言本身并不强制要求结尾的分号，不过使用分号既可以让代码更易读，也能支持在一行书写多条语句。代码清单4-2展示了一段脚本，其中有两条语句，它们是在页面中的`script`标签中定义的。

代码清单4-2 JavaScript语句

```

<!DOCTYPE HTML>
<html>
  <head>
    <title>Example</title>
    <script type="text/javascript">

```



```

        console.log("This is a statement");
        console.log("This is also a statement");
    </script>
</head>
<body>
    This is a simple example
</body>
</html>

```

浏览器会顺序执行每条语句。在本例中，我只是输出了两条消息到控制台，输出结果如下。

```

This is a statement
This is also a statement

```

## 4.3 函数

浏览器加载script元素后，就会立即逐条执行这些JavaScript语句。JavaScript脚本还有另外一种书写方式，即像代码清单4-3那样把一些语句封装成一个函数。浏览器在遇到调用这个函数的语句之前不会执行被封装的代码。

代码清单4-3 定义JavaScript函数

```

<!DOCTYPE HTML>
<html>
<head>
    <title>Example</title>
    <script type="text/javascript">
        function myFunc() {
            console.log("This is a statement");
        };
        myFunc();
    </script>
</head>
<body>
    This is a simple example
</body>
</html>

```

函数中的语句被一对大括号包住，被包住的语句以及包住它们的大括号又合称代码块。上面这些代码定义了一个名为myFunc的函数，它的代码块中只有一条语句。JavaScript语言区分字母大小写，这意味着关键字function必须小写。在浏览器遇到下面这条调用语句之前，函数体内的语句不会执行：

```
myFunc();
```

执行该语句可产生以下输出结果：

```
This is a statement
```

在这个例子中，myFunc函数在定义完之后立即被调用，因此除了演示如何定义函数，这个例子并不是那么实用。在第二部分中，我们会见到很多实用函数的例子。

### 4.3.1 带参数的函数

如代码清单4-4所示，和绝大多数其他编程语言一样，JavaScript支持带参数的函数。

代码清单4-4 带参数的函数

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    function myFunc(name, weather) {
      console.log("Hello " + name + ".");
      console.log("It is " + weather + " today");
    };

    myFunc("Adam", "sunny");
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

我给myFunc函数加了两个参数，分别是name和weather。JavaScript是一种动态类型语言，也就是说定义函数时不必声明参数的数据类型。后面我们讲到JavaScript变量时还会接着讨论动态类型。调用带参数的函数时，我们应该这样提供参数值：

```
...
myFunc("Adam", "sunny");
...
```

上面这个代码清单的输出结果如下：

---

```
Hello Adam.
It is sunny today
```

---

在调用函数时，我们提供的参数个数不必与函数定义的参数个数相同。如果在调用函数时提供的参数不够多，缺失参数的值就是undefined。如果提供了过多的参数，那些多余的参数就会被直接忽略。<sup>①</sup>

由此得出一个结论：如果我们创建了两个名字相同而参数不同的函数，根本不能指望JavaScript根据参数差异正确地调用我们希望调用的函数。换句话说，尽管Java和C#等语言支持多态，JavaScript却不支持。在JavaScript中，如果你定义了两个名字相同的函数，后面定义的那个就会直接替掉前面那个。

---

① 这种说法不是很准确，是否“扔掉”那些看上去“多余”的参数是由程序员决定的。函数是能够得到并处理这些多余参数的。当然，编程实践中的大部分函数都没有处理而是“直接”扔掉了那些参数。——译者注

### 4.3.2 有返回值的函数

我们可以在函数中使用`return`关键字返回某个值。代码清单4-5演示了一个有返回值的函数。

代码清单4-5 让函数返回一个结果

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    function myFunc(name) {
      return ("Hello " + name + ".");
    };
    console.log(myFunc("Adam"));
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

这个函数定义了一个参数，并利用该参数生成一个简单的返回值。下面的代码调用了这个函数，并将它的返回值作为参数传递给`console.log`函数：

```
...
console.log(myFunc("Adam"));
...
```

注意，我们不必事先声明函数会不会返回值，也不必声明返回值的类型。上面代码清单的运行结果如下：

---

```
Hello Adam.
```

---

## 4.4 变量和类型

JavaScript使用`var`关键字定义变量，并且可以在定义变量的同时为变量赋值。在函数内定义的变量是局部变量，这样的变量仅可以在函数内使用。直接定义在脚本里的变量是全局变量，到处都可以使用，甚至能被其他脚本使用。代码清单4-6演示了局部变量和全局变量的用法。

代码清单4-6 使用局部变量和全局变量

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Example</title>
    <script type="text/javascript">
      var myGlobalVar = "apples";
```

```
        function myFunc(name) {
            var myLocalVar = "sunny";
            return ("Hello " + name + ". Today is " + myLocalVar + ".");
        };
        console.log(myFunc("Adam"));
    </script>
    <script type="text/javascript">
        console.log("I like " + myGlobalVar);
    </script>
</head>
<body>
    This is a simple example
</body>
</html>
```

我们前面说过，JavaScript是一种动态类型语言，但这并不是说JavaScript语言没有类型。JavaScript确实不需要显式声明变量的类型，并且允许把不同类型的数据赋给同一个变量，这毫无问题。在我们给一个变量赋值时，JavaScript能根据变量的值自动推导出数据的类型，并且根据变量的上下文自动转换变量的类型。代码清单4-6的输出如下：

```
Hello Adam. Today is sunny.
I like apples
```

#### 4.4.1 基本数据类型

JavaScript定义了几种基本数据类型：字符串类型、数字类型和布尔类型。这确实有点少，不过JavaScript终究还是想方设法使用这3种基本类型实现了对种种问题的灵活处理。

##### 1. 字符串

如代码清单4-7所示，我们可以使用单引号或双引号定义字符串。

代码清单4-7 定义字符串

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Example</title>
    <script type="text/javascript">
        var firstString = "This is a string";
        var secondString = 'And so is this';
    </script>
</head>
<body>
    This is a simple example
</body>
</html>
```

引号必须配对。以单引号开始双引号结束是不行的，反之亦然。这个代码清单中的代码不会产生任何控制台输出。

## 2. 布尔值

布尔类型只有两个值：`true`和`false`。代码清单4-8展示了布尔类型的使用，布尔类型最有用的地方是条件判断语句，比如`if`语句。

代码清单4-8 定义布尔值

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var firstBool = true;
    var secondBool = false;
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

## 3. 数字

数字类型用来表示整数和浮点数（即实数）。代码清单4-9演示了数字在JavaScript中的用法。

代码清单4-9 定义数值

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var daysInWeek = 7;
    var pi = 3.14;
    var hexValue = 0xFFFF;
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

你只管使用数字，没有必要指定数字的类型，JavaScript会自动根据数字的值处理类型。在代码清单4-9中，我定义了一个整数、一个浮点数，还通过前置的`0x`定义了一个十六进制的数字。（这个代码清单中的代码不会产生任何控制台输出。）

### 4.4.2 生成对象

JavaScript支持对象，而且支持多种对象创建方式。代码清单4-10给出了一个简单示例。

代码清单4-10 生成一个对象

```
<!DOCTYPE HTML>
<html>
```

```

<head>
  <title>Example</title>
  <script type="text/javascript">
    var myData = new Object();
    myData.name = "Adam";
    myData.weather = "sunny";

    console.log("Hello " + myData.name + ". ");
    console.log("Today is " + myData.weather + ".");
  </script>
</head>
<body>
  This is a simple example
</body>
</html>

```

调用new Object()即得到一个对象，然后我把新创建的对象赋值给变量myData。在对象生成之后，我使用下面这样的赋值语句为对象定义了两个属性。

```

...
myData.name = "Adam";
...

```

在执行上面这条语句之前，对象myData并没有name属性。执行这条语句之后，它才有这个属性并得到一个值Adam。像下面这样，在变量名myData之后加上小数点和属性名就能读取name属性的值。

```

...
console.log("Hello " + myData.name + ". ");
...

```

该清单输出结果如下。

---

```

Hello Adam.
Today is sunny.

```

---

### 1. 对象字面量

我们可以使用对象字面量，在得到一个对象的同时定义它的属性。代码清单4-11演示了这种做法。

**代码清单4-11 对象字面量**

```

<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var myData = {
      name: "Adam",
      weather: "sunny"
    };

    console.log("Hello " + myData.name + ". ");
    console.log("Today is " + myData.weather + ".");
  </script>
</head>

```

```
<body>
  This is a simple example
</body>
</html>
```

每个属性和它的值之间由冒号(:)分隔,属性之间使用逗号(,)分隔。<sup>①</sup>所得效果与上面的示例相同,该示例输出结果如下。

```
Hello Adam.
Today is sunny.
```

4

## 2. 作为方法使用的函数

JavaScript语言中我最喜欢的一个特性是可以直接在对象里把一个属性定义成函数。如果把一个函数定义成对象的属性,它就自动成为对象的一个方法。说不出具体的理由,我就是觉得这种方式既优雅又使人愉快。代码清单4-12展示了如何以这种方式为对象定义方法。

代码清单4-12 在对象中定义方法

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var myData = {
      name: "Adam",
      weather: "sunny",
      printMessages: function() {
        console.log("Hello " + this.name + ".");
        console.log("Today is " + this.weather + ".");
      }
    };
    myData.printMessages();
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

在这个例子中,printMessages属性的值是一个函数,因此它是myData对象的一个方法。注意,要在函数内引用对象的属性必须使用this关键字。当一个函数作为方法使用时,函数会自动得到一个特殊的本地变量this,它是调用对象的引用。上面的代码清单有如下输出。

```
Hello Adam.
Today is sunny.
```

① 虽然Firefox/Chrome等现代浏览器支持最后一个属性之后的逗号,但绝大多数版本的IE都不支持。为确保不出问题,最后一个属性之后一定不要放逗号。——译者注



### 4.4.3 使用对象

在生成对象之后，我们可以利用对象做很多事。接下来我会讲解对象的各种用法，这些知识在后面的内容里都用得着。

#### 1. 读取和修改属性值

显然，我们可以读取和修改对象属性的值。访问属性的语法有两种，参见代码清单4-13。

代码清单4-13 读取和修改属性

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var myData = {
      name: "Adam",
      weather: "sunny",
    };

    myData.name = "Joe";
    myData["weather"] = "raining";

    console.log("Hello " + myData.name + ".");
    console.log("It is " + myData["weather"]);
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

第一种风格绝大多数程序员都很熟悉，我在前面的例子里也使用了这种风格。像下面这样，我们用小数点连接对象变量名和属性名：

```
...
myData.name = "Joe";
...
```

使用小数点连接的对象名和属性名可以读取属性的值，如果像上面这条语句那样再加上一个等号和一个新值，可以修改属性的值。第二种风格类似于数组索引：

```
...
myData["weather"] = "raining";
...
```

在这种风格中，我们不使用句点，而是把属性的名字用一对中括号括起来。当我们使用一个变量引用属性名字时，只能使用这种风格，因为只有这种风格能支持变量名表示的属性名：

```
...
var myData = {
  name: "Adam",
  weather: "sunny",
};

var propName = "weather";
```

```
myData[propName] = "raining";  
...
```

这是接下来迭代对象属性的基础。该代码清单从中控台的输出结果如下：

```
Hello Joe.  
It is raining
```

## 2. 迭代对象属性

我们可以使用for...in语句迭代处理对象的属性。代码清单4-14演示了这一用法。

代码清单4-14 迭代对象属性

```
<!DOCTYPE HTML>  
<html>  
<head>  
  <title>Example</title>  
  <script type="text/javascript">  
    var myData = {  
      name: "Adam",  
      weather: "sunny",  
      printMessages: function() {  
        console.log("Hello " + this.name + ".");  
        console.log("Today is " + this.weather + ".");  
      }  
    };  
  
    for (var prop in myData) {  
      console.log("Name: " + prop + " Value: " + myData[prop]);  
    }  
  </script>  
</head>  
<body>  
  This is a simple example  
</body>  
</html>
```

在代码清单4-14中，for...in循环逐一处理myData对象的属性（执行代码块中的语句）。在每次迭代中，它都会把属性的名字赋给prop变量，而我则使用数组索引风格的语法得到属性的值。上面的代码清单产生如下输出结果（为了容易阅读，我调整了输出结果的格式）：

```
Name: name Value: Adam  
Name: weather Value: sunny  
Name: printMessages Value: function () {  
  console.log("Hello " + this.name + ".");  
  console.log("Today is " + this.weather + ".");  
}
```

在上面的结果中，我们定义的函数也一样迭代了。这也是JavaScript语言灵活处理函数的结果。

### 3. 属性与方法的添加和删除

即使是使用对象字面量定义对象，我们也能为对象添加新的属性。代码清单4-15演示了这种做法。（此部分的代码清单没有任何中控台输出。）

**代码清单4-15 给对象添加新的属性**

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var myData = {
      name: "Adam",
      weather: "sunny",
    };

    myData.dayOfWeek = "Monday";
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

在代码清单4-15中，我使用句点表示法（用句点连接对象和属性名）为对象添加了一个新属性 `dayOfWeek`，不过也可以轻松改成使用数组索引表示的语法。现在，你或许期待着给对象添加一个新方法。代码清单4-16演示了如何为已有的对象添加方法。

**代码清单4-16 为已有对象添加方法**

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var myData = {
      name: "Adam",
      weather: "sunny",
    };

    myData.SayHello = function() {
      console.write("Hello");
    };
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

在代码清单4-17中可以看到，`delete`关键字用来删除对象的属性或者方法。

**代码清单4-17 删除对象的属性**

```
<!DOCTYPE HTML>
```

```

<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var myData = {
      name: "Adam",
      weather: "sunny",
    };

    delete myData.name;
    delete myData["weather"];
    delete myData.SayHello;
  </script>
</head>
<body>
  This is a simple example
</body>
</html>

```

#### 4. 检查对象是否拥有某个属性

如代码清单4-18所示，我们可以使用in表达式检查对象是否拥有某个属性。

代码清单4-18 检查对象是否有某个属性

```

<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var myData = {
      name: "Adam",
      weather: "sunny",
    };

    var hasName = "name" in myData;
    var hasDate = "date" in myData;

    console.log("HasName: " + hasName);
    console.log("HasDate: " + hasDate);
  </script>
</head>
<body>
  This is a simple example
</body>
</html>

```

在这个例子中，我们成功测试出一个属性存在，而另一个属性不存在，即hasName变量的值为true，hasDate变量的值为false，如下所示：

```

HasName: true
HasDate: false

```

## 4.5 JavaScript 运算符

JavaScript支持许多标准的运算符，表4-1中列出了最有用的一些。

表4-1 常用JavaScript运算符

运 算 符	描 述
++、--	预/后增1或减1
+, -, *, /, %	加、减、乘、除、取余
<, <=, >, >=	小于、小于等于、大于、大于等于
==, !=	相等、不等
===, !==	恒等、不恒等
&&,	逻辑与和逻辑或
=	赋值
+	字符串连接
?:	三目运算

### 4.5.1 条件语句

许多JavaScript运算符都是在条件语句中使用的，而我在本书中偏好使用if/else和switch语句。代码清单4-19演示了这两类语句的用法（如果你了解别的编程语言，应该很熟悉这些东西了）。

代码清单4-19 使用if/else和switch条件语句

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">

    var name = "Adam";

    if (name == "Adam") {
      console.log("Name is Adam");
    } else if (name == "Jacqui") {
      console.log("Name is Jacqui");
    } else {
      console.log("Name is neither Adam or Jacqui");
    }

    switch (name) {
      case "Adam":
        console.log("Name is Adam");
        break;
      case "Jacqui":
        console.log("Name is Jacqui");
        break;
      default:
```

```
        console.log("Name is neither Adam or Jacqui");
        break;
    }
</script>
</head>
<body>
    This is a simple example
</body>
</html>
```

该代码清单输出结果如下：

---

```
Name is Adam
Name is Adam
```

---

## 4.5.2 相等运算符和恒等运算符

相等运算符和恒等运算符值得一书。相等运算符会在必要时先转换操作数的类型再进行比较，如果你能意识到这一点，这是一个挺贴心的功能。代码清单4-20演示了相等运算符的用法。

代码清单4-20 使用相等运算符

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Example</title>
    <script type="text/javascript">

        var firstVal = 5;
        var secondVal = "5";

        if (firstVal == secondVal) {
            console.log("They are the same");
        } else {
            console.log("They are NOT the same");
        }
    </script>
</head>
<body>
    This is a simple example
</body>
</html>
```

这个脚本的输出如下：

---

```
They are the same
```

---

如果需要，JavaScript会把两个操作数先转换为同一类型，然后再进行比较。本质上说，相等运算符仅测试操作数的值是否相等，而不关心操作数的类型。如果我们希望既能检测到值相等又能检测到

类型相同,如代码清单4-21所示,应该使用恒等运算符(===,3个等号而不是相等运算符的两个等号)。

代码清单4-21 使用恒等运算符

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">

    var firstVal = 5;
    var secondVal = "5";

    if (firstVal === secondVal) {
      console.log("They are the same");
    } else {
      console.log("They are NOT the same");
    }
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

在这个示例中,恒等运算符认为两个变量不同。这个运算符不作任何类型转换,上面脚本的结果是:

---

They are NOT the same

---

对于JavaScript基本类型来说,比较运算比较的是值,然而对于JavaScript对象来说,比较运算比较的却是引用。代码清单4-22展示了JavaScript处理对象的相等测试和恒等测试的方式。

代码清单4-22 对象的相等与恒等测试

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">

    var myData1 = {
      name: "Adam",
      weather: "sunny",
    };

    var myData2 = {
      name: "Adam",
      weather: "sunny",
    };

    var myData3 = myData2;
```



```

    var test1 = myData1 == myData2;
    var test2 = myData2 == myData3;
    var test3 = myData1 === myData2;
    var test4 = myData2 === myData3;

    console.log("Test 1: " + test1 + " Test 2: " + test2);
    console.log("Test 3: " + test3 + " Test 4: " + test4);
  </script>
</head>
<body>
  This is a simple example
</body>
</html>

```

上面脚本的运行结果如下:

---

```

Test 1: false Test 2: true
Test 3: false Test 4: true

```

---

代码清单4-23展示了针对基本数据类型所做的同样的测试。

#### 代码清单4-23 基本类型的相等与恒等测试

```

<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">

    var myData1 = 5;
    var myData2 = "5";
    var myData3 = myData2;

    var test1 = myData1 == myData2;
    var test2 = myData2 == myData3;
    var test3 = myData1 === myData2;
    var test4 = myData2 === myData3;

    console.log("Test 1: " + test1 + " Test 2: " + test2);
    console.log("Test 3: " + test3 + " Test 4: " + test4);
  </script>
</head>
<body>
  This is a simple example
</body>
</html>

```

上面脚本的运行结果如下:

---

```

Test 1: true Test 2: true
Test 3: false Test 4: true

```

---

### 4.5.3 显式类型转换

字符串连接运算符(+)比加法运算符(也是+)的优先级高,也就是说相对于加法运算JavaScript会优先进行字符串连接运算。如代码清单4-24所示,由于JavaScript悄悄地自动转换操作数的类型,有些时候会产出我们不期望的结果,造成困扰。

代码清单4-24 字符串连接运算符的优先级

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">

    var myData1 = 5 + 5;
    var myData2 = 5 + "5";

    console.log("Result 1: " + myData1);
    console.log("Result 2: " + myData2);

  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

上面脚本的运行结果如下:

---

```
Result 1: 10
Result 2: 55
```

---

你一定会惊讶于第二个结果。由于JavaScript的运算符优先级规则及其“过度热心”的类型转换机制,原本我们打算做加法运算,却被解释成了字符串连接运算。要避免这一问题,我们可以明确地主动转换操作数的数据类型,以保证运算正确。

#### 1. 数字转换为字符串

假设有几个数字变量需要当成字符串进行连接,如代码清单4-25所示,我们可以使用toString方法把它们转换成字符串类型。

代码清单4-25 数字的toString方法

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var myData1 = (5).toString() + String(5);
    console.log("Result: " + myData1);
  </script>
```

```

</head>
<body>
  This is a simple example
</body>
</html>

```

注意，我先用小括号把数值括起来，然后才调用了toString方法。这是因为在调用数字对象的方法之前，必须先把字面量的数字转换为数字对象。<sup>①</sup>在上例中我还展示了另一种把数字转换为字符串的方法，就是调用String函数并以数值为参数。这两种方法的效果完全相同，都是把一个数字转换为字符串，告诉+运算符执行字符串连接运算而不是加法运算。上面脚本的输出如下：

---

Result: 55

---

数字类型的对象还拥有一些支持把数字转换成字符串的其他方法，只是它们的行为有所不同。在表4-2中我简要列出了这些方法的用法和用途，其中列出的所有方法都是由数字类型对象定义的。

表4-2 数字转换为字符串的实用方法

方 法	描 述	返 回 值
toString()	返回十进制数字字符串	string
toString(2)、toString(8)、toString(16)	分别返回二进制、八进制、十六进制表示的数字字符串	string
toFixed(n)	表示成一个实数的字符串，小数点后面保留n位小数	string
toExponential(n)	返回一个以指数记数法表示的数字字符串，该字符串中小数点之前有一位有效数字，小数点后有n位数字	string
toPrecision(n)	返回一个n位有效数字的数字字符串，必要时可使用指数记数法	string

## 2. 字符串转换为数字

除了将数字转换为字符串，我们有时候需要对字符串数字做加法运算而不是字符串连接。如代码清单4-26所示，我们可以使用Number函数进行类型转换。

### 代码清单4-26 字符串转换为数字

```

<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var firstVal = "5";
    var secondVal = "5";

    var result = Number(firstVal) + Number(secondVal);

    console.log("Result: " + result);
  </script>

```

① 严格说来这个理由并不正确，之所以加括号是因为如果不加括号数字后面表示方法调用的句点会被解释成小数点。其实不加括号也可以，比如说要把5转换为字符串"5"，可以这样：5.toString()。没错，就是两个句点。——译者注

```
</head>
<body>
  This is a simple example
</body>
</html>
```

以上脚本的输出如下：

---

Result: 10

---

`Number`函数对要转换为数字的字符串值要求非常严格，不过还有两个比较灵活的函数，它们能忽略数字后面的非数字字符，这就是`parseInt`和`parseFloat`。表4-3列出了这3个函数的用法。

表4-3 字符串转换为数字的实用方法

方 法	描 述
<code>Number(str)</code>	解析str得到一个整数或者实数
<code>parseInt(str)</code>	解析str得到一个整数
<code>parseFloat(str)</code>	解析str得到一个整数或实数

## 4.6 数组

JavaScript语言中数组的行为与绝大多数编程语言中的数组极其相似。代码清单4-27演示了JavaScript中如何创建数组和增加数组元素。

代码清单4-27 创建数组并增加数组元素

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">

    var myArray = new Array();
    myArray[0] = 100;
    myArray[1] = "Adam";
    myArray[2] = true;

  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

这里通过调用`new Array()`得到一个新的空数组，并把它赋值给`myArray`变量。后面几条语句在数组的几个索引位置赋了值。（该代码清单在控制台没有任何消息输出。）

在这个例子中有两点需要注意：首先，创建数组时不必指定数组的元素个数（JavaScript数组会主

动调整自身大小以容纳更多的元素);其次,不必声明数组元素的数据类型(每一个JavaScript数组都能容纳任意类型的数据)。在本例中,我为数组设置了3个元素:一个数字、一个字符串和一个布尔值。

### 4.6.1 数组字面量

如代码清单4-28所示,我们可以使用数组字面量在创建数组的同时定义数组的元素。

代码清单4-28 使用数组字面量

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">

    var myArray = [100, "Adam", true];

  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

在这个例子中,两个中括号([])定义了一个新的数组,中括号之间(逗号分隔的数据)是数组的元素。这个数组被赋值给myArray变量。(该代码清单在控制台没有任何消息输出。)

### 4.6.2 读取和修改数组元素

如代码清单4-29所示,要读取某个元素的值,我们使用中括号括起需要访问元素的数组索引。

代码清单4-29 使用索引读取数组数据

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var myArray = [100, "Adam", true];
    console.log("Index 0: " + myArray[0]);
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

通过简单地指定索引并为该索引所在位置的元素赋一个新值,我们能够修改JavaScript数组中的任意元素。我们不但可以改变元素的值,还可以改变值的类型,没有任何问题。该清单的输出结果如下。

---

Index 0: 100

---

代码清单4-30演示了修改数组元素的方法。

代码清单4-30 修改数组元素

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var myArray = [100, "Adam", true];
    myArray[0] = "Tuesday";
    console.log("Index 0: " + myArray[0]);
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

在这个例子中，我把数组的第0个元素重新赋值为一个字符串，这个元素原来的值是一个数字，并得到以下输出：

---

```
Index 0: Tuesday
```

---

### 4.6.3 枚举数组内容

使用for循环可以枚举数组的内容。代码清单4-31展示了如何使用循环显示一个简单数组的内容。

代码清单4-31 枚举数组的内容

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var myArray = [100, "Adam", true];
    for (var i = 0; i < myArray.length; i++) {
      console.log("Index " + i + ": " + myArray[i]);
    }
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

JavaScript的循环正如其他语言中的循环。我们使用数组的length属性得到数组的元素个数，上面的代码清单输出如下。

```

Index 0: 100
Index 1: Adam
Index 2: true

```

## 4.6.4 数组内建方法

JavaScript数组对象内建了许多实用的方法，其中一些常用方法见表4-4。

表4-4 实用数组方法

方 法	描 述	返 回 值
concat(otherArray)	把otherArray的元素添加到数组的末尾，支持多个otherArray参数 <sup>①</sup>	Array
join(separator)	把数据所有的元素连接成一个字符串，separator参数用来分隔这些元素	string
pop()	把数组当成栈，删除数组的最后一个元素并返回该元素（出栈）	object
push(item)	把数组当成栈，把item作为数组的最后一个元素追加到数组中（入栈）	void
reverse()	得到与原数组元素顺序相反的新数组	Array
shift()	和pop()类似，不过它删除并返回的是数组的第一个元素	object
slice(start, end)	返回一个子数组	Array
sort()	返回元素排序后的新数组	Array
splice(index, count)	从指定的索引index开始，在数组中删除count个元素	Array
unshift(item)	和push()类似，不过它把新元素添加到数组的开头	void

## 4.7 错误处理

JavaScript使用try...catch语句处理错误。由于本书的主题是全面讲解jQuery知识，而非常规编程技术，因此对于本书中的绝大多数内容来说，你不必纠结于书中代码的错误。代码清单4-32展示了这种语句的用法。

代码清单4-32 处理异常

```

<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    try {
      var myArray;
      for (var i = 0; i < myArray.length; i++) {
        console.log("Index " + i + ": " + myArray[i]);
      }
    } catch (e) {
      console.log("Error: " + e);
    }
  </script>
</head>
</html>

```

<sup>①</sup> concat()方法的参数可以不是数组，此时参数会被作为一个元素添加到数组的末尾。——译者注



```

    }
  </script>
</head>
<body>
  This is a simple example
</body>
</html>

```

上面这段脚本中有一个很常见的问题。我试图使用一个未正确初始化的变量。我把可能出问题的代码封装在了try子句中，如果代码没有错误，这些语句就会正常执行，catch子句会被忽略。但是，因为代码中出现了错误，try子句中的代码就会立刻终止执行，把控制权交给catch子句并在控制台输出以下结果：

---

```
Error: TypeError: Cannot read property 'length' of undefined
```

---

JavaScript把错误信息保存在一个Error对象中，并把这个对象传递给catch子句。表4-5列出了Error对象定义的属性。

表4-5 Error对象

属 性	描 述	返 回 值
message	错误信息	string
name	错误的名字，默认值是Error	string
number	返回对应该错误的错误号	number

catch子句让我们在发生错误时有机会做一些补救或者清理工作。如代码清单4-33所示，如果无论是否发生错误都有一些语句需要执行，我们可以把这些语句放到可选的finally子句之中。

#### 代码清单4-33 使用finally子句

```

<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    try {
      var myArray;
      for (var i = 0; i < myArray.length; i++) {
        console.log("Index " + i + ": " + myArray[i]);
      }
    } catch (e) {
      Console.log("Error: " + e);
    } finally {
      console.log("Statements here are always executed");
    }
  </script>
</head>
<body>
  This is a simple example
</body>
</html>

```

该清单在控制台的输出结果如下。

---

```
Error: TypeError: Cannot read property 'length' of undefined
Statements here are always executed
```

---

## 4.8 比较 undefined 和 null 值

JavaScript有两个特殊的值，即undefined和null，在比较时需要格外小心。当我们读取一个变量，而该变量尚未被赋值的时候，或者试图读取一个不存在的对象属性时，代码就会返回undefined值。代码清单4-34演示了JavaScript中undefined的用法。

代码清单4-34 特殊的undefined

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">
    var myData = {
      name: "Adam",
      weather: "sunny",
    };
    console.log("Prop: " + myData.doesntexist);
  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

上面脚本的输出如下。

---

```
Prop: undefined
```

---

有人觉得JavaScript“古怪”，因为它还定义了一个特殊对象null。null与undefined有细微的差别。当值未定义时代码会返回undefined，而null用于表示值确实已经定义，但这个值却不属于任何一种合法的对象（字符串、数字或布尔值等）。也就是说，我们定义了一个没有值的值。为了帮助大家搞清楚这个问题，代码清单4-35展示了undefined值是如何变迁为null的。

代码清单4-35 使用undefined和null

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">

    var myData = {
```

```
        name: "Adam",
    });

    console.log("Var: " + myData.weather);
    console.log("Prop: " + ("weather" in myData));

    myData.weather = "sunny";
    console.log("Var: " + myData.weather);
    console.log("Prop: " + ("weather" in myData));

    myData.weather = null;
    console.log("Var: " + myData.weather);
    console.log("Prop: " + ("weather" in myData));

</script>
</head>
<body>
    This is a simple example
</body>
</html>
```

我创建了对象myData，并试图读取它并不存在的weather属性。

```
...
console.log("Var: " + myData.weather);
console.log("Prop: " + ("weather" in myData));
...
```

由于myData对象没有weather属性，因此myData.weather返回undefined值，并且我们使用in关键字检测myData对象是否具有weather属性也返回false。上面两条语句的输出如下。

---

```
Var: undefined
Prop: false
```

---

然后，我给weather属性赋了一个值，从而给myData对象添加了这一属性。

```
...
myData.weather = "sunny";
console.log("Var: " + myData.weather);
console.log("Prop: " + ("weather" in myData));
...
```

这时我再次读取weather属性的值，并检查myData对象是否拥有weather属性。和你料想的一样，myData对象确实定义了weather属性，并且它的值为sunny。

---

```
Var: sunny
Prop: true
```

---

接着，我把weather属性的值设置为null。

```
...
myData.weather = null;
...
```

这导致了非常特殊的效果：myData对象仍然拥有weather属性，可它声称自己没有值。接着我再次执行和上面同样的检查，得到以下结果。

---

```
Var: null
Prop: true
```

---

当我们需要比较undefined和null时，它们之间的差异就变得重要起来。null是一个对象，而undefined本质上是一种数据类型。

### 4.8.1 检查变量或者属性是否为null或undefined

如果需要检查属性是否为null或undefined，只要在if语句中使用取反运算符(!)就可以。具体例子见代码清单4-36。

代码清单4-36 检查一个属性是否是null或undefined

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">

    var myData = {
      name: "Adam",
      city: null
    };

    if (!myData.name) {
      console.log("name IS null or undefined");
    } else {
      console.log("name is NOT null or undefined");
    }

    if (!myData.city) {
      console.log("city IS null or undefined");
    } else {
      console.log("city is NOT null or undefined");
    }

  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

这项技术依赖于JavaScript的自动类型转换，（由于进行的是取反运算）因而我们希望检查的操作数会被当成布尔值处理。如果一个变量或属性是null或undefined，它们就会被自动转换为布尔值false。该清单的输出结果如下。

---

```
name is NOT null or undefined
city IS null or undefined
```

---

## 4.8.2 区分null和undefined

如果我们打算比较两个值，就要做个选择。如果希望对undefined和null一视同仁，我们可以使用相等运算符(==)并依赖JavaScript的自动类型转换。此时，undefined变量等于一个null变量。如果我们需区分null和undefined，就应该使用恒等运算符(===)。代码清单4-37演示了这两种比较。

代码清单4-37 null与undefined之间的相等和恒等测试

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Example</title>
  <script type="text/javascript">

    var firstVal = null;
    var secondVal;

    var equality = firstVal == secondVal;
    var identity = firstVal === secondVal;

    console.log("Equality: " + equality);
    console.log("Identity: " + identity);

  </script>
</head>
<body>
  This is a simple example
</body>
</html>
```

上面脚本的输出如下。

---

```
Equality: true
Identity: false
```

---

## 4.9 小结

本章，我们一起了解了贯穿全书的JavaScript核心特性。还记得吗？我们在本章前面说过，理解这些基本概念是使用jQuery的基础。在本书第二部分，我将适当介绍jQuery，并告诉你如何使用它。

本章我们开始学习第一个jQuery脚本。这个脚本非常简单，但展示了jQuery中几个至关重要的特性：在页面中怎么选择元素、怎么表示选中元素，以及jQuery与内建的原生DOM API（已成为HTML标准的一部分）之间关系的本质。表5-1概述了本章内容。

表5-1 本章概要

问 题	解决方法	代码清单
如何把jQuery添加到HTML页面	使用link元素导入保存在网站Web服务器或某个CDN上的jQuery库文件。再添加一个script元素存放我们编写的jQuery代码	1
如何动态决定是导入jQuery 1.x还是2.x版本	使用条件注释	2
如何选取页面元素	传递一个CSS选择器给\$（或jQuery）函数	3、4、10
如何重命名\$函数	使用noConflict方法	5、6
如何等到DOM就绪之后才执行我们的jQuery脚本	在全局的document变量上注册ready事件处理函数或者传递给\$主函数一个函数并把我们的脚本包含在这个函数中	7、8
如何控制ready事件的触发时机	使用holdReady事件	9
如何限制页面元素的选取范围	为\$函数提供上下文参数	11、12
如何得到创建jQuery对象时使用的上下文	读取context属性	13
如何使用HTMLElement对象生成jQuery对象	把HTMLElement对象用作\$函数的参数	14
如何遍历jQuery对象包含的元素	把jQuery对象当成数组处理或者使用each方法	15、16
如何得到jQuery对象中指定顺序号的元素	使用index或get方法	17~19
如何同时对jQuery对象中包含的多个元素执行同一个操作	在这个jQuery对象上直接调用jQuery方法	20
如何对一个jQuery结果集执行多个操作	链式调用这些方法	21~23
如何处理事件	使用jQuery事件处理函数	24

## 与第一版相比jQuery发生的变化

在本书第一版中，我使用jQuery对象的selector属性得到查询字符串，以便将来多次查询。在jQuery 1.9中，selector属性已经过时（并未删除），不再推荐使用。

jQuery 1.9/2.0的选择器引擎（Sizzle）支持一些新的CSS3选择器（即使浏览器尚不支持）。这些选择器是：`:nth-last-child`、`:nth-of-type`、`:nth-last-of-type`、`:first-of-type`、`:last-of-type`、`:only-of-type`、`:target`、`:root`和`:lang`。

## 5.1 安装 jQuery

要使用jQuery，我们首先要把它添加到正在开发的页面中。代码清单5-1即本书第一部分中出现过的花店页面，在这里增加了jQuery库。

代码清单5-1 花店示例页面

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
        <div id="row1" class="drow">
          <div class="dcell">
            <label for="aster">Aster:</label>
            <input name="aster" value="0" required>
          </div>
          <div class="dcell">
            <label for="daffodil">Daffodil:</label>
            <input name="daffodil" value="0" required >
          </div>
          <div class="dcell">
            <label for="rose">Rose:</label>
            <input name="rose" value="0" required>
          </div>
        </div>
        <div id="row2" class="drow">
          <div class="dcell">
            <label for="peony">Peony:</label>
            <input name="peony" value="0" required>
          </div>
          <div class="dcell">
            <label for="primula">Primula:</label>
            <input name="primula" value="0" required>
          </div>
          <div class="dcell">
            <label for="snowdrop">Snowdrop:</label>
            <input name="snowdrop" value="0" required>
          </div>
        </div>
      </div>
    </div>
  </form>
</body>
</html>
```



```

        </div>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
</form>
</body>
</html>

```

为方便大家关注页面内容，我把页面中的CSS样式挪到了单个文件styles.css中（可参阅第3章），并使用下面的标签把jQuery库添加到了页面中：

```

...
<script src="jquery-2.0.2.js" type="text/javascript"></script>
...

```

一旦我们决定了使用哪个jQuery产品线（1.x或2.x，详见第3章），就可以到jquery.com下载适合自己的版本：以.js结尾的版本，或者是以.min.js扩展名结尾的版本。在我写到这里时，jQuery 2.x产品线的最新版本的文件名分别是jquery-2.0.2.js和jquery-2.0.2.min.js。

在网站或应用程序的开发过程中通常使用jquery-2.0.2.js文件。这个文件大约有240 KB，里面都是标准JavaScript代码。我们可以打开并阅读这个文件，学习jQuery是如何实现它的各个功能的，也可以利用浏览器调试工具来找出错误的代码。

---

**提示** jQuery的开发非常活跃，因此当你读到这里的时候，几乎可以肯定已经有了更新的版本。不过请放心，虽然我们提到一些方法已经过时，并且jQuery也已分为两个产品线，jQuery的API还是令人难以置信地稳定。我在本书中讲到的所有技术都能在jQuery 1.x和2.x以及未来的版本正常使用。

---

另一个文件是jQuery.2.0.2.min.js，当我们把网站或者应用程序部署到生产环境时，应该使用这个文件。这个文件中的JavaScript代码与第一个文件完全相同，但是经过了压缩处理。也就是说，代码中所有的注释和不必要的空白都被删除，而且为了减小体积，有意义的变量名也被替换成很短的变量名。这个版本的jQuery库可读性极差，几乎不能用于调试，但它最大的优点是体积非常小。这个压缩文件的体积只有82 KB，如果许多页面都需要用到jQuery，使用压缩版的jQuery能为我们节省很多带宽。

---

**提示** 可以从jquery.com下载一份源码地图（扩展名为.min.map）。有了源码地图，我们就能更容易地调试压缩代码。当我写到这里时，这个新方法尚未得到广泛使用。访问<http://www.html5rocks.com/en/tutorials/developertools/sourcemaps>可了解此方法的一个简单演示。

---

## 使用CDN版的jQuery

还有一种方法，即使用托管在公共CDN（Content Delivery Network，内容分发网络）上的jQuery库。CDN是一种内容分发网络，当用户请求其数据时，CDN能智能的分配离用户最近的服务器提供服务。使用CDN有两个优点：首先，用户体验更好了。因为jQuery库不是来自我们的服务器，而是来自离用户最近的CDN服务器（通常会更快）。由于jQuery库是如此流行，用户浏览器很可能已经缓存了

来自其他网站的同一个jQuery库，这意味着用户很可能不必真的向CDN服务器请求文件（而是直接从缓存中读取）。第二个好处是节省了宝贵的传输jQuery库所需的带宽。对于一些流量很大的站点来说，这么做能显著节省费用。

如果我们打算使用CDN，就必须信赖选用的CDN服务。我们肯定希望用户总是能接收到正确的文件，并且服务不会中断。谷歌和微软都免费提供jQuery（和其他流行的JavaScript库）托管服务。两家公司在运营高可靠性服务方面都极富经验，它们也不会中途篡改jQuery。你可以访问[www.asp.net/ajaxlibrary/cdn.ashx](http://www.asp.net/ajaxlibrary/cdn.ashx)详细了解微软的CDN服务，并访问<http://code.google.com/apis/libraries/devguide.html>了解谷歌提供的服务。

CDN方案并不适合那些内联网（intranet）应用程序。使用CDN方案，浏览器就必须访问因特网以得到jQuery库，不如使用更近、更快也更节省带宽的本地服务器。

## 条件注释

在代码清单5-1中，我加载的jQuery版本是2.0.2。在第1章已经说过，这个版本速度最快，但是不支持老版本的IE浏览器。

选择性能还是选择兼容？不必纠结，有一项技术可以帮助我们自动加载适当的jQuery版本。这就是条件注释，微软公司在IE5.0及更高版本中提供了这一非标准的HTML增强技术。代码清单5-2展示了如何利用这一技术在示例页面中动态加载适用的jQuery版本。

代码清单5-2 利用条件注释动态加载jQuery 1.x或2.x

```
...
<head>
  <title>Example</title>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <!--[if lt IE 9]>
    <script src="jquery-1.10.1.js" type="text/javascript"></script>
  <![endif]-->
  <!--[if gte IE 9]><!-->
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <!--<![endif]-->
</head>
...
```

利用条件注释，如果是比IE9更老的IE浏览器，就加载jQuery 1.10.1版本，否则加载jQuery 2.0.2版本。在复制这些代码时一定要小心，务必一字不差地复制，因为这里很容易出错。

---

**提示** 对于条件注释，本书没有详细介绍。如果你希望深入了解条件注释，可访问[http://en.wikipedia.org/wiki/Conditional\\_comment](http://en.wikipedia.org/wiki/Conditional_comment)。

---

## 5.2 第一个jQuery脚本

我们已经把jQuery库添加到页面中，现在可以编写第一个jQuery脚本了。代码清单5-3是一个简单

的例子，演示了jQuery的几个基础功能。

### 代码清单5-3 第一个jQuery脚本

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script type="text/javascript">
    $(document).ready(function () {
      $("img:odd").mouseenter(function(e) {
        $(this).css("opacity", 0.5);
      }).mouseout(function(e) {
        $(this).css("opacity", 1.0);
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
        <div id="row1" class="drow">
          <div class="dcell">
            <label for="aster">Aster:</label>
            <input name="aster" value="0" required>
          </div>
          <div class="dcell">
            <label for="daffodil">Daffodil:</label>
            <input name="daffodil" value="0" required >
          </div>
          <div class="dcell">
            <label for="rose">Rose:</label>
            <input name="rose" value="0" required>
          </div>
        </div>
        <div id="row2" class="drow">
          <div class="dcell">
            <label for="peony">Peony:</label>
            <input name="peony" value="0" required>
          </div>
          <div class="dcell">
            <label for="primula">Primula:</label>
            <input name="primula" value="0" required>
          </div>
          <div class="dcell">
            <label for="snowdrop">Snowdrop:</label>
            <input name="snowdrop" value="0" required>
          </div>
        </div>
      </div>
    </div>
  </form>
</body>
</html>
```

```

    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
</html>

```

虽然这个脚本很短，但它演示了jQuery中最重要的几个功能和特性。我会在本章逐行讲解这个脚本，不过读过本书后面的部分之后你才能真正理解这个脚本的功能。首先来看图5-1，它展示了这一脚本产生的效果。

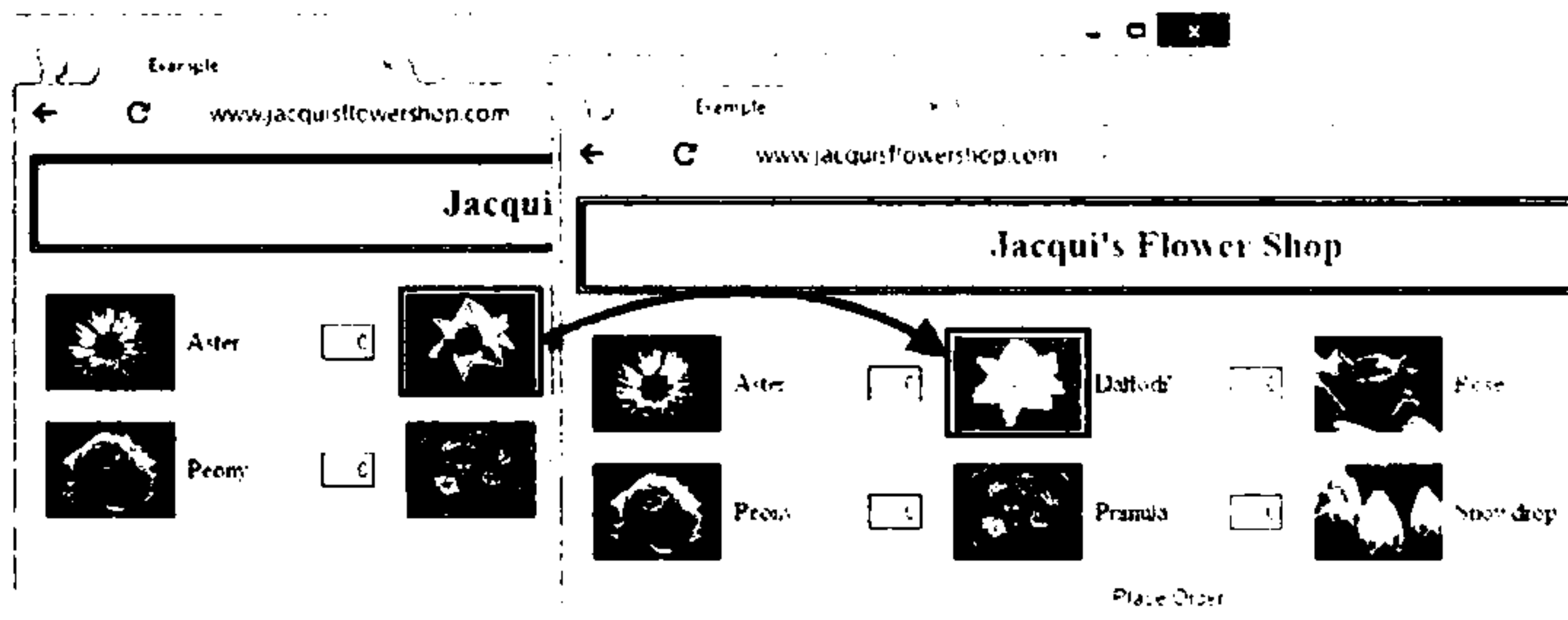


图5-1 改变图片的透明度

**提示** 你也许已经注意到，在上面的代码清单里只加载了jQuery 2.0.2，而没有使用条件注释，本书中的绝大部分代码清单都是如此。这是因为我希望让示例代码保持简单清晰。但是我强烈建议你使用条件注释，在我的项目中大量使用了这一技术，它相当有用。

当我们把鼠标放到水仙、牡丹和雪滴花图片上时，脚本就会改变图片的透明度。图片会变得比原来亮，并且有些褪色。当鼠标移开时，图片又恢复到原来的透明度。然而，紫菀、玫瑰和报春花图片却不会受脚本影响。

### 5.3 理解 jQuery 的\$函数

我们利用`$(...)`函数使用jQuery的功能，描述方便起见，后面会把它简写为`$`函数。`$`函数是jQuery精彩世界的入口，也是jQuery函数的简写形式。我们也可以使用`$`函数的全名重写刚才的脚本（代码清单5-4）。

#### 代码清单5-4 使用jQuery函数替换简写形式

```

...
<script type="text/javascript">
  jQuery(document).ready(function () {
    jQuery("img:odd").mouseenter(function(e) {
      jQuery(this).css("opacity", 0.5);
    });
  });

```

```
        }).mouseout(function(e) {  
            jQuery(this).css("opacity", 1.0);  
        });  
    });  
</script>  
...
```

这个脚本的功能与上一个完全相同，只是需要多打几个字，好处是一眼就能让人看出使用了 jQuery。

这一点非常有用，因为除 jQuery 之外，还有一些别的 JavaScript 库也使用 \$ 符号。这意味着在同一个页面若加载了多个库，就有可能产生冲突。如果遇到这种情况，我们可以调用 jQuery.noConflict 方法让 jQuery 放弃使用 \$ 符号（参见代码清单 5-5）。

代码清单 5-5 让 jQuery 放弃 \$ 符号

```
...  
<script type="text/javascript">  
    jQuery.noConflict();  
    jQuery(document).ready(function () {  
        jQuery("img:odd").mouseenter(function(e) {  
            jQuery(this).css("opacity", 0.5);  
        }).mouseout(function(e) {  
            jQuery(this).css("opacity", 1.0);  
        });  
    });  
</script>  
...
```

我们也可以自己定义 jQuery 简写符号，只需把 noConflict 方法的返回值赋给一个变量即可（参见代码清单 5-6）。

代码清单 5-6 自定义 jQuery 简写符号

```
...  
<script type="text/javascript">  
    var jq = jQuery.noConflict();  
    jq(document).ready(function () {  
        jq("img:odd").mouseenter(function(e) {  
            jq(this).css("opacity", 0.5);  
        }).mouseout(function(e) {  
            jq(this).css("opacity", 1.0);  
        });  
    });  
</script>  
...
```

在这个例子中，我先是创建了一个自己的简写符号 jq，接着在后面的脚本中一直使用这个符号。

---

**提示** 纵贯全书，我会一直使用 \$ 符号表示 jQuery 函数。一来这是 jQuery 的习惯简写符号，二来我也不会使用那些也使用了 \$ 符号的其他库。

---

不管使用哪种方式引用jQuery主函数，它支持的参数类型始终一样。表5-2列出了其中最重要的一些参数。在本章后面，我会详细讲解这些参数（不包括最后一种参数，那个参数要到第7章才会讲到）。

表5-2 jQuery主函数的参数

参 数	描 述
<code>\$(function)</code>	指定DOM就绪后才执行的函数
<code>\$(selector) \$(selector, context)</code>	从页面中选择元素
<code>\$(HTMLElement) \$(HTMLElement[])</code>	基于一个HTMLElement对象或一个HTMLElement对象数组生成jQuery对象
<code>\$()</code>	选择0个元素
<code>\$(HTML) \$(HTML, map)</code>	基于HTML代码片段生成新元素，支持可选的map对象参数，利用它定义新元素的属性（更多详细信息，请参阅第7章）

## 5.4 等待 DOM 就绪

在第2章，我把script元素放到页面的末尾，这是为了让浏览器在准备好DOM中所有的对象之后才执行我的脚本。利用下面的技术，jQuery能帮我们更完美地解决这个问题。

```
...
<script type="text/javascript">
    $(document).ready(function () {
        // 要执行的代码
    });
</script>
...
```

我不再把JavaScript语句直接放入script元素之中，而是把document对象（参见第1章）作为参数传递给\$函数，接着调用ready方法，把我希望在浏览器加载完所有HTML内容之后才会执行的函数作为参数传递给ready方法。代码清单5-7展示了如何在示例页面中使用这一技术。

代码清单5-7 等待DOM就绪

```
...
<script type="text/javascript">
    $(document).ready(function () {
        $("img:odd").mouseenter(function (e) {
            $(this).css("opacity", 0.5);
        }).mouseout(function (e) {
            $(this).css("opacity", 1.0);
        });
    });
</script>
...
```

使用ready方法我们可以安全地把script元素放到页面的任意位置，因为jQuery会保证这些代码不会过早执行。我一般把script元素放到页面的head标签之中，没有别的什么原因，仅仅是我的个人偏好而已。

**注意** 以函数为参数传递给ready方法，这本质上是为jQuery的ready事件绑定事件处理函数。第9章会详细讲解jQuery事件，现在你只需要知道一点：传递给ready方法的函数会在页面加载完成（DOM就绪）之后自动执行。

### 遗漏函数声明

一个常见的错误是漏掉这段“咒语”中的function部分，结果传给ready方法一堆JavaScript语句。这样不行：浏览器会立即执行这些语句，根本不会等到DOM就绪。请看下面的脚本。

```
...
<script type="text/javascript">
    function countImgElements() {
        return $("img").length;
    }

    $(document).ready(function() {
        console.log("Ready function invoked. IMG count: " + countImgElements());
    });

    $(document).ready(
        console.log("Ready statement invoked. IMG count: " + countImgElements())
    );
</script>
...
```

在这个例子中，我调用了ready方法两次，一次传入了一个函数，而另一次传递了一条普通的JavaScript语句。两种情况我都调用了countImgElements函数，这个函数会返回DOM中img元素的个数。现在你不必着急弄清countImgElements函数的工作原理。我会在本章后面解释length属性。当我们载入此页面，浏览器会执行这些脚本并在控制台输出以下信息：

```
Ready statement invoked. IMG count: 0
Ready function invoked. IMG count: 6
```

看到了没？页面还在加载，浏览器还没有解析到页面中的img元素，未包含在函数中的语句就执行了，得出这样的结果一点也不奇怪。

## 5.4.1 另一种写法

如果我们愿意，也可以把函数直接作为参数传递给jQuery主函数\$。这种做法的效果与\$(document).ready方式完全相同。代码清单5-8演示了这种写法。

代码清单5-8 延迟函数的执行，直到DOM就绪

```
...
<script type="text/javascript">
    $(function() {
        $("img:odd").mouseenter(function(e) {
            $(this).css("opacity", 0.5);
        }).mouseout(function(e) {
```



```

        $(this).css("opacity", 1.0);
    });
});
</script>
...

```

### 5.4.2 延迟ready事件的触发时间

我们可以使用holdReady方法控制ready事件的触发时机。当我们需要动态载入外部资源（不怎么样的高级技术）时，这个方法很有用。holdReady方法必须在ready事件触发之前调用，这样当我们确认准备好了之后才触发ready事件。代码清单5-9演示了这个方法的使用。

代码清单5-9 使用holdReady方法

```

...
<script type="text/javascript">

    $.holdReady(true);

    $(document).ready(function() {
        console.log("Ready event triggered");
        $("img:odd").mouseenter(function(e) {
            $(this).css("opacity", 0.5);
        }).mouseout(function(e) {
            $(this).css("opacity", 1.0);
        });
    });

    setTimeout(function() {
        console.log("Releasing hold");
        $.holdReady(false);
    }, 5000);

</script>
...

```

在这个例子中，脚本的第一句就使用参数true调用了holdReady方法，这表示我们希望人工控制ready事件的发生时机。接着我们定义了ready事件发生时要执行的函数：和本章一开始的那个函数一样，修改一些图片的透明度。

最后，我使用setTimeout函数设置了一个定时器，该定时器将在5000 ms（5 s）后调用一个函数。这个函数会调用holdReady方法（使用false参数），告诉jQuery触发ready事件。最终结果是成功延迟了ready事件5s。当页面成功载入浏览器时，我添加的一些调试消息将以下内容输出到控制台。

```

Releasing hold
Ready event triggered

```

**提示** 我们可以多次调用holdReady方法，不过有一点必须注意：在ready事件被真正触发之前，使用true参数调用holdReady方法的次数一定要与使用false参数调用的holdReady方法的次数相同



## 5.5 选择元素

从DOM中选择元素是jQuery最重要的功能之一。在下面的示例脚本中（代码清单5-10），我选中了所有奇数位置的img元素。

代码清单5-10 从DOM中选择元素

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("img:odd").mouseenter(function(e) {
            $(this).css("opacity", 0.5);
        }).mouseout(function(e) {
            $(this).css("opacity", 1.0);
        })
    });
</script>
...
```

要选择元素，你只需要把选择器参数传递给\$函数。jQuery不但支持我在第3章讲到的所有选择器，而且额外支持一些选择器，这些选择器不但给我们带来了方便，还能实现粒度更细的控制。上面的示例使用了:odd伪选择器，它会选取主选择器（本例中是img，表示选择所有img元素，参见第3章）中所有的奇数元素。:odd（还有:even）从0开始计数，也就是说第1个元素其实是第0个，是偶数元素。如果你是第一次接触这个选择器，也许会有一点点困惑（习惯就好啦）。表5-3列出了最有用的jQuery选择器。

表5-3 jQuery扩充的选择器

选 择 器	描 述
:animated	选择所有正在处理动画的元素
:contains(text)	选择包含指定文本的元素
:eq(n)	选择第n个元素（从0开始计数）
:even	选择所有的偶数元素（从0开始计数）
:first	选择第一个匹配的元素
:gt(n)	选择序号大于n的所有元素（从0开始计数）
:has(selector)	选择至少包含一个匹配指定选择器的元素的元素
:last	选择最后一个匹配的元素
:lt(n)	选择序号小于n的所有元素（从0开始计数）
:odd	选择所有奇数元素（从0开始计数）
:text	选择所有的输入文本框元素

**提示** 调用\$函数但不提供任何参数（\$()），这样就可以选择“空”（即什么都不选择）。之所以在这儿提到这个功能只是为了保持介绍的完整性，老实说，这个功能我一次也没有用过。

之所以说这些选择器最有用,这是因为仅使用CSS选择器很难实现这些功能。这些选择器就像CSS中的伪选择器,可以单独使用,也可以配合DOM中所有的元素使用,比如:

```
...
$(" :even")
...
```

还可以与其他选择器配合使用,从而限制它们的作用范围, 比如:

```
...
$("img:even")
...
```

除了这些选择器, jQuery还定义了一些类型选择器(参见表5-4)。

表5-4 jQuery扩展的类型选择器

选 择 器	描 述
:button	选择所有的按钮
:checkbox	选择所有的复选框
:file	选择所有的文件上传输入框
:header	选择所有的标题元素(h1、h2等)
:hidden	选择所有被隐藏的元素
:image	选择所有的图片元素
:input	选择所有的input元素
:last	选择最后一个匹配的元素 <sup>①</sup>
:parent	选择所有拥有至少一个子元素的元素
:password	选择所有的密码输入框
:radio	选择所有的单选框
:reset	选择所有的表单重置按钮
:selected	选择所有的状态为已选中的元素
:submit	选择所有的表单提交按钮
:visible	选择所有的可见元素

### 关于选择器性能

如果你已经花了一些时间了解jQuery,你一定看到过有关选择器性能的讨论。很多人花了很多的时间来使用不同方式的选择器分析比较,力求榨出jQuery性能的最后一滴油。

我的观点很简单:这不是问题,即便它造成了问题,也只是表明是其他某个地方出了问题。jQuery的(选择器)性能相当棒,特别是在主流浏览器的最新版本,这些浏览器都拥有极快的

<sup>①</sup> 这个选择器更适合出现在表5-3中,并且确实已经出现过了。此外,表5-3的:text选择器其实更适合出现在表5-4中。——译者注

JavaScript引擎。当开发者在优化选择器性能时，通常他们都正在处理大量的HTML元素，执行一次选择需要几百毫秒。当一次用户操作需要几次这样的选择时，用户就会注意到我们的产品反应迟缓。

这不能怪jQuery，因为我们发送了超出浏览器处理能力的过多数据。浏览器正变得越来越强大，越来越能干，但再强大的浏览器也有极限，特别是那些运行在移动设备上的一些比较老的浏览器。

如果发现选择元素时不够快，先不要急着优化选择器，审视我们是如何使用HTML标签的：在服务器端做适当的处理以便尽量减少发送给浏览器的HTML内容，Web应用就是Web应用，它们不是桌面应用程序。

## 使用上下文限制搜索范围

jQuery默认在整个DOM中搜索元素。多给\$函数提供一个参数，我们便能够限制搜索的范围。这个参数规定了搜索的上下文，jQuery用它作为查找元素的起点。代码清单5-11演示了这一功能。

代码清单5-11 使用上下文限制搜索范围

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("img:odd", $(".drow")).mouseenter(function(e) {
            $(this).css("opacity", 0.5);
        }).mouseout(function(e) {
            $(this).css("opacity", 1.0);
        })
    });
</script>
...
```

在这个例子中，我将一个选择器用作另一个选择器的上下文。jQuery先计算上下文选择器，得到拥有drow类的所有元素，然后把把这些元素作为img:odd选择器的上下文进一步匹配元素。

如果提供的上下文选择器匹配多个元素，那么其中的每个元素都会成为搜索的一个起点。这很有趣，也很微妙。先把匹配上下文选择器的元素收集到一起，然后再匹配主选择器。在本例中，这意味着img:odd选择器会应用在.drow选择器的结果上，也就是说这次匹配的奇数元素与在整个页面中匹配奇数元素有所不同。最后，每个拥有drow类的div元素中的奇数图片（水仙花和报春花）支持透明度特效。如果我们省略上下文的话，那么就是水仙花、牡丹花和雪滴花图片支持特效。

如果希望搜索从页面的某个特定位置开始，我们可以将HTMLElement对象用作上下文。代码清单5-12提供了这样一个示例。在下一节中，我们很快就会发现从jQuery的世界切换到HTMLElement对象的世界是多么容易。

代码清单5-12 将HTMLElement对象用作上下文

```
...
<script type="text/javascript">
    $(document).ready(function() {
```

```

    var elem = document.getElementById("oblock");

    $("img:odd", elem).mouseenter(function(e) {
        $(this).css("opacity", 0.5);
    }).mouseout(function(e) {
        $(this).css("opacity", 1.0);
    });
});
</script>
...

```

这段脚本仅在oblock元素内搜索奇数图片。当然，使用CSS中的后代选择器也能达成同样的目的。将HTMLElement对象用作上下文有一个优点，即在需要限制搜索范围时不必构造选择器字符串。这种方案的一个典型应用场景是事件处理。我们将在第9章详细介绍事件及那种情况下HTMLElement对象是如何产生的。

## 5.6 理解选择结果

在使用jQuery从DOM中选择元素时，人们为\$函数的返回结果取了一个令人困惑的名字：jQuery对象，它代表着0个或多个DOM元素。事实上，当我们执行修改元素的jQuery操作时，这些操作返回的很可能就是一个jQuery对象。这是jQuery框架中非常重要的一个特征，稍后我还会详细介绍这一点。

jQuery对象中定义了许多方法和属性，实际上它们构成了本书余下的内容。不过本章只会简要介绍jQuery对象的一些标准成员（参见表5-5）。

表5-5 jQuery对象的标准成员

选 择 器	描 述	返 回 值
context	选择元素时使用的上下文对象	HTMLElement
each(function)	在每个选中元素上运行给定的function（并返回function修改之后的jQuery对象）	jQuery
get(index)	返回给定索引值（index）对应的HTMLElement对象	HTMLElement
index(HTMLElement)	返回给定HTMLElement在jQuery对象中的索引序号	number
index(jQuery)	返回给定jQuery对象中的第一个元素在主jQuery对象中的索引序号	number
index(selector)	返回主jQuery对象中的第一个元素在选择器参数对应的结果集中的索引序号	number
length	返回jQuery对象中包含元素的个数	number
selector	返回选择器	string
size()	返回jQuery对象中包含元素的个数	number
toArray()	返回一个由jQuery对象中包含的元素构成的HTMLElement对象数组	HTMLElement[]

### 5.6.1 确定jQuery对象对应的上下文对象

context属性中保存着创建jQuery对象时使用的上下文对象。在创建jQuery对象时，如果我们使用的是单一HTMLElement对象，那么context属性就是这个HTMLElement对象。如果没有提供context参数或者使用了多个元素作为context参数（比如本章前面的例子），那么context属性就是表示整个文档的document对象。代码清单5-13演示了这个属性的用法。

代码清单5-13 确定创建jQuery对象时使用的上下文对象

```

...
<script type="text/javascript">
    $(document).ready(function() {
        var jq1 = $("img:odd");
        console.log("No context: " + jq1.context.tagName);

        var jq2 = $("img:odd", $('<div>.drow'));
        console.log("Multiple context elements: " + jq2.context.tagName);

        var jq3 = $("img:odd", document.getElementById("oblock"));
        console.log("Single context element: " + jq3.context.tagName);
    });
</script>
...

```

上面这段脚本展示了3种情况<sup>①</sup>：没有上下文、多元素上下文（一个jQuery对象）、单一HTMLElement对象上下文。输出如下：

```

No context: undefined
Multiple context elements: undefined
Single context element: DIV

```

## 5.6.2 处理DOM对象

jQuery并不能代替DOM。不过要是没有jQuery，我们就不可能如此方便地处理DOM。第2章中讲到的HTMLElement对象之后还会用到。jQuery提供了一种极其简单的方法帮助我们在两种对象之间进行切换。在传统DOM和jQuery对象之间来回切换极其容易，这本身就是jQuery库优雅设计的一部分。这种设计能有效帮助我们处理jQuery脚本和非jQuery脚本之间的兼容问题。

### 1. 由DOM对象生成jQuery对象

如果我们把一个HTMLElement对象或一个HTMLElement对象数组作为参数传递给\$函数，就得到了相应的jQuery对象。在需要改写未使用jQuery的JavaScript代码时，或者在必须使用底层DOM对象的场合（比如处理事件时）中，我们经常会用到这一功能。代码清单5-14给出了这样一个例子。

代码清单5-14 由DOM对象生成jQuery对象

```

...
<script type="text/javascript">
    $(document).ready(function() {

        var elems = document.getElementsByTagName("img");

        $(elems).mouseenter(function(e) {

```

① 第一种和第二种情况的上下文对象都是document对象，而document对象并没有tagName属性，所以会输出undefined。估计作者并未认真查阅文档，仅仅根据这个输出，就误以为上下文对象也是undefined，所以得出了错误的结论。

```

        $(this).css("opacity", 0.5);
    }).mouseout(function(e) {
        $(this).css("opacity", 1.0);
    });
});
</script>
...

```

在这个例子中，我先是使用`document.getElementsByTagName`方法，而不是直接使用jQuery加一个选择器的方式选中页面中的所有元素。接着，我把返回结果（一个HTML`Element`数组）作为参数传递给\$函数，从而得到一个标准的，与前面几个例子类似的jQuery对象。

这段脚本还演示了如何使用单一HTML`Element`对象生成jQuery对象。

```

...
$(this).css("opacity", 1.0);
...

```

当我们处理事件时，jQuery会把正在处理事件的HTML`Element`对象赋值给`this`变量。由于我们会在第9章详细介绍jQuery事件，因此现在不必深入介绍这一主题。不过，稍后本章确实又一次用到了包含这些语句的函数。

## 2. 把jQuery对象视为数组

我们可以把jQuery对象看成是一个HTML`Element`对象数组。也就是说，我们在享受jQuery高级功能的同时仍然可以直接访问DOM。我们不但能使用`length`属性或者`size`方法直接确定jQuery对象中共有多少个元素，还能使用数组风格的索引语法（使用中括号`[]`）直接访问具体的DOM对象。

---

**提示** 我们也可以使用`toArray`方法把HTML`Element`对象从jQuery对象中释放出来，得到一个真正的数组。我更喜欢使用jQuery对象，不过有时候也需要使用DOM对象，比如在处理那些未采用jQuery的遗留代码时。

---

代码清单5-15演示了如何枚举jQuery对象的内容并访问其中的HTML`Element`对象。

### 代码清单5-15 把jQuery对象当成数组

```

...
<script type="text/javascript">
    $(document).ready(function() {
        var elems = $("img:odd");
        for (var i = 0; i < elems.length; i++) {
            console.log("Element: " + elems[i].tagName + " " + elems[i].src);
        }
    });
</script>
...

```

在上面的代码中，我使用\$函数选中奇数元素，迭代它们并在控制台输出`tagName`和`src`属性的值。输出结果如下。

```

Element: IMG http://www.jacquisflowershop.com/jquery/daffodil.png
Element: IMG http://www.jacquisflowershop.com/jquery/peony.png
Element: IMG http://www.jacquisflowershop.com/jquery/snowdrop.png

```

### 3. 迭代处理jQuery对象内含的DOM对象

each方法允许我们定义一个回调函数，迭代处理jQuery对象中所有的DOM对象，jQuery对象内有多少个DOM对象，这个函数就会被调用多少次。代码清单5-16给出了一个例子。

代码清单5-16 使用each方法

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $('img:odd').each(function(index, elem) {
            console.log("Element: " + elem.tagName + " " + elem.src);
        });
    });
</script>
...

```

jQuery会传递两个参数给我们定义的函数。第一个是当前元素在结果集中的序号（从0开始），第二个是当前DOM对象。在这个例子中，我在控制台输出了DOM对象的标签名和src属性值，产生的结果与上一个例子完全相同。

### 4. 找出元素的序号或指定的元素

index方法能帮助我们得到jQuery对象内某个HTMLElement对象的序号。index方法的参数既可以是HTMLElement对象，也可以是jQuery对象。如果我们将jQuery对象用作index方法的参数，那么返回jQuery对象中第一个匹配元素的序号。代码清单5-17给出了一个例子。

代码清单5-17 确定一个HTMLElement的序号

```

...
<script type="text/javascript">
    $(document).ready(function() {

        var elems = $('body *');

        // 使用原生DOM API找出元素的序号
        var index = elems.index(document.getElementById("oblock"));
        console.log("Index using DOM element is: " + index);

        // 利用另一个jQuery对象找出同一元素的序号
        index = elems.index($('#oblock'));
        console.log("Index using jQuery object is: " + index);
    });
</script>
...

```

在这个例子中，我先是使用DOM API中的getElementById方法，根据元素id属性的值选中一个div元素，这个方法会返回一个HTMLElement对象。接着，我在一个jQuery对象上调用index方法，以确定代表这个div元素的对象顺序号。最后，我以对应同一个div元素的jQuery对象作为参数再次调用index



方法。我把两种方法的返回值都输出到控制台，得到以下结果。

---

```
Index using DOM element is: 2
Index using jQuery object is: 2
```

---

index方法也接受字符串参数。这种情况下，index方法会把这个字符串解释成一个选择器。不过，将字符串用作参数时，index方法的行为将与前面的例子显著不同。代码清单5-18展示的就是这种情况。

**代码清单5-18** 使用选择器作为index方法的参数

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var imgElems = $('img:odd');
        // 使用选择器确定元素顺序号
        index = imgElems.index("body *");
        console.log("Index using selector is: " + index);

        // 使用jQuery对象做同一件事
        index = $("body *").index(imgElems);
        console.log("Index using jQuery object is: " + index);

    });
</script>
...
```

当我们传递一个字符串参数给index方法，进行运算的结果集元素的顺序就变了。假设jQuery根据选择器参数得到的元素集合是A，调用index方法的jQuery对象内含的元素集合是B，那么最后返回的是B集合中的第一个元素在A集合中的序号。也就是说，下面这个语句：

```
...
index = imgElems.index("body *");
...
```

等同于下面这个语句：

```
...
index = $("body *").index(imgElems);
...
```

实际上，传递一个字符串参数给index方法会翻转进行运算的两个结果集。该代码段输出结果如下：

---

```
Index using selector is: 10
Index using jQuery object is: 10
```

---

---

**提示** 我们也可以使用index方法而不提供任何参数，从而得到一个元素相对于邻居的序号。用jQuery遍历DOM这种方法非常有用，它也是第7章的主题。

---



get方法是index方法有力的补充，它接受一个序号参数，得到jQuery对象内相应的HTMLElement对象。get方法的效果与我们在本章前面提到的数组风格的索引语法完全相同。代码清单5-19给出了get方法的一个例子。

代码清单5-19 获取指定索引位置的HTMLElement对象

```
...
<script type="text/javascript">
  $(document).ready(function() {
    var elem = $("img:odd").get(1);
    console.log("Element: " + elem.tagName + " " + elem.src);
  });
</script>
...
```

在这个脚本中，我先是选中奇数img元素，然后使用get方法得到序号1对应的HTMLElement对象，并把它.tagName和src属性值输出到控制台。结果如下：

---

```
Element: IMG http://www.jacquisflowershop.com/jquery/peony.png
```

---

## 5.7 修改多个元素与链式方法调用

几乎所有jQuery对象内建方法都支持直接修改内含的全部元素，这也是jQuery如此简洁、表达能力极强的原因。注意，我在这里的用词是几乎所有，也就是说还有一些方法不支持修改全部内含元素，在后面的几章中你会看到这样的例子。在代码清单5-20中，演示了如何使用DOM API处理多个元素。

代码清单5-20 使用DOM API操作多个元素

```
...
<script type="text/javascript">
  $(document).ready(function() {

    var labelElems = document.getElementsByTagName("label");
    for (var i = 0; i < labelElems.length; i++) {
      labelElems[i].style.color = "blue";
    }
  });
</script>
...
```

script元素内的代码选中页面中所有的label元素，并把这些元素的CSS color属性的值修改为blue。代码清单5-21演示了使用jQuery完成这一任务有多么容易。

代码清单5-21 使用jQuery处理多个元素

```
...
<script type="text/javascript">
  $(document).ready(function () {
    $("label").css("color", "blue");
  });
</script>
...
```

```

    });
</script>
...

```

使用jQuery处理这一任务只需要一行代码，与使用DOM API相比，这样更省事——我承认在这儿算不上多么巨大的改进，然而在复杂的Web应用中，小的改进将会不断累积。而且jQuery语句的可读性更好，更容易理解，这有助于我们对JavaScript代码的长期维护。

jQuery对象的另一个优雅之处在于它灵活聪明的API。不论什么时候调用一个方法修改jQuery内元素的内容，这个方法的返回结果都是一个jQuery对象。这听起来没有什么了不起的，然而正是这一设计使得绝大多数jQuery方法支持链式调用（代码清单5-22）。

#### 代码清单5-22 jQuery方法的链式调用

```

...
<script type="text/javascript">
    $(document).ready(function() {

        $("label").css("color", "blue").css("font-size", ".75em");

        var labelElems = document.getElementsByTagName("label");
        for (var i = 0; i < labelElems.length; i++) {
            labelElems[i].style.color = "blue";
            labelElems[i].style.fontSize = ".75em";
        }
    });
</script>
...

```

在这个例子中，我先使用\$函数生成一个jQuery对象，调用css方法设置color属性，然后再一次调用css方法，设置font-size属性。为便于比较，我又使用原生DOM API实现了同样的功能。由于我们已经写好了遍历选中元素的循环，使用原生API达成同样目标也没有花费太多气力。

只有当我们使用链式调用对jQuery对象内含元素作出更多实质的改变时，才开始体会到jQuery灵活的API会带来多么大的便利。代码清单5-23就是这样一个例子。

#### 代码清单5-23 一个较复杂的链式调用示例

```

...
<script type="text/javascript">
    $(document).ready(function() {

        $("label").css("color", "blue").add("input[name='rose']")
            .filter("[for!='snowdrop']").css("font-size", ".75em");

        var elems = document.getElementsByTagName("label");
        for (var i = 0; i < elems.length; i++) {
            elems[i].style.color = "blue";
            if (elems[i].getAttribute("for") != "snowdrop") {
                elems[i].style.fontSize = ".75em";
            }
        }
        elems = document.getElementsByTagName("input");
    });
</script>
...

```

```
        for (var i = 0; i < elems.length; i++) {  
            if (elems[i].getAttribute("name") != "rose") {  
                elems[i].style.fontSize = ".75em";  
            }  
        }  
    });  
</script>  
...
```

这个例子有点夸张，但它演示了jQuery的灵活性。下面我们剖析链式调用的这些方法，搞清背后发生的事情。我们从下面这条语句开始：

```
...  
$("label").css("color", "blue")  
...
```

这行代码选中了页面中所有的label元素，然后把这些元素的color属性设置为blue。现在看下一步：

```
...  
$("label").css("color", "blue").add("input[name!='rose']")  
...
```

add方法能把匹配其参数选择器的元素添加到原来的jQuery对象。在本例中，我们选择了所有name属性不等于rose的input元素。如此一来，jQuery对象中既包含了第一步匹配的label元素，又包含了第二步操作匹配的input元素。我们在第6章详细介绍add方法。接下来看下一步：

```
...  
$("label").css("color", "blue").add("input[name!='rose']").filter("[for!='snowdrop']")  
...
```

filter方法滤除了jQuery对象中所有的不满足过滤条件的元素。在第6章我会详细讲解filter方法，此刻你只需要知道它能帮我们删除jQuery对象中所有for属性等于snowdrop的元素。

```
...  
$("label").css("color", "blue").add("input[name!='rose']")  
    .filter("[for!='snowdrop']").css("font-size", ".75em");  
...
```

最后一步，我们又一次调用了css方法，这次把font-size属性设置为.75em。这条语句最终对页面造成以下影响：

- (1) 把所有label元素的color属性都设置为blue；
- (2) 把除for属性等于snowdrop之外的所有label元素的font-size属性设置为.75em；
- (3) 把除name属性等于rose之外的所有input元素的font-size属性设置为.75em。

使用原生DOM API实现同样的效果是一个相当复杂的任务，在编写这个脚本时我遇到了不少困难。比方说，我本想用第2章讲过的document.querySelectorAll方法使用input[name!='rose']选出所有的input元素，这个方法却不支持这种属性过滤器。为避免重复编写设置font-size值的代码，我试图合并两个getElementsByTagName的结果，结果合并本身又是非常痛苦的体验。我不想在这里唠叨太多，特别是既然你正在阅读本书，就一定对jQuery有所期待，jQuery确实提供了原生DOM API不可及的灵活性和表达能力。

## 5.8 事件处理

现在回到本章最初的脚本，即代码清单5-24。你会发现我连续调用两个方法，形成了一个方法链（见代码清单5-24中的加粗部分）。

代码清单5-24 示例脚本中的方法链式调用

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("img:odd").mouseenter(function(e) {
            $(this).css("opacity", 0.5);
        }).mouseout(function(e) {
            $(this).css("opacity", 1.0);
        });
    });
</script>
...
```

在上面的脚本中，我链式调用了`mouseenter`和`mouseout`方法。这两个方法分别为`mouseenter`和`mouseout`事件定义了事件处理函数（参见第2章）。我会在第9章详细讲解jQuery事件，在这儿只是演示一下如何利用jQuery对象的这种行为，为选中的全部元素指定同一个事件处理函数。

## 5.9 小结

本章我们一起完成了第一个jQuery脚本，并利用这个脚本了解了jQuery库的几个关键特征：`$`函数、`ready`事件、jQuery对象，以及jQuery是如何增强而不是替换原生DOM API（它是HTML标准的一部分）的。



大多数时候，我们以相当清晰的“两步”模式使用jQuery。第一步是选取元素，第二步是对选中的元素执行某些操作。本章我们重点研究第一步，也就是学习如何控制jQuery结果集并“裁剪”它以满足需要。我们还会学习如何使用jQuery遍历DOM。不论是哪一种情况，我们总是先选择一些元素，然后做一些处理，直到结果集恰好满足需要为止。事实上，根据需要，开始选取的元素与最终需要的元素之间的关系可能简单，也可能很复杂。表6-1列出了本章概要。

表6-1 本章概要

问 题	解决方法	代码清单
如何选取更多元素	使用add方法	1
如何从结果集中得到某一个元素	使用first、last或者eq方法	2
如何从结果集中截取一个子集	使用slice方法	3
如何过滤结果集从而得到一个子集	使用filter或not方法	4、5
如何根据选中元素是否拥有后代元素过滤结果集	使用has方法	6
如何在一个结果集中再进行一次选择	使用map方法	7
如何判断结果集中至少有一个元素符合指定条件	使用is方法	8
如何得到上次选择的结果	使用end方法	9
如何得到上次结果集与本次结果集的合集	使用addBack方法	10
如何得到选中元素的子元素或者后代元素	使用children方法和find方法	11~13
如何得到选中元素的父元素	使用parent方法	14
如何得到选中元素的一系列祖先元素	使用parents方法	15
如何得到某一个祖先元素	使用parentsUntil方法	16、17
如何得到离自己最近的匹配选择器参数或者具体元素的祖先元素	使用closest方法	18、19
如何得到离自己最近的定位祖先元素	使用offsetParent方法	20
如何得到选中元素的兄弟元素	使用siblings方法	21、22
如何得到当前元素之前或之后的兄弟元素	使用next、prev、nextAll、prevAll、nextUntil或prevUntil方法	23

## 新版jQuery中与本章有关的变化

jQuery 1.9中新增了addBack方法，并用它替掉了jQuery 1.8中的addSelf方法。这两个方法功能完全相同，具体例子参见6.5节。

## 6.1 选择更多元素

add方法允许我们往jQuery对象中添加更多元素。表6-2列出了add方法支持的参数。

表6-2 add方法支持的参数类型

参 数	描 述
add(selector), add(selector, context)	把匹配选择器（可选上下文对象）的所有元素添加到调用add方法的jQuery对象
add(HTMLElement) add(HTMLElement[])	把一个或一组HTMLElement添加到jQuery对象
add(jQuery)	把参数jQuery对象中的元素添加到当前jQuery对象

和许多其他jQuery方法一样，add方法也返回jQuery对象，这样我们就能接着调用其他jQuery方法，包括add()方法。代码清单6-1演示了如何使用add方法往jQuery对象中添加更多元素。

代码清单6-1 add方法示例

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script type="text/javascript">
    $(document).ready(function() {

      var labelElems = document.getElementsByTagName("label");
      var jq = $("img[src*=daffodil]");

      $("img:even").add("img[src*=primula]").add(jq)
        .add(labelElems).css("border", "thick double red");
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
        <div id="row1" class="drow">
          <div class="dcell">
            <label for="aster">Aster:</label>
            <input name="aster" value="0" required>
          </div>
          <div class="dcell">
```

```

        <label for="daffodil">Daffodil:</label>
        <input name="daffodil" value="0" required >
    </div>
    <div class="dcell">
        <label for="rose">Rose:</label>
        <input name="rose" value="0" required>
    </div>
</div>
<div id="row2" class="drow">
    <div class="dcell">
        <label for="peony">Peony:</label>
        <input name="peony" value="0" required>
    </div>
    <div class="dcell">
        <label for="primula">Primula:</label>
        <input name="primula" value="0" required>
    </div>
    <div class="dcell">
        <label for="snowdrop">Snowdrop:</label>
        <input name="snowdrop" value="0" required>
    </div>
</div>
</div>
<div id="buttonDiv"><button type="submit">Place Order</button></div>
</form>
</body>
</html>

```

**警告** 以为remove方法与add方法行为相反，因此用它来缩小选择范围，这是一个很常见的错误。事实上，remove方法会从页面中删除元素，从而改变DOM的结构（参见第7章）。你可以从6.2节中选择一个方法达到目的。

上面例子中的脚本先后使用了3种方式往初始jQuery对象中添加元素：使用了另一个选择器、一些HTMLElement对象和另一个jQuery对象。添加完元素之后，它接着调用了css方法设置选中元素的边框属性，如图6-1所示，为选中元素加上了又粗又红的边框。

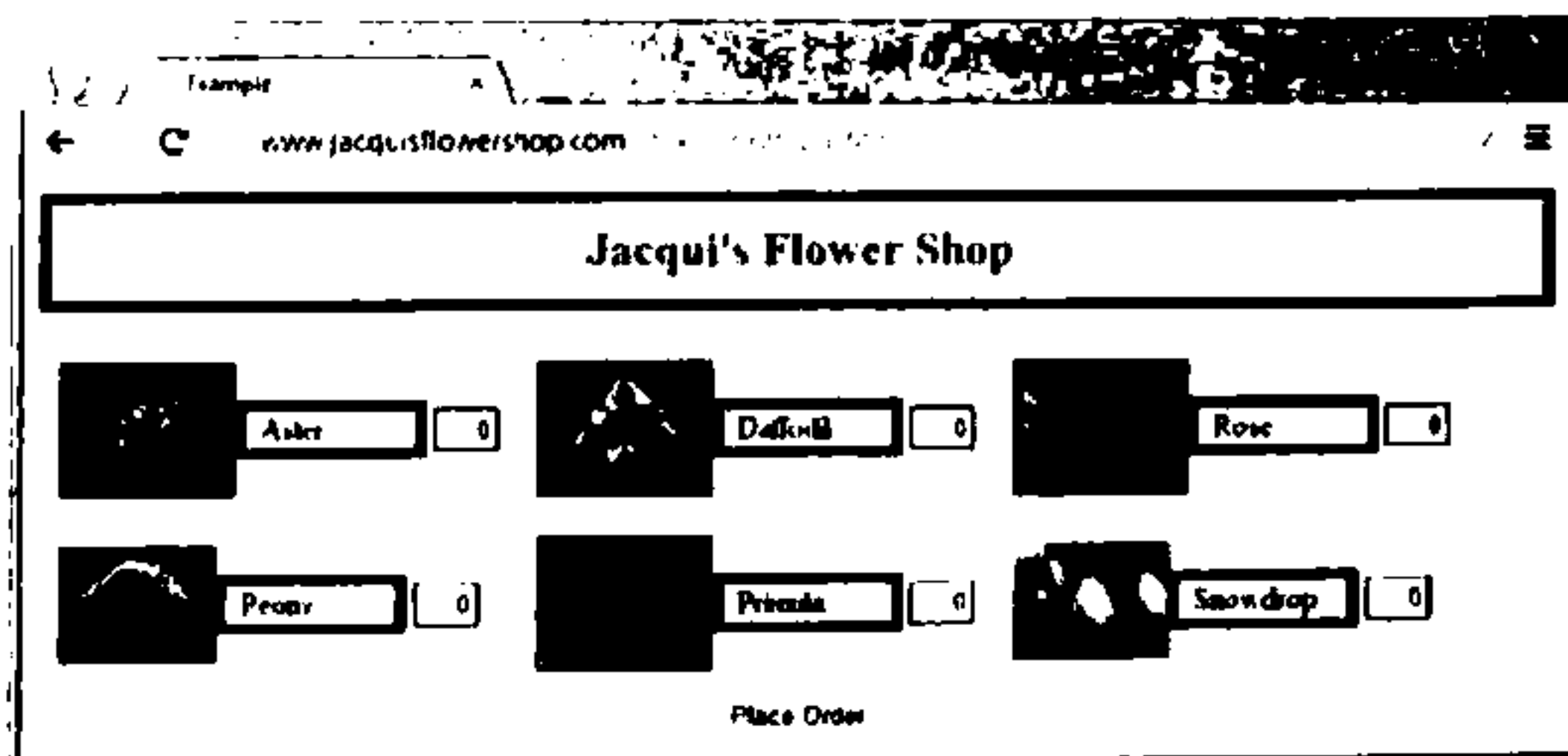


图6-1 使用add方法向jQuery对象中添加元素

## 6.2 限制选择范围

jQuery提供了一些方法帮助我们从结果集中删除元素，这些方法见表6-3。它们都返回包含着新结果集的jQuery对象，并不修改调用这些方法的当前jQuery对象。

表6-3 用来过滤元素的方法

方 法	描 述
eq(index)	得到第index个元素(删除其他元素)
filter(condition)	得到符合condition参数的元素(删除不符合条件的元素)。该方法condition参数的细节，请参考本章后面的具体讨论
first()	得到第一个元素(删除其他元素)
has(selector)、has(jQuery) has(HTMLElement)has(HTMLElement[])	得到匹配selector后代元素的元素，或者包含jQuery对象内的元素或HTMLElement对应元素的元素(删除没有匹配后代的元素)
last()	得到最后一个元素(删除其他元素)
not(condition)	删除匹配condition参数的元素，该方法condition参数的细节，请参考本章后面的具体讨论
slice(start, end)	得到start、end参数指定范围的元素子集，删除子集之外的其他元素

### 6.2.1 从结果集中获取某个元素

first、last和end方法是3个最基础的，用来得到某一元素的方法。我们可以使用这些方法基于这些元素在jQuery对象中的存放位置得到一个具体的元素。代码清单6-2演示了这些方法的用法。

代码清单6-2 基于元素的位置得到具体元素

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var jq = $("label");

        jq.first().css("border", "thick double red");
        jq.last().css("border", "thick double green");
        jq.eq(2).css("border", "thick double black");
        jq.eq(-2).css("border", "thick double black");

    });
</script>
...
```

注意，上例中我调用了eq方法两次。当参数为正值时，从jQuery对象中的第一个元素开始计数。当参数为负值时，则从jQuery对象中的最后一个元素开始计数。图6-2展示了这段脚本的效果。



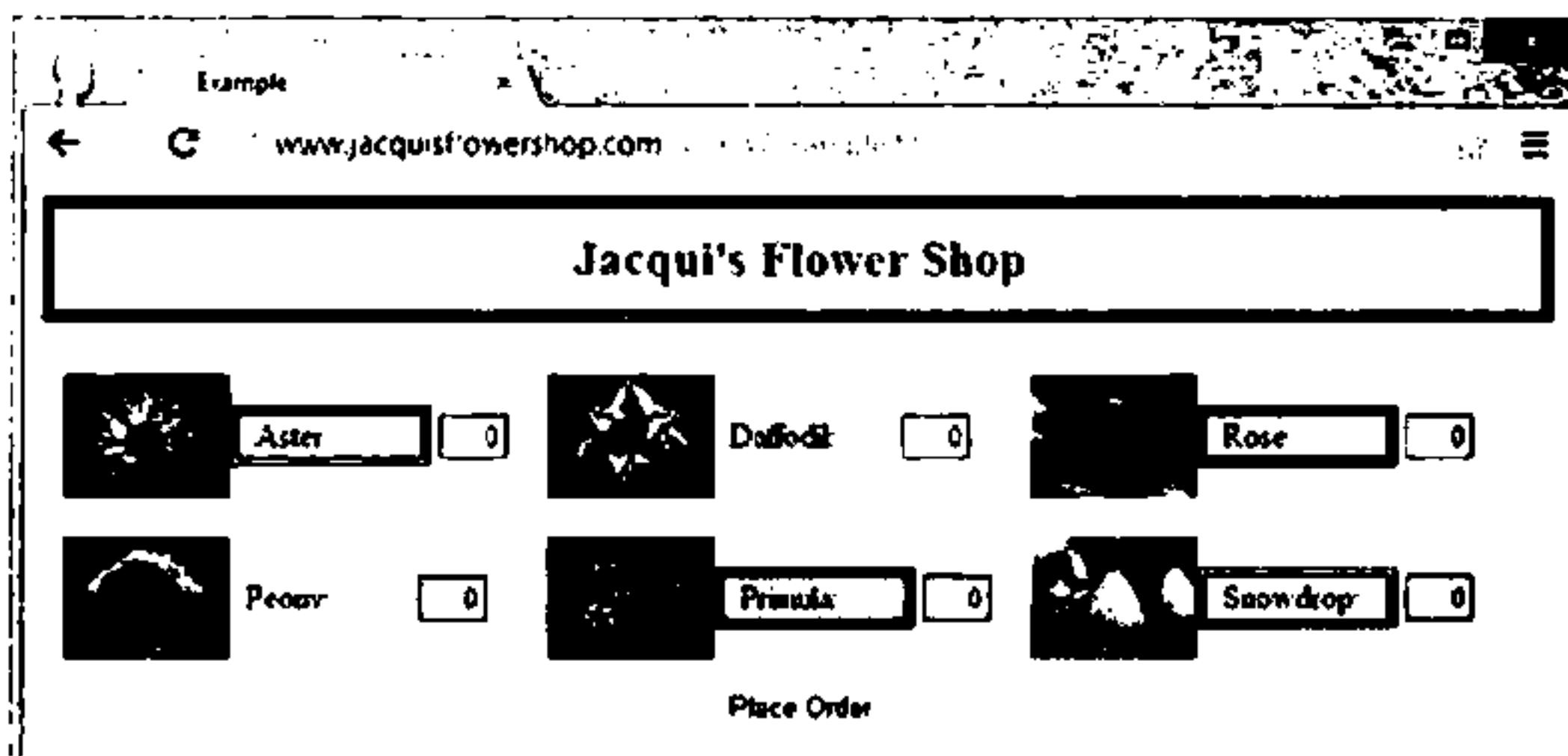


图6-2 删减结果集以得到某个具体元素

## 6.2.2 获取元素子集

slice方法用来获取特定的一组子元素，使用方法见代码清单6-3。

代码清单6-3 slice方法示例

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var jq = $("label");

        jq.slice(0, 2).css("border", "thick double black");
        jq.slice(4).css("border", "thick solid red");
    });
</script>
...
```

slice方法支持两个参数，分别是选择的起始索引号和结束索引号。索引从0开始计数，因此在上例中我使用的参数（0和2）表示选择前两个元素。如果我们省略第二个参数，就意味着子集从起点开始，一直延续到原始元素集合的末尾。在包含6个元素的jQuery对象中，只提供一个参数4调用slice方法，我们会得到最后两个元素（索引号分别为4和5）。图6-3展示了上面脚本的显示结果。

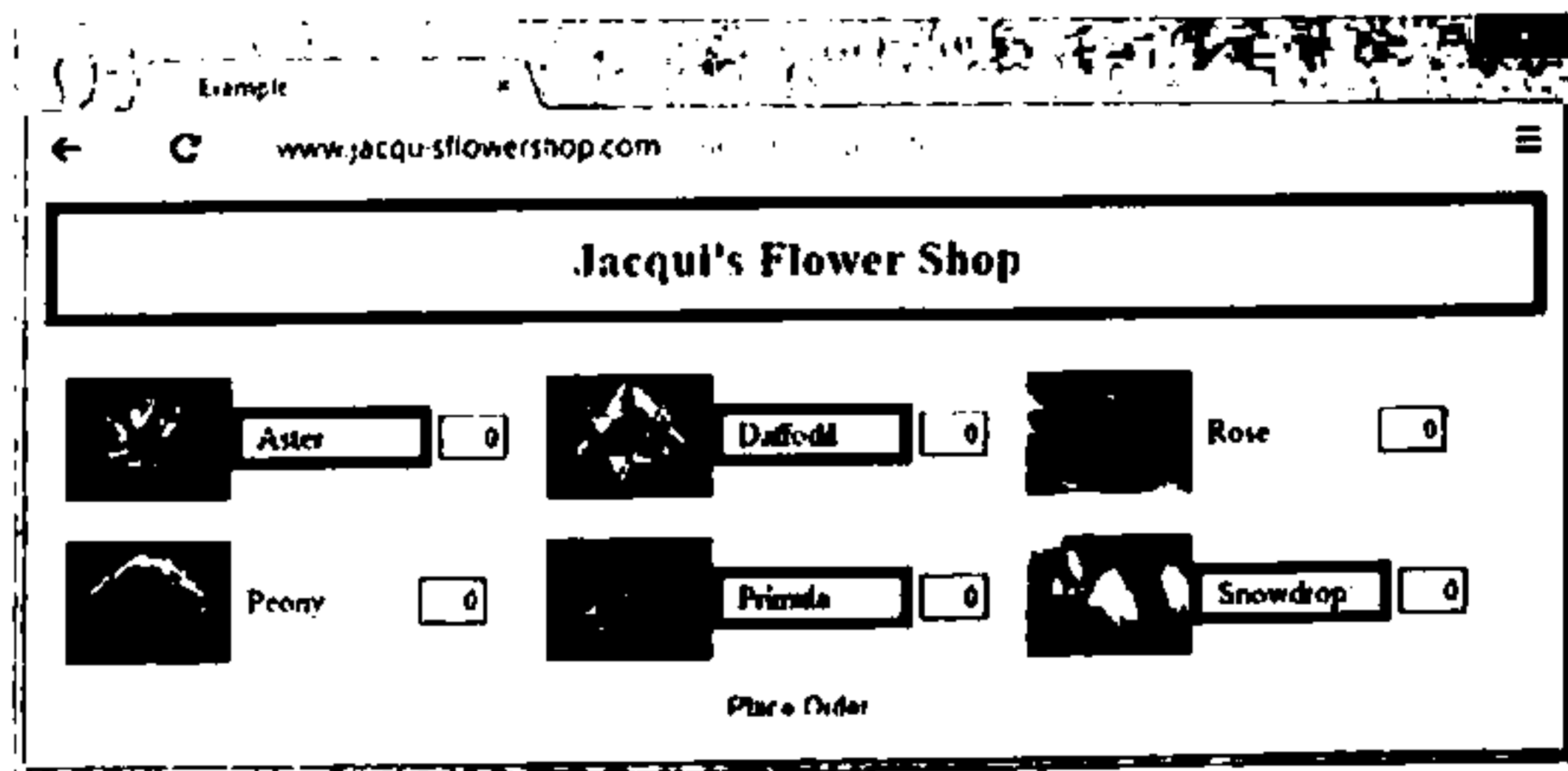


图6-3 获取元素集合的子集

### 6.2.3 过滤元素

使用filter方法可以将所有不满足某个指定条件的元素剔除。表6-4列出了filter方法支持的各种表示过滤条件的参数。

表6-4 filter方法的参数

参 数	描 述
filter(selector)	得到匹配选择器的元素（删除不匹配的元素）
filter(HTMLElement)	得到参数对应的元素（删除其他元素）
filter(jQuery)	得到原始集合与参数jQuery对象所含元素集合的交集
filter(function(index))	每个元素调用一次参数函数，若某个元素调用参数函数的返回结果为false，就删除该元素

代码清单6-4展示了指定过滤器的4种方式。

#### 代码清单6-4 指定过滤器

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("img").filter("[src*=s]").css("border", "thick double red");

        var jq = $("[for*=p]");
        $("label").filter(jq).css("color", "blue");

        var elem = document.getElementsByTagName("label")[1];
        $("label").filter(elem).css("font-size", "1.5em");

        $("img").filter(function(index) {
            return this.getAttribute("src") == "peony.png" || index == 4;
        }).css("border", "thick solid red");
    });
</script>
...
```

前三种方法一目了然，无非是基于选择器、HTMLElement或者另一个jQuery对象来过滤元素。

第四种方法使用了一个函数，值得我们多花些时间深入了解。jQuery取出jQuery对象中包含的每一个元素，运行参数函数：如果参数函数返回true，就会保留这个元素；如果返回false，就删除这个元素。

jQuery会把元素的索引号作为参数传递给这个函数。除此之外，this变量还被设置为需要处理的对应当前元素的HTMLElement对象。在这个例子中，如果元素的索引号等于4或者src属性为“peony.png”，函数就返回true，如图6-4所示。

**提示** 你或许会奇怪：为什么不直接使用src属性获取值，而非要用getAttribute方法获取src属性？

这是因为getAttribute方法会忠实地返回我在页面中设置的src值（一个相对URL），而src属性却会返回一个绝对URL。对于这个例子来说，相对URL更简单好用。

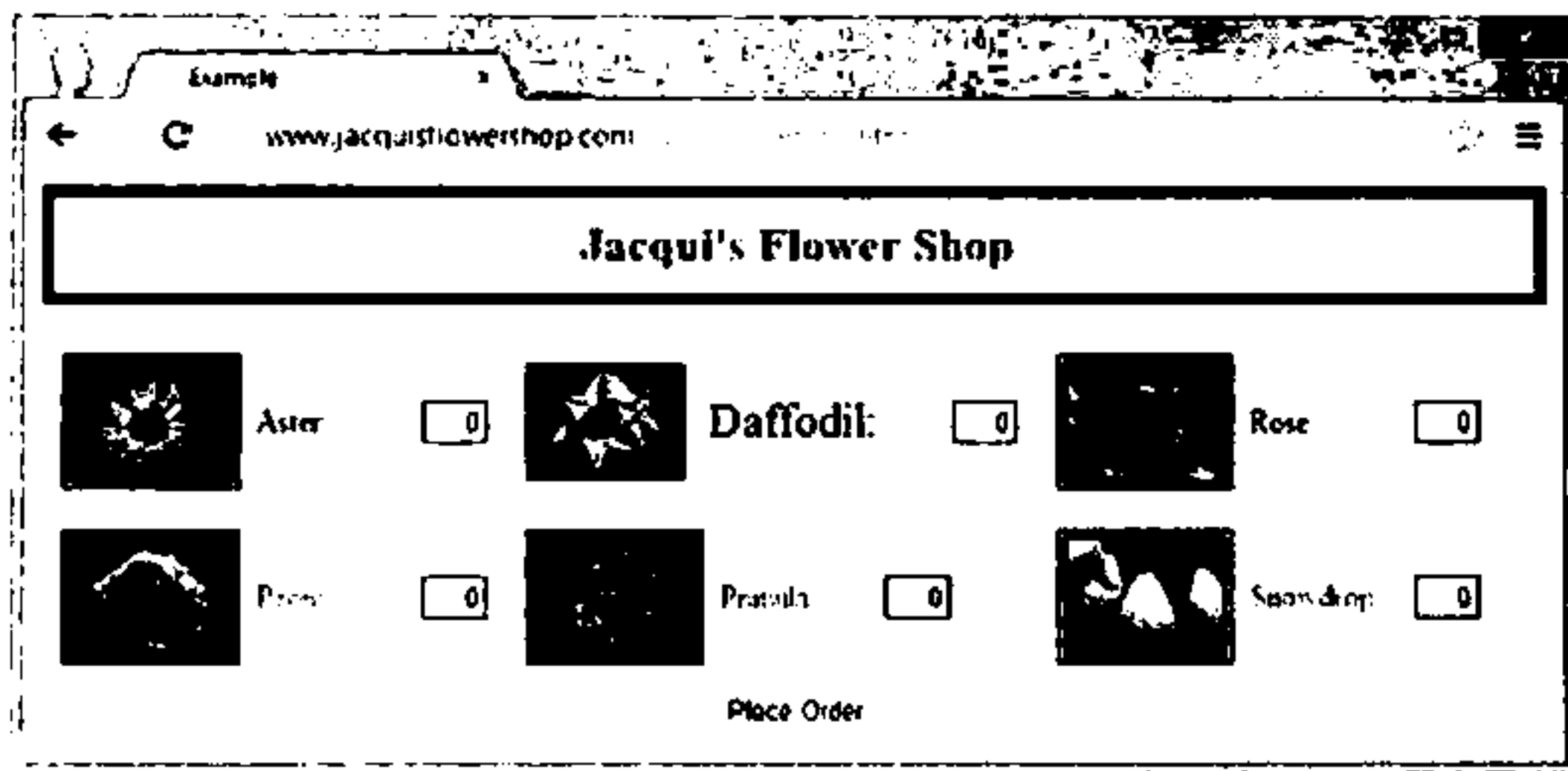


图6-4 filter方法

not方法是filter方法的有效补充,在很大程度上它的工作方式与filter正好相反。表6-5列出了not方法支持的参数。

表6-5 not方法的参数

参 数	描 述
not(selector)	删除匹配选择器的元素
not(HTMLElement[]), not(HTMLElement)	删除指定的元素或者元素集合
not(jQuery)	删除原始集合与参数jQuery对象所含元素集合的交集
not(function(index))	每个元素调用一次参数函数,若某个元素调用参数函数的返回结果为true,就删除该元素

代码清单6-5在上一个例子的基础上展示了not方法的用法。

代码清单6-5 not方法示例

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("img").not("[src*=s]").css("border", "thick double red");

        var jq = $("[for*=p]");
        $("label").not(jq).css("color", "blue");

        var elem = document.getElementsByTagName("label")[1];
        $("label").not(elem).css("font-size", "1.5em");

        $("img").not(function(index) {
            return this.getAttribute("src") == "peony.png" || index == 4;
        }).css("border", "thick solid red");
    });
</script>
...
```

这个脚本的效果见图6-5。显而易见,这个例子的效果与前一个例子正好相反。

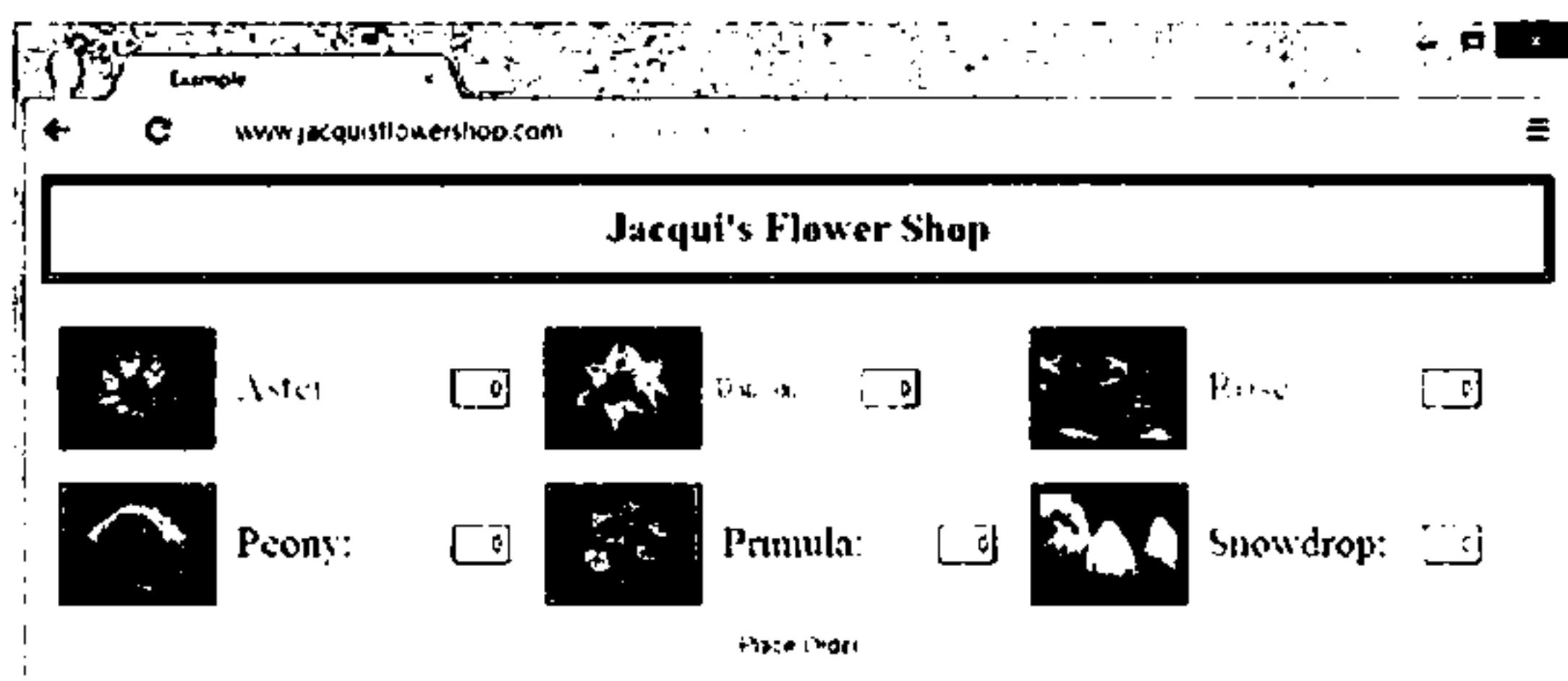


图6-5 使用not方法过滤元素

### 6.2.4 基于后代元素过滤结果集

我们可以使用has方法指定一个选择器、一个或多个HTMLElement对象为参数，从而得到拥有指定后代的元素。代码清单6-6展示了has方法的用法。

代码清单6-6 使用has方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("div.dcell").has("img[src*=aster]").css("border", "thick solid red");
        var jq = $("[for*=p]");
        $("div.dcell").has(jq).css("border", "thick solid blue");

    });
</script>
...
```

在第一个案例里我使用了一个选择器过滤结果集，得到结果元素至少有一个后代img元素且src属性值中有aster字符串。在第二个案例中，我使用一个jQuery对象代替第一个案例中的选择器，得到至少有一个后代元素具有for属性，且for属性值中必须有字母p的结果元素。这段脚本的运行结果见图6-6。

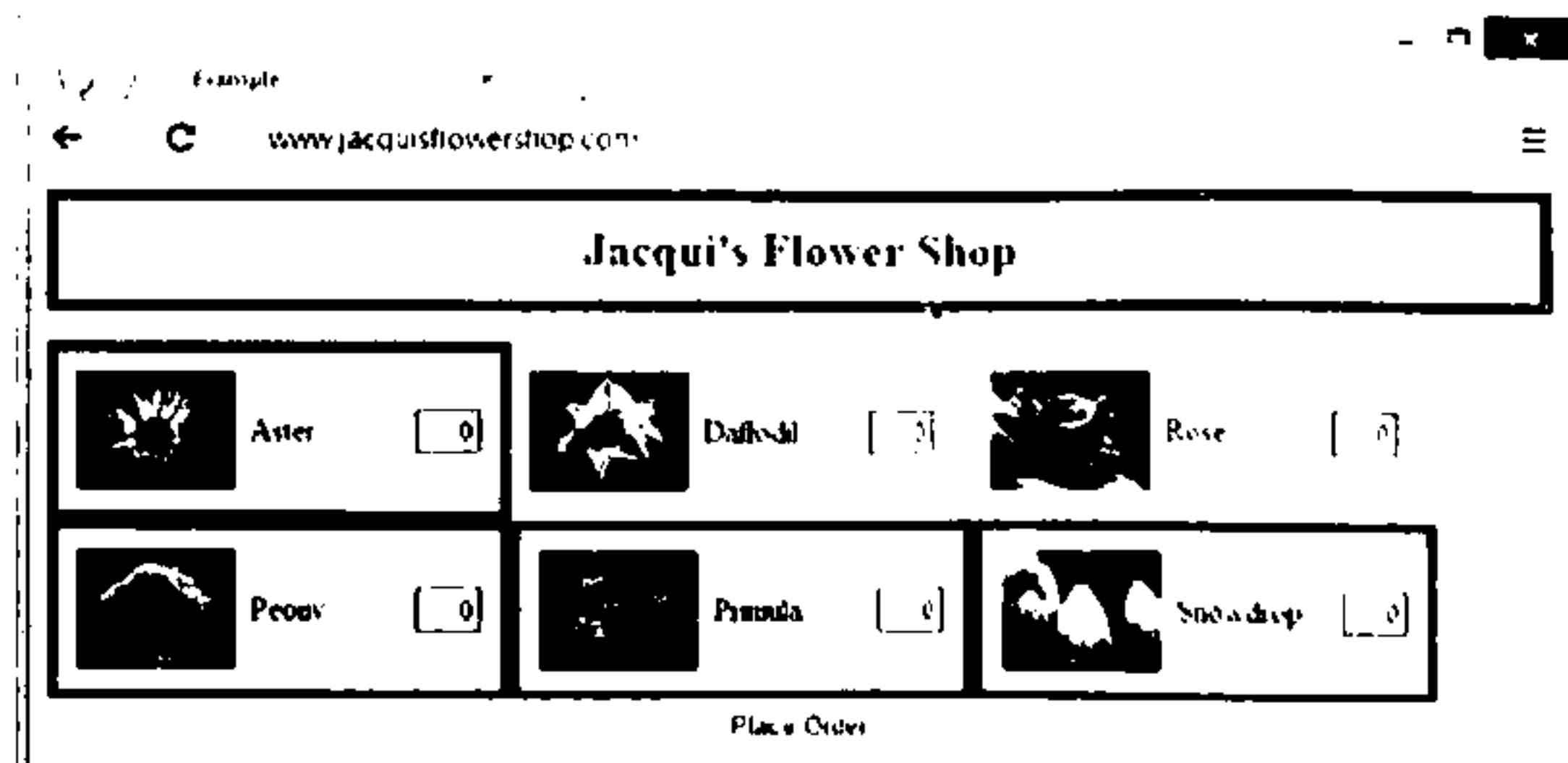


图6-6 使用has方法过滤结果集

## 6.3 以映射方式处理结果集

以一个函数为参数，map方法能帮助我们灵活处理一个jQuery对象，从而得到满足需要的另一个jQuery对象。针对源jQuery对象中的每一个元素都调用一次这个函数，而函数返回的HTMLElement对象将包含在结果jQuery对象中，组成新的结果集（参见代码清单6-7）。

代码清单6-7 map方法的用法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("div.dcell").map(function(index, elem) {
            return elem.getElementsByTagName("img")[0];
        }).css("border", "thick solid red");

        $("div.dcell").map(function(index, elem) {
            return $(elem).children()[1];
        }).css("border", "thick solid blue");

    });
</script>
...
```

在这段脚本中，我先后调用了两次map方法。第一次调用使用DOM API返回当前元素中的第一个img元素；第二次调用使用了jQuery，返回当前元素children方法返回的结果集中的第二个元素。（本章后面会仔细讲解children方法；顾名思义，它返回当前jQuery对象中每个元素的子元素。）结果如图6-7所示。

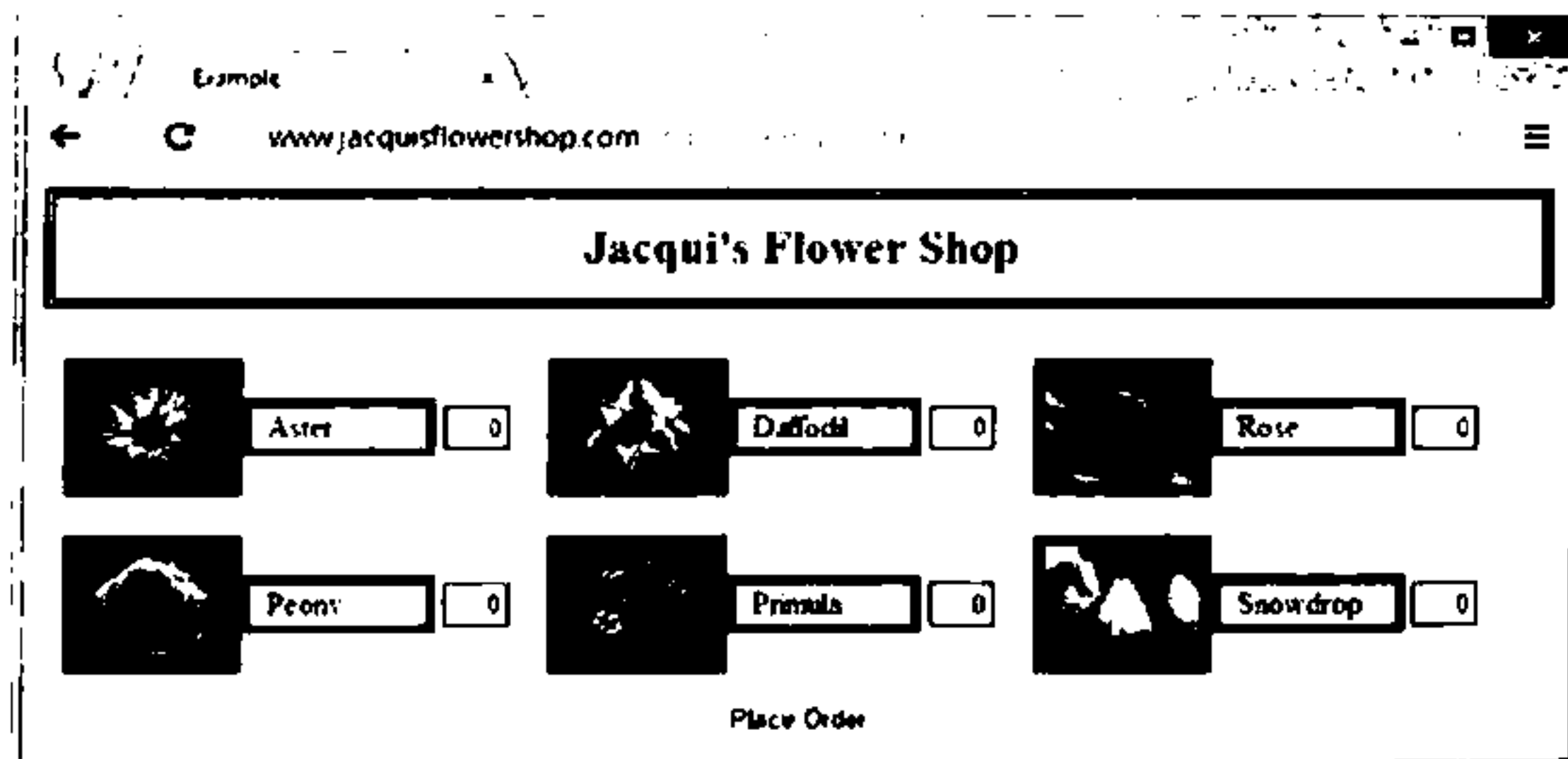


图6-7 使用map方法

**提示** 这个函数每次只能返回一个元素。如果希望从每一个源元素中得到多个目标元素，请参考第8章，综合使用each方法和add方法达成目标。

## 6.4 检测结果集

我们可以用is方法确定jQuery对象中的某个或某些元素是否满足测试条件。表6-6列出了is方法支持的参数。

表6-6 is方法的参数

参 数	描 述
is(selector)	如果结果集中至少有一个元素匹配选择器selector, 返回true
is(HTMLElement[]), is(HTMLElement)	如果结果集中包含指定的HTMLElement或者至少包含指定HTMLElement[]中的一个元素, 返回true
is(jQuery)	如果结果集中至少包含一个参数对象中的元素, 返回true
is(function(index))	如果至少有一次参数函数返回true, 则返回true

当我们指定一个函数为is方法的参数时, jQuery会针对结果集中的每个元素调用一次这个函数。jQuery会把当前元素在结果集中的序号(从0开始)传递给这个函数, 并把该函数中的this变量设置为当前元素。代码清单6-8演示了is方法的用法。

**注意** 这个方法返回布尔值(而不是jQuery对象)。第5章已经说过, 并不是所有的jQuery方法都返回jQuery对象。

### 代码清单6-8 is方法的用法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var isResult = $("img").is(function(index) {
            return this.getAttribute("src") == "rose.png";
        });
        console.log("Result: " + isResult);

    });
</script>
...
```

这段脚本检测jQuery结果集, 查看是否有一个元素的src属性值为rose.png, 最后把结果输出到控制台。结果如下:

```
Result: true
```

## 6.5 修改、回退结果集

当我们调用方法链来修改结果集的时候，jQuery维护着一个历史结果集栈。如表6-7所示，我们可以使用jQuery内建的两个方法利用这个栈。

表6-7 展开结果集栈的方法

方 法	描 述
end()	扔掉当前结果集，返回jQuery对象中缓存着的上一个结果集
addBack()、addBack(selector)	得到原结果集与当前选择结果的合集，支持可选的选择器参数，利用这个选择器参数可过滤原结果集

我们可以利用end方法得到上一个结果集。这样我们就能够做到选择一些元素、增加或者缩小选择范围、执行一些操作，然后再回到最初的选择结果。代码清单6-9演示了end方法的使用。

代码清单6-9 使用end方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("label").first().css("border", "thick solid blue")
        .end().css("font-size", "1.5em");

    });
</script>
...
```

在这个脚本中，我先选中页面中所有的label元素，然后调用first方法得到结果集中的第一个元素，使用css方法把它的样式设置为蓝色边框，即只改变了第一个选中元素的CSS属性。然后，我调用end方法回到上一个结果集（最初的结果集，全部label元素），接着调用css方法把所有label元素的字号设置为1.5em。如图6-8所示，所有label元素的CSS属性都发生了变化。

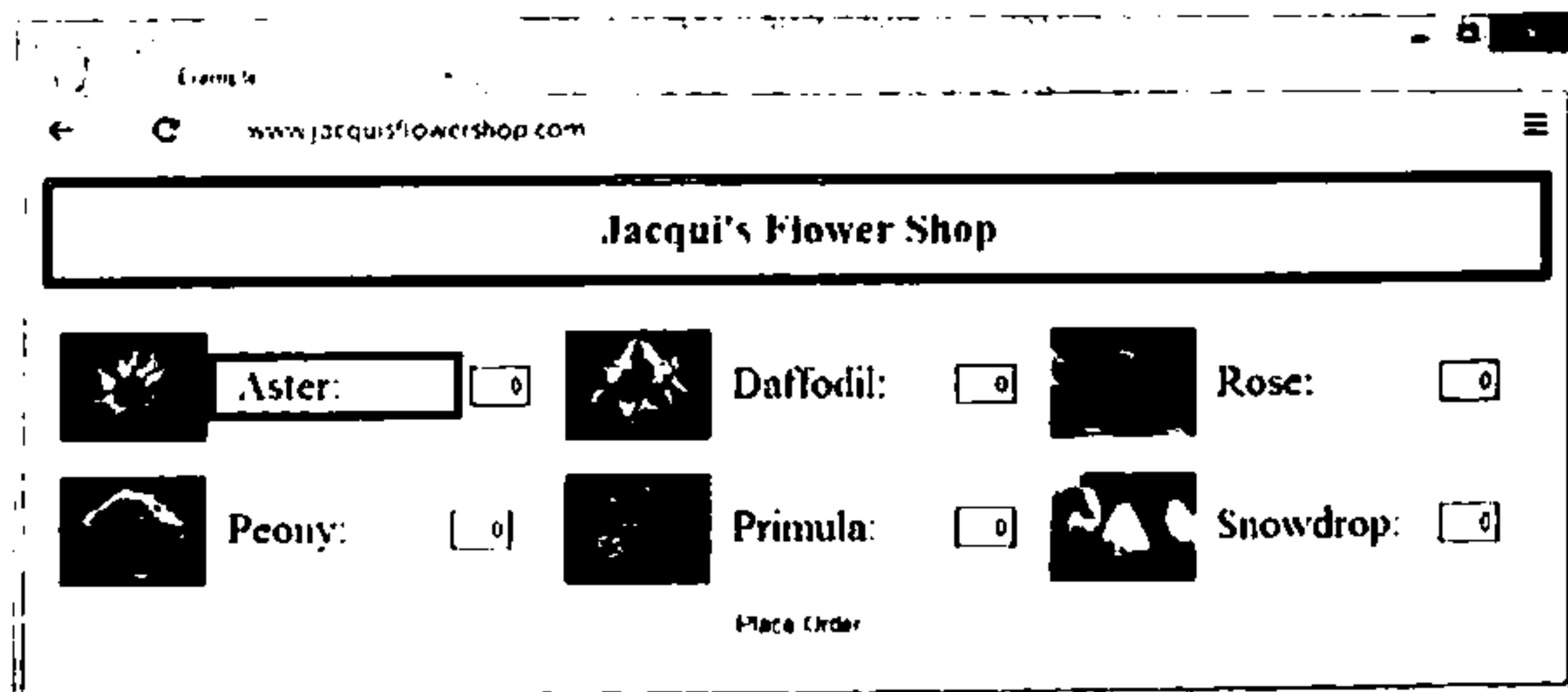


图6-8 使用end方法

addBack方法返回当前结果集与上一个结果集的合集。代码清单6-10演示了addBack方法的使用。

## 代码清单6-10 使用addBack

```
...  
<script type="text/javascript">  
    $(document).ready(function() {  
        $("div.dcell").children("img").addBack().css("border", "thick solid blue");  
    });  
</script>  
...
```

**注意** 在jQuery 1.9/2.0中，使用addBack方法代替了原有的addSelf方法。新方法的功能与addSelf方法功能相同，并且支持一个可选的选择器参数，用来过滤原结果集。

在这个例子中，我先选中所有具有dcell类的div元素，使用children方法得到这些div元素所有的img子元素（6.6节详细讲解了children方法）。接着，我调用了addBack方法，把上一个结果集（那些div元素）添加到当前集合（img元素）当中，得到一个新的结果集。最后，我使用css方法为新结果集里的所有元素设置了边框样式。这段脚本的运行结果见图6-9。

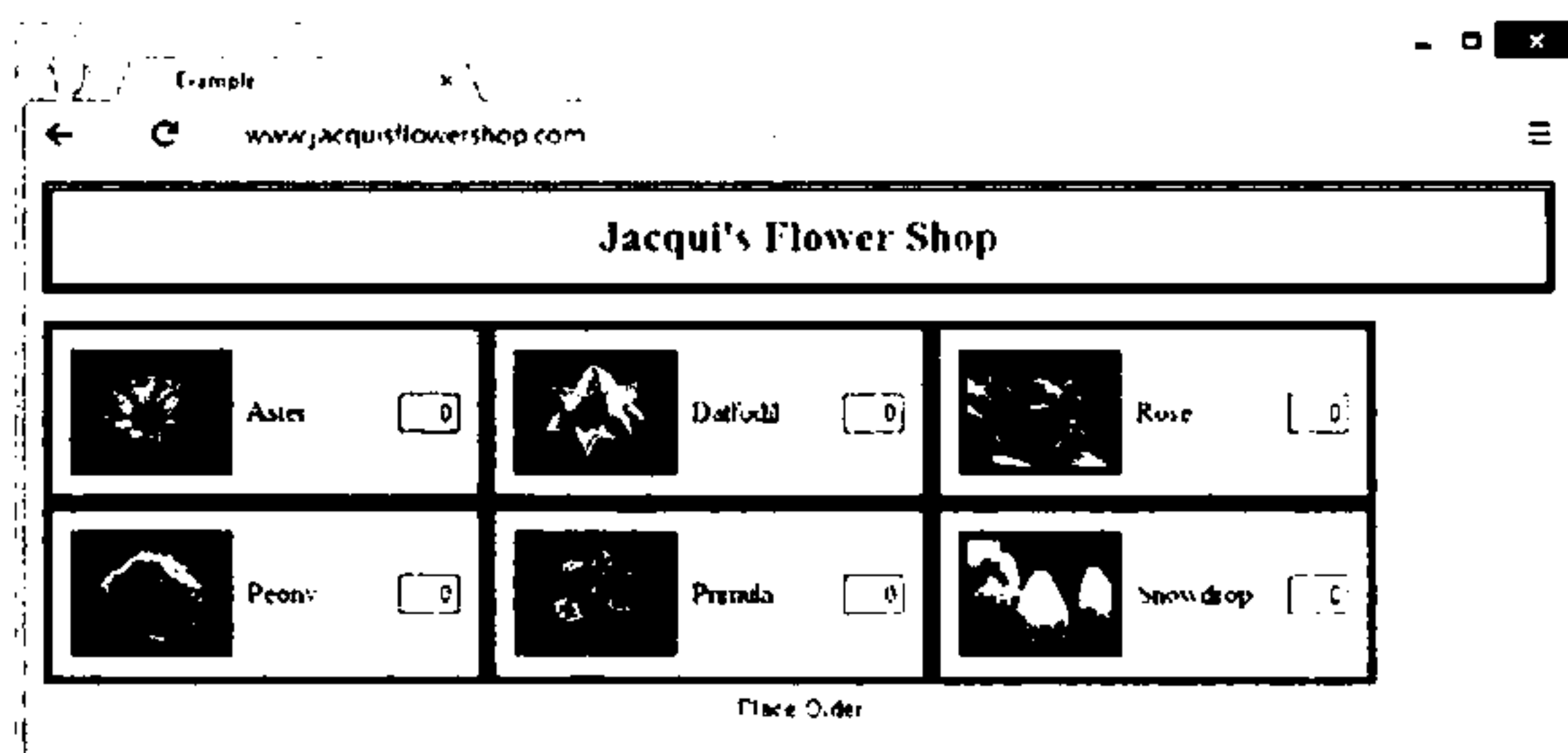


图6-9 addBack方法的用法

## 6.6 访问 DOM

以一个结果集为起点，我们可以在DOM中访问其他部分，使用一个结果集得到另一个结果集。在接下来的内容中，我会讲解并演示jQuery访问DOM的方法。第2章已经讲解过页面上各个元素之间的种种关系。

**提示** 以下各节中讲到的所有方法都返回jQuery对象。如果有匹配结果，返回的jQuery对象中就包含着匹配元素；如果没有匹配，就返回一个空的jQuery对象（length属性等于0）。



### 6.6.1 访问后代元素

所谓访问后代元素，就是在一个jQuery对象内访问结果集内元素的后代元素。表6-8列出了有关的jQuery方法。

表6-8 jQuery中用来访问后代元素的方法

方 法	描 述
children()	得到结果集内所有元素的子元素
children(selector)	得到结果集内所有元素的匹配selector选择器的子元素
contents()	得到结果集内所有元素的子元素和文本内容
find()	得到结果集内所有元素的后代元素
find(selector)	得到结果集内所有元素的匹配selector选择器的后代元素
find(jQuery)、find(HTMLElement)、find(HTMLElement[])	得到结果集内所有元素的子元素与参数对象对应元素的交集

children方法用来得到结果集元素的直接子元素，它支持一个可选的用来过滤结果的选择器参数。find方法用来得到所有后代元素，不仅仅是子元素。contents方法不但返回直接子元素节点，还返回文本节点<sup>①</sup>。代码清单6-11演示了children和find方法的用法。

代码清单6-11 使用children和find方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var childCount = $("div.drow").children().each(function(index, elem) {
            console.log("Child: " + elem.tagName + " " + elem.className);
        }).length;
        console.log("There are " + childCount + " children");

        var descCount = $("div.drow").find("img").each(function(index, elem) {
            console.log("Descendant: " + elem.tagName + " " + elem.src);
        }).length;
        console.log("There are " + descCount + " img descendants");

    });
</script>
...
```

这个例子在一个语句中使用了不带参数的children方法，而在另一个语句里使用了带参数的find方法。这个脚本在控制台产生以下输出。

```
Child: DIV dcell
Child: DIV dcell
Child: DIV dcell
Child: DIV dcell
```

① 还包括注释节点。——译者注

Child: DIV dcell

Child: DIV dcell

一共6个子元素

Descendant: IMG http://www.jacquisflowershop.com/jquery/aster.png

Descendant: IMG http://www.jacquisflowershop.com/jquery/daffodil.png

Descendant: IMG http://www.jacquisflowershop.com/jquery/rose.png

Descendant: IMG http://www.jacquisflowershop.com/jquery/peony.png

Descendant: IMG http://www.jacquisflowershop.com/jquery/primula.png

Descendant: IMG http://www.jacquisflowershop.com/jquery/snowdrop.png

一共6个img后代元素

children和find方法最大的优点是返回的结果中没有重复元素。代码清单6-12展示了这一优点。

#### 代码清单6-12 生成含有重复元素的结果集

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("div.drow").add("div.dcell").find("img").each(function(index, elem) {
            console.log("Element: " + elem.tagName + " " + elem.src);
        });
    });
</script>
...
```

这个例子先生成一个包含所有div.drow元素和div.dcell元素的jQuery对象。需要注意的是，每个div.dcell元素都包含在某个div.drow元素之内。由于img元素既是.drow元素的后代元素又是.dcell元素的后代元素，因此使用find元素查找img后代元素时会遭遇重复元素问题。不过jQuery及时“拯救”了我们，find方法返回的结果中不会出现重复元素。这个例子的输出如下。

```
Element: IMG http://www.jacquisflowershop.com/jquery/aster.png
Element: IMG http://www.jacquisflowershop.com/jquery/daffodil.png
Element: IMG http://www.jacquisflowershop.com/jquery/rose.png
Element: IMG http://www.jacquisflowershop.com/jquery/peony.png
Element: IMG http://www.jacquisflowershop.com/jquery/primula.png
Element: IMG http://www.jacquisflowershop.com/jquery/snowdrop.png
```

#### 使用find方法得到元素的交集

我们也可以传递一个jQuery对象，或者一个HTMLElement对象，或者一个由HTMLElement对象组成的数组作为find方法的参数。当使用这类参数时，我们就会得到源jQuery对象包含元素的后代元素与参数对象对应元素的交集。代码清单6-13演示了这种用法。

#### 代码清单6-13 使用find方法得到一个交集

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var jq = $("label").filter("[for*=p]").not("[for=peony]");
```

```
        $("div.drow").find(jq).css("border", "thick solid blue");
    });
</script>
...

```

正如这个脚本演示的那样，这种方法的优点在于我们能够很明确地指定需要与后代元素取交集的元素。我们先生成一个基础jQuery对象，然后顺序使用filter和not方法过滤元素，最后把它作为参数传递给包含了所有div.drow元素的jQuery对象的find方法。最后的结果集是div.drow元素的后代元素与参数对象jq包含元素的交集。这个脚本的最终效果见图6-10。

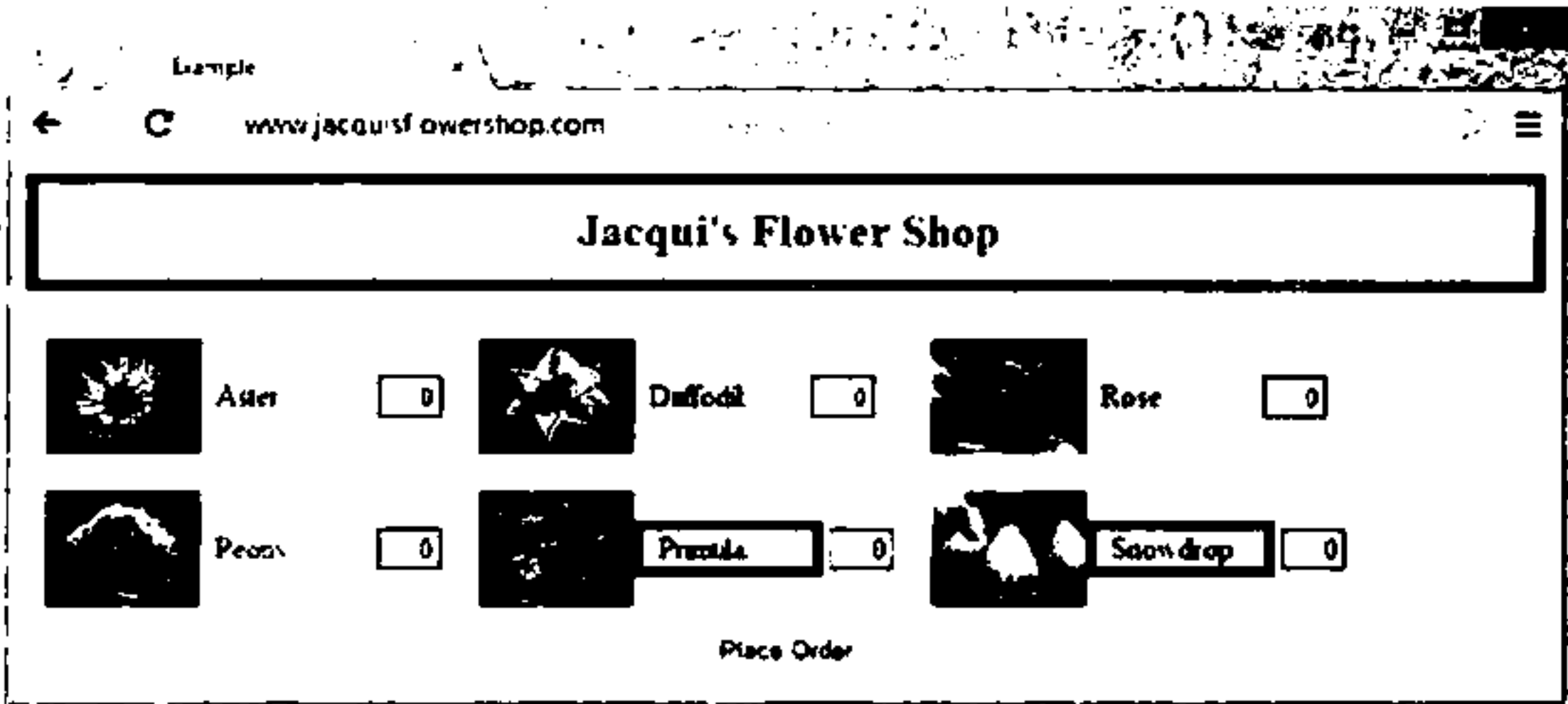


图6-10 使用find方法得到一个交集

6.6.2 访问祖先元素

所谓访问祖先元素，就是指访问当前jQuery对象包含元素的父元素和祖先元素。表6-9列出了jQuery中可用于访问祖先元素的方法。

表6-9 访问祖先元素的方法

方 法	描 述
closest(selector)、closest(selector, context)	得到结果集内元素的祖先元素中匹配selector选择器的最接近的那个祖先元素
closest(jQuery)、closest(HTMLElement)	得到结果集内元素的祖先元素与参数元素的交集
offsetParent()	得到距离最近的祖先定位元素（使用fixed、absolute或relative定位的元素）
parent()、parent(selector)	得到结果集内元素的父元素（可选匹配selector选择器的元素）
parents()、parents(selector)	得到结果集内元素的父元素（可选匹配selector选择器的元素）
parentsUntil(selector) parentsUntil (selector, selector)	得到结果集内元素的匹配selector选择器的祖先元素，可选的第二个选择器参数用来过滤选择结果
parentsUntil(HTMLElement) parentsUntil(HTMLElement, selector) parentsUntil(HTMLElement[]) parentsUntil(HTMLElement[], selector)	得到结果集内元素的祖先元素与参数元素的交集，可选的第二个选择器参数用来过滤选择结果

### 1. 得到父元素

parent方法用来选取父元素，该方法支持一个可选的选择器参数，用来过滤选择结果。代码清单6-14演示了parent方法的用法。

代码清单6-14 使用parent方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("div.dcell").parent().each(function(index, elem) {
            console.log("Element: " + elem.tagName + " " + elem.id);
        });

        $("div.dcell").parent("#row1").each(function(index, elem) {
            console.log("Filtered Element: " + elem.tagName + " " + elem.id);
        });

    });
</script>
...
```

在这段脚本中，我先选取所有的div.dcell元素，然后调用parent方法选择父元素；这里还演示了使用选择器参数的parent方法。我使用each方法把结果集中元素的一些信息输出到控制台：

---

```
Element: DIV row1
Element: DIV row2
Filtered Element: DIV row1
```

---

### 2. 选取祖先元素

parents方法（注意不是parent）用来选取所有祖先元素，而不仅仅是父元素。同样，我们可以使用可选的选择器参数过滤选择结果。代码清单6-15演示了parents方法的用法。

代码清单6-15 使用parents方法

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("img[src*=peony], img[src*=rose]").parents().each(function(index, elem) {
            console.log("Element: " + elem.tagName + " " + elem.className + " "
                + elem.id);
        });
    });
</script>
...
```

在这个例子中，我选择了两个img元素，并使用parents方法得到了它们的祖先元素。接着我又把这些祖先元素的信息输出到控制台：

---

```

Element: DIV dcell
Element: DIV drow row2
Element: DIV dcell
Element: DIV drow row1
Element: DIV dtable
Element: DIV oblock
Element: FORM
Element: BODY
Element: HTML

```

---

还有一个方法，也就是`parentsUntil`，它也用来选择祖先元素。对于jQuery对象中的每一个元素，`parentsUntil`方法延着DOM树向上查找，选中沿途的祖先元素，直到当前祖先元素匹配参数选择器为止。代码清单6-16演示了`parentsUntil`的用法。

代码清单6-16 `parentsUntil`方法

```

...
<script>
    $(document).ready(function() {
        $("img[src*=peony], img[src*=rose]").parentsUntil("form")
            .each(function(index, elem) {
                console.log("Element: " + elem.tagName + " " + elem.className
                    + " " + elem.id);
            });
    });
</script>
...

```

在这个例子中，jQuery对象中各元素的祖先元素均被选中，直到遇到form元素为止。这个脚本产生的控制台输出如下。

---

```

Element: DIV dcell
Element: DIV drow row2
Element: DIV dcell
Element: DIV drow row1
Element: DIV dtable
Element: DIV oblock

```

---

注意选择结果并不包含匹配参数选择器的元素，也就是说，本例中结果集中并不包含form元素。我们还可以提供第二个选择器参数，这样就能进一步过滤选择结果（代码清单6-17）。

代码清单6-17 进一步过滤`parentsUntil`方法产生的元素结果集

```

...
<script type="text/javascript">
    $(document).ready(function() {

        $("img[src*=peony], img[src*=rose]").parentsUntil("form", ":not(.dcell)")
            .each(function(index, elem) {

```

```

        console.log("Element: " + elem.tagName + " " + elem.className
            + " " + elem.id);
    });

});
</script>
...

```

在这个例子中，我使用了另一个选择器过滤掉那些没有dcell类的元素。该脚本输出如下：

---

```

Element: DIV drow row2
Element: DIV drow row1
Element: DIV dtable
Element: DIV oblock

```

---

### 3. 选择第一个匹配的祖先元素

closest方法用来得到匹配参数选择器的第一个祖先元素。代码清单6-18演示了该方法的用法。

代码清单6-18 closest方法

```

...
<script type="text/javascript">
    $(document).ready(function() {

        $("img").closest(".drow").each(function(index, elem) {
            console.log("Element: " + elem.tagName + " " + elem.className
                + " " + elem.id);
        });

        var contextElem = document.getElementById("row1");
        $("img").closest(".drow", contextElem).each(function(index, elem) {
            console.log("Context Element: " + elem.tagName + " " + elem.className
                + " " + elem.id);
        });

    });
</script>
...

```

在这个例子中，我先选中页面中的所有img元素，然后找出它们祖先元素中那些拥有drow类的元素。该函数也支持使用上下文对象（第二个参数，HTMLElement对象）限制祖先元素的选择范围：非上下文对象或其后代的元素将被忽略。该脚本输出如下。

---

```

Element: DIV drow row1
Element: DIV drow row2
Context Element: DIV drow row2

```

---

如果我们指定一个jQuery对象，或者一个HTMLElement对象，或者一个HTMLElement对象数组为closest方法的参数，jQuery将沿DOM树向上求索，直到当前祖先元素匹配第一个参数为止。代码清单6-19演示了这种用法。

## 代码清单6-19 closest方法（以jQuery结果集为参数）

```

...
<script type="text/javascript">
    $(document).ready(function() {

        var jq = $("#row1, #row2, form");

        $("img[src*=rose]").closest(jq).each(function(index, elem) {
            console.log("Context Element: " + elem.tagName + " " + elem.className
                + " " + elem.id);
        });

    });
</script>
...

```

在这个例子里，我先是在页面中选中一个img元素，然后使用closest方法匹配它的祖先元素。我将一个包含form、#row1、#row2元素的jQuery结果集用作closest方法的参数。jQuery会选取离img元素最近的，匹配这一结果集内任一元素的祖先元素。也就是说，它将沿DOM树向上查找，直到当前祖先元素匹配参数对象中的某个元素为止。该脚本输出如下。

---

```
Context Element: DIV drow row1
```

---

offsetParent方法是closest方法的一个变种，它匹配祖先元素中最近的定位元素（所谓定位元素，即CSS position属性值为relative、absolute或fixed的元素）。这种元素又称为定位祖先元素，在制作动画时（第10章讲解了jQuery对动画的支持）找到定位祖先元素非常有用。代码清单6-20演示了该方法的一种应用。

## 代码清单6-20 使用offsetParent方法

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <style type="text/css">
        #oblock {position: fixed; top: 120px; left: 50px}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $("img[src*=aster]").offsetParent().css("background-color", "lightgrey");
        });
    </script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">

```

```

<div class="dcell">
  <label for="aster">Aster:</label>
  <input name="aster" value="0" required>
</div>
<div class="dcell">
  <label for="daffodil">Daffodil:</label>
  <input name="daffodil" value="0" required >
</div>
<div class="dcell">
  <label for="rose">Rose:</label>
  <input name="rose" value="0" required>
</div>
</div>
</div>
<div id="buttonDiv"><button type="submit">Place Order</button></div>
</form>
</body>
</html>

```

这是示例文档的一个删减版，在这个文档中我使用CSS为#oblock元素的position属性设置了一个值。然后，我在脚本中使用jQuery选中一个img元素并调用offsetParent方法查找离它最近的定位祖先元素。该方法从img元素起沿DOM树向上查找，直到遇到定位元素为止。接着我使用css方法为该定位元素设置了背景色（参见图6-11）。

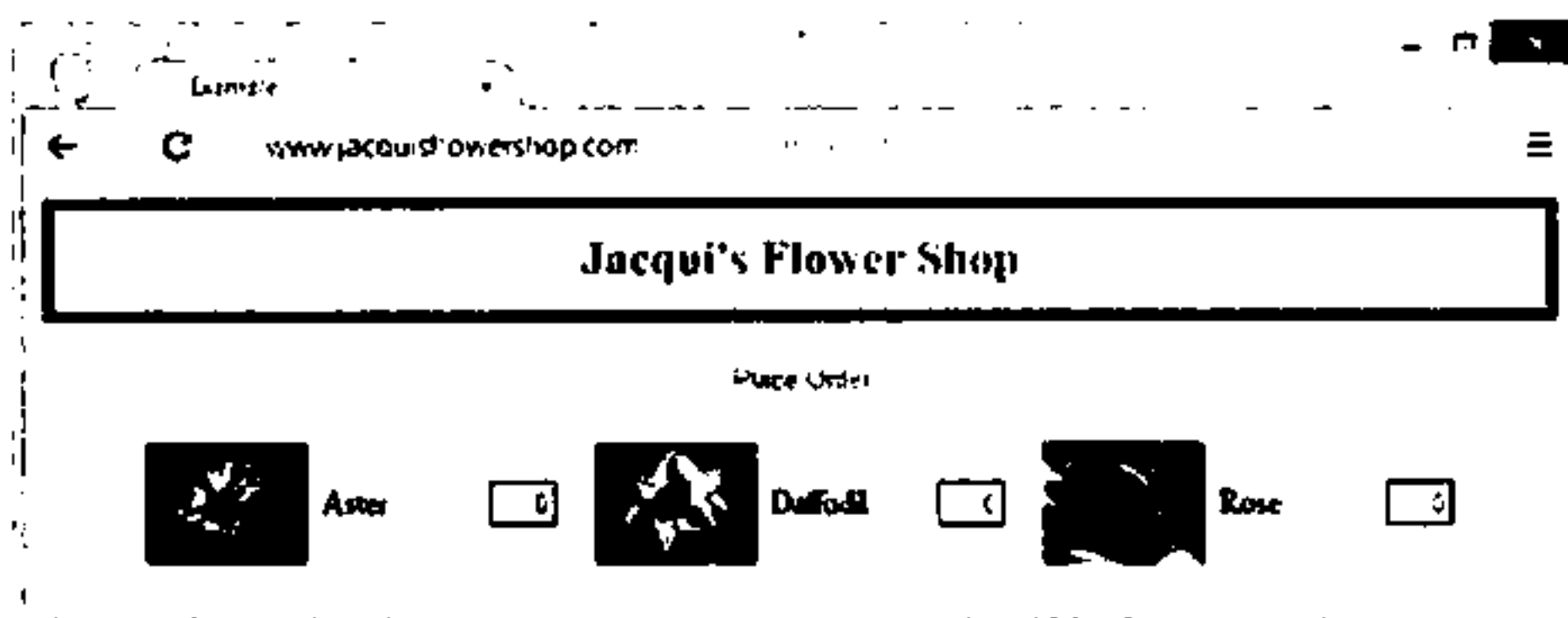


图6-11 寻找最近的定位祖先元素

### 6.6.3 访问兄弟元素

这是访问DOM的最后一种形式。jQuery提供了一系列方法用于访问兄弟元素，见表6-10。

表6-10 访问兄弟元素的方法

方 法	描 述
next()、next(selector)	得到下一个兄弟元素，可选的selector参数用来过滤选择结果
nextAll()、nextAll(selector)	得到后面的所有兄弟元素，可选的selector参数用来过滤选择结果
nextUntil((selector)	得到后面的兄弟元素，直到（但不包括）匹配参数（选择器、jQuery对象、HTMLElement对象或HTMLElement对象数组）。可选的第二个选择器参数用来进一步过滤选择结果
nextUntil(selector, selector)	
nextUntil(jQuery)	
nextUntil(jQuery, selector)	
nextUntil(HTMLElement[])	
nextUntil(HTMLElement[], selector)	



(续)

方 法	描 述
<code>prev()</code> 、 <code>prev(selector)</code>	得到上一个兄弟元素，可选的 <code>selector</code> 参数用来过滤选择结果
<code>prevAll()</code> 、 <code>prevAll(selector)</code>	得到前面的所有兄弟元素，可选的 <code>selector</code> 参数用来过滤选择结果
<code>prevUntil(selector)</code>	得到前面的兄弟元素，直到（但不包括）匹配参数（选择器、jQuery对象、HTMLElement对象或HTMLElement对象数组）。可选的第二个选择器参数用来进一步过滤选择结果
<code>prevUntil(selector, selector)</code>	
<code>prevUntil(jQuery)</code>	
<code>prevUntil(jQuery, selector)</code>	
<code>prevUntil(HTMLElement[])</code>	
<code>prevUntil(HTMLElement[], selector)</code>	
<code>siblings()</code>	选择所有的兄弟元素，可选的过滤器用来过滤选择结果
<code>siblings(selector)</code>	

### 1. 选择全部兄弟元素

`siblings`方法用来选中jQuery对象包含元素的所有兄弟元素。代码清单6-21演示了这个方法的使用。（在这个例子中，我完全恢复了代码清单6-1所示的花店页面。）

代码清单6-21 `siblings`方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script type="text/javascript">
    $(document).ready(function () {
      $("img[src*=aster], img[src*=primula]")
        .parent().siblings().css("border", "thick solid blue");
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
        <div id="row1" class="drow">
          <div class="dcell">
            <label for="aster">Aster:</label>
            <input name="aster" value="0" required>
          </div>
          <div class="dcell">
            <label for="daffodil">Daffodil:</label>
            <input name="daffodil" value="0" required>
          </div>
          <div class="dcell">
            <label for="rose">Rose:</label>
            <input name="rose" value="0" required>
          </div>
        </div>
      </div>
    </div>
  </form>
</body>
</html>
```

```

</div>
<div id="row2" class="drow">
  <div class="dcell">
    <label for="peony">Peony:</label>
    <input name="peony" value="0" required>
  </div>
  <div class="dcell">
    <label for="primula">Primula:</label>
    <input name="primula" value="0" required>
  </div>
  <div class="dcell">
    <label for="snowdrop">Snowdrop:</label>
    <input name="snowdrop" value="0" required>
  </div>
</div>
</div>
<div id="buttonDiv"><button type="submit">Place Order</button></div>
</form>
</body>
</html>

```

在这个例子中,我先得得到两个元素,然后调用parent方法选中它们的父元素,接着调用siblings方法得到这两个父元素的兄弟元素。得到所有的兄弟元素(之前的和之后的)之后,我马上调用css方法为这些元素设置边框样式(参见图6-12,其中4个图片部分均被蓝色边框包围)。(调用parent方法使样式的效果更清楚明了。)

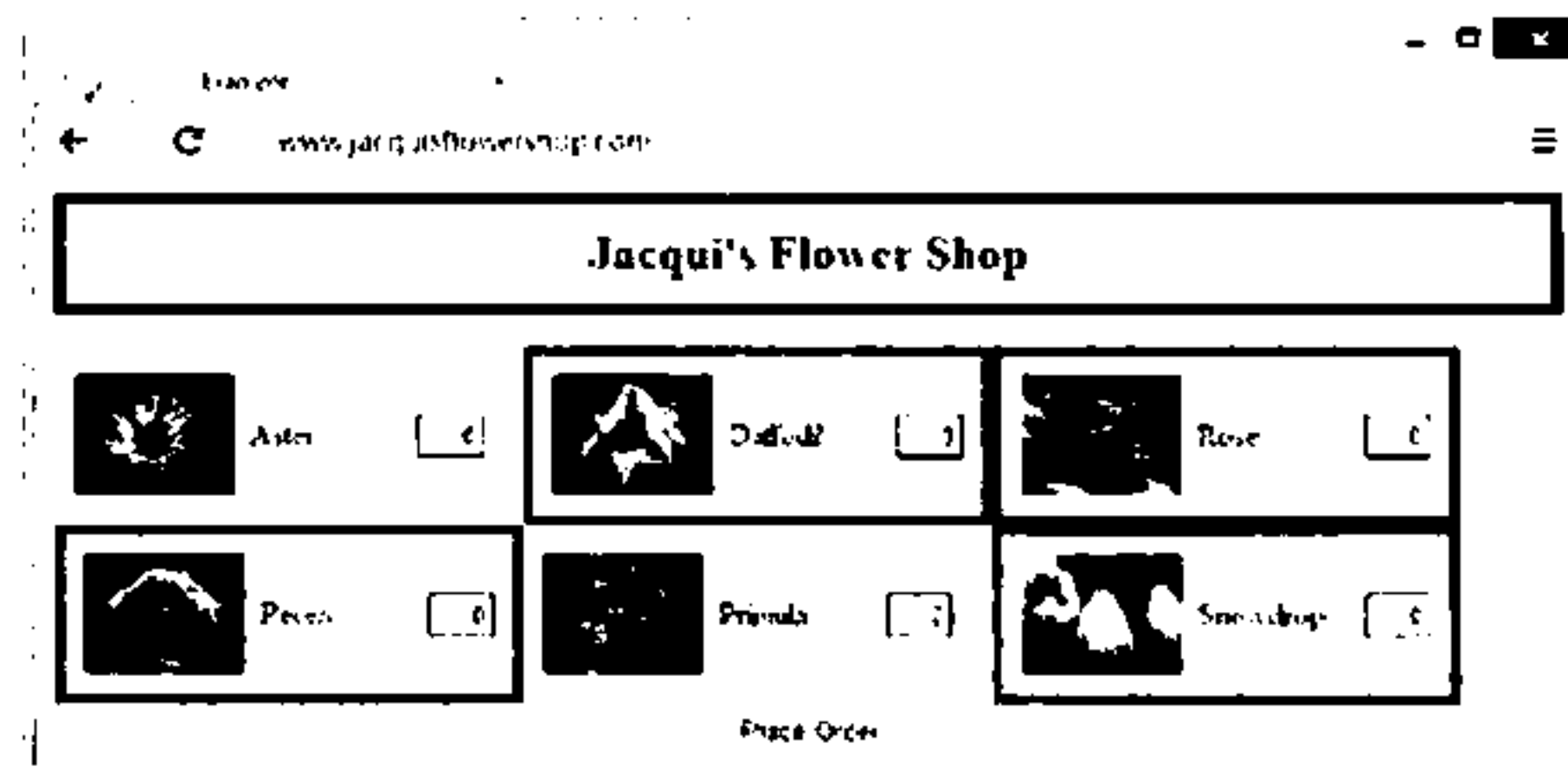


图6-12 选择兄弟元素

注意,选择元素A的兄弟元素时,结果集中并不包含元素A。当然啦,如果jQuery对象中包含的元素A恰好是元素B的兄弟元素,那就另当别论了(参见代码清单6-22)。

#### 代码清单6-22 重叠的兄弟元素结果集

```

...
<script type="text/javascript">
  $(document).ready(function() {
    $("#row1 div.dcell").siblings().css("border", "thick solid blue");
  });
</script>
...

```

在这段脚本中，我选中#row1元素的所有div.dcell后代元素，然后调用siblings方法。如图6-13所示（其中3个图片部分均被蓝色边框包围），结果集中的每个元素互为兄弟元素。

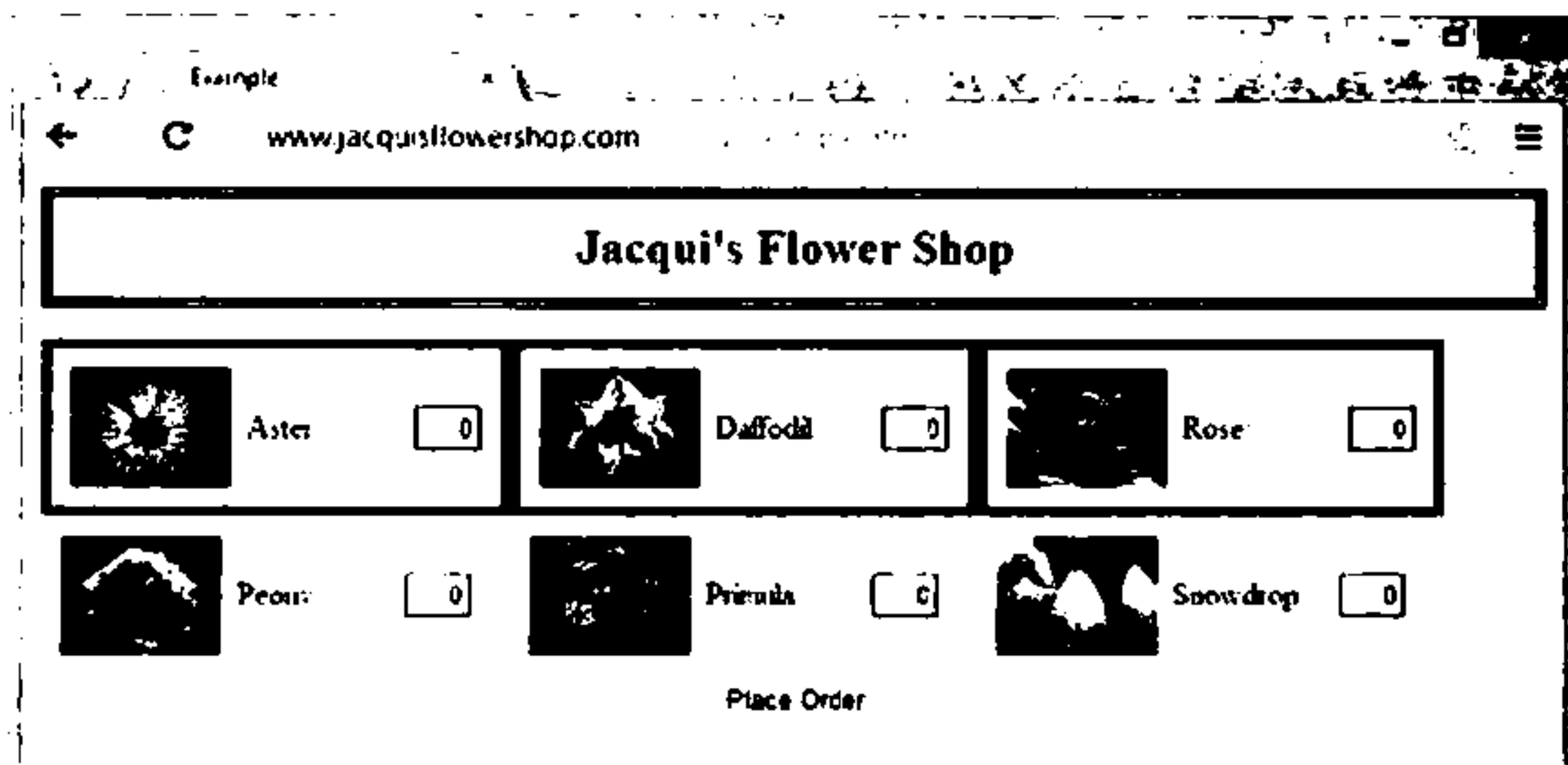


图6-13 重叠的兄弟元素

## 2. 选择后面的和前面的兄弟元素

由于选择下一个/上一个兄弟元素的方法工作方式都一样，这里不会为每个方法都给出示例。代码清单6-23演示了nextAll和prevAll方法的使用。

### 代码清单6-23 nextAll方法和prevAll方法

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("img[src*=aster]").parent().nextAll().css("border", "thick solid blue");
        $("img[src*=primula]").parent().prevAll().css("border", "thick double red");
    });
</script>
...
```

这段脚本中，第一行选中的是紫菀花图片的父元素之后的所有兄弟元素，第二行选中的则是报春花图片父元素之前的所有兄弟元素。这段脚本的实际效果见图6-14。

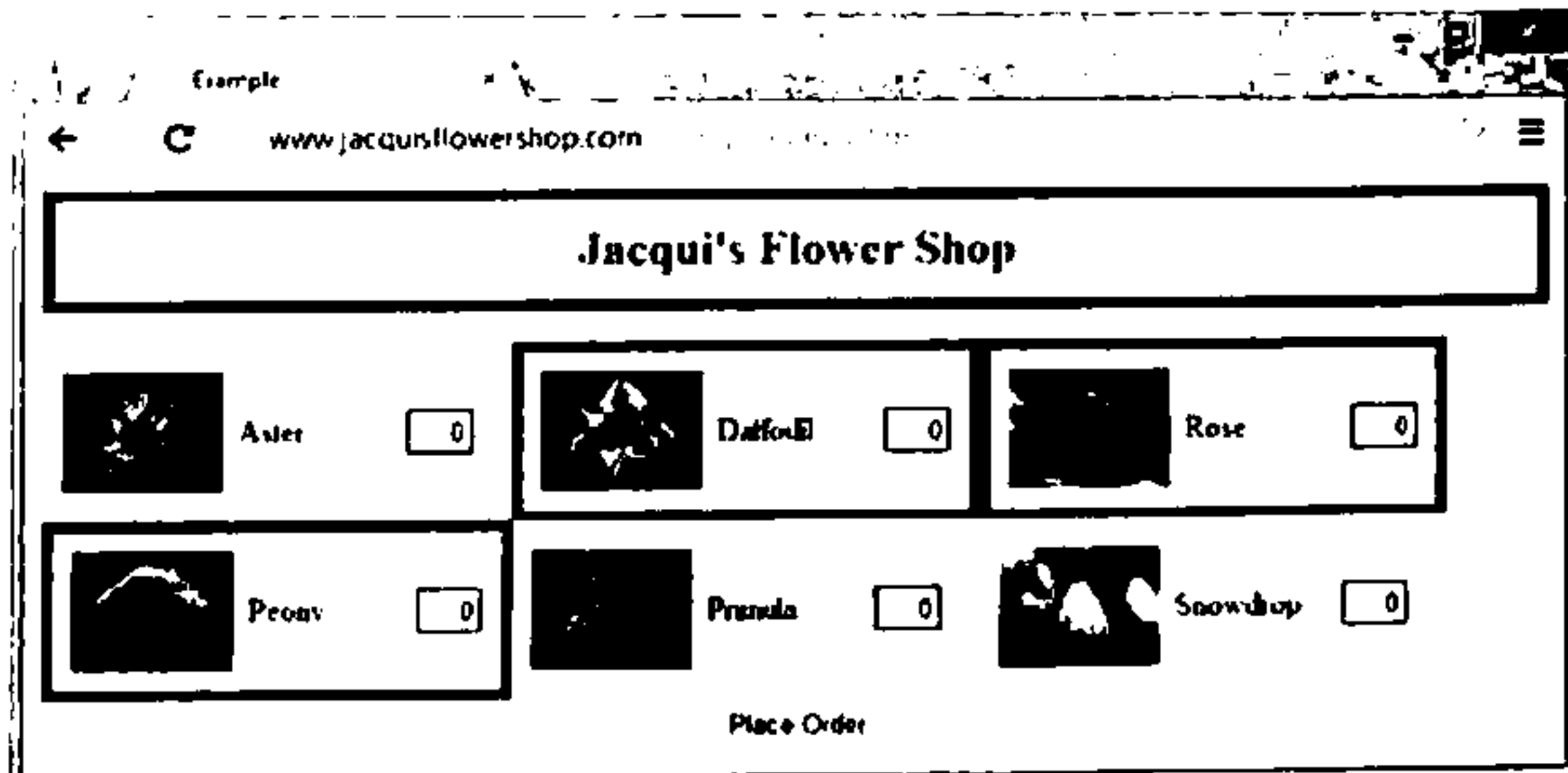


图6-14 选择后面或前面的兄弟元素

## 6.7 小结

本章演示了控制和裁剪jQuery结果集的方法以满足实际需要，其中包括追加元素、过滤元素、遍历处理元素及测试结果集是否满足某个条件。我还演示了以某个jQuery结果集为起点访问DOM的方法，以及以一个结果集为起点得到页面中另一个结果集的方法。在第7章，我将介绍如何使用选择结果处理DOM，利用jQuery方法来创建、删除和修改HTML元素。



上一章展示了如何选择元素，而利用选择结果改变HTML文档本身的结构，也就是通常所说的操控DOM，是jQuery最强大的一个功能。本章，我们将一起学习改变页面结构的种种方法，包括添加元素作为其他元素的子元素、父元素或者兄弟元素。我还会讲解如何创建新元素、在页面中移动元素，以及彻底删除元素。表7-1列出了本章概要。

表7-1 本章概要

问 题	解决方法	代码清单
如何创建新元素	以HTML片段为参数传递给\$函数，或者使用clone方法，或者使用DOM API	1~3
如何让元素以某元素子元素的形式插入到该元素内容的最后面	使用append方法	4
如何让元素以某元素子元素的形式插入到该元素内容的最前面	使用prepend方法	5、6
如何把已有的元素插入到其他地方	克隆它，然后把克隆后的元素插入到目标位置	7、8
如果把jQuery对象内含元素以子元素的形式插入到其他元素之中	使用appendTo或prependTo方法	9
如何动态插入元素	使用函数参数调用append或prepend方法	10
如何插入元素作为已有元素的父元素	使用wrap方法	11
如何插入元素作为几个已有元素的父元素	使用wrapAll方法	12、13
如何插入元素包住已有元素的内容	使用wrapInner方法	14
如何动态插入元素包住已有的元素或内容	使用函数参数调用wrap或wrapInner方法	15
如何插入兄弟元素	使用after、before、insertAfter或insertBefore方法	16、17
如何动态插入兄弟元素	使用函数参数调用before或after方法	18
如何替换元素	使用replaceWith或replaceAll方法	19
如何动态替换元素	使用函数参数调用replaceWith方法	20
如何从DOM中删除元素	使用remove或detach方法	21~23
如何删除元素的内容	使用empty方法	24
如何删除元素的祖先元素	使用unwrap方法	25

## 新版jQuery中与本章有关的变化

对本章来说, jQuery 1.9/2.0最重要的变化发生在对HTML字符串的解释上, 新版本采用了更严格的规则解释HTML字符串。不过这一变更在1.10和2.0.1版本中已经回退为旧版规则, 具体可参见侧边栏《HTML解析器的变化》中的内容。

还有一些底层技术的变化, 在新版jQuery中, after、before、replaceWith、appendTo、insertBefore、insertAfter和replaceAll方法底层处理jQuery对象的方式, 与其他DOM处理方法保持了一致。令人欣慰的是, 这些变化并不会影响到本章讲到的有关技术。

## 7.1 创建新元素

编写Web应用时, 在把新元素插入到DOM中的目标位置之前, 我们往往需要先创建它们(也可以插入已有的元素, 后面会讲到)。在接下来的几节中, 我们将一起学习创建新元素的几种方式。

**提示** 有一点非常重要, 即创建新元素这个动作并不会自动地把新元素插入到DOM中。我们需要明确告诉jQuery应将新元素插入到哪个位置, 本章后面会讲解怎么做。

### 7.1.1 使用\$函数创建新元素

把一段HTML文本作为参数调用\$方法, 我们就能得到与HTML文本对应的新元素。jQuery会自动解析HTML文本并生成相应的DOM对象。具体的例子见代码清单7-1。

代码清单7-1 使用\$函数创建新元素

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-1.7.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script type="text/javascript">
    $(document).ready(function() {

      var newElems = $('<div class="dcell"></div>');

      newElems.each(function (index, elem) {
        console.log("New element: " + elem.tagName + " " + elem.className);
      });

      newElems.children().each(function(index, elem) {
        console.log("Child: " + elem.tagName + " " + elem.src);
      });
    });
  </script>
</head>
```

```
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
        <div id="row1" class="drow">
          <div class="dcell">
            <label for="aster">aster:</label>
            <input name="aster" value="0" required>
          </div>
          <div class="dcell">
            <label for="daffodil">Daffodil:</label>
            <input name="daffodil" value="0" required >
          </div>
          <div class="dcell">
            <label for="rose">Rose:</label>
            <input name="rose" value="0" required>
          </div>
        </div>
        <div id="row2" class="drow">
          <div class="dcell">
            <label for="peony">Peony:</label>
            <input name="peony" value="0" required>
          </div>
          <div class="dcell">
            <label for="primula">Primula:</label>
            <input name="primula" value="0" required>
          </div>
          <div class="dcell">
            <label for="snowdrop">Snowdrop:</label>
            <input name="snowdrop" value="0" required>
          </div>
        </div>
      </div>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
</html>
```

在这个例子中，我使用HTML片段新建了两个元素：一个div元素，一个img元素。既然是HTML片段，当然可以有结构。在本例中，img元素是div元素的子元素。

### HTML解析器的变化

当我们传递一个字符串给\$函数时，jQuery要判断这是一个选择器，还是一个HTML字符串。在1.9版本之前，如果字符串的任意位置包括一个HTML标签，这个字符串就会当成是一段HTML代码（参阅第2章详细了解HTML标签）。在某些极端情况下，一个复杂的选择器可能会被解释成HTML字符串从而造成问题。为了解决这一问题，jQuery 1.9/2.0引入了一个变化，即只有字符串以<开头的情况下才会被视为HTML代码。

jQuery 1.10/2.0.1版本回退了这一改变，证明这一变化并不受开发者欢迎。不过这也提醒人们，检测HTML字符串的方式未来仍然存在变数。如果你在处理的字符串有可能引发歧义，建议你使用parseHTML方法，这样就不会出现期望当做HTML处理却被当成选择器处理的问题。

由\$函数返回的jQuery对象只包含HTML片段的顶级（最外层）元素。为了演示这一性质，我使用each方法把jQuery内含元素的基本信息输出到了控制台。至于顶级元素的子元素，jQuery并没有把它们扔掉，我们可以使用访问DOM的普通方法（详见第6章）访问这些元素。出于演示目的，我先在jQuery对象上调用children方法，然后同样把每个子元素的基本信息输出到控制台。这段脚本的输出如下：

```
New element: DIV dcell
Child: IMG http://www.jacquisflowershop.com/jquery/lily.png
```

**提示** 还可以额外提供一个map对象参数来指定HTML元素的属性。在第12章中，有一个这样调用\$函数的例子。

### 7.1.2 通过克隆已有元素生成新元素

我们可以使用clone方法以已有的元素为模子生成新元素。clone方法会复制jQuery对象内包含后代元素在内的所有元素，具体的例子见代码清单7-2。

代码清单7-2 克隆元素

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var newElems = $('div.dcell').clone();

        newElems.each(function (index, elem) {
            console.log("New element: " + elem.tagName + " " + elem.className);
        });

        newElems.children('img').each(function(index, elem) {
            console.log("Child: " + elem.tagName + " " + elem.src);
        });

    });
</script>
...
```

在这段脚本中，我选取并克隆了所有的div.dcell元素。为演示克隆结果中同样包含着后代元素，我使用带选择器参数的children方法得到克隆结果中所有的img元素。我把div元素和img元素的基本信息输出到了控制台，结果如下：



---

```

New element: DIV dcell
New element: DIV dcell
New element: DIV dcell
New element: DIV dcell
New element: DIV dcell
New element: DIV dcell
Child: IMG http://www.jacquisflowershop.com/jquery/aster.png
Child: IMG http://www.jacquisflowershop.com/jquery/daffodil.png
Child: IMG http://www.jacquisflowershop.com/jquery/rose.png
Child: IMG http://www.jacquisflowershop.com/jquery/peony.png
Child: IMG http://www.jacquisflowershop.com/jquery/primula.png
Child: IMG http://www.jacquisflowershop.com/jquery/snowdrop.png

```

---

**提示** 如果你提供true参数给clone方法，就会把事件处理函数及关联数据一并克隆到目标元素。若省略这个参数或者将false用作参数，它就会忽略事件处理函数和关联数据。我会在第8章讲解如何为元素关联数据，在第9章讲解jQuery的事件处理。

7

### 7.1.3 使用DOM API创建新元素

我们也可以使用DOM API直接生成HTMLElement对象。使用其他技术生成新元素时，其实是jQuery在幕后悄悄地调用DOM API。我不会在这里深入剖析DOM API，代码清单7-3是一个简单的例子，可以让你知道这个技术大体是怎么回事。

代码清单7-3 使用DOM API创建新元素

```

...
<script type="text/javascript">
    $(document).ready(function() {

        var divElem = document.createElement("div");
        divElem.classList.add("dcell");

        var imgElem = document.createElement("img");
        imgElem.src = "lily.png";

        divElem.appendChild(imgElem);

        var newElems = $(divElem);

        newElems.each(function (index, elem) {
            console.log("New element: " + elem.tagName + " " + elem.className);
        });

        newElems.children('img').each(function(index, elem) {
            console.log("Child: " + elem.tagName + " " + elem.src);
        });

    });

```

```
</script>
...
```

如同前一个例子,我在这个例子里也创建并配置了一个div元素和一个img元素,让img元素成为div元素的子元素。以这种方式创建元素完全没有问题,不过既然这是一本讲jQuery的书,为了避免离题太远,关于DOM API的内容就此打住,浅尝辄止。

我把HTMLElement元素作为jQuery \$函数的参数,以方便像前几个例子那样使用each方法处理元素。控制台输出如下:

```
New element: DIV dcell
Child: IMG http://www.jacquisflowershop.com/jquery/lily.png
```

## 7.2 添加子元素或后代元素

创建元素后,就可以着手将它们添加到文档中。我们先从把一个元素插入到另一个元素内,生成该元素的子元素或后代元素开始。表7-2列出了有关方法。

表7-2 用于添加子元素或后代元素的方法

方 法	描 述
append(HTML)、append(jQuery)、append(HTMLElement[])	把参数指定的元素插入到所有jQuery内含元素内容末尾,成为它们的最后一个子元素
prepend(HTML)、prepend(jQuery)、prepend(HTMLElement[])	把参数指定的元素插入到所有jQuery内含元素内容之前,成为它们的第一个子元素
appendTo(jQuery)、appendTo(HTMLElement[])	把jQuery内含元素插入到参数指定元素内容的末尾,成为参数元素的最后一个子元素
prependTo(HTML)、prependTo(jQuery) prependTo(HTMLElement[])	把jQuery内含元素插入到参数指定元素内容之前,成为参数元素的第一个子元素
append(function)、prepend(function)	把参数函数的返回值插入到jQuery对象内含元素内容的末尾或者之前,成为它们的子元素

**提示** 我们也可以使用wrapInner方法插入子元素(见7.3.2节)。这个方法会在一个元素和它的子元素之间插入一个新的子元素。还有一种技术,那就是使用html方法(详见第8章)。

注意,由于参数元素将被插入到jQuery对象内含的所有元素之内,因此我们在第6章学过的,剪裁选择结果以确保jQuery对象只包含我们需要的元素这一技术,就变得异常重要。代码清单7-4演示了append方法的用法。

### 代码清单7-4 append方法

```
...
<script type="text/javascript">
```

```

$(document).ready(function() {
    var newElems = $("<div class='dcell'></div>")
        .append("<img src='lily.png' />")
        .append("<label for='lily'>Lily:</label>")
        .append("<input name='lily' value='0' required />");

    newElems.css("border", "thick solid red");

    $('#row1').append(newElems);
});
</script>
...

```

在上面的脚本中，我展示了append方法的两种用法。第一种用法是先创建一组新元素，然后把这组元素添加到页面中。由于这是你第一次接触DOM操控方法，接下来我会多花一些时间，详细讲解代码的行为，帮你避开那些与DOM有关的最常见jQuery错误。不过，首先来看一下这个脚本的效果：添加新元素到页面中（参见图7-1）。

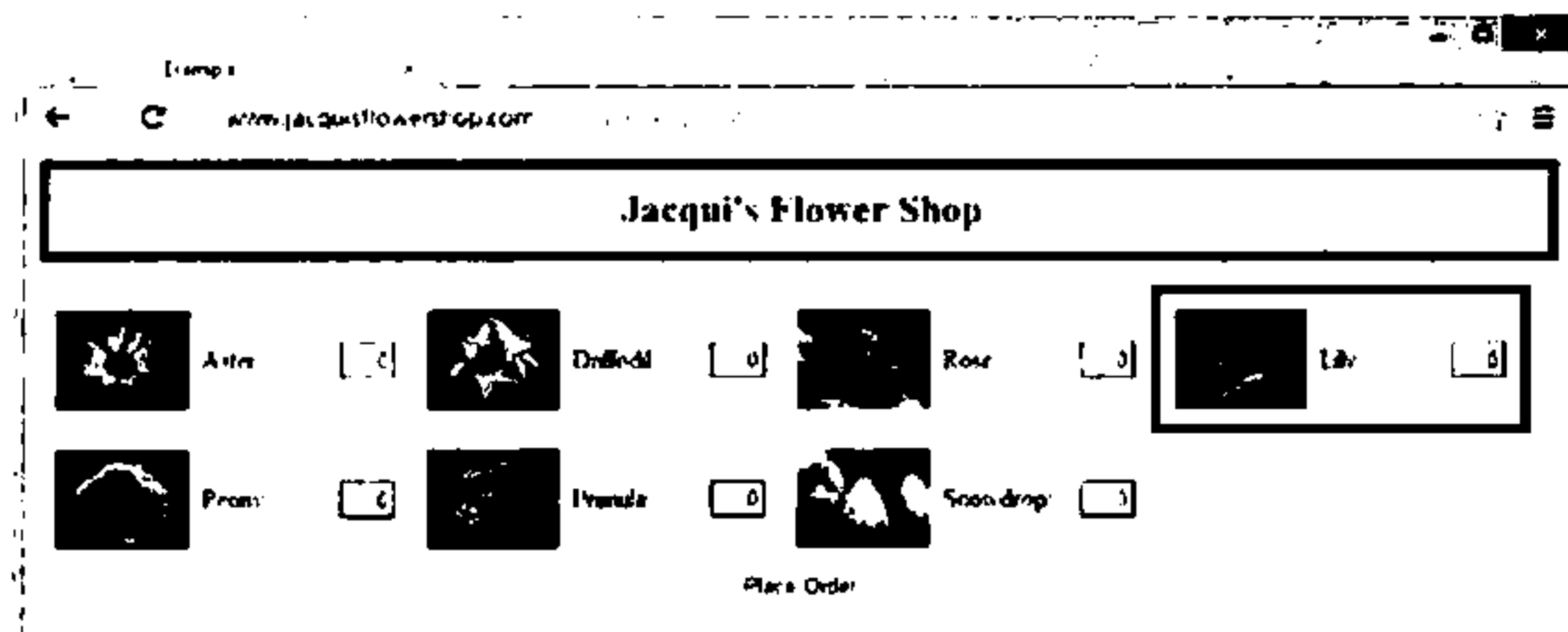


图7-1 在页面中插入新元素

我们先来看看怎么用append方法创建新元素：

```

...
var newElems = $("<div class='dcell' />")
    .append("<img src='lily.png' />")
    .append("<label for='lily'>Lily:</label>")
    .append("<input name='lily' value='0' required />");
...

```

我创建了单个包含着一些元素的较大HTML片段。不过，要提醒大家注意有关处理DOM的一个关键行为，上面这些append方法调用返回的jQuery对象内含的元素始终不变，一直是最初的div.dcell元素。

也就是说，最初jQuery对象中包含着一个div元素，而每个append方法返回的jQuery对象始终包含着最初那个div元素，而不是我添加的元素。这说明链式append调用的结果只是为最初的div元素添加了一些新创建的子元素。

需要指出的第二个行为是：尽管这些新创建的元素可能还没有添加到页面，我们仍然可以使用jQuery访问并修改它们。比如，我希望新元素边框高亮，就这样写：

```

...
newElems.css("border", "thick solid red");
...

```

这个功能很有用。利用这个特点，我们能够先生成很复杂的元素并完全设置好它们的样式，再将它们添加到页面中。

最后，使用下面这条语句，我把这些新元素添加到页面上：

```
...
$('#row1').append(newElems);
...
```

新元素会追加到结果集中的每个元素之内。不过这个结果集里只有一个元素（#row1），因此在咱们花店页面上就新添了一个品种：百合花。

### 7.2.1 插入第一个子元素

与append方法的“追加”（插到最后面）行为相对应，prepend方法会插入新元素到jQuery所有内含元素之内，使之成为相应元素的第一个子元素（插到最前面）。具体的例子见代码清单7-5。

代码清单7-5 prepend方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var orchidElems = $("<div class='dcell' />")
            .append("<img src='orchid.png' />")
            .append("<label for='orchid'>Orchid:</label>")
            .append("<input name='orchid' value='0' required />");

        var newElems = $("<div class='dcell' />")
            .append("<img src='lily.png' />")
            .append("<label for='lily'>Lily:</label>")
            .append("<input name='lily' value='0' required />").add(orchidElems);

        newElems.css("border", "thick solid red");

        $('#row1, #row2').prepend(newElems);
    });
</script>
...
```

上面这段脚本不仅演示了prepend方法的用法，还演示了jQuery处理DOM的另一个特点：作为参数传递给append/prepend方法的所有元素，会添加到jQuery内含的所有元素之中，成为它们的子元素。在本例中，我创建了两个div元素，一个用于百合花（lily），一个用于兰花（orchid），并使用add方法把这两个div元素合并到一个jQuery对象之中。

---

**提示** add方法也可以接受字符串形式的HTML片段。我们也可以利用这一特点使用jQuery对象创建新元素。

---

接着我又创建了一个包含着#row1和#row2元素的jQuery对象，然后使用prepend方法把兰花和百合花添加到页面中。这段脚本的效果见图7-2。这些新元素都有着醒目的红色边框。如图所示，我们不但

把百合花和兰花添加到了第一行，而且添加到了第二行。

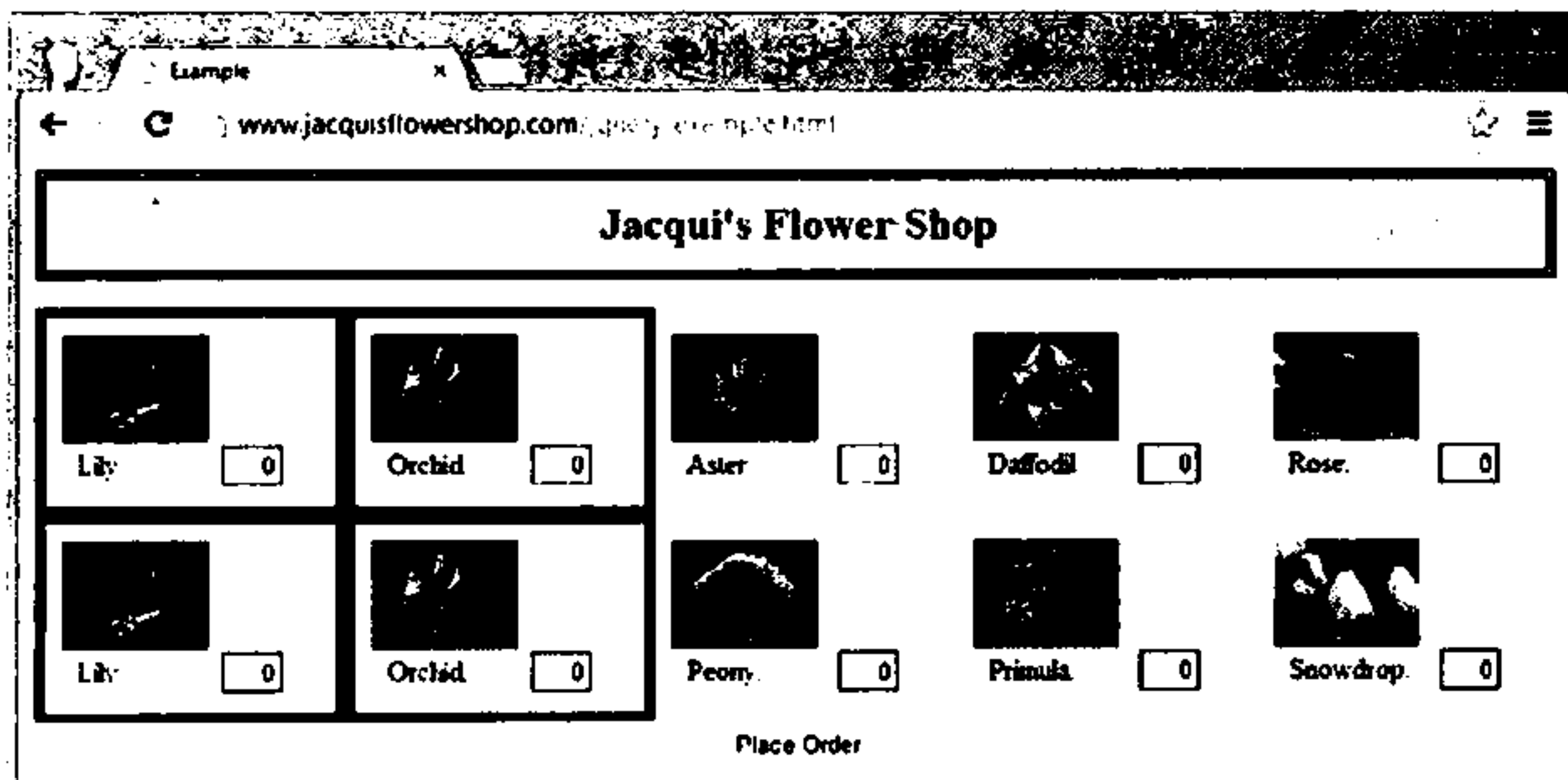


图7-2 把新建元素插入到jQuery对象内的多个元素

除了上例中的add方法，还有一种方法可以把多个元素传递给这些DOM修改方法，如代码清单7-6所示。该清单产生的结果与图7-2相同。

代码清单7-6 传递多个元素给prepend的另一种方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var orchidElems = $("<div class='dcell' />")
            .append("<img src='orchid.png' />")
            .append("<label for='orchid'>Orchid:</label>")
            .append("<input name='orchid' value='0' required />");

        var lilyElems = $("<div class='dcell' />")
            .append("<img src='lily.png' />")
            .append("<label for='lily'>Lily:</label>")
            .append("<input name='lily' value='0' required />");

        lilyElems.css("border", "thick solid red");

        $('#row1, #row2').prepend(lilyElems, orchidElems);
    });
</script>
...
```

**提示** 这里使用独立的一条语句调用css方法设置CSS border属性，只是为了让这个例子更容易理解。事实上，我完全可以像其他jQuery方法一样链式调用css方法。

## 7.2.2 把同一组元素插入到页面的不同位置

我们只可以把新元素插入到页面一次。现在把它们多次传给DOM插入方法只会移动元素，而非复制它们。代码清单7-7演示了这个问题。

代码清单7-7 尝试两次把新元素添加到页面

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var orchidElems = $("<div class='dcell' />")
            .append("<img src='orchid.png' />")
            .append("<label for='orchid'>Orchid:</label>")
            .append("<input name='orchid' value='0' required />");

        var newElems = $("<div class='dcell' />")
            .append("<img src='lily.png' />")
            .append("<label for='lily'>Lily:</label>")
            .append("<input name='lily' value='0' required />").add(orchidElems);

        newElems.css("border", "thick solid red");

        $('#row1').append(newElems);
        $('#row2').prepend(newElems);
    });
</script>
...
```

这段脚本的目标很明确：把新元素追加到#row1并插入到#row2内容的最前面。当然，它失败了。如图7-3所示，这段脚本未能如愿。

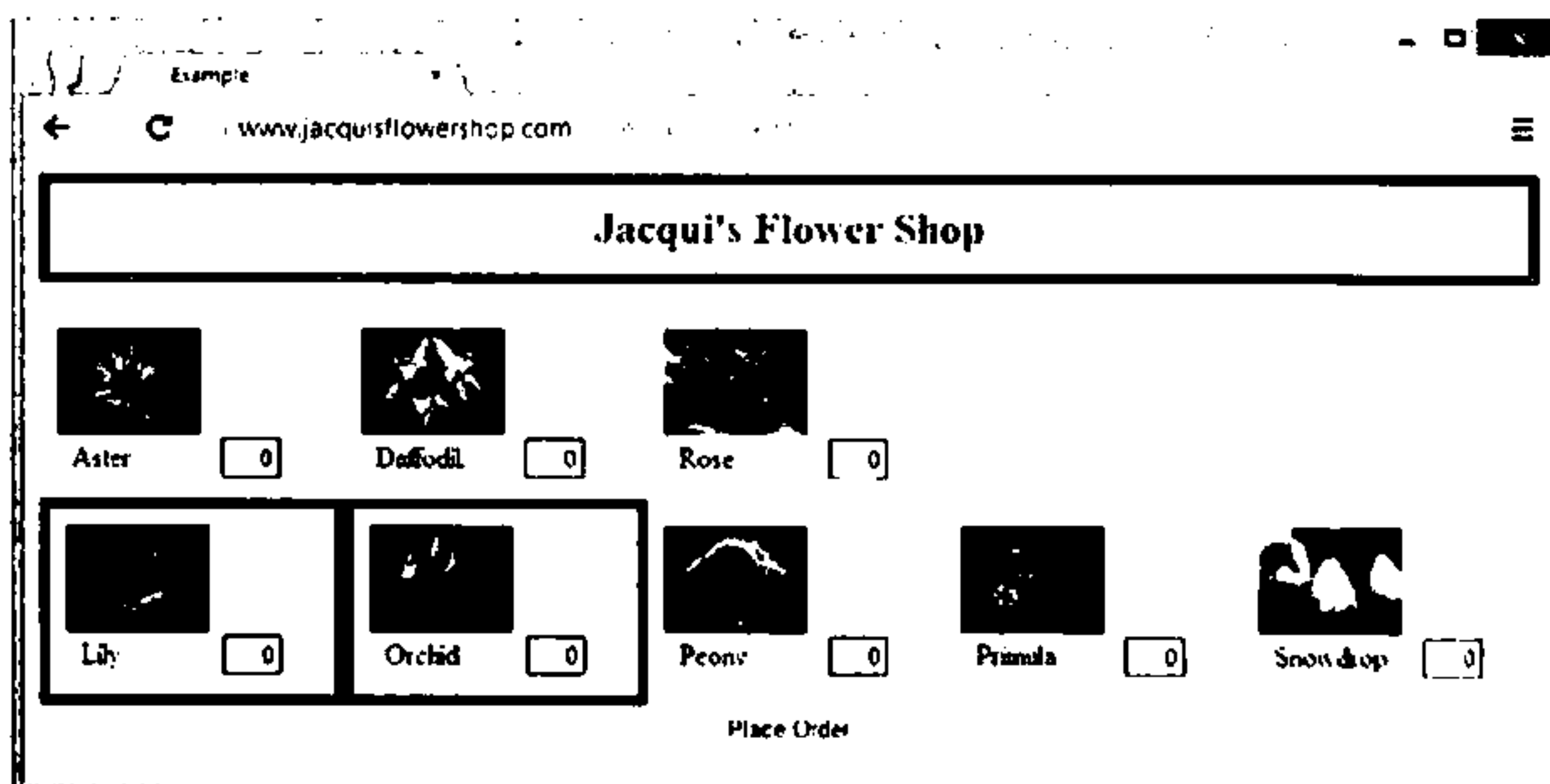


图7-3 尝试把新元素两次添加到页面（未成功）

元素确实被追加到了#row1内，然而接下来企图把新元素插入到#row1最前面的行为却是把新元素从#row1移动，而非复制到#row2。要解决这个问题，我们需要使用clone方法先生成新元素的一个副本，再执行插入操作。代码清单7-8展示了问题修正之后的代码。

代码清单7-8 为了支持把新元素多次添加到页面而克隆元素

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var orchidElems = $("<div class='dcell' />")
            .append("<img src='orchid.png' />")
            .append("<label for='orchid'>Orchid:</label>")
            .append("<input name='orchid' value='0' required />");

        var newElems = $("<div class='dcell' />")
            .append("<img src='lily.png' />")
            .append("<label for='lily'>Lily:</label>")
            .append("<input name='lily' value='0' required />").add(orchidElems);

        newElems.css("border", "thick solid red");

        $('#row1').append(newElems);
        $('#row2').prepend(newElems.clone());
    });
</script>
...
```

如图7-4所示，这一次我们把新元素和它的副本分别插入到了预期的位置。

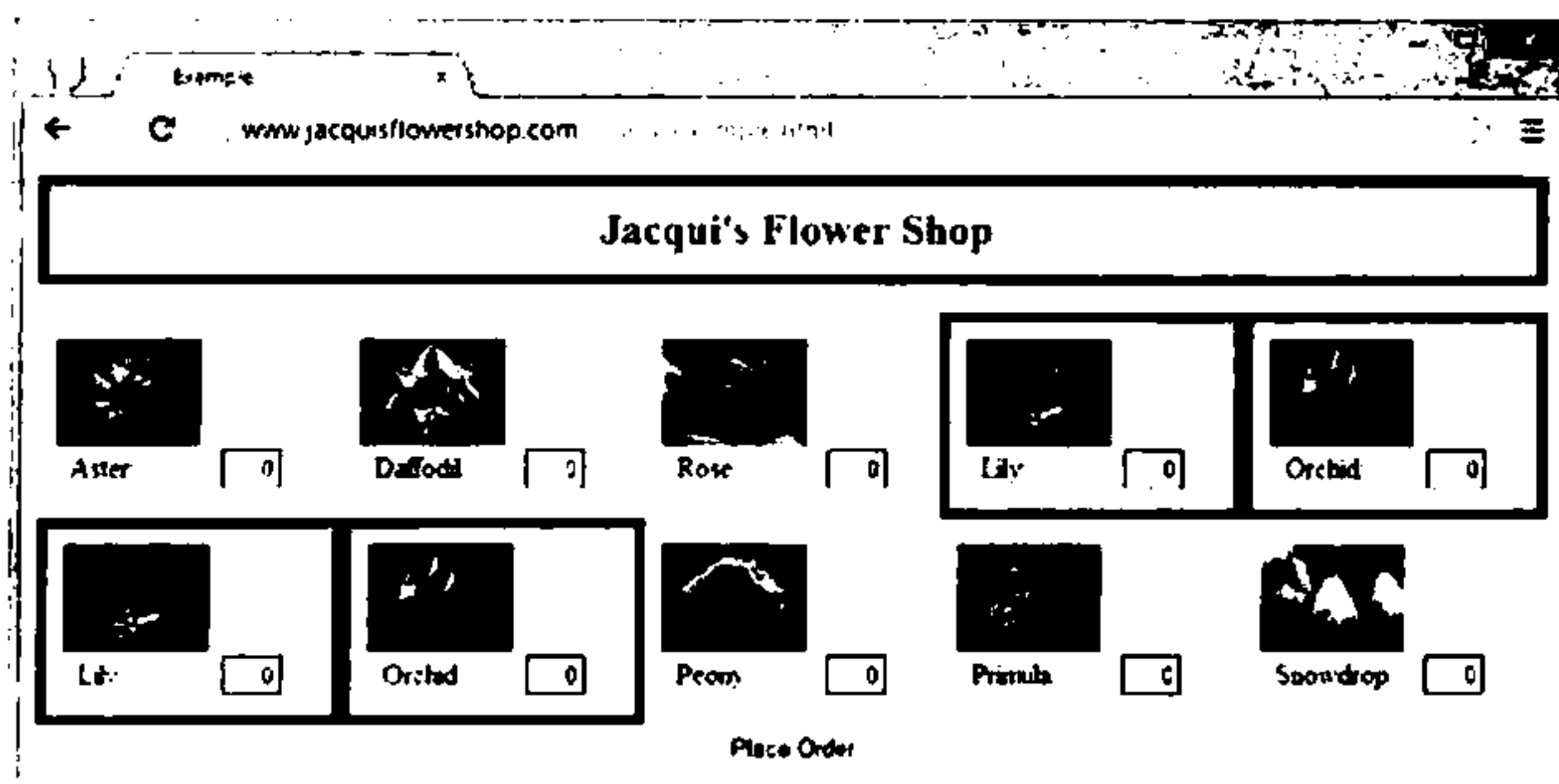


图7-4 克隆并插入元素

### 7.2.3 插入jQuery对象

我们也可以使用appendTo和prependTo方法来改变元素之间的关系，如代码清单7-9所示。

代码清单7-9 appendTo方法

```
...
<script type="text/javascript">
    $(document).ready(function() {
```

```

    var newElems = $("

在这段脚本中，我先新建了一个div.dcell元素，然后使用appendTo方法把页面中的img元素追加到这个jQuery对象作为div.dcell的子元素。这个例子的实际效果见图7-5。如图所示，这段脚本先是把页面中的img元素移到了新建的div元素，然后再把这个新div元素追加到#row1元素中。



图7-5 使用appendTo方法



## 7.2.4 使用回调函数动态插入子元素



append和prepend方法也接受函数参数。如代码清单7-10所示，我们可以动态地挑选jQuery对象中的元素，为不同的元素添加不同子元素。



代码清单7-10 使用回调函数动态地插入子元素



```

...
<script type="text/javascript">
  $(document).ready(function() {

    var orchidElems = $("
```


```



```

$(orchidElems).add(lilyElems).css("border", "thick solid red");

$('div.drow').append(function(index, html) {
    if (this.id == "row1") {
        return orchidElems;
    } else {
        return lilyElems;
    }
});
});
</script>
...

```

jQuery对象中的每个元素都会调用这个函数一次。传递给这个函数的参数有两个，一个是当前元素的索引，一个是当前元素对应的HTML。另外，函数中的this对象被设置为对应当前元素的HTMLElement对象。这个函数返回值会被追加或者插入到当前元素内。我们可以返回一段HTML片段、一个或多个HTMLElement对象，或者一个jQuery对象。

在这个例子中，我事先为百合花（lily）和兰花（orchid）创建好了一些元素，然后在append方法的回调函数参数中（根据当前处理元素的id属性）返回相应的元素。这个例子的实际效果见图7-6。

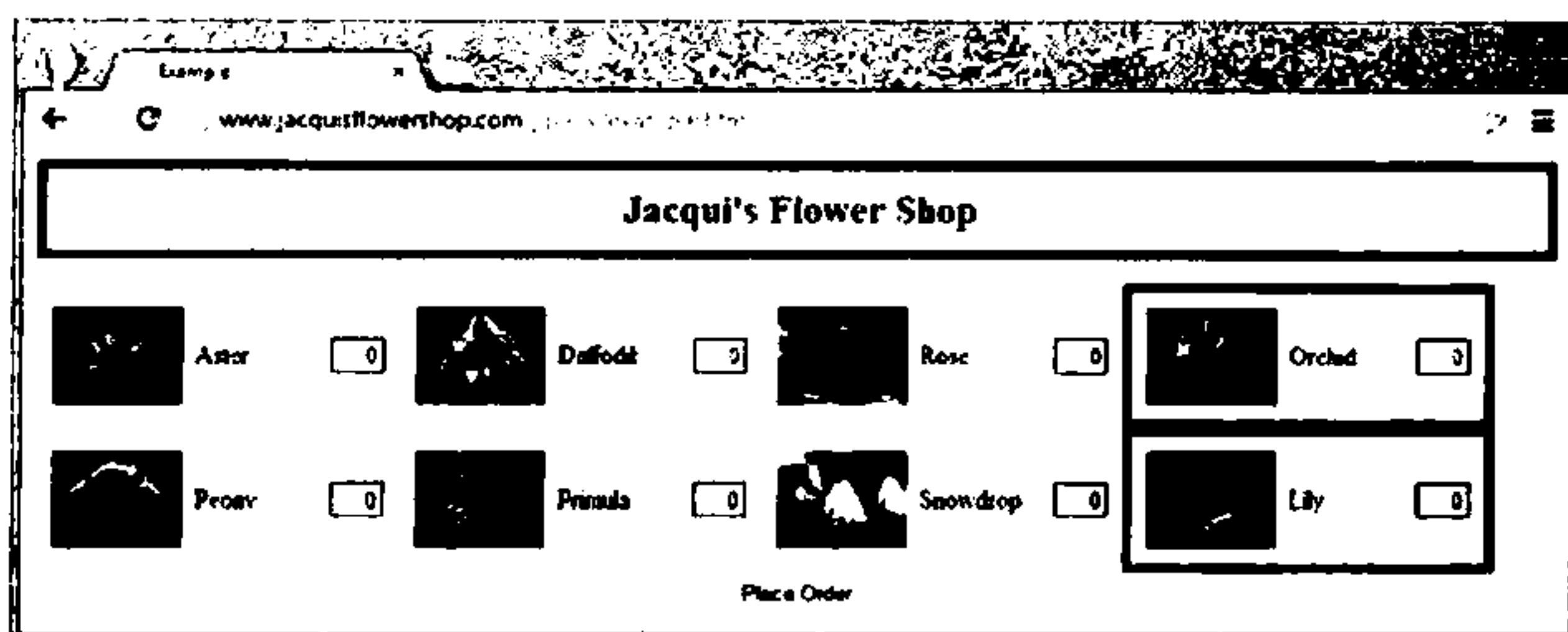


图7-6 使用回调函数动态插入子元素

## 7.3 封装（包裹）元素

jQuery还提供了一些把新元素作为现有元素的父元素或者祖先元素插入到页面的方法。这些方法又称为封装方法（因为是以一个元素包裹住另一个元素）。表7-3列出了这些方法。

表7-3 封装元素的方法

| 方 法  | 描 述                            |
|--|--------------------------------|
| wrap(HTML)、wrap(jQuery)、wrap(HTMLElement[])          | 使用参数元素封装jQuery对象内的每个元素         |
| wrapAll(HTML)、wrapAll(jQuery)、wrapAll(HTMLElement[]) | 把jQuery对象中的所有元素组合到一起，再使用参数元素封装 |

(续)

| 方 法   | 描 述                    |
|---|------------------------|
| <code>wrapInner(HTML)</code> 、 <code>wrapInner(jQuery)</code> 、 <code>wrapInner(HTMLElement[])</code> | 使用参数元素封装jQuery对象内元素的内容 |
| <code>wrap(function)</code> 、 <code>wrapInner(function)</code>  | 使用回调函数动态地封装元素          |

**提示** 与封装方法相对应的是反封装方法 (`unwrap`)，详见7.6节。

在实施封装时，我们可以作为参数传递多个元素，但一定要保证这些元素最多只能有一个内部元素，否则jQuery就会不知所措。换言之，参数中的每个元素至多有一个父元素并且至多有一个子元素。代码清单7-11演示了`wrap`方法的用法。

#### 代码清单7-11 `wrap`方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var newElem = $("<div/>").css("border", "thick solid red");
        $('div.drow').wrap(newElem);

    });
</script>
...
```

这段脚本生成了一个新的`div`元素，并使用`css`方法为它加了一个红色边框。然后，我使用`wrap`方法把这个`div`作为页面中所有`div.drow`元素的父元素插入到页面当中。这个脚本的实际效果见图7-7。

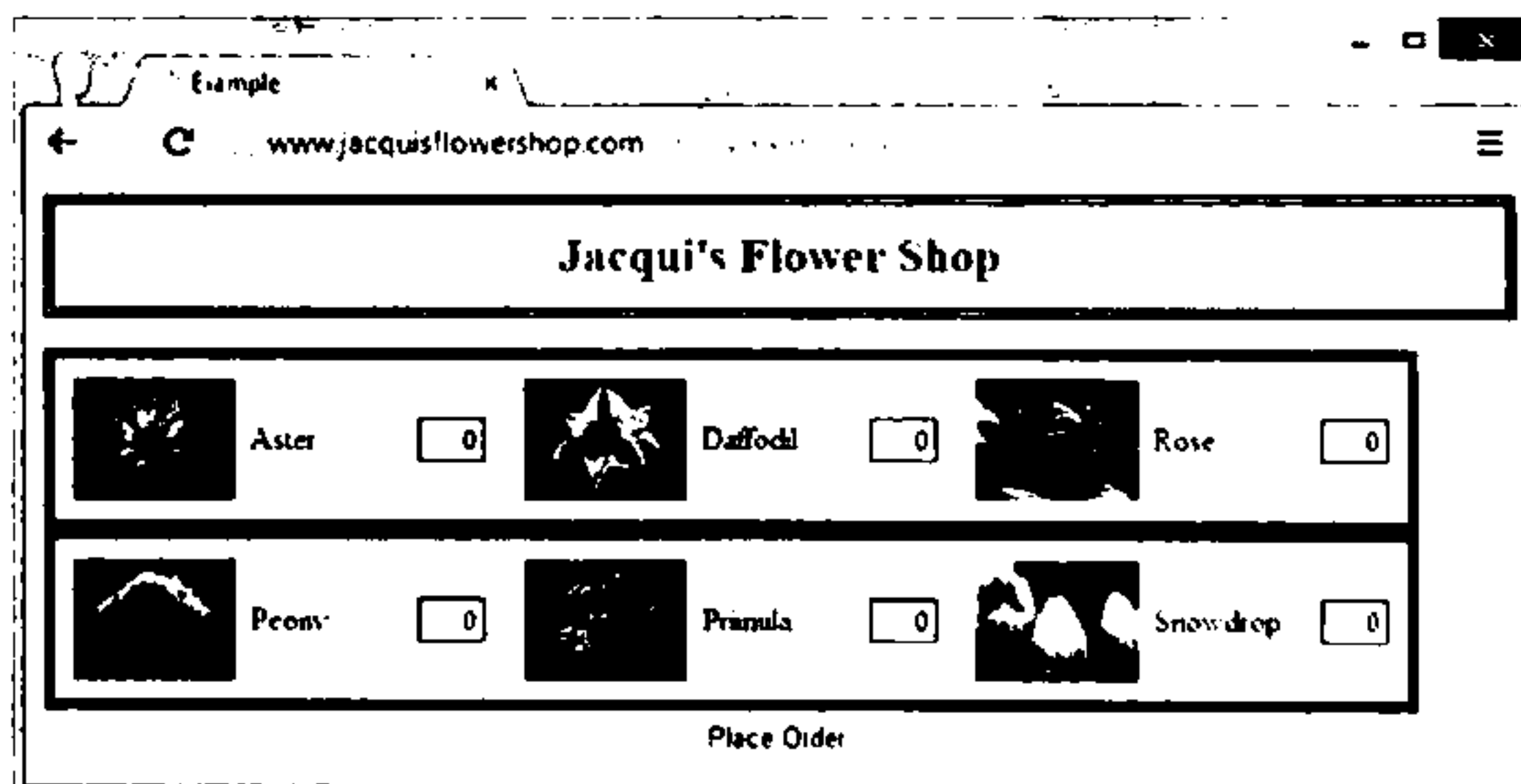


图7-7 使用`wrap`方法为元素添加父元素

jQuery会把参数元素插入到jQuery对象中每个元素与它们当前的父元素之间。因此，下面的HTML片段：

```
...
<div class="dtable">
```

```

    <div id="row1" class="drow">
        ...
    </div>
    <div id="row2" class="drow">
        ...
    </div>
</div>
...

```

就演变成:

```

...
<div class="dtable">
    <div style="...style properties...">
        <div id="row1" class="drow">
            ...
        </div>
    </div>
    <div style="...style properties...">
        <div id="row2" class="drow">
            ...
        </div>
    </div>
</div>
...

```

7

### 7.3.1 把多个元素封装到一个元素之中

当我们使用wrap方法，jQuery会克隆参数元素来封装jQuery对象中的每个元素，让它们都得到一个全新的父元素。如代码清单7-12所示，使用wrapAll方法可以实现用一个元素封装一堆元素的操作。

代码清单7-12 使用wrapAll方法

```

...
<script type="text/javascript">
    $(document).ready(function() {

        var newElem = $("<div/>").css("border", "thick solid red");
        $('div.drow').wrapAll(newElem);

    });
</script>
...

```

与上一个例子相比，我们只是把wrap方法换成了wrapAll。这个例子的具体效果见图7-8（图中花朵部分整体被红色边框包围）。

如图所示，参数元素成了选中元素共同的父元素，也就是说，脚本把HTML演变成了下面这个样子：

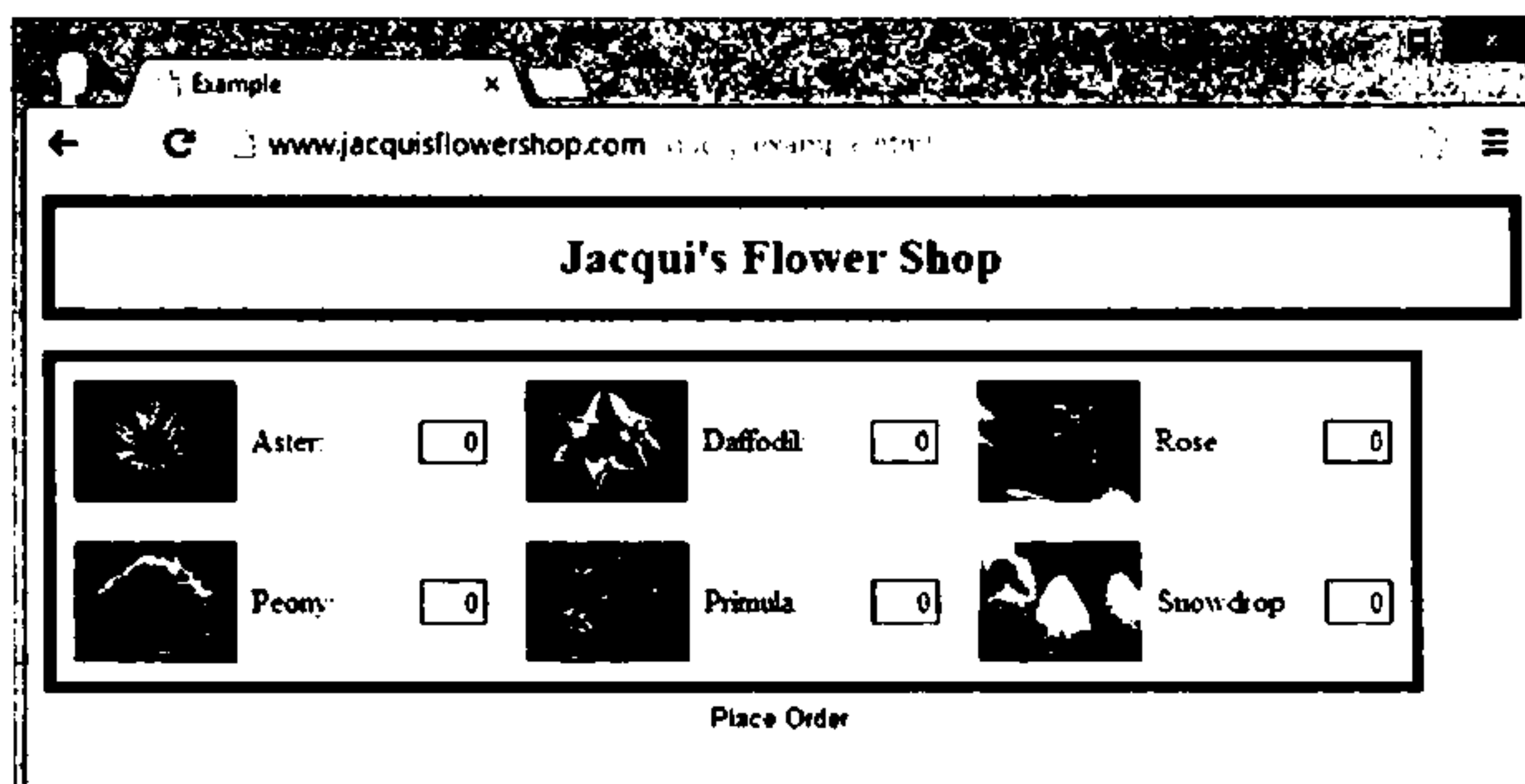


图7-8 应用wrapAll方法

```
...
<div class="dtable">
  <div style="...style properties...">
    <div id="row1" class="drow">
      ...
    </div>
    <div id="row2" class="drow">
      ...
    </div>
  </div>
</div>
...
```

要慎重使用这个方法。如果选中的元素本身不是兄弟元素（没有共同的父元素），参数元素就成为结果集中第一个元素的父元素，然后jQuery会移动所有结果集中的其他元素，让它们成为第一个元素的兄弟元素。代码清单7-13演示了wrapAll方法的这一特性。

#### 代码清单7-13 使用wrapAll方法封装没有共同父元素的元素

```
...
<script type="text/javascript">
  $(document).ready(function() {

    var newElem = $("<div/>").css("border", "thick solid red");
    $('img').wrapAll(newElem);

  });
</script>
...
```

我选取了页面中的所有img元素，它们的父元素各不相同。这个脚本产生的效果见图7-9。jQuery把新div元素作为aster图片的父元素插入到页面中，而把其他img元素当做aster图片的兄弟元素插入到页面中。

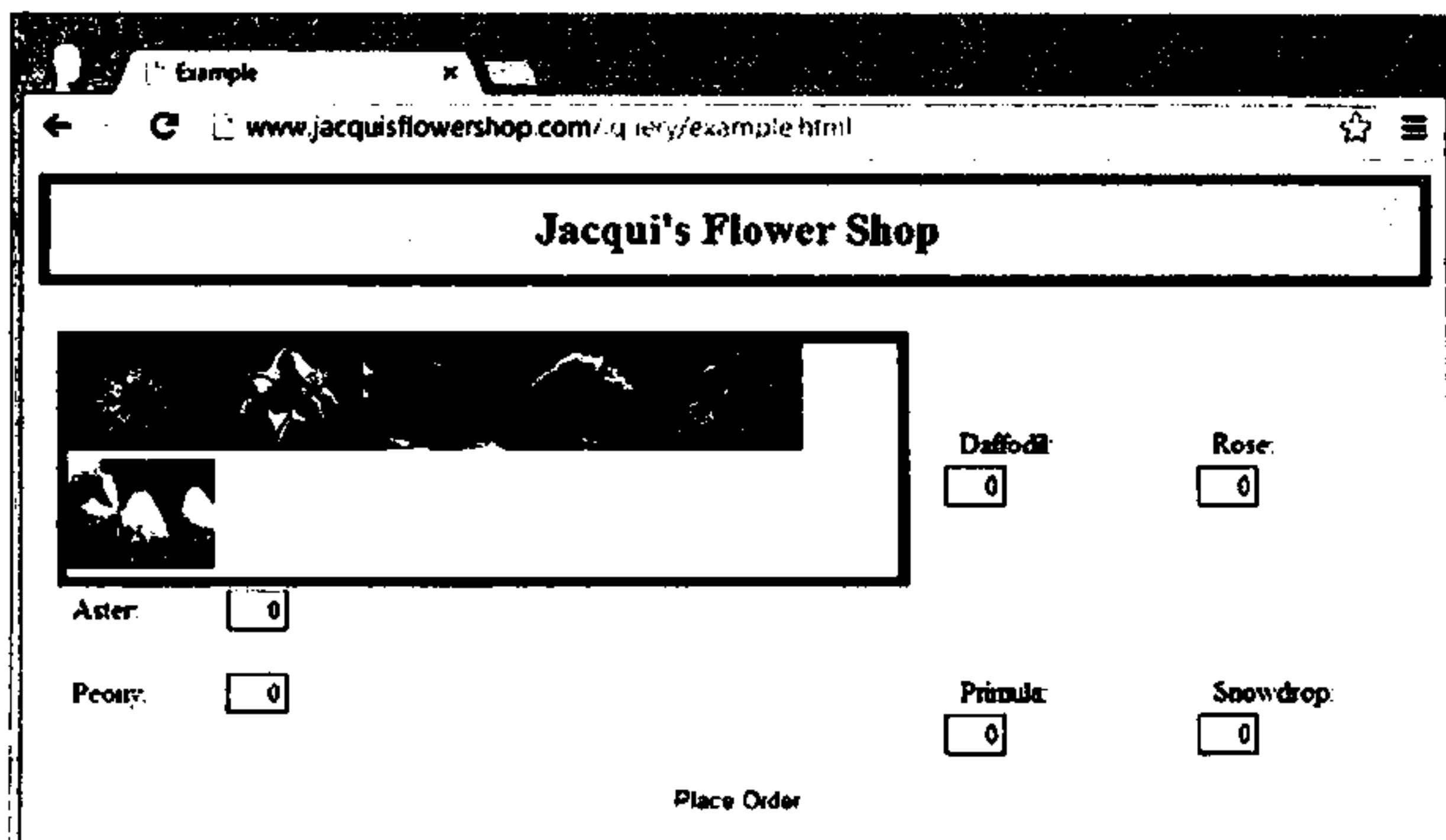


图7-9 使用wrapAll方法封装没有共同父元素的元素

7

### 7.3.2 封装元素的内容

wrapInner方法封装jQuery对象中元素的内容(而非元素本身)。代码清单7-14就是一个这样的例子。

代码清单7-14 使用wrapInner方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var newElem = $("<div/>").css("border", "thick solid red");
        $('.dcell').wrapInner(newElem);

    });
</script>
...
```

jQuery把参数元素插入到jQuery对象中元素与它们的内容之间。在这段脚本中，我先选中.dcell元素，然后用新div元素封装它们的内容。具体效果见图7-10。

使用append方法也可以达到wrapInner方法的效果。下面是与wrapInner方法等效的脚本，仅供参考：

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var newElem = $("<div/>").css("border", "thick solid red");
        $('.dcell').each(function(index, elem) {
            $(elem).append(newElem.clone().append($(elem).children()));
        });

    });
</script>
...
```

我并不建议这种做法（wrapInner方法可读性更好，使用起来更方便），不过这是一个使用jQuery可以有多种不同方式解决同一问题的好例子。

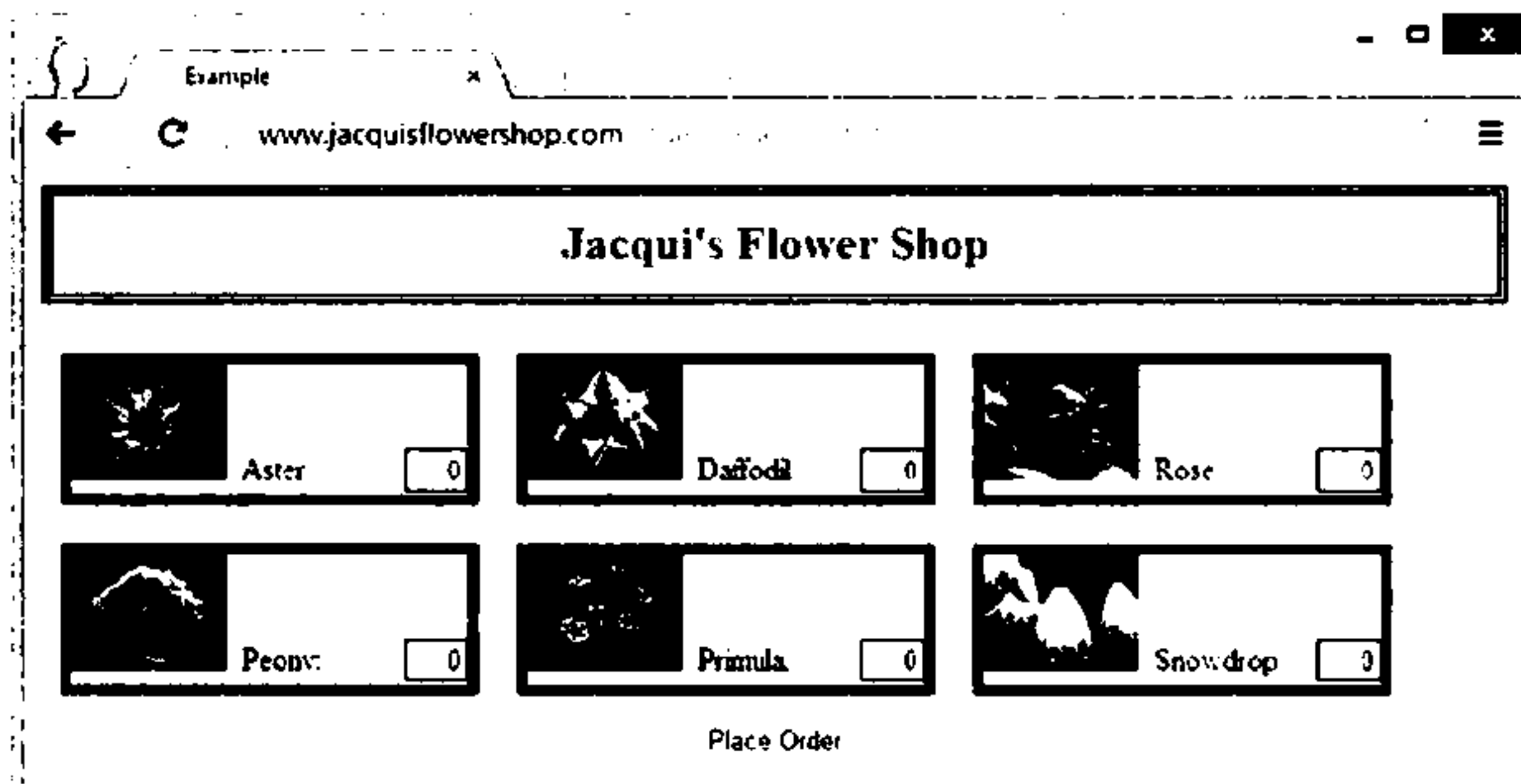


图7-10 应用wrapInner方法

### 7.3.3 使用回调函数封装元素

我们也可以传递一个函数给wrap或wrapInner方法以动态方式生成封装元素。针对结果集中的每一个元素都会调用一次这个函数，并把当前元素在结果集中的索引（序号从0开始）作为参数传递给这个函数。在该函数中特殊变量this被设置为当前正在处理的元素（对应的HTMLElement对象）。代码清单7-15演示了如何动态生成封装元素。

代码清单7-15 动态封装元素

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('.drow').wrap(function(index) {
            if ($(this).has('img[src*=rose]').length > 0) {
                return $("<div/>").css("border", "thick solid blue");
            } else {
                return $("<div/>").css("border", "thick solid red");
            }
        });

    });
</script>
...
```

在这个例子中，封装元素的样子由每个选中元素的后代元素决定。这个例子的实际效果见图7-11。

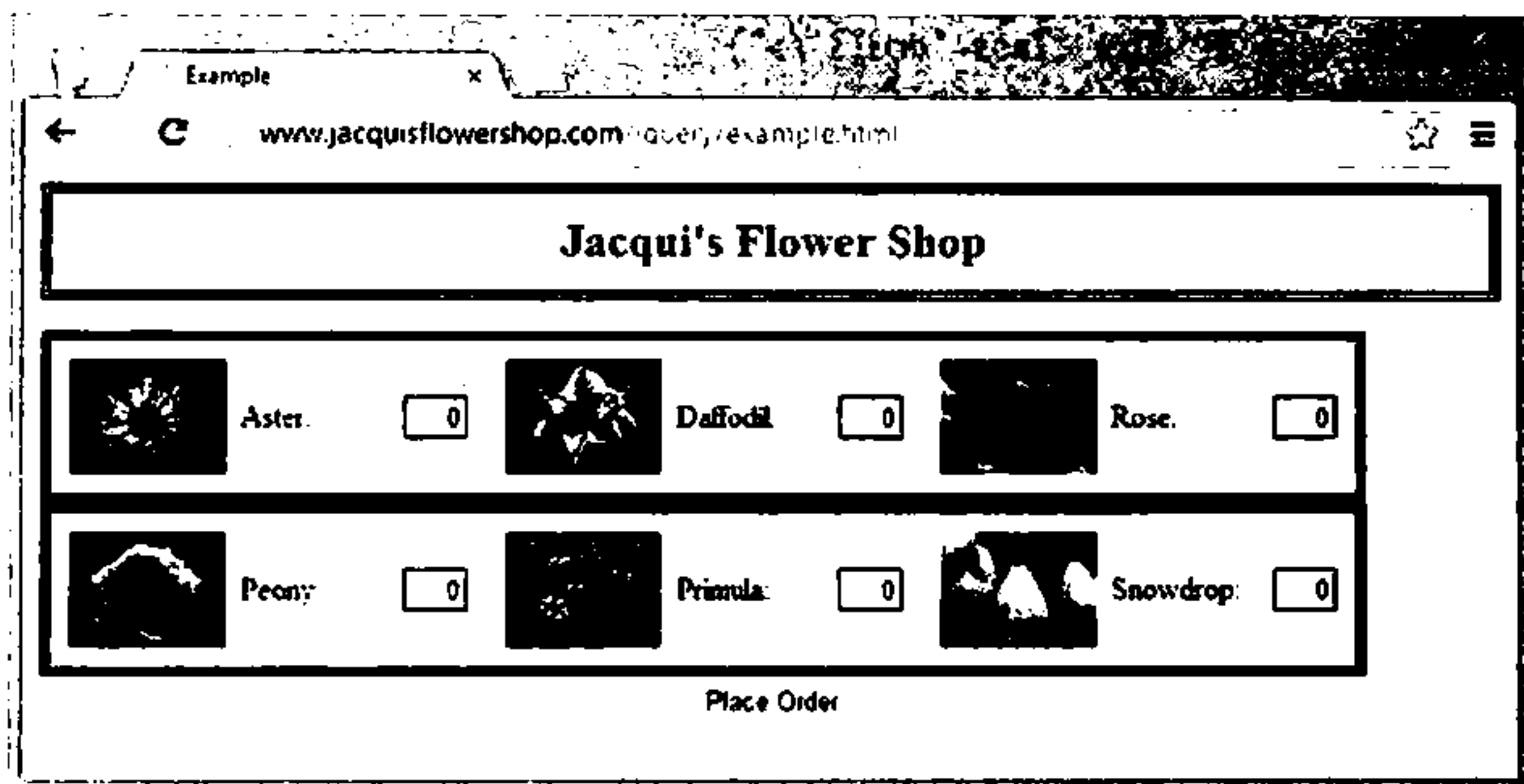


图7-11 以回调函数为参数调用wrap方法，以动态生成封装元素

## 7.4 插入兄弟元素

jQuery也提供了插入兄弟元素到页面其他元素旁的方法，见表7-4。

表7-4 插入兄弟元素的方法

方 法	描 述
after(content[,content])	把指定元素作为下一个兄弟元素添加到 jQuery对象中的每个元素之后
before(content[,content])	把指定元素作为上一个兄弟元素添加到jQuery对象中的每个元素之前
insertAfter(content[,content])	把jQuery对象中的元素作为下一个兄弟元素插入到参数指定的每个元素之后
insertBefore(content[,content])	把jQuery对象中的元素作为下一个兄弟元素插入到参数指定的每个元素之前
after(function) . before(function)	使用回调函数动态添加兄弟元素

当我们需要在页面中为某些元素添加兄弟元素时，before和after方法的工作模式一模一样。代码清单7-16演示了这两个方法的用法。

代码清单7-16 使用before和after方法

```
...
<script type="text/javascript">
  $(document).ready(function() {

    var orchidElems = $("<div class='dcell' />")
      .append("<img src='orchid.png' />")
      .append("<label for='orchid'>Orchid:</label>")
      .append("<input name='orchid' value='0' required />");
```

```

var lilyElems = $("<div class='dcell' />")
    .append("<img src='lily.png' />")
    .append("<label for='lily'>Lily:</label>")
    .append("<input name='lily' value='0' required />");

$(orchidElems).add(lilyElems).css("border", "thick solid red");

$("#row1 div.dcell").after(orchidElems);
$("#row2 div.dcell").before(lilyElems);

});
</script>
...

```

在这个脚本中，我为兰花（orchid）和百合花（lily）生成了一些元素，然后把它们作为兄弟元素插入到.dcell元素旁边。其中，兰花元素作为下一个兄弟元素插入到#row1 div.dcell元素之后，百合花元素作为上一个兄弟元素插入到#row2 div.dcell元素之前。这个脚本的实际效果见图7-12（其中6个花朵部分被红色边框包围）。

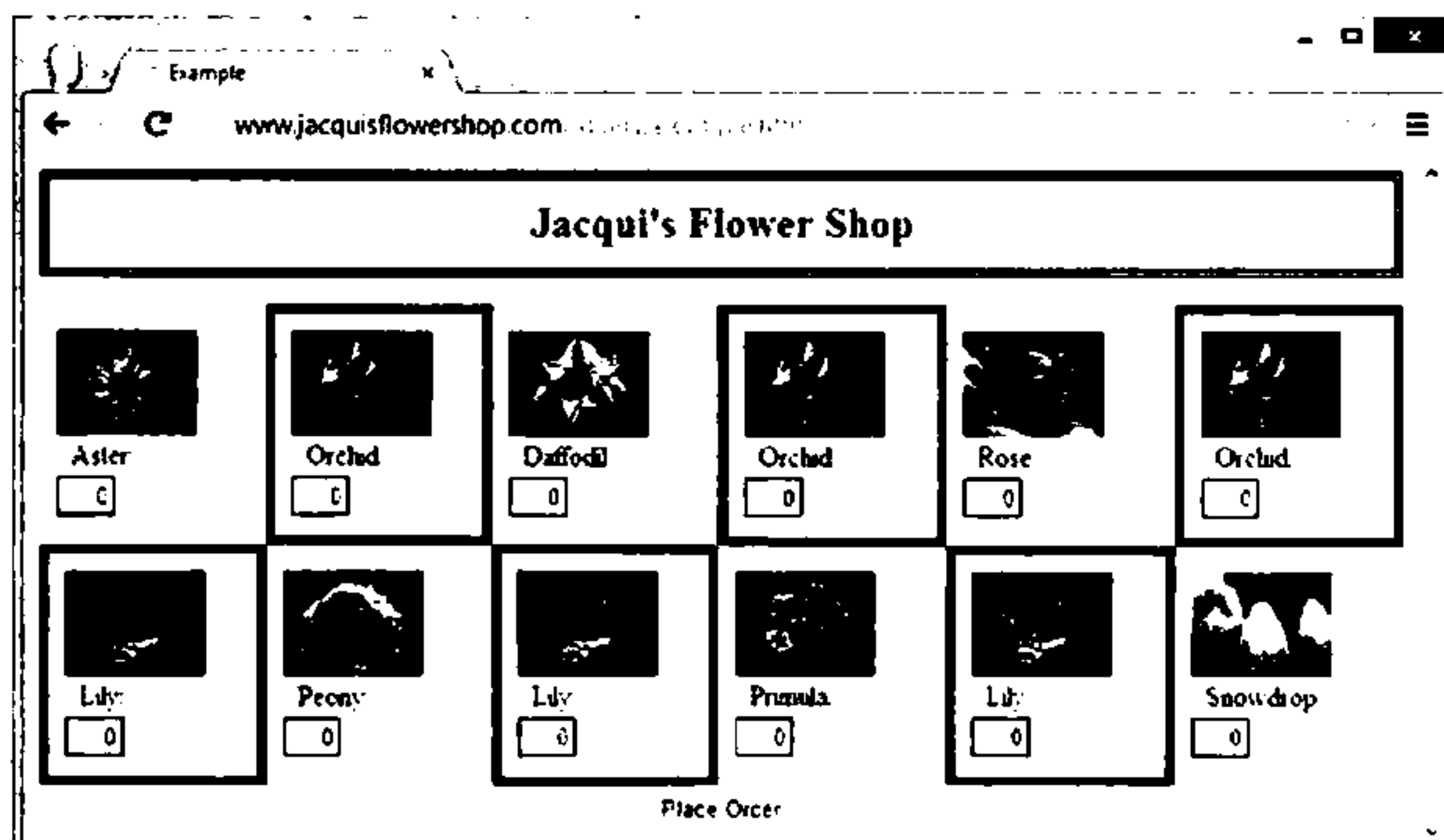


图7-12 使用before或after方法添加兄弟元素

### 7.4.1 把jQuery对象中的元素插入为兄弟元素

insertAfter与insertBefore方法把jQuery对象中的元素作为兄弟元素插入到参数元素之后或之前。和after与before方法一样，它们都是把一些元素插到另一些元素之后或者之前，只不过jQuery对象内元素与参数元素之间的关系正好相反。代码清单7-17演示了这些方法的用法。如图7-12所示，这个例子最终的效果与上例相同。

#### 代码清单7-17 insertAfter和InsertBefore方法

```

...
<script type="text/javascript">

```



```

$(document).ready(function() {

    var orchidElems = $("<div class='dcell' />")
        .append("<img src='orchid.png' />")
        .append("<label for='orchid'>Orchid:</label>")
        .append("<input name='orchid' value='0' required />");

    var lilyElems = $("<div class='dcell' />")
        .append("<img src='lily.png' />")
        .append("<label for='lily'>Lily:</label>")
        .append("<input name='lily' value='0' required />");

    $(orchidElems).add(lilyElems).css("border", "thick solid red");

    orchidElems.insertAfter('#row1 div.dcell');
    lilyElems.insertBefore('#row2 div.dcell');

});
</script>
...

```

### 7.4.2 使用回调函数动态插入兄弟元素

类似于我们前面对父元素和子元素的处理, after和before方法也支持使用回调函数动态插入兄弟元素。代码清单7-18演示了如何动态插入兄弟元素。

代码清单7-18 使用回调函数动态插入兄弟元素

```

...
<script type="text/javascript">
    $(document).ready(function() {

        $('#row1 div.dcell').after(function(index, html) {
            if (index == 0) {
                return $("<div class='dcell' />")
                    .append("<img src='orchid.png' />")
                    .append("<label for='orchid'>Orchid:</label>")
                    .append("<input name='orchid' value='0' required />")
                    .css("border", "thick solid red");
            } else if (index == 1) {
                return $("<div class='dcell' />")
                    .append("<img src='lily.png' />")
                    .append("<label for='lily'>Lily:</label>")
                    .append("<input name='lily' value='0' required />")
                    .css("border", "thick solid red");
            }
        });

    });
</script>
...

```

在本例中, 我根据jQuery对象内元素的索引index(只处理第0个和第1个元素)生成不同兄弟元素。这个例子的实际效果见图7-13。

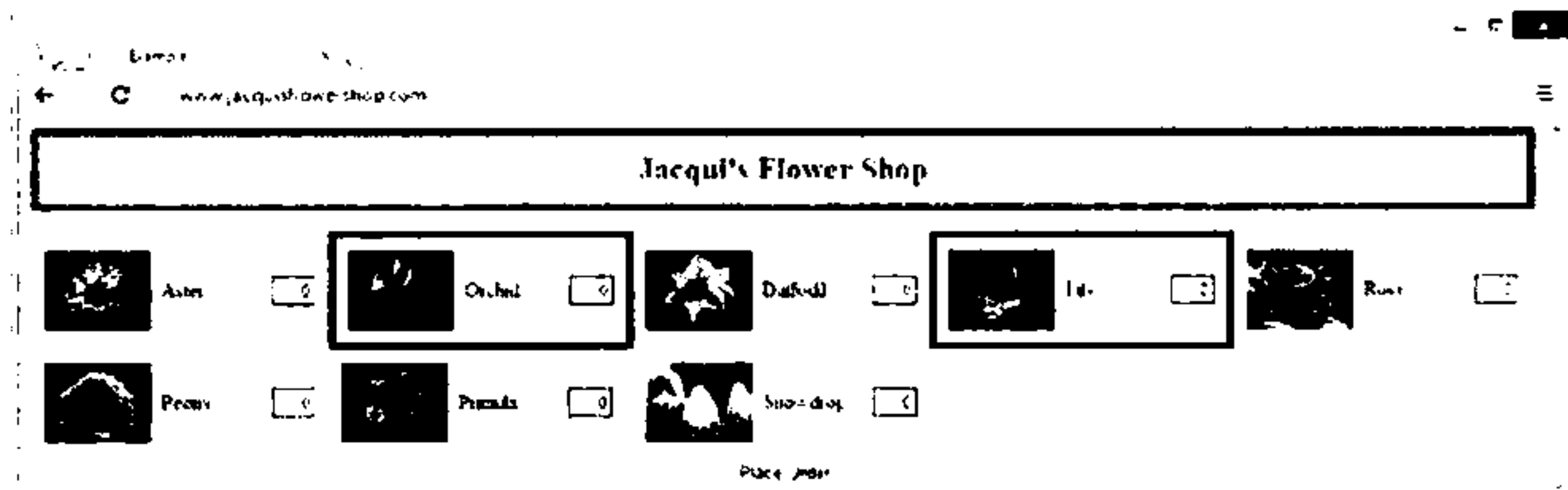


图7-13 使用回调函数动态插入兄弟元素

## 7.5 替换元素

表7-5中列出的这些方法可使用另一些元素替换已有元素。

表7-5 替换元素的方法

方 法	描 述
<code>replaceWith(HTML)</code> 、 <code>replaceWith(jQuery)</code> 、 <code>replaceWith(HTMLElement[])</code>	使用指定内容替换jQuery中的元素
<code>replaceAll(jQuery)</code> 、 <code>replaceAll(HTMLElement[])</code>	使用jQuery对象中的元素替换参数元素
<code>replaceWith(function)</code>	使用回调函数动态替换元素

`replaceWith`和`replaceAll`方法的作用都是替换元素，不同在于jQuery对象内元素与参数元素的角色正好相反。代码清单7-19演示了这两个方法的使用。

代码清单7-19 `replaceWith`和`replaceAll`方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var newElems = $("<div class='dcell' />")
            .append("<img src='orchid.png' />")
            .append("<label for='orchid'>Orchid:</label>")
            .append("<input name='orchid' value='0' required />")
            .css("border", "thick solid red");

        $('#row1').children().first().replaceWith(newElems);
        $("<img src='carnation.png' />").replaceAll('#row2 img')
            .css("border", "thick solid red");

    });
</script>
...
```

在本例中，利用`replaceWith`方法我们使用新的内容换掉`#row1`元素中的第一个子元素（结果是使用兰花换掉了未分类花卉）。我还使用了`replaceAll`方法把`#row2`元素的所有`img`子元素都换成了康乃馨（`carnation`）。这个脚本的具体效果见图7-14。

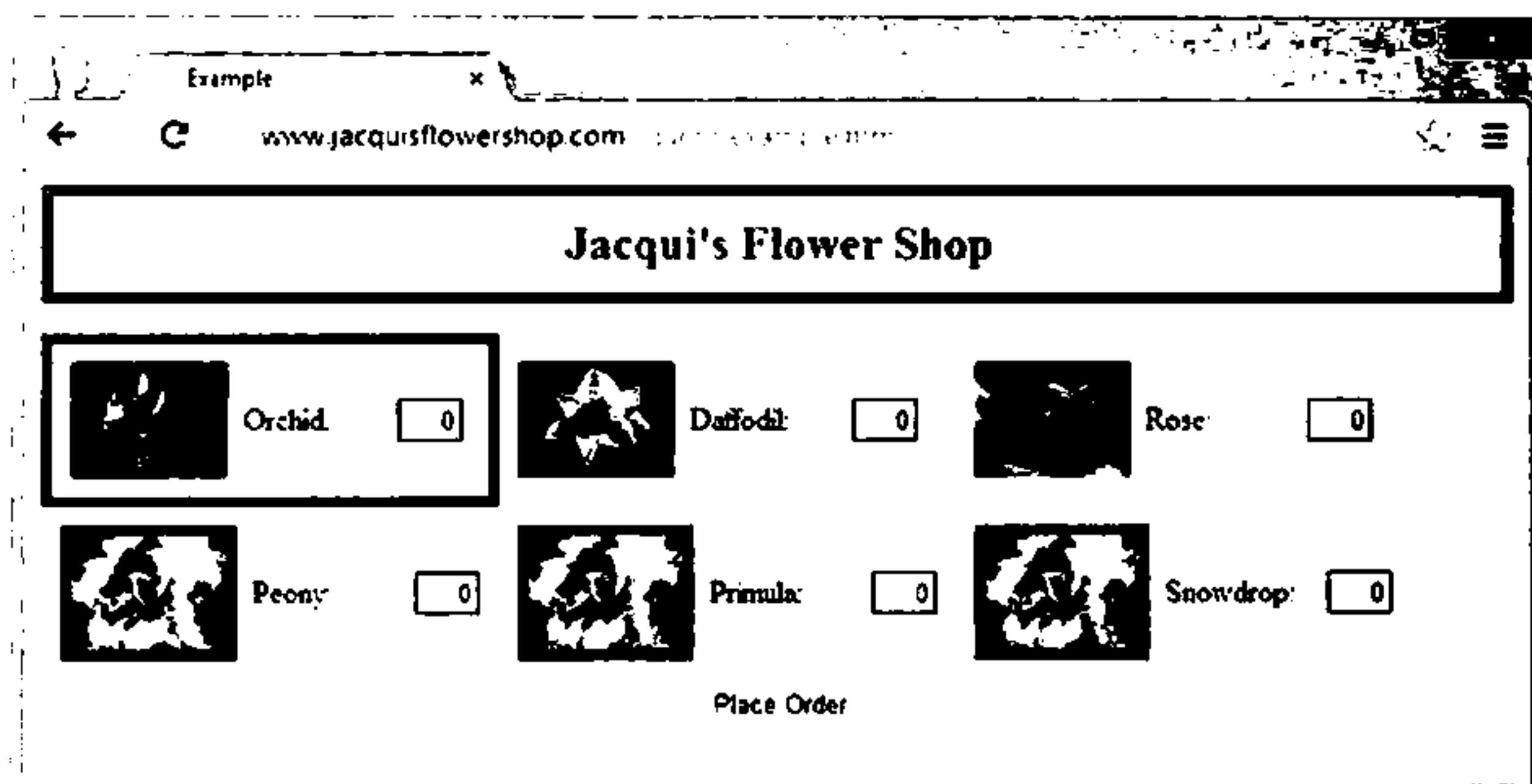


图7-14 使用replaceWith和replaceAll方法替换元素

## 使用回调函数动态替换元素

7

replaceWith方法也支持使用回调函数动态地替换元素。回调函数不接受任何参数，不过this变量被设置为对应当前正在处理元素的HTMLElement对象。代码清单7-20演示了这一方法的使用。

代码清单7-20 使用回调函数动态替换元素

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $('div.drow img').replaceWith(function() {
            if (this.src.indexOf("rose") > -1) {
                return $("<img src='carnation.png' />").css("border", "thick solid red");
            } else if (this.src.indexOf("peony") > -1) {
                return $("<img src='lily.png' />").css("border", "thick solid red");
            } else {
                return $(this).clone();
            }
        });
    });
</script>
...
```

在本例中，我基于img元素的src属性替换元素。如果src属性值包含rose字样，我就把它替换成carnation.png（康乃馨）。如果src属性值包含peony字样，我就把它换成lily（水仙）。新元素都带着红色边框，在页面中非常醒目。

对于其他所有元素，代码返回当前处理元素的一个副本（克隆），从而实现用自己的副本代替自己的效果。这个脚本的具体效果见图7-15。

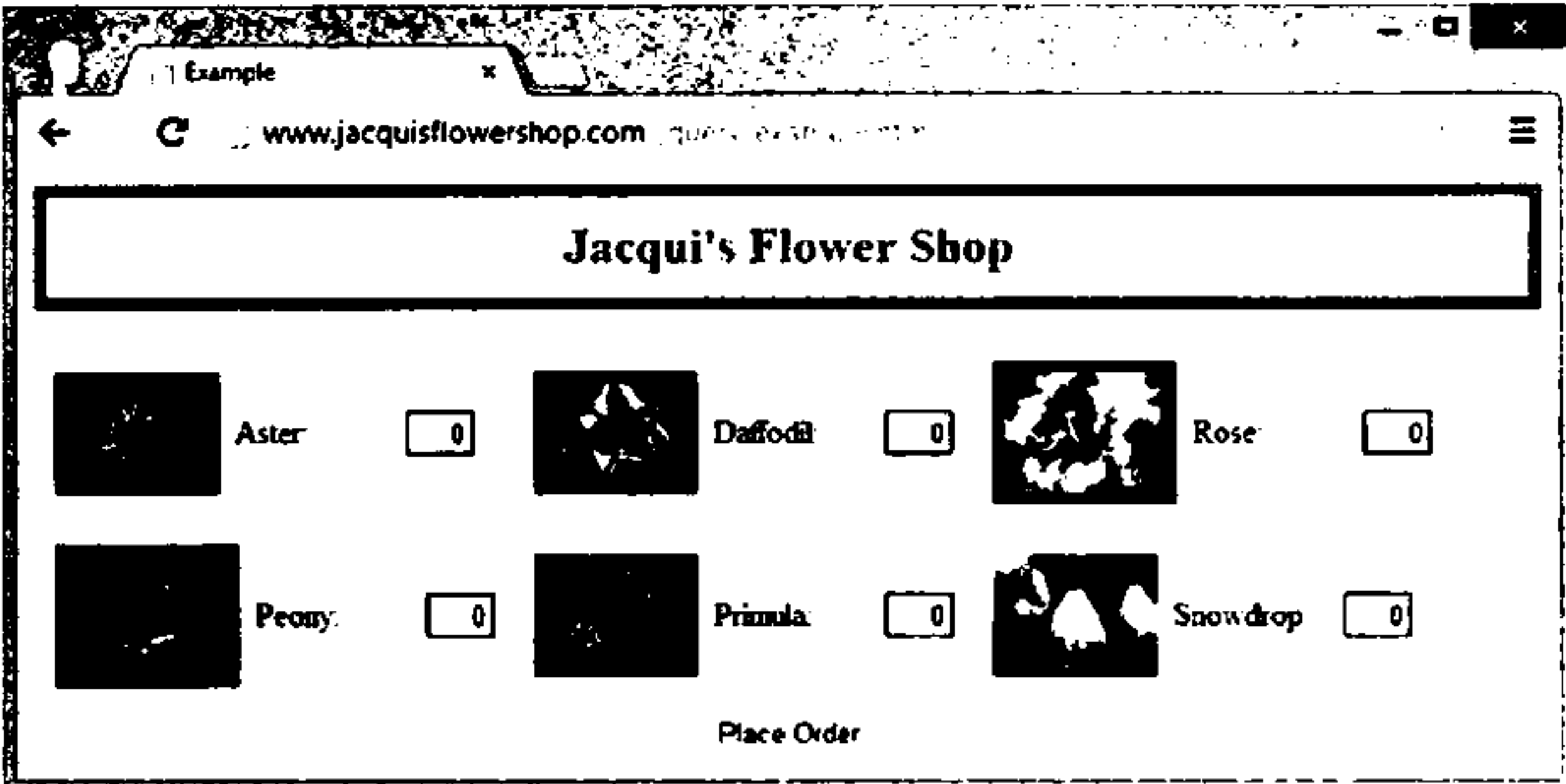


图7-15 使用回调函数动态替换元素

**提示** 如果你不想替换一个元素，就返回一个它的副本。如果没有返回副本，jQuery会删除这个元素。当然，我们也可以通过事先过滤要替换的元素避免这个问题，然而有些时候无法做到这一点。

### 7.6 删除元素

作为对插入元素、替换元素的补充，jQuery还提供了一些从DOM中删除元素的方法，见表7-6。

表7-6 删除元素的方法

方 法	描 述
<code>detach()</code> 、 <code>detach(selector)</code>	从DOM中删除元素并保留关联在元素上的数据
<code>empty()</code>	删除jQuery对象中所有元素的子元素
<code>remove()</code> 、 <code>remove(selector)</code>	从DOM中删除元素，不保留关联在元素上的数据
<code>unwrap()</code>	删除jQuery对象内每个元素的父元素

代码清单7-21演示如何使用remove方法从DOM中删除元素。

代码清单7-21 使用remove方法从DOM中删除元素

```
...
<script type="text/javascript">
  $(document).ready(function() {
    $("img[src*=daffodil], img[src*=snow]").parent().remove();
  });
</script>
...
```

这段脚本先选中src属性中包含daffodil或者snow的img元素，再获取它们的父元素，最后删除这些父元素。如代码清单7-22所示，我们也可以给remove方法提供一个选择器参数过滤打算要删除的元素。

## 代码清单7-22 只删除匹配选择器参数的元素

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $("div.dcell").remove(":has(img[src*=snow], img[src*=daffodil])");
    });
</script>
...

```

如图7-16所示，这段脚本与上一段脚本的效果完全相同。

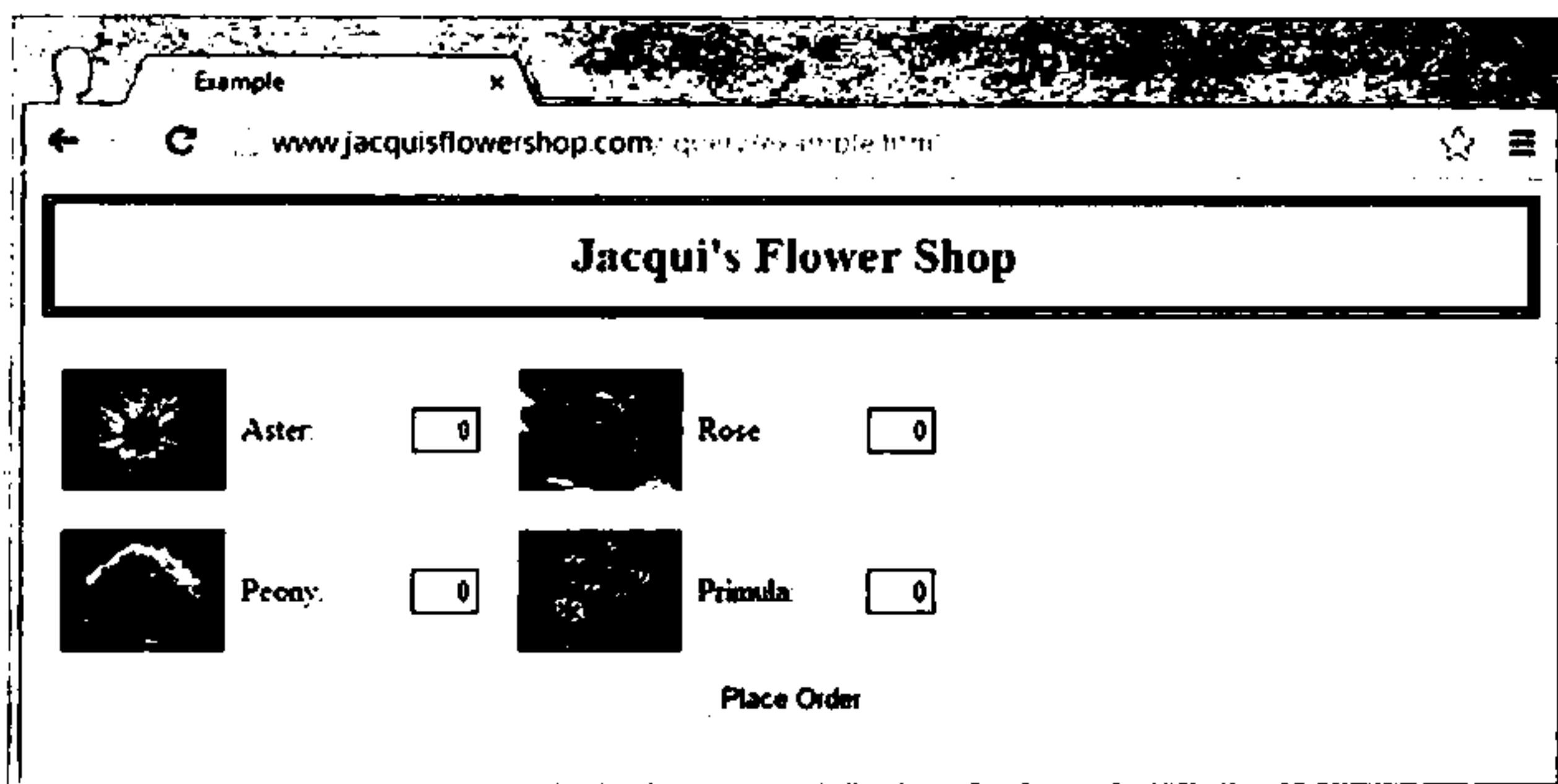


图7-16 从DOM中删除元素

**提示** remove方法返回的jQuery对象中包含着最初的选择结果集，也就是说，无法根据这个方法的返回结果判断目标元素是否已被删除。

### 7.6.1 分离元素

detach方法与remove方法的工作方式完全相同，只不过该方法会保留那些关联在被删除元素上的数据。我会在第8章细致讲解元素关联数据方面的知识。就本章来说，你只要知道detach方法通常是最好的删除方法就够了，如果我们还打算把删除的元素再插入到页面的某个地方的话。代码清单7-23演示了detach方法的用法。

**代码清单7-23 使用detach方法删除元素并保留与元素关联的数据**

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $("#row2").append($("#img[src*=aster]").parent().detach());
    });
</script>
...

```

上面这个脚本先是删除了src属性包含aster字样的img元素的父元素,然后又使用本章前面讲过的append方法把删除掉的元素追加到#row2元素的末尾。由于在这个地方即使不使用detach方法也能达到同样的效果,我很不情愿地在这里使用了detach方法(纯粹是想演示detach方法的用法)。我们完全可以把上面脚本中最关键的那条语句改写成下面这样:

```
...
$("#row2").append($("#img[src*=aster]").parent());
...
```

这个脚本的具体效果见图7-17。

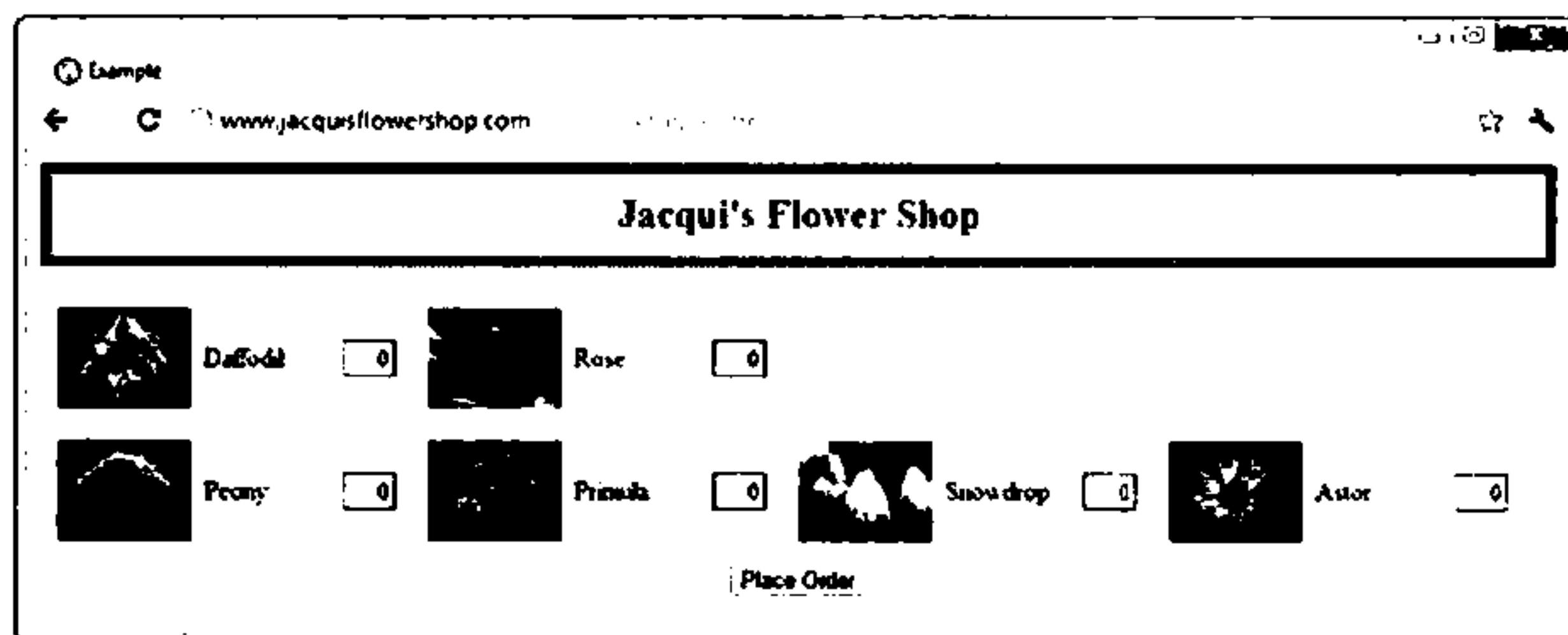


图7-17 使用detach方法

## 7.6.2 清空元素

empty方法删除jQuery对象中所有元素的后代元素,如代码清单7-24演示的那样,这些元素本身仍留在页面中。

### 代码清单7-24 使用empty方法

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#row1").children().eq(1).empty().css("border", "thick solid red");
    });
</script>
...
```

在这个脚本中,我选中#row1元素的第二个子元素(index值为1),然后调用empty方法。为了让变更清晰可见,我使用css方法为这个元素加上了红色边框。脚本的效果见图7-18。

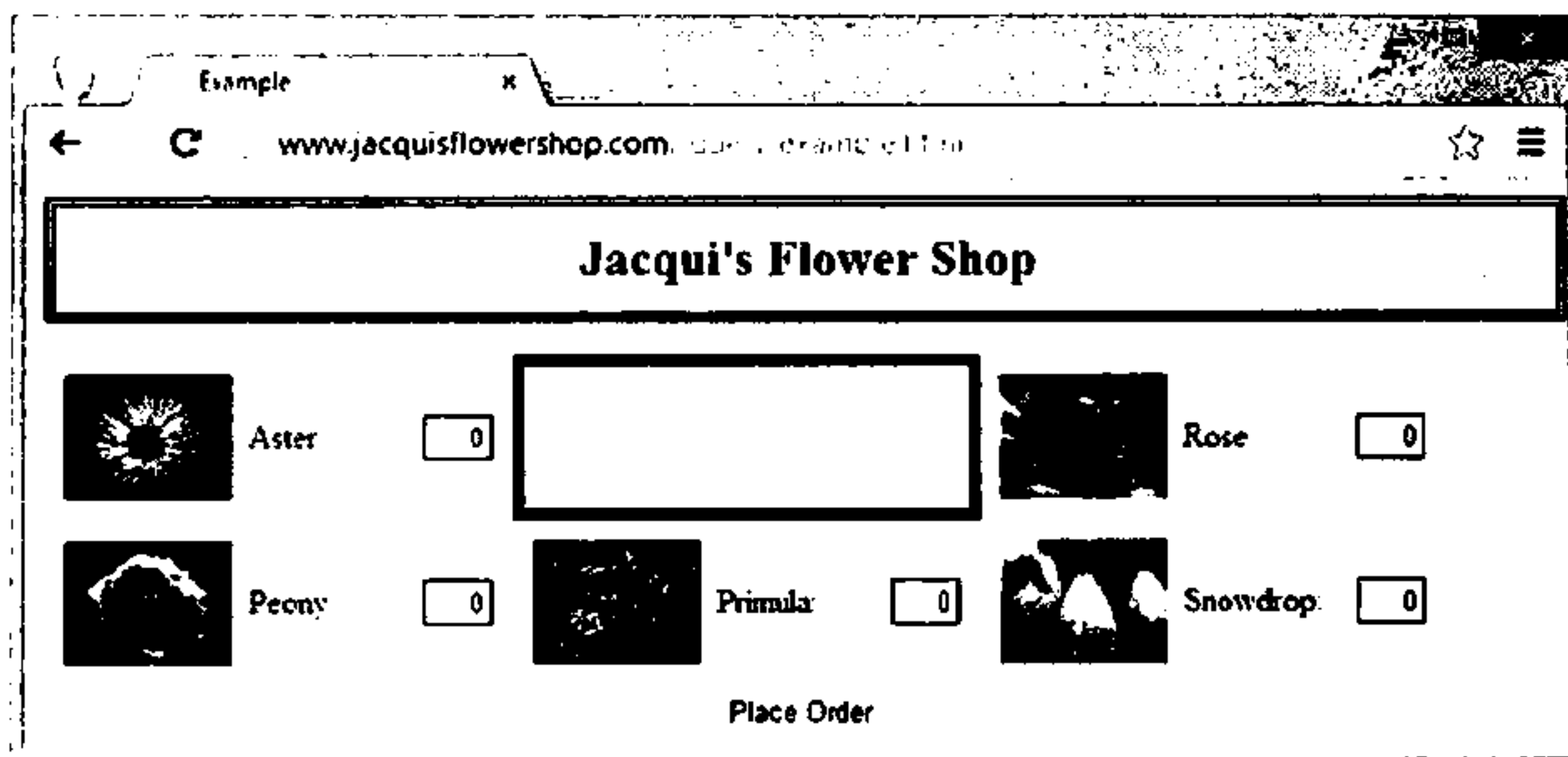


图7-18 empty方法的作用效果

### 7.6.3 删除元素的父元素

unwrap方法用来删除jQuery对象内元素的父元素。unwrap方法成功执行之后，被选中的元素自动成为祖父元素的子元素。代码清单7-25演示了unwrap方法的用法。

代码清单7-25 unwrap方法

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $(".dcell").unwrap();
    });
</script>
...
```

在这个脚本中，我先选中具有dcellclass的div元素，然后调用unwrap方法。结果#row1和#row2元素被删除，如图7-19所示。

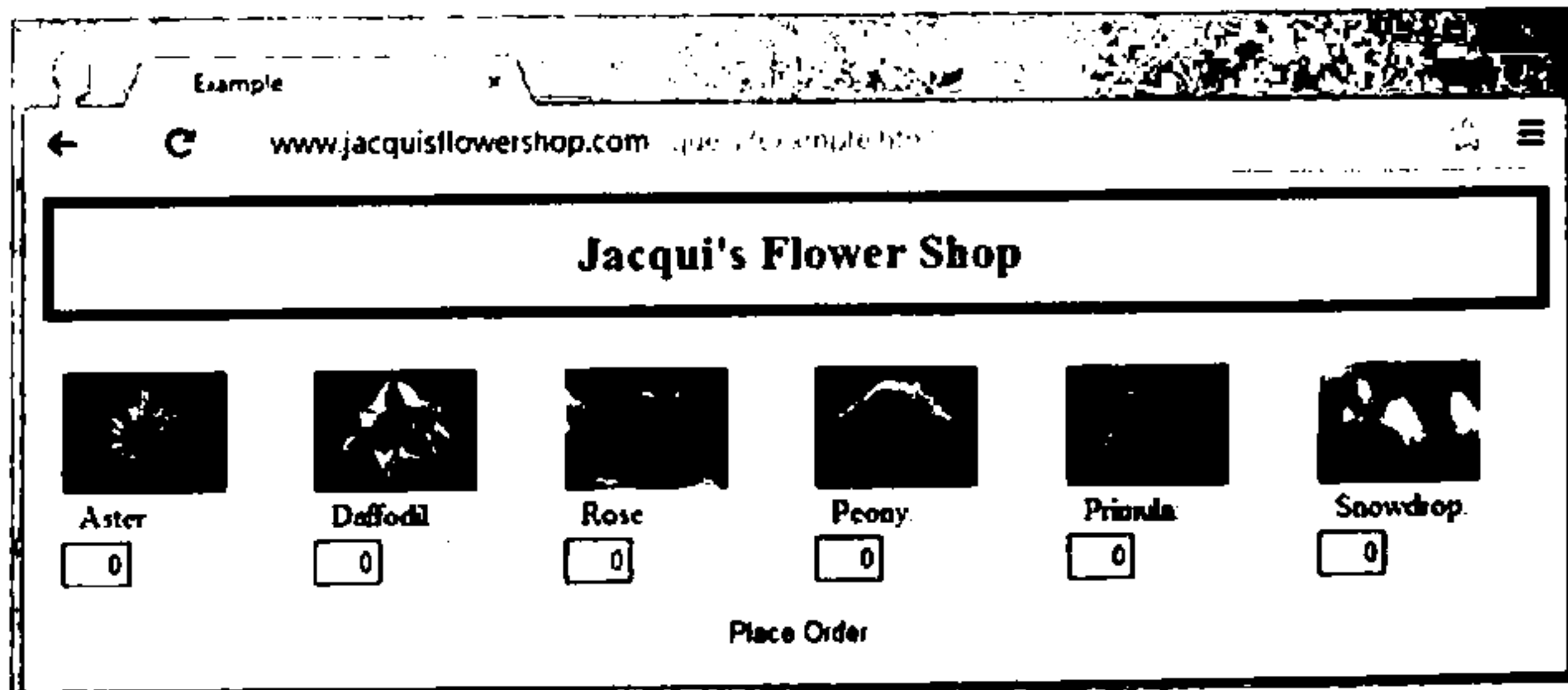


图7-19 应用unwrap方法的效果

## 7.7 小结

本章演示了使用jQuery处理DOM的技术，包括如何创建新元素，以及把新元素或者已有元素作为子元素、父元素、兄弟元素插入到页面的种种方法。还演示了在页面中如何移动元素和删除元素。下一章将介绍如何利用jQuery处理DOM元素。





在本章中，我们一起学习用jQuery处理元素的方法，包括获取和设置元素的属性、使用jQuery便捷方法处理样式类和CSS属性、获取和设置元素的HTML或文本内容。我还会介绍一个特别棒的功能，它用于关联数据与元素。表8-1列出了本章概要。

表8-1 本章概要

问 题	解决方法	代码清单
如何得到jQuery对象中第一个元素的某个属性的值	使用attr方法	1
如何得到jQuery对象中每个元素某个属性的值	综合使用each方法和attr方法	2
如何为jQuery对象中的每个元素设置某个属性的值	使用attr方法，属性值可以是一个回调函数	3
如何一次调用设置多个属性	使用attr方法并将一个映射对象用作它的参数	4、5
如何删除属性	使用removeAttr方法	6
如何读取或设置HTMLElement对象中定义的那些属性	使用attr方法的“影子”方法prop方法	7
如何控制元素的class属性	使用addClass、hasClass、removeClass，参数可以是一个回调函数	8~10
如何切换元素的class属性值	使用toggleClass方法	11~16
如何设置元素style属性的内容	使用css方法	17~21
如何得到元素详细的位置信息	使用处理CSS相关属性的专用方法	22~24
如何获取和设置元素的文本或者HTML内容	使用text或者html方法	25~27
如何获取和设置表单元素的值	使用val方法	28~30
如何为元素挂接上与它关联的数据	使用data方法	31

#### 新版jQuery中与本章有关的变化

jQuery 1.9/2.0升级了css方法，新版css方法允许一次获取多个css属性，8.3.2节详细介绍了这一特性。

在jQuery 1.9/2.0中，jQuery强迫attr方法和prop方法各行其事，二者的区别越来越大。在早期版本中，jQuery允许在应该调用prop方法的地方使用attr方法，以便兼容更早版本（小于1.6的版本）的jQuery。prop方法正是在jQuery 1.6版本中引入的。

本版jQuery的另一个变化是允许使用attr方法设置input元素的type属性（如果浏览器支持的话）。要慎重使用这一功能，在那些不支持这种做法的浏览器上（包括旧版本的IE浏览器），jQuery会抛出异常。如果需要不同type的input元素，我建议直接使用新元素替掉旧元素从而完全避免这个问题。

## 8.1 处理元素字面属性和元素对象定义属性

jQuery提供了一系列处理元素属性的方法，见表8-2。

表8-2 处理元素属性的方法

方 法	描 述
<code>attr(name)</code>	得到jQuery对象中第一个元素的name对应的属性的值
<code>attr(name, value)</code>	把jQuery对象中所有元素的name属性的值设置为value
<code>attr(map)</code>	为jQuery对象中的所有元素设置由map对象指定的属性和值
<code>attr(name, function)</code>	把jQuery对象中所有元素的name属性的值设置为回调函数function的返回值
<code>removeAttr(name)</code> 、 <code>removeAttr(name[])</code>	删除jQuery对象中所有元素的name属性
<code>prop(name)</code>	得到jQuery对象中第一个元素的name对应的由元素对象定义的属性的值
<code>prop(name, value)</code> 、 <code>prop(map)</code>	设置jQuery对象中所有元素的name属性或由map对象定义的多个属性的值
<code>prop(name, function)</code>	把jQuery对象中所有元素的由元素对象定义的name属性的值设置为回调函数function的返回值
<code>removeProp(name)</code>	删除jQuery对象中所有元素的name属性

调用attr方法时，如果只提供一个参数，它就会返回jQuery对象中第一个元素的参数对应属性的值（参见代码清单8-1）。

### 代码清单8-1 读取一个属性的值

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script type="text/javascript">
    $(document).ready(function() {
      var srcValue = $('img').attr('src');
      console.log("Attribute value: " + srcValue);
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
        <div id="row1" class="drow">
          <div class="dcell">
            <label for="aster">Aster:</label>
            <input name="aster" value="0" required />
          </div>
          <div class="dcell">
            <label for="daffodil">Daffodil:</label>
```

```

        <input name="daffodil" value="0" required />
    </div>
    <div class="dcell">
        <label for="rose">Rose:</label>
        <input name="rose" value="0" required />
    </div>
</div>
<div id="row2" class="drow">
    <div class="dcell">
        <label for="peony">Peony:</label>
        <input name="peony" value="0" required />
    </div>
    <div class="dcell">
        <label for="primula">Primula:</label>
        <input name="primula" value="0" required />
    </div>
    <div class="dcell">
        <label for="snowdrop">Snowdrop:</label>
        <input name="snowdrop" value="0" required />
    </div>
</div>
</div>
<div id="buttonDiv"><button type="submit">Place Order</button></div>
</form>
</body>
</html>

```

在这个脚本中，我先是选中页面中所有的img元素，然后使用attr方法获取src属性的值。当我们使用attr方法读取属性值的时候，它的返回值是一个字符串，而我把它输出到了控制台。这个脚本的输出如下：

---

```
Attribute value: aster.png
```

---

我们可以结合each方法使用attr方法读取jQuery对象中每一个元素的属性。我在第5章讲解过each方法，而代码清单8-2演示了此种情况下each方法的用法。

**代码清单8-2 使用each和attr方法读取多个对象的属性值**

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $('img').each(function(index, elem) {
            var srcValue = $(elem).attr('src');
            console.log("Attribute value: " + srcValue);
        });
    });
</script>
...

```

在这个脚本中，我把HTMLElement对象作为参数传递给\$函数从而得到新的jQuery对象。这个对象

仅包含一个元素，完美满足attr方法的要求。这个脚本的输出如下：

---

```
Attribute value: aster.png
Attribute value: daffodil.png
Attribute value: rose.png
Attribute value: peony.png
Attribute value: primula.png
Attribute value: snowdrop.png
```

---

### 8.1.1 设置属性值

使用attr方法设置属性值时，jQuery对象中所有元素的这一属性都会更改。这与该方法在读取属性时仅返回第一个元素值的行为形成鲜明对比。代码清单8-3演示了如何设置属性。

代码清单8-3 设置属性值

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("img").attr("src", "lily.png");
    });
</script>
...
```

---

**提示** 当我们设置属性值的时候，attr方法的返回值是jQuery对象。这意味着，这种情况下我们能够进行链式调用。

---

在这段脚本中，我选中了所有的img元素并把src属性设置为lily.png。这一条语句影响了所有选中的元素，具体效果见图8-1。

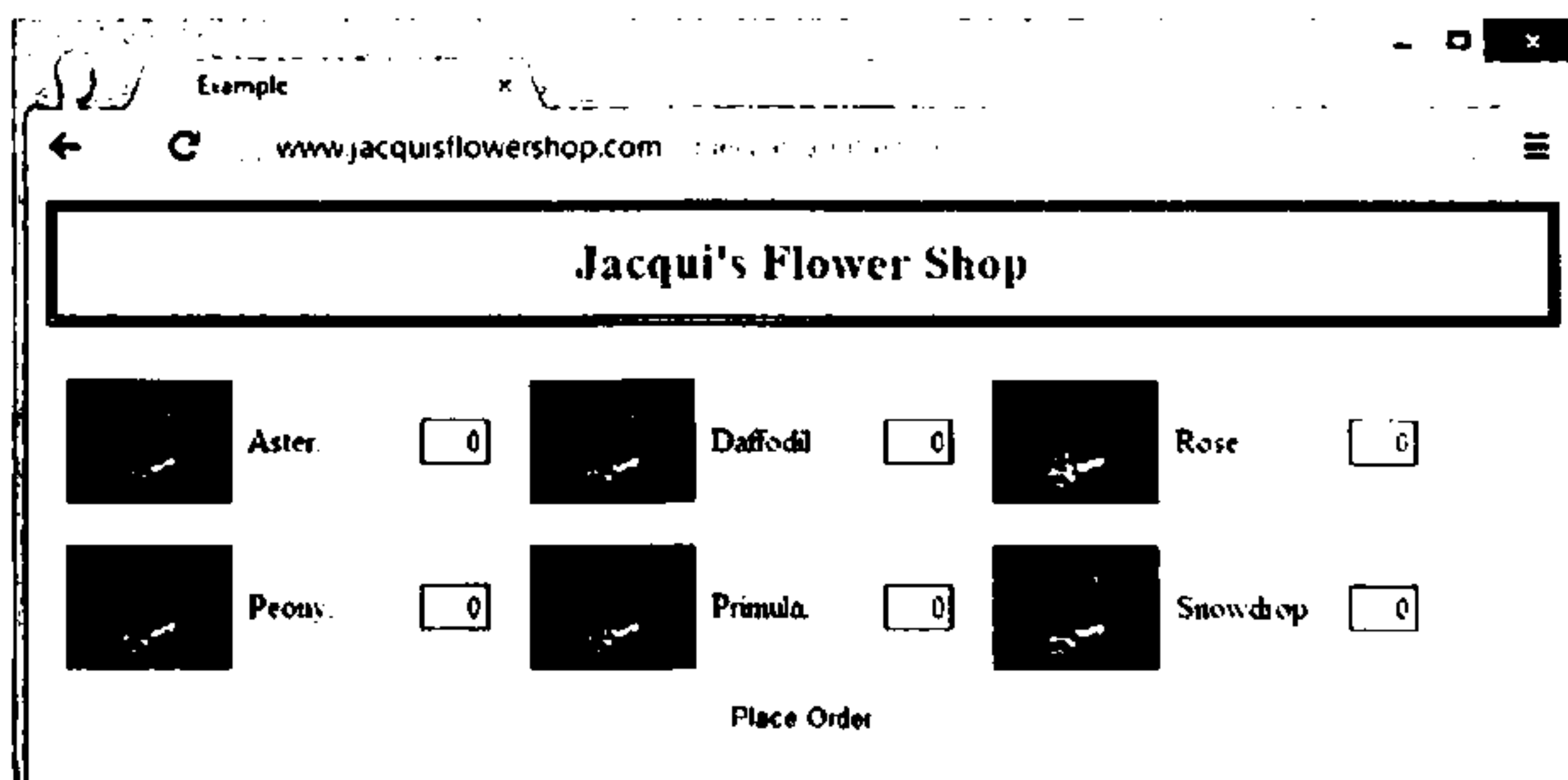


图8-1 把多个元素的某个属性设置为同一个值

### 8.1.2 一次设置多个属性

传递一个对象给attr方法，我们就可以做到在一次attr方法调用中设置多个属性。这个对象的属性名被解释为元素的属性名，对象的属性值则被用作元素的属性值。这种对象又叫映射对象。代码清单8-4给出了一个这样的示例。

代码清单8-4 使用映射对象一次设置多个属性

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var attrValues = {
            src: "lily.png",
            style: "border: thick solid red"
        };

        $("img").attr(attrValues);
    });
</script>
...
```

在这个脚本中我创建了一个映射对象，它有两个属性，分别是src和style。我选中了页面中所有的img元素，然后以这个映射对象为参数调用attr方法。这个脚本的效果见图8-2。

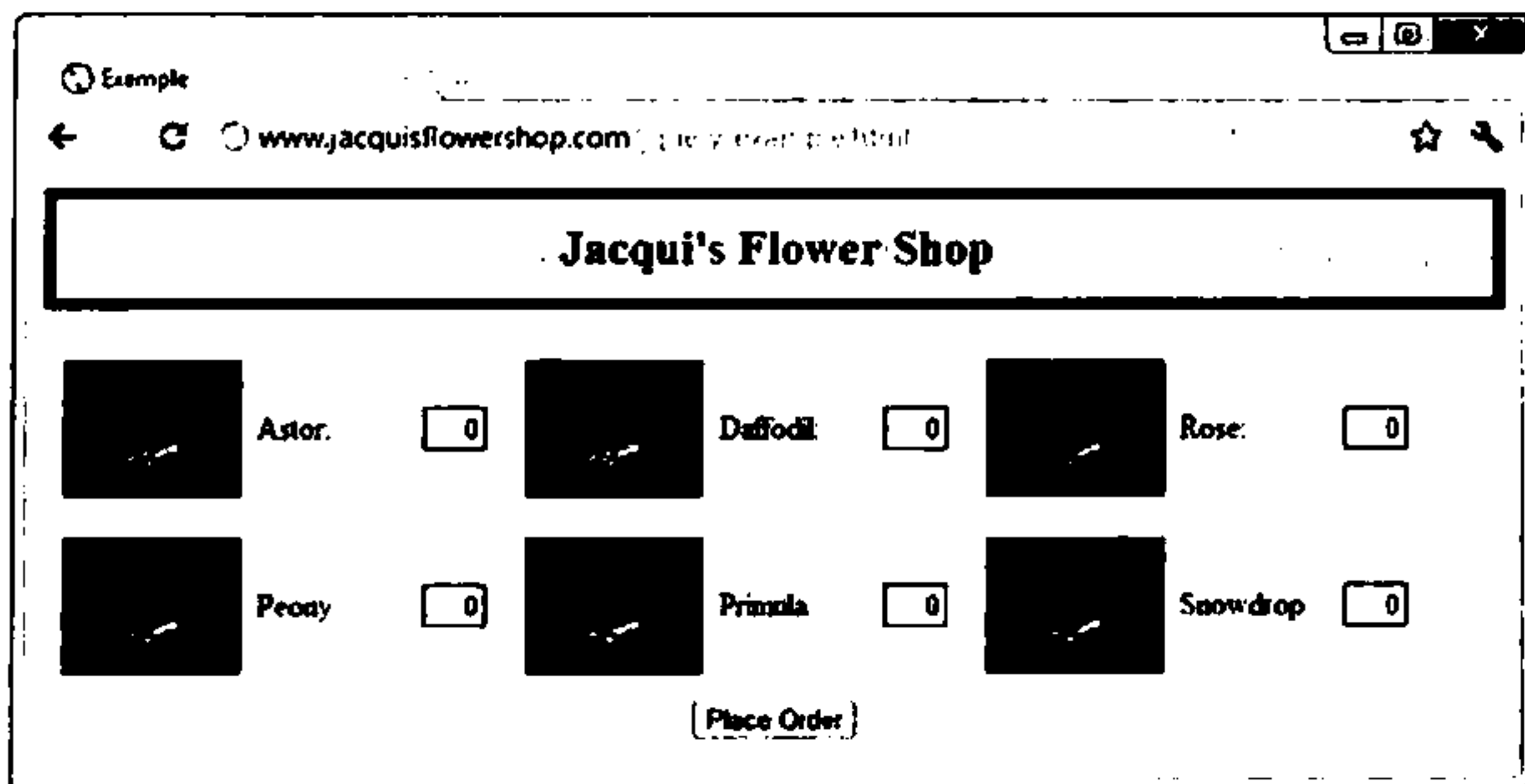


图8-2 使用attr方法一次设置多个属性

**提示** 在这个例子中我显式设置了style属性，其实jQuery还提供了一些专门设置样式的方法，具体内容可参阅8.3节。

### 8.1.3 动态设置属性值

如果将函数用作attr方法的第二个参数，我们就能根据情况动态地设置属性值。代码清单8-5演示了这种方法。

## 代码清单8-5 使用函数设置属性值

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $("img").attr("src", function(index, oldVal) {
            if (oldVal.indexOf("rose") > -1) {
                return "lily.png";
            } else if ($(this).closest("#row2").length > 0) {
                return "carnation.png";
            }
        });
    });
</script>
...

```

这里的回调函数会收到两个参数，一个是当前元素的索引，一个是当前元素该属性的原始值。函数内this变量被设置为对应当前元素的HTMLElement对象。如果你希望修改属性，就让这个函数返回一个字符串值，这个值将会被设置为属性的新值。如果不返回任何值，则使用属性的原始值。在代码清单8-5中，我利用回调函数有选择地修改了img元素的src属性。这个例子的具体效果见图8-3。

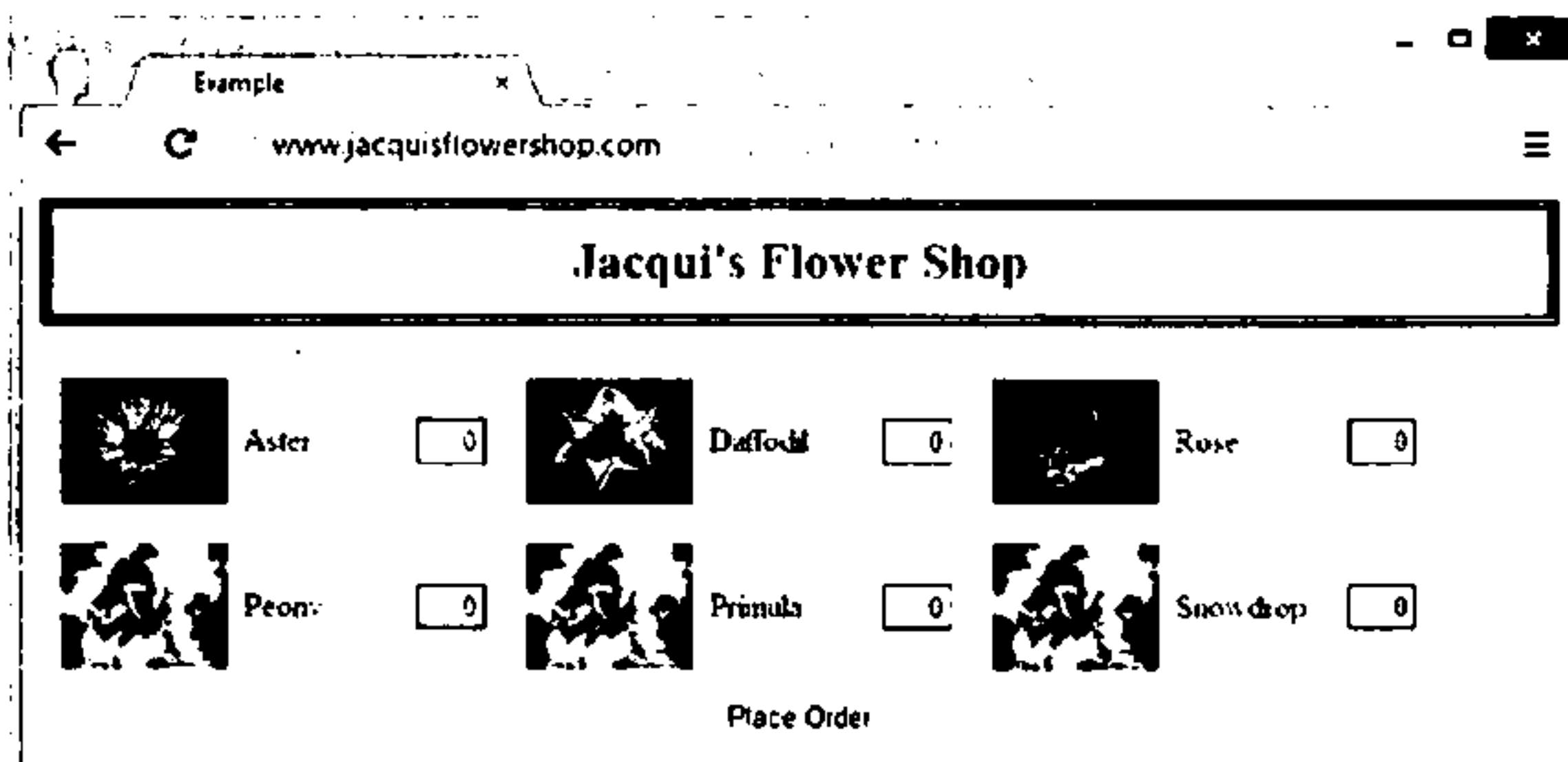


图8-3 使用函数修改属性值

## 8.1.4 删除属性

如代码清单8-6所示，removeAttr方法用来删除元素的属性。

## 代码清单8-6 删除元素属性

```

...
<script type="text/javascript">
    $(document).ready(function() {

        $("img").attr("style", "border: thick solid red");
        $("img:odd").removeAttr("style");
    });
</script>
...

```

在这个例子中，我先用attr方法设置了style属性，接着又使用removeAttr方法删除了奇数元素的这一属性。这一脚本的具体效果见图8-4。

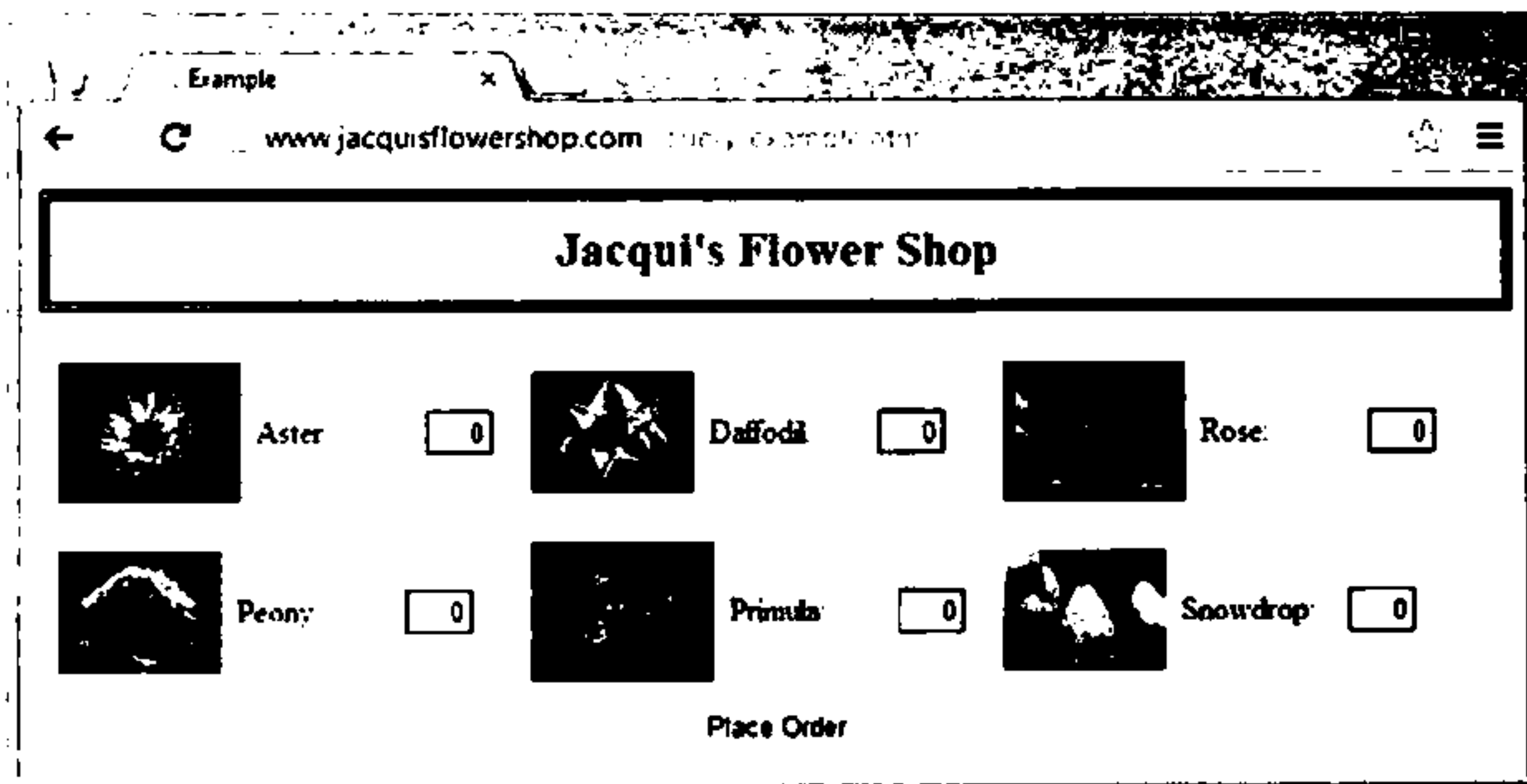


图8-4 删除元素属性

### 8.1.5 使用prop方法

attr方法与prop方法如影随行。二者的区别主要在于prop方法处理HTMLElement对象定义的属性，而不仅仅是我们能够看到的那些字面属性。我们在HTML标签中看到的字面属性（attribute）通常与元素对象所定义的属性（property）相同，不过也有例外。比如元素字面属性class（attribute），在HTMLElement对象中就使用className（property）而非class（attribute）表示。代码清单8-7演示了如何使用prop方法读取className属性。

代码清单8-7 使用prop方法读取属性

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("*[class]").each(function(index, elem) {
            console.log("Element:" + elem.tagName + " " + $(elem).prop("className"));
        });
    });
</script>
...
```

在这个例子中，我先选中所有拥有class属性的元素，然后使用each方法迭代处理选择结果：显示每一个元素的元素类型和className属性值。该例子在控制台产生的输出如下：

```
Element:DIV dtable
Element:DIV drow
Element:DIV dcell
Element:DIV dcell
Element:DIV dcell
Element:DIV drow
```

```

Element:DIV dcell
Element:DIV dcell
Element:DIV dcell

```

## 8.2 处理 class 属性

我们也可以使用处理属性的常规方法处理class属性，不过jQuery还提供了一套更方便的方法。表8-3简要列出了这些方法的功能与用法。在网页中class属性一般用来告诉浏览器应用事先定义好的一组CSS属性规则。如果需要详细了解class属性，请阅读第3章。

表8-3 处理class属性的方法

方 法	描 述
<code>addClass(name)</code>	为jQuery对象内包含的所有元素添加name指定的class（name可以是空格分格的多个class）
<code>addClass(function)</code>	为jQuery对象内含元素动态地添加class
<code>hasClass(name)</code>	如果jQuery对象内至少有一个元素拥有name指定的class，返回true，否则返回false
<code>removeClass(name)</code>	为jQuery对象内含元素删除name指定的class（name可以是空格分格的多个class）
<code>removeClass(function)</code>	为jQuery对象内含元素动态地删除class
<code>toggleClass()</code>	切换jQuery内含元素的class()
<code>toggleClass(boolean)</code>	单向切换jQuery内含元素的所有class
<code>toggleClass(name)</code>	为jQuery内含元素切换一个或多个命名class（name可以是空格分格的多个class）
<code>toggleClass(name, boolean)</code>	单向切换jQuery内含元素的一个命名class
<code>toggleClass(function, boolean)</code>	使用函数动态切换jQuery内含元素的class

`addClass`方法用于为元素添加class，`removeClass`方法用于删除class。如果想知道一个元素是否拥有某个class，请使用`hasClass`方法。代码清单8-8演示了这3个方法的用法。

代码清单8-8 添加、删除以及检测是否拥有某个class

```

...
<style type="text/css">
    img.redBorder {border: thick solid red}
    img.blueBorder {border: thick solid blue}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $("img").addClass("redBorder");
        $("img:even").removeClass("redBorder").addClass("blueBorder");

        console.log("All elements: " + $("img").hasClass("redBorder"));
        $("img").each(function(index, elem) {
            console.log("Element: " + $(elem).hasClass("redBorder") + " " + elem.src);
        });
    });

```



```
});
</script>
...
```

代码一开始,在style元素内我使用class定义了两行样式规则。这些class并非一定用来管理样式,不过它们确实简化了本章例子的演示。

我首先选中页面中所有的img元素,使用addClass方法为它们添加上redBorder class。然后,我选中所有的偶数img元素,使用removeClass方法删除刚刚添加上的redBorder class,接着又使用addClass方法为它们添加上blueBorder class。

---

**提示** addClass方法仅仅把新class额外添加到这些元素上,不会删除元素原有的class。

---

最后,我使用hasClass方法检测每个img元素是否拥有redBorder class(只要结果集内有一个元素拥有redBorder class就返回true)。图8-5列出了这些class产生的效果。

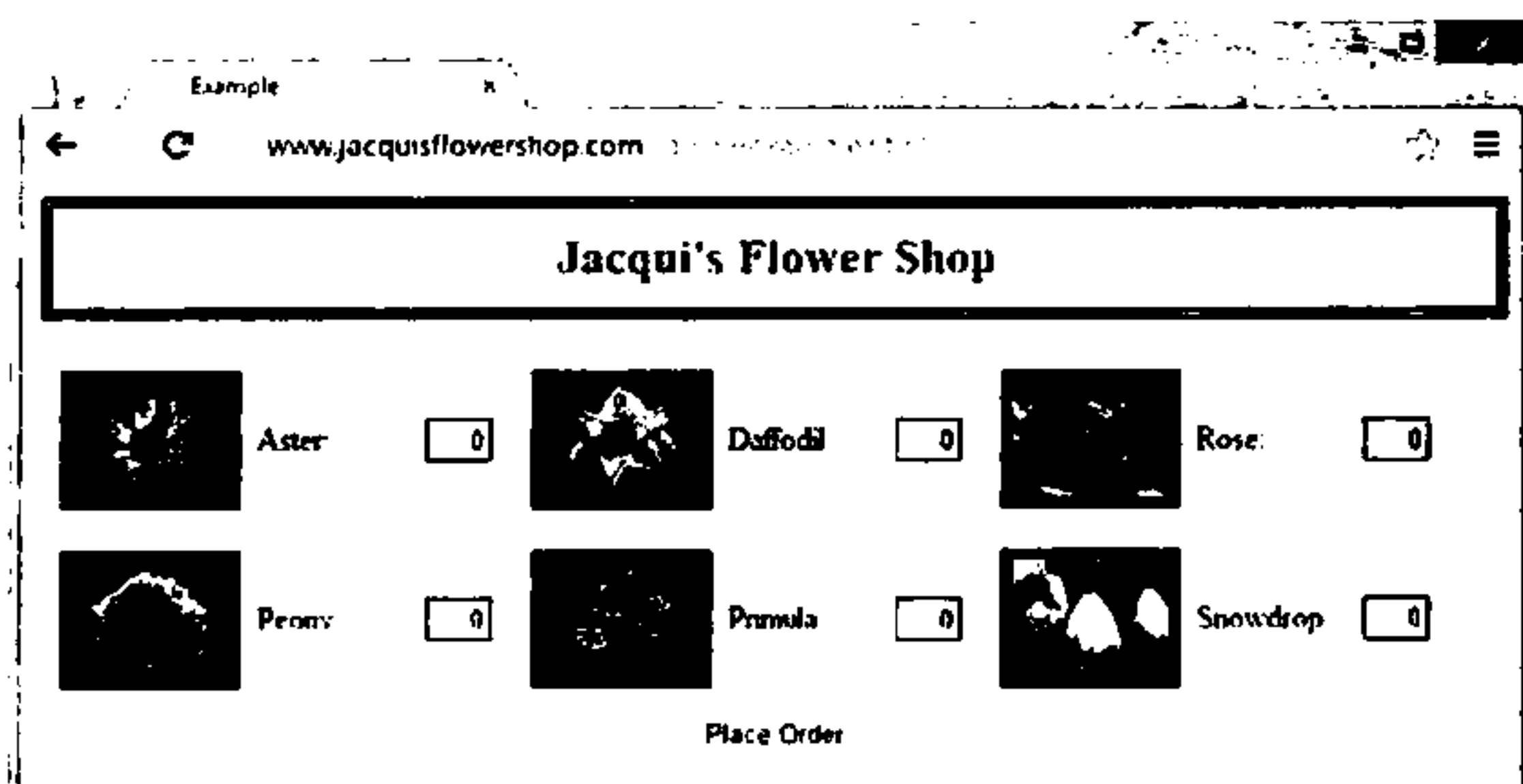


图8-5 使用class应用样式

以上测试img元素是否拥有redBorder class的脚本输出如下:

```
All elements: true
Element: false http://www.jacquisflowershop.com/jquery/aster.png
Element: true http://www.jacquisflowershop.com/jquery/daffodil.png
Element: false http://www.jacquisflowershop.com/jquery/rose.png
Element: true http://www.jacquisflowershop.com/jquery/peony.png
Element: false http://www.jacquisflowershop.com/jquery/primula.png
Element: true http://www.jacquisflowershop.com/jquery/snowdrop.png
```

### 8.2.1 使用函数动态添加或删除class

我们可以将一个函数用作addClass或removeClass方法的参数,以便动态决定添加或删除哪些class。代码清单8-9演示了addClass方法的这种用法。

## 代码清单8-9 使用函数参数动态地添加class

```

...
<style type="text/css">
    img.redBorder {border: thick solid red}
    img.blueBorder {border: thick solid blue}
</style>
<script type="text/javascript">
    $(document).ready(function() {
        $('img').addClass(function(index, currentClasses) {
            if (index % 2 == 0) {
                return "blueBorder";
            } else {
                return "redBorder";
            }
        });
    });
</script>
...

```

作为参数的回调函数有两个参数：一个是index，它是当前处理元素在jQuery对象内的索引（第几个）；一个是currentClasses，它是当前处理元素拥有的class。与其他类似的函数一样，jQuery将this变量设置为与当前处理元素相对应的HTMLElement对象。这个函数返回应该添加的class。在本例中，我使用index参数来决定为哪些元素添加blueBorder class，又为哪些元素添加redBorder class。这段脚本的效果和图8-5所示一样。

动态删除元素的class与之类似，如代码清单8-10所示，传递一个类似的函数给removeClass方法就行

## 代码清单8-10 使用函数参数动态地删除元素的class

```

...
<style type="text/css">
    img.redBorder {border: thick solid red}
    img.blueBorder {border: thick solid blue}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('img').filter(':odd').addClass("redBorder").end()
            .filter(':even').addClass("blueBorder");

        $('img').removeClass(function(index, currentClasses) {
            if ($(this).closest('#row2').length > 0
                && currentClasses.indexOf('redBorder') > -1) {
                return "redBorder";
            } else {
                return "";
            }
        });
    });
</script>
...

```

在这段脚本中，我传递给removeClass方法的函数使用了作为this变量的HTMLElement对象，以及当前元素拥有的class(currentClasses)：如果当前处理的img元素是#row2的后代元素并且拥有redBorder class，就删除它的这个redBorder class。这段脚本产生的效果见图8-6。

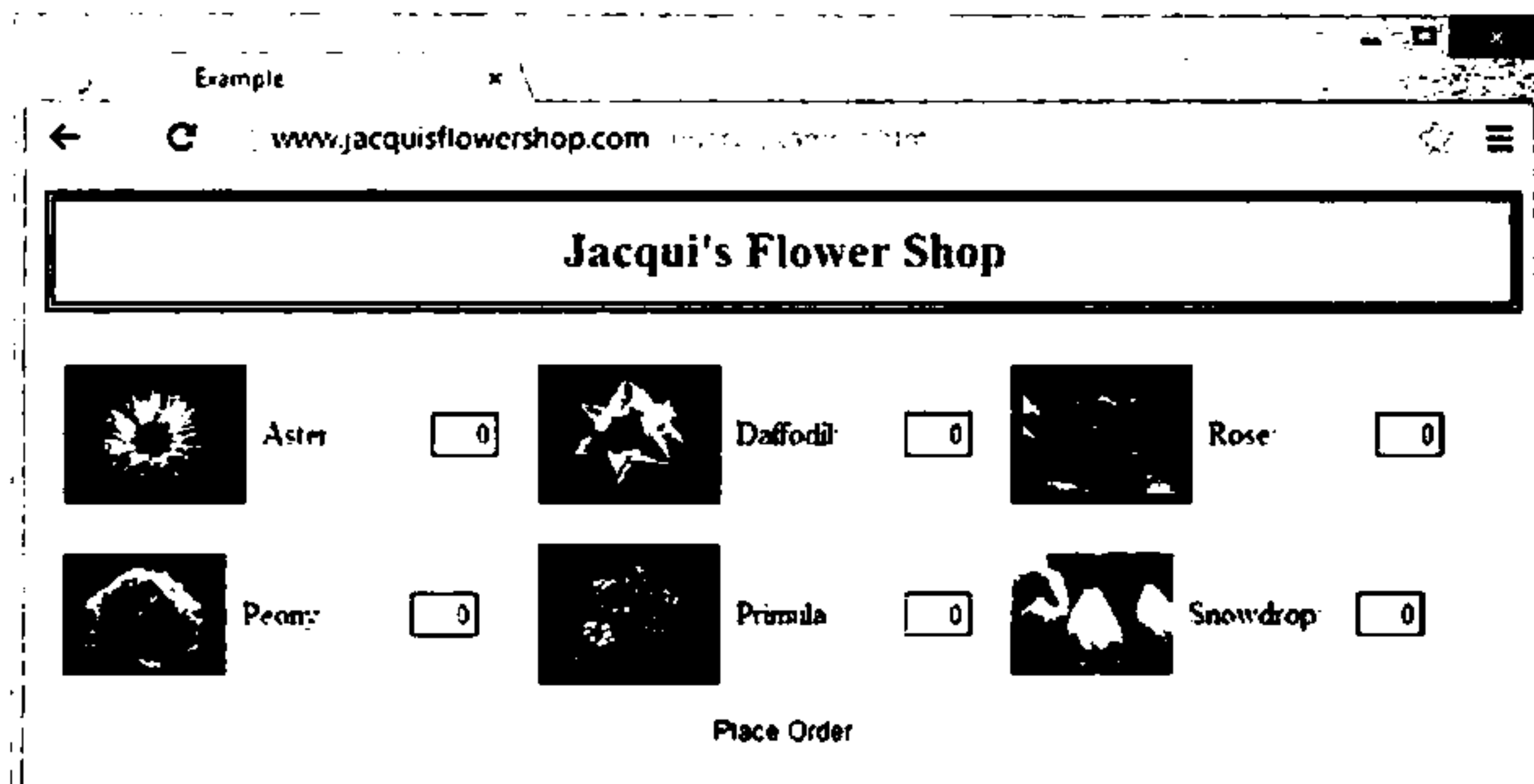


图8-6 使用函数动态删除class

**注意** 如果不需要删除任何类，我就让函数返回空字符串。如果函数没有返回值，jQuery就会删除这个元素所有的class。

## 8.2.2 切换class

toggleClass方法最常见的用途是切换class。所谓切换某个class，即若jQuery内含的元素有这个class就移除它，否则就加上它。要实现这种效果，如代码清单8-11所示，只需把要切换的class名作为参数传递给toggleClass方法。

代码清单8-11 toggleClass方法

```
...
<style type="text/css">
  img.redBorder {border: thick solid red}
  img.blueBorder {border: thick solid blue}
</style>
<script type="text/javascript">
  $(document).ready(function() {

    $("img").filter(":odd").addClass("redBorder").end()
      .filter(":even").addClass("blueBorder");

    $("<button>Toggle</button>").appendTo("#buttonDiv").click(doToggle);

    function doToggle(e) {
      $("img").toggleClass("redBorder");
    }
  });
</script>
```

```

        e.preventDefault();
    });
</script>
...

```

在上面的脚本中，我们先给奇数img元素添加了redBorder class，给偶数元素添加上blueBorder class。然后我们创建了一个新按钮并把它追加到div#buttonDiv元素内。这样新按钮就被放置到页面中已有按钮的右边。我还使用click方法为该按钮指定了事件处理函数，该函数在用户点击这个按钮时自动执行。这部分属于jQuery事件处理，更详细的介绍见第9章。用户点击按钮时自动调用的函数是doToggle，在doToggle函数中最关键的一条语句是：

```

...
$("img").toggleClass("redBorder");
...

```

这条语句选中页面中所有的img元素，并切换它们的redBorder class。doToggle语句的参数e以及后面那条e.preventDefault()语句在本章并不重要，我会在第9章详细介绍它们。这段脚本的具体效果见图8-7。对于这种例子来说，真正动手在浏览器中载入这个示例页面并点击这个按钮亲身体会一下，这比只看书上的截图更有意义。

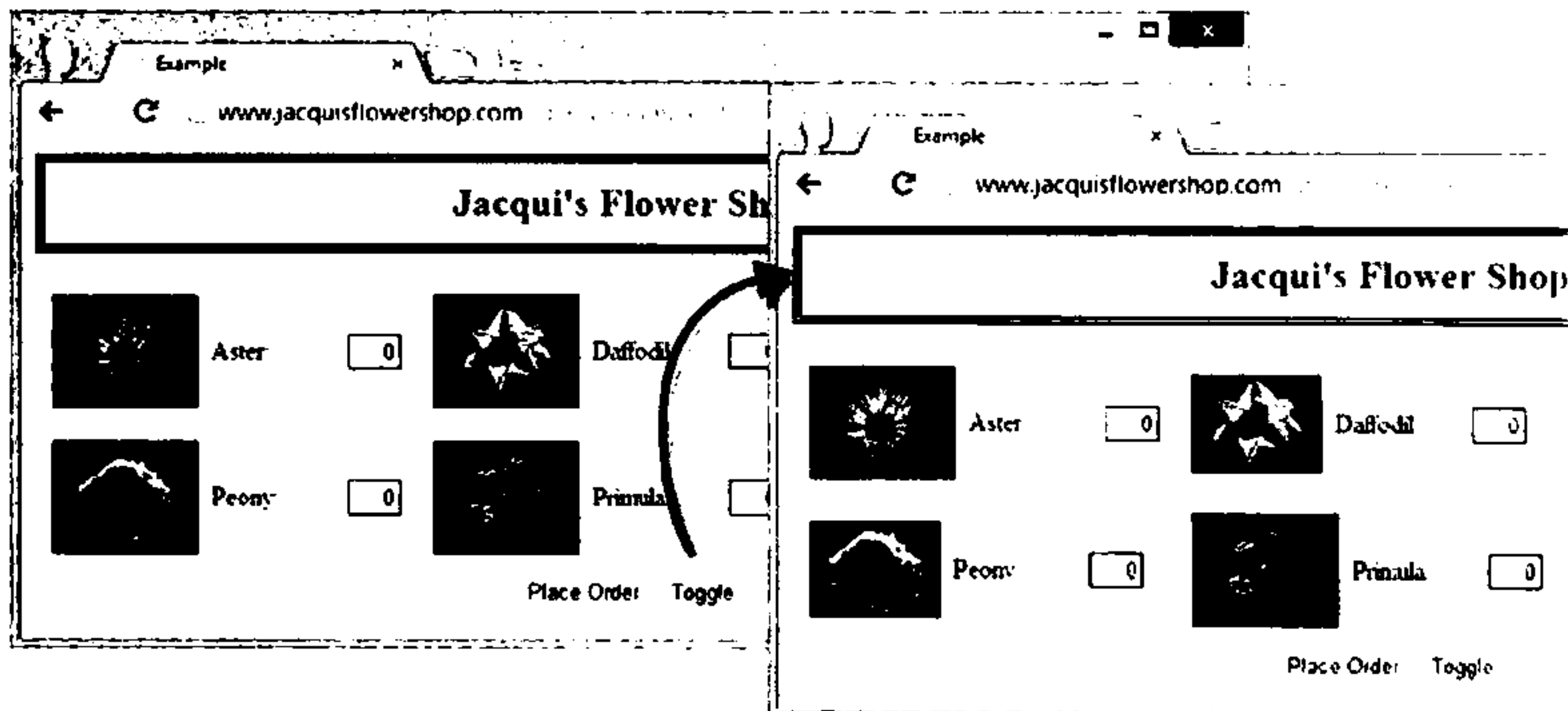


图8-7 使用toggleClass方法切换class

如果你比较细心，就会发现图8-7有点儿怪。那些原来有红色边框的元素不再有红色边框，而那些有蓝色边框的元素仍然有蓝色边框。为什么会这样？原来jQuery删除了奇数img元素的redBorder class，并把redBorder class加到偶数img元素上。这和我们期望的一样，但被添加redBorder class的img元素同时还拥有blueBorder class。由于在style元素中blueBorder样式定义于redBorder样式之后，因此它拥有更高的优先级（关于样式优先级详见第3章）。所以，虽然class切换行为确实发生了，我们却不得不仔细研究样式的微妙之处。如果希望原来具有蓝色边框的元素显示红色边框，如代码清单8-12所示，我们可以在样式声明中调换两个class的位置。

## 代码清单8-12 调整样式声明以符合class切换要求

```

...
<style type="text/css">
  img.blueBorder {border: thick solid blue}
  img.redBorder {border: thick solid red}
</style>
<script type="text/javascript">
  $(document).ready(function() {
    $("img").filter(":odd").addClass("redBorder").end()
      .filter(":even").addClass("blueBorder");

    $("<button>Toggle</button>").appendTo("#buttonDiv").click(doToggle);

    function doToggle(e) {
      $("img").toggleClass("redBorder");
      e.preventDefault();
    };
  });
</script>
...

```

现在若一个元素既有blueBorder class又有redBorder class, 浏览器就会使用redBorder class定义的规则。修改样式声明后的效果见图8-8。

8

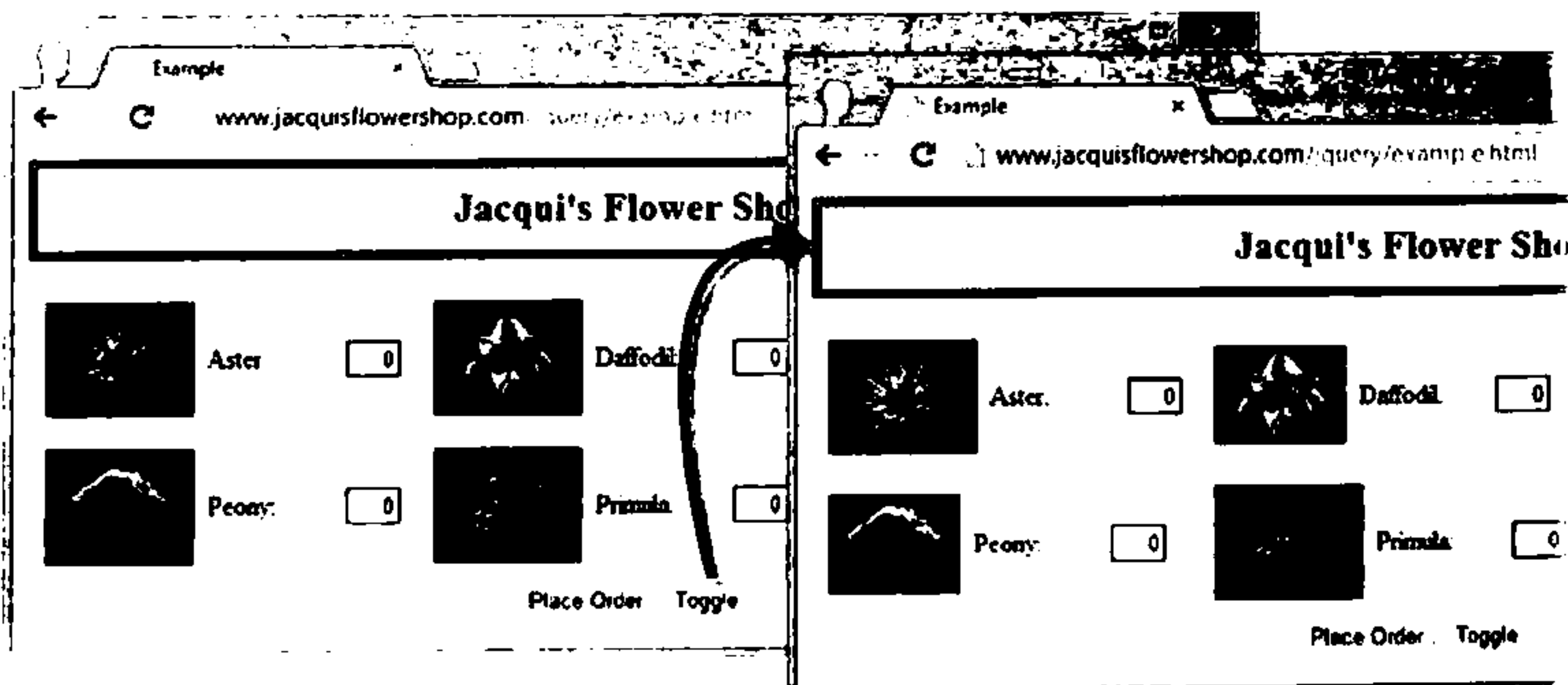


图8-8 调整了样式声明, 满足class切换要求后的效果

## 8.2.3 切换多个class

如果我们使用空格分隔的多个class名作为toggleClass方法的参数, 那么对于选中的元素来说, 这些class都会参与切换。具体的例子见代码清单8-13。

## 代码清单8-13 切换多个class

```

...
<style type="text/css">

```

```

    img.blueBorder {border: thick solid blue}
    img.redBorder {border: thick solid red}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('img').filter(':odd').addClass("redBorder").end()
            .filter(':even').addClass("blueBorder");

        $("<button>Toggle</button>").appendTo("#buttonDiv").click(doToggle);

        function doToggle(e) {
            $('img').toggleClass("redBorder blueBorder");
            e.preventDefault();
        };

    });
</script>
...

```

在这个例子中，我让所有img元素同时切换redBorder和blueBorder class。上面脚本的实际效果见图8-9。

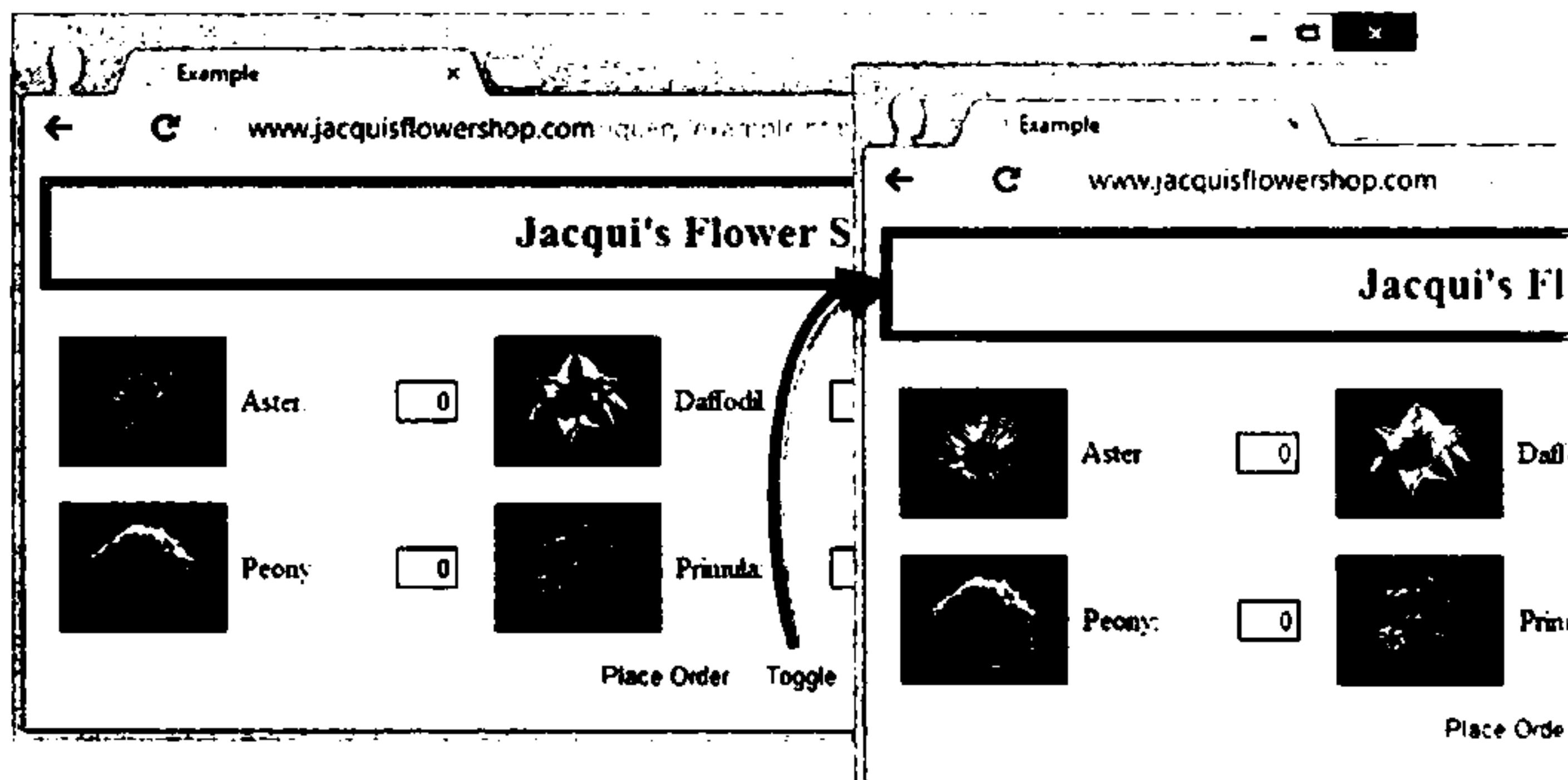


图8-9 切换多个class

#### 8.2.4 切换全部的class

如果不提供任何参数给toggleClass方法，结果就会切换每个元素拥有的全部class。由于jQuery自动保存需要切换的class，人们觉得这项技术非常精妙。代码清单8-14演示了这一方法的使用。

##### 代码清单8-14 切换选中元素所有的class

```

...
<style type="text/css">

```

```

img.blueBorder {border: thick solid blue}
img.redBorder {border: thick solid red}
label.bigFont {font-size: 1.5em}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $('img').filter(':odd').addClass("redBorder").end()
            .filter(':even').addClass("blueBorder");
        $('label').addClass("bigFont");

        $("<button>Toggle</button>").appendTo("#buttonDiv").click(doToggle);

        function doToggle(e) {
            $('img, label').toggleClass();
            e.preventDefault();
        };

    });
</script>
...

```

在这个例子中，我使用addClass方法为img和label元素添加了一些class。如果点击Toggle按钮，我选中同样的元素，并且无参数调用toggleClass方法。如图8-10所示，我们得到了非常特殊的效果。

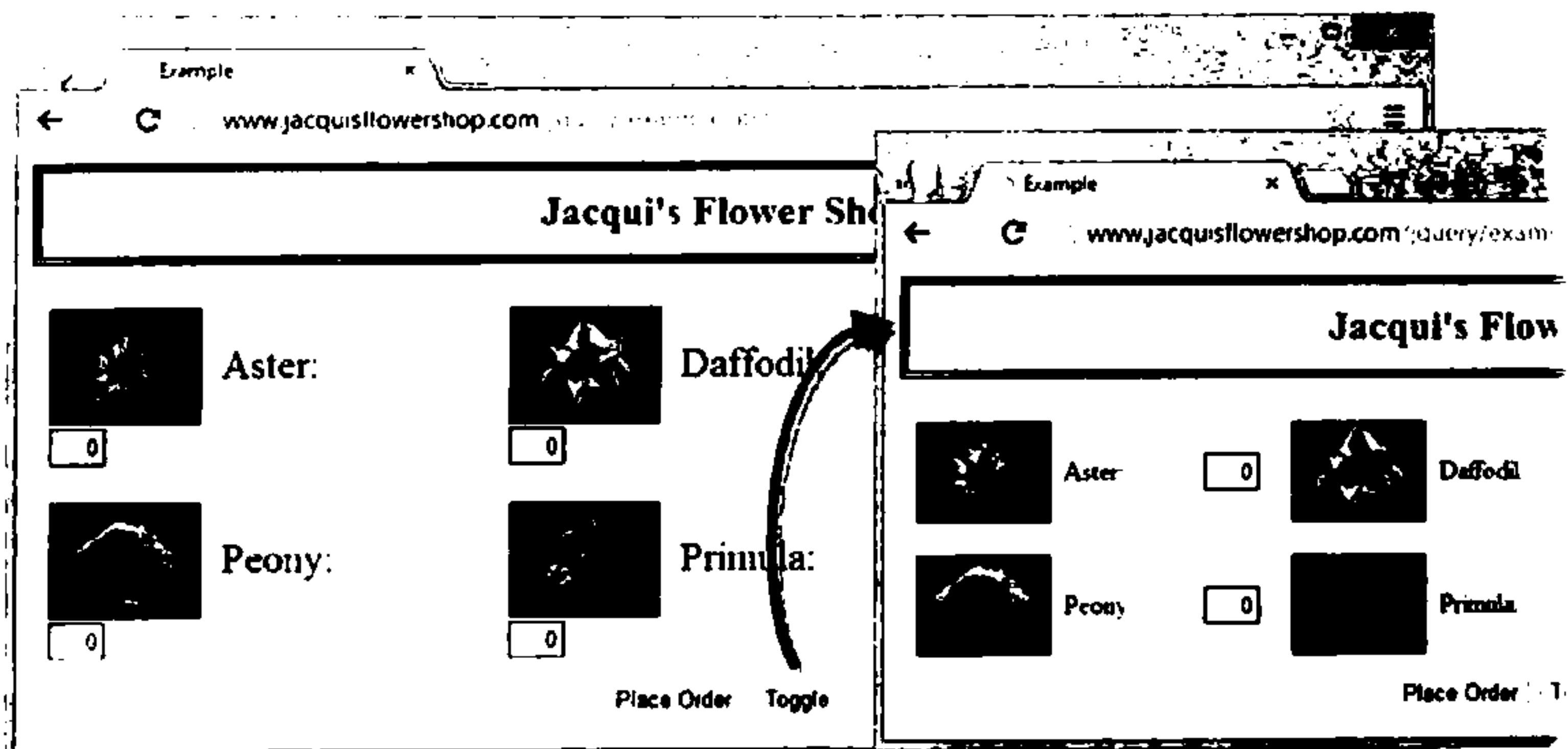


图8-10 切换元素所有的class

第一次点击Toggle按钮时，选中元素的所有class都被删除。jQuery会记住每个元素删除了哪些class，这样当我们再次点击Toggle按钮时，这些类就会自动恢复。

### 8.2.5 单方向切换class

如果传递一个布尔值参数给toggleClass方法，我们就能够控制切换的方向。若参数为false，toggleClass方法只会做删除动作；若参数为true，它只做添加动作。参见代码清单8-15。

代码清单8-15 控制切换的方向

```

...
<style type="text/css">
  img.blueBorder {border: thick solid blue}
  img.redBorder {border: thick solid red}
</style>
<script type="text/javascript">
  $(document).ready(function() {

    $('img').filter(':odd').addClass("redBorder").end()
      .filter(':even').addClass("blueBorder");

    $("<button>Toggle On</button>").appendTo("#buttonDiv").click(doToggleOn);
    $("<button>Toggle Off</button>").appendTo("#buttonDiv").click(doToggleOff);

    function doToggleOff(e) {
      $('img, label').toggleClass("redBorder", false);
      e.preventDefault();
    };

    function doToggleOn(e) {
      $('img, label').toggleClass("redBorder", true);
      e.preventDefault();
    };
  });
</script>
...

```

我在页面中添加了两个按钮，一个用来删除class，一个用来添加class。由于每个按钮都是单向操作，只有轮流点击这两个按钮才会有切换class的效果，如图8-11所示。

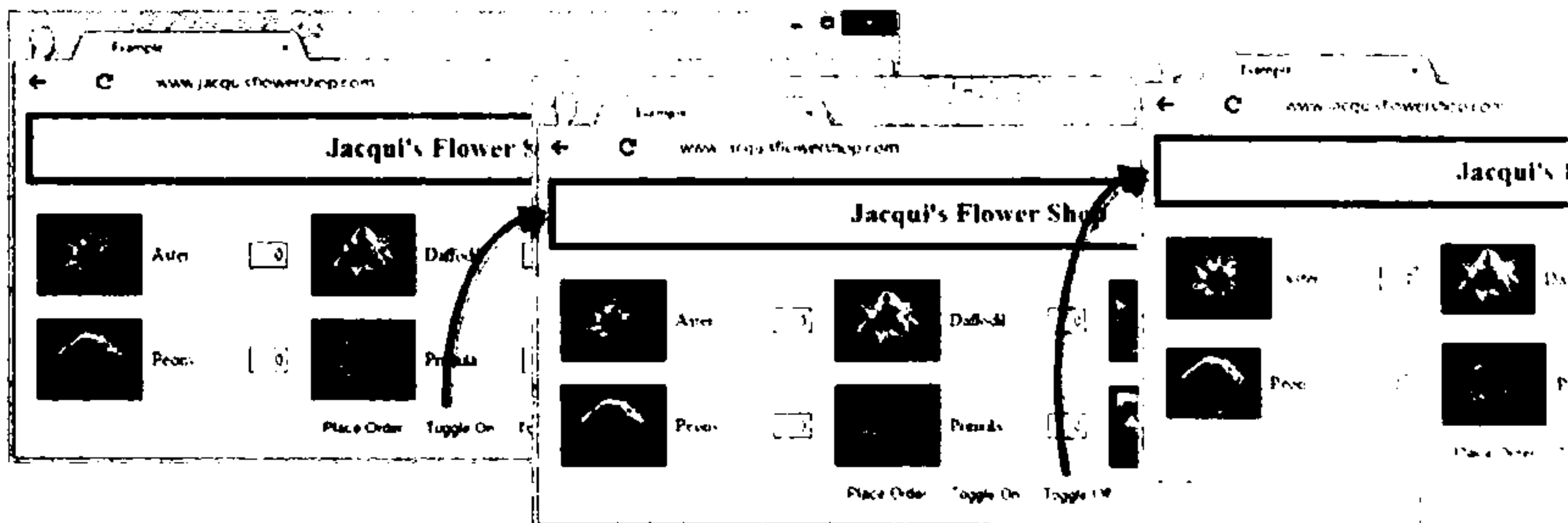


图8-11 单方向切换class

### 8.2.6 动态切换class

把一个回调函数作为参数传递给toggleClass方法，这样就能实现动态切换class的动作。代码清单8-16给出了一个简单的例子。



## 代码清单8-16 使用回调函数切换class

```

...
<style type="text/css">
  img.blueBorder {border: thick solid blue}
  img.redBorder {border: thick solid red}
</style>
<script type="text/javascript">
  $(document).ready(function() {

    $('img').addClass("blueBorder");
    $('img:even').addClass("redBorder");

    $("<button>Toggle</button>").appendTo("#buttonDiv").click(doToggle);

    function doToggle(e) {
      $('img').toggleClass(function(index, currentClasses) {
        if (index % 2 == 0) {
          return "redBorder";
        } else {
          return "";
        }
      });
      e.preventDefault();
    };
  });
</script>
...

```

我先给所有img元素加上blueBorder class，再给偶数img元素加上redBorder class。回调函数的参数index是当前处理元素在jQuery对象内的索引（第几个），参数currentClasses是当前处理元素拥有的class。另外，函数内的this变量被设置为与当前处理元素相对应的HTMLElement对象。这个函数返回需要切换的class。如果我不需要切换任何类，就让函数返回空字符串。注意，如果函数没有返回值，jQuery就会切换这个元素所有的class。该代码清单产生的效果如图8-12所示。

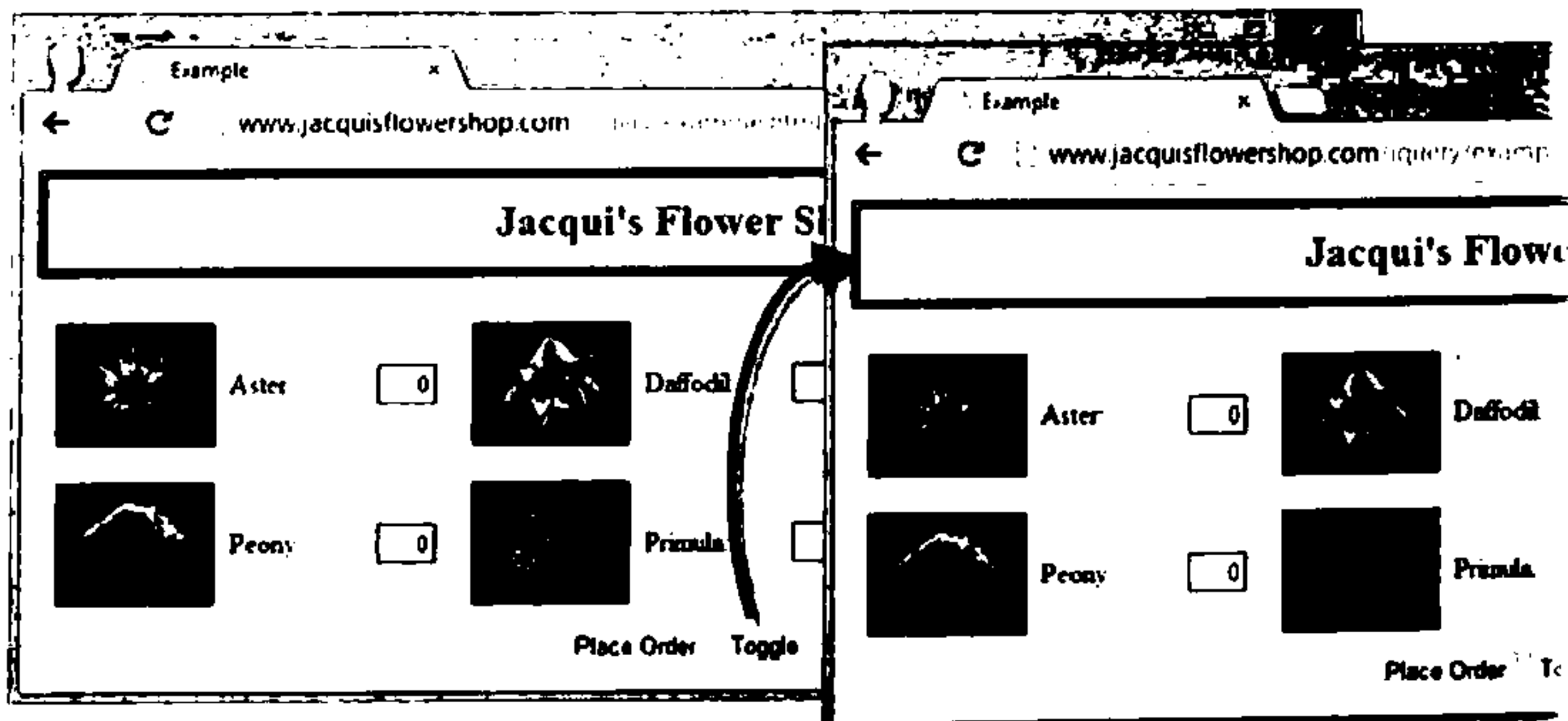


图8-12 动态切换class

## 8.3 处理 CSS 样式

在前面的一个例子里，我使用基础的属性处理方法attr设置元素的style属性，因此能够一次给许多元素设置CSS属性。jQuery还提供了一套使用起来更便捷容易的专门处理样式的方法。这些方法中应用最广的一个是css方法，它的具体用法见表8-4。

表8-4 css方法

方 法	描 述
css(name)	得到jQuery对象中第一个元素的name对应的CSS属性的值
css(names)	以数组形式提供names参数，得到多个CSS属性的值
css(name, value)	设置jQuery对象中所有元素的name属性对应的CSS属性的值为value
css(map)	设置jQuery对象中所有元素的由map对象定义的多个CSS属性的值
css(name, function)	使用回调函数动态设置jQuery对象中所有元素的name属性对应的CSS样式

**提示** 这些方法实际操作的是元素的style属性。如果你希望使用定义在样式表里的样式，就应该使用本章前面所介绍的class相关方法。

### 8.3.1 获取并设置单个CSS值

要读取CSS属性的值，只需把属性名称传递给css方法，就能得到jQuery对象中第一个元素的该属性的值。如果使用css方法设置CSS属性，则是对jQuery对象中的所有元素都有效。代码清单8-17演示了css方法的基本用法。

代码清单8-17 使用css方法获取和设置样式

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var sizeVal = $('label').css("font-size");
        console.log("Size: " + sizeVal);
        $('label').css("font-size", "1.5em");
    });
</script>
...
```

**提示** 虽然我使用的属性名是font-size而不是HTMLElement对象中定义的驼峰式fontSize，让人开心的是，聪明的jQuery同时支持这两种写法。

在这个脚本中，我先是选中页面中的label元素，然后使用css方法获取font-size属性值，并把它输出到控制台。然后，我再次选中所有的label元素，并且为结果集中的所有元素设置font-size属性值。

这段脚本的输出如下：

---

```
Size: 16px
```

---

**提示** 把一个样式属性的值设置为空字符串（""），这样即可从元素的style属性中删除这个属性。

---

### 8.3.2 获取多个CSS属性

把多个CSS属性名称以数组形式传递给css方法，即可一次获取多个CSS属性的值。这时css方法的返回结果是一个对象，参数数组的每个值对应着结果对象的一个键，每个键的值对应着结果集中第一个元素的相应CSS属性的值。代码清单8-18演示了如何使用css方法一次获取三个CSS属性的值。

代码清单8-18 使用css方法一次获取多个CSS属性的值

```
...
<script type="text/javascript">
    $(document).ready(function () {
        var propertyNames = ["font-size", "color", "border"];
        var cssValues = $("label").css(propertyNames);
        for (var i = 0; i < propertyNames.length; i++) {
            console.log("Property: " + propertyNames[i]
                + " Value: " + cssValues[propertyNames[i]]);
        }
    });
</script>
...
```

8

**提示** 这个加强版的css方法是在jQuery 1.9/2.0版本中引入的。

---

首先，创建一个以我希望拿到值的三个CSS属性名为内容的数组，这三个属性名分别是font-size、color和border。然后把这个数组传递给css方法，得到包含期望值的对象，该对象内容如下：

---

```
{font-size: "16px", color: "rgb(0, 0, 0)", border: "0px none rgb(0, 0, 0)"}
```

---

接着迭代参数数组，依次获取每个属性的值，最终在控制台显示如下结果：

---

```
Property: font-size Value: 16px
Property: color Value: rgb(0, 0, 0)
Property: border Value: 0px none rgb(0, 0, 0)
```

---

### 8.3.3 一次设置多个CSS样式属性

有两种方法可以一次设置多个样式属性。第一种方法，如代码清单8-19所示，只需要简单地链式调用css方法。

代码清单8-19 链式调用css方法

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("label").css("font-size", "1.5em").css("color", "blue");
    });
</script>
...
```

在这段脚本中，我设置了元素的font-size和color属性。如代码清单8-20所示，我们也可以使用一个映射对象达到同一目标。这个映射对象的格式与上一节中css方法返回的包含多个CSS属性值的对象格式相同。

代码清单8-20 使用映射对象设置多个样式属性

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var cssVals = {
            "font-size": "1.5em",
            "color": "blue"
        };

        $("label").css(cssVals);
    });
</script>
...
```

以上两个脚本的执行效果见图8-13。

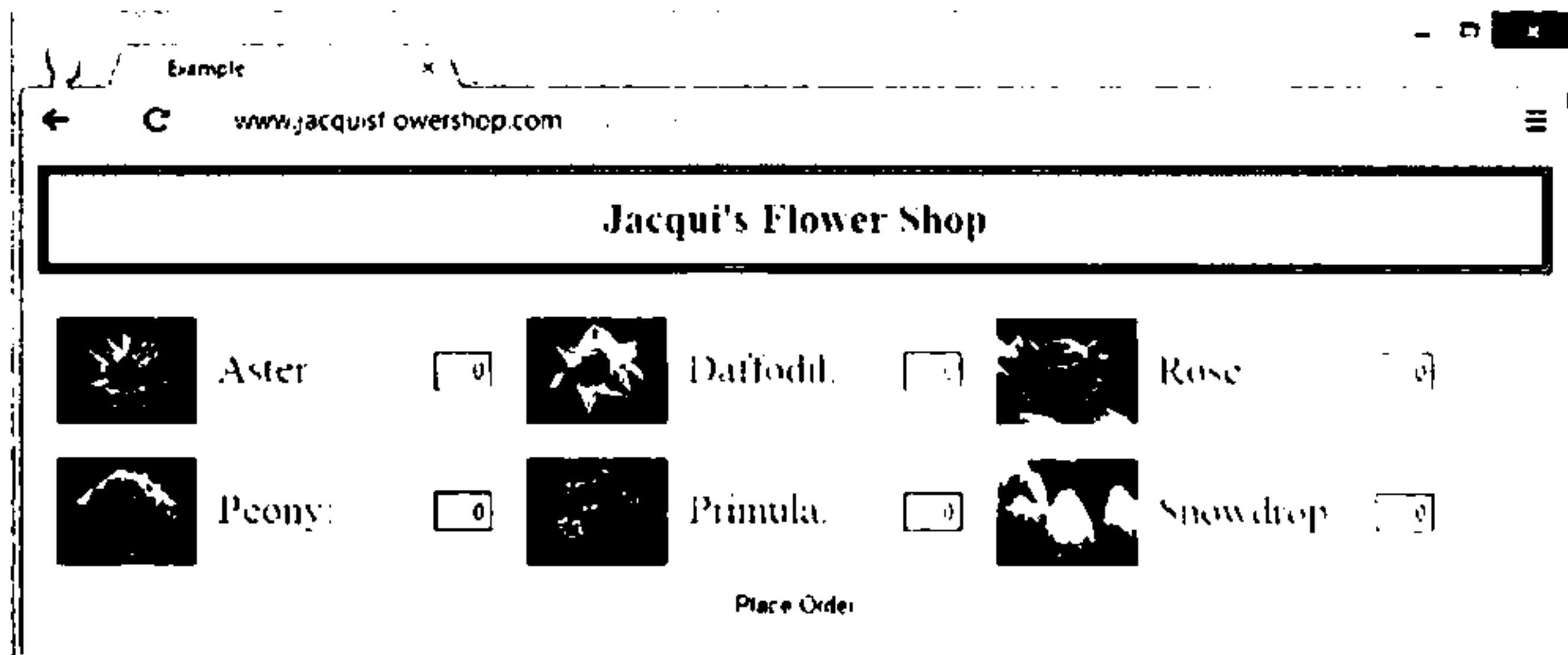


图8-13 设置多个样式属性

### 8.3.4 以相对值设置样式属性

css方法支持属性相对值。在数字值之前加上+=或者-=，这样就会在当前值的基础上增加或者减少值。这个技术仅适用于使用数值单位表示的CSS属性。具体例子见代码清单8-21。

代码清单8-21 在css方法中使用相对值

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('label:odd').css("font-size", "+=5")
        $('label:even').css("font-size", "-=5")

    });
</script>
...
```

jQuery读取这个属性时是什么单位，就会把这些值当成什么单位的值。在本例中，我把奇数label元素的字号增加了5像素，把偶数label元素的字号减小了5像素，实际效果见图8-14。

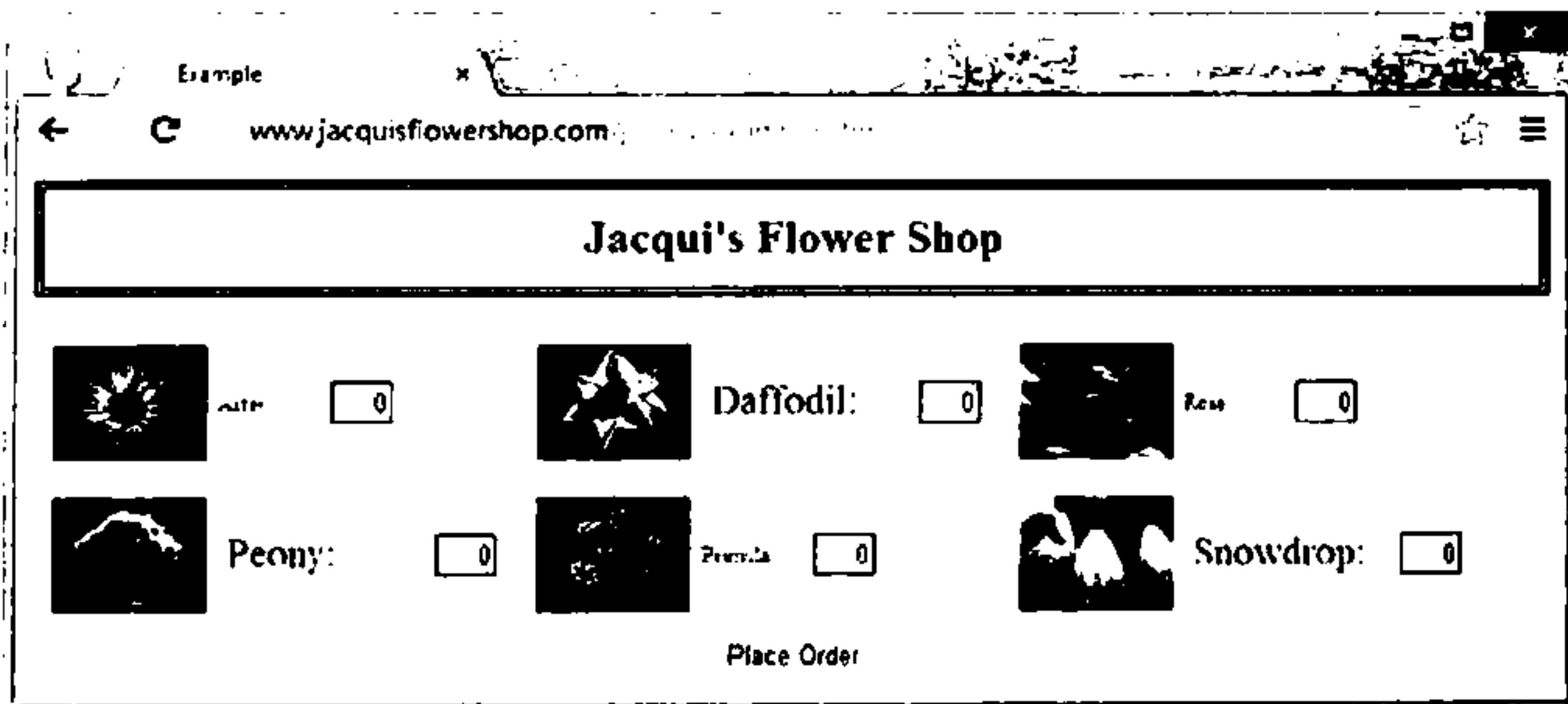


图8-14 使用相对值

### 8.3.5 使用回调函数设置样式属性

我们也可以将回调函数用作css方法的第二个参数，这样就能动态设置某个样式属性的值。示例代码见代码清单8-22。传递给回调函数的两个参数分别是当前元素的索引和属性的当前值。函数内的this变量也被设置为当前处理元素对应的HTMLElement对象。css方法将这个函数的返回值作为该属性的新值。

代码清单8-22 使用回调函数动态设置样式属性

```
...
<script type="text/javascript">
    $(document).ready(function() {
```

```

    $("label").css("border", function(index, currentValue) {
        if ($(this).closest("#row1").length > 0) {
            return "thick solid red";
        } else if (index % 2 == 1) {
            return "thick double blue";
        }
    });
});
</script>
...

```

脚本的实际效果见图8-15。

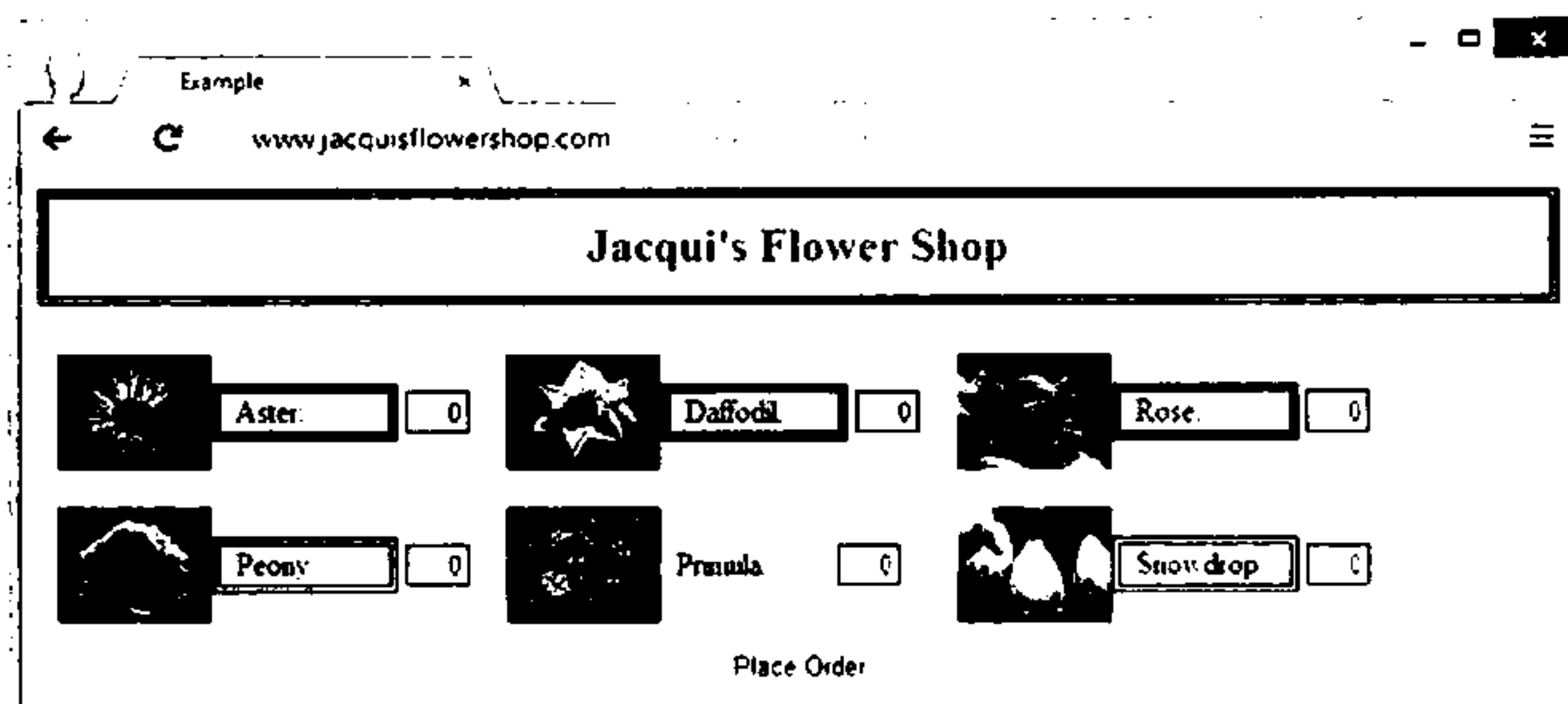


图8-15 使用回调函数动态设置样式属性

### 8.3.6 针对具体属性的CSS便捷方法

除css方法之外，jQuery还定义了许多方法，用来获取或设置常用CSS属性及其相关信息，详见表8-5。

表8-5 处理具体CSS属性的方法

方 法	描 述
height()	得到jQuery对象中第一个元素的高度（一个不带单位的数值，其计量单位为像素）
height(value)	设置jQuery对象中所有元素的高度为value
innerHeight()	得到jQuery对象中第一个元素的内部高度，即包括内边距（padding），但不包括边界（border）和外边距（margin）
innerWidth()	得到jQuery对象中第一个元素的内部宽度，即包括内边距，但不包括边界和外边距
offset()	得到jQuery对象中第一个元素相对于页面左上角的坐标
outerHeight(boolean)	得到jQuery对象中第一个元素的高度，包括内边距和边界，参数的值决定是否包括外边距
outerWidth(boolean)	得到jQuery对象中第一个元素的宽度，包括内边距和边界，参数的值决定是否包括外边距

(续)

方 法	描 述
position()	得到jQuery对象中第一个元素相对于父元素左上角的坐标
scrollLeft()、scrollTop()	得到jQuery对象中第一个元素横向或竖向滚动条的位置
scrollLeft(value)、scrollTop(value)	设置jQuery对象中所有元素的横向或竖向滚动条的位置
width()	得到jQuery对象中第一个元素的宽度
width(value)	设置jQuery对象中所有元素的宽度
height(function)、width(function)	使用回调函数动态设置jQuery对象中所有元素的高度和宽度

这里的绝大多数方法都是可以望文知义的，只有两个方法需要讲解。offset和position方法所返回的对象使用top和left这两个属性表示元素的当前位置。代码清单8-23演示了position方法的使用。

代码清单8-23 position方法

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var pos = $("#img").position();
        console.log("Position top: " + pos.top + " left: " + pos.left);
    });
</script>
...
```

这段脚本会输出position方法返回值的top和left属性，如下：

```
Position top: 108.078125 left: 18
```

#### 使用回调函数设置宽度和高度

将一个回调函数用作width或者height方法的参数，我们就可以动态设置结果集内所有元素的宽度或者高度。传递给回调函数的两个参数分别是当前元素的索引和属性的当前值。和你料想的一样，函数内的this变量也被设置为当前处理元素对应的HTMLElement对象，而该函数的返回值则会被当成该样式属性的设置值。示例代码见代码清单8-24。

代码清单8-24 使用回调函数动态设置元素高度

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#row1 img").css("border", "thick solid red")
        .height(function(index, currentValue) {
            return (index + 1) * 25;
        });
    });
</script>
...
```

在这个脚本中，我将index的值加1作为height的乘数。实际效果见图8-16。

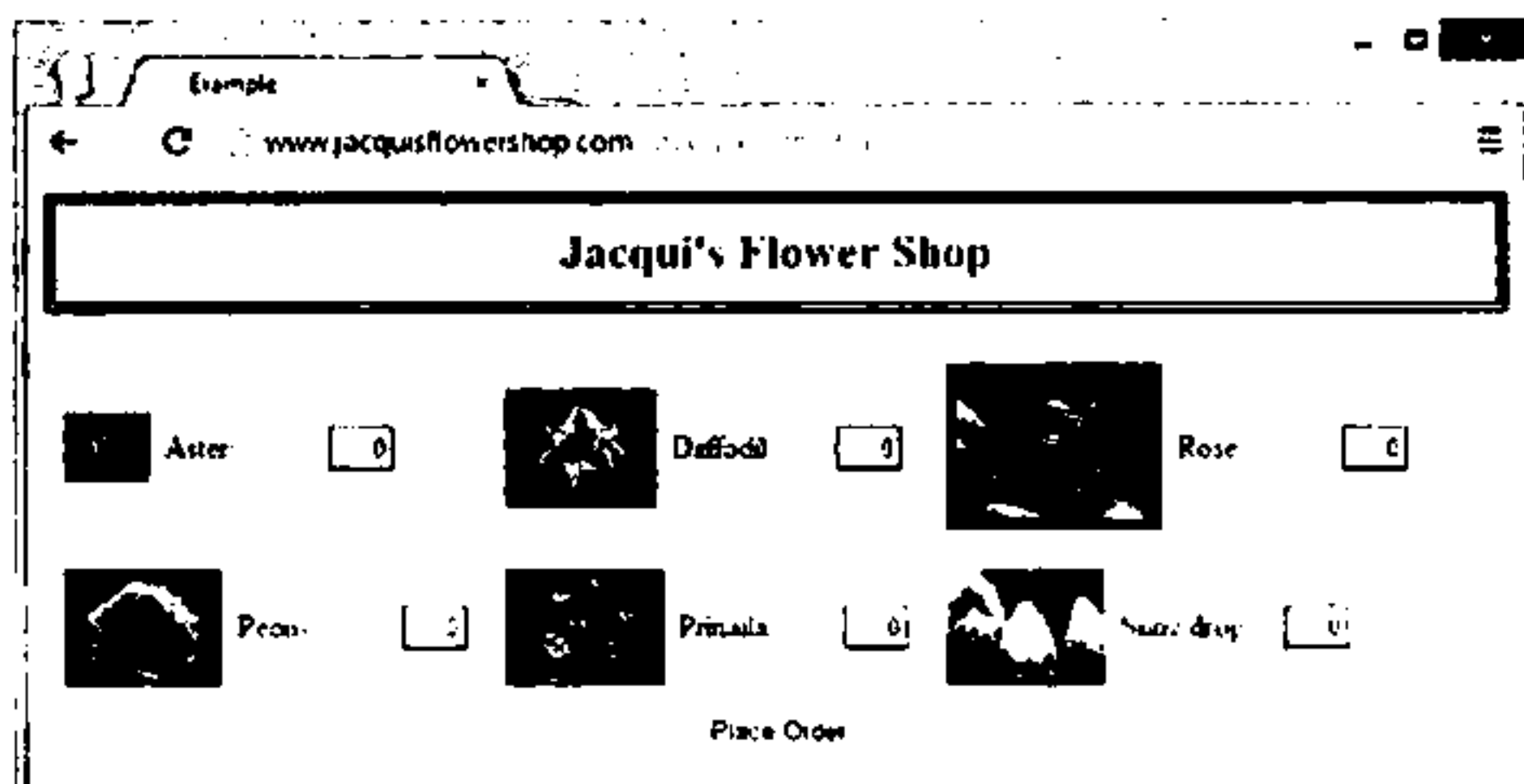


图8-16 使用回调函数动态设置元素高度

## 8.4 处理元素内容

本章从开始到现在，都是在讲如何处理元素属性，其实jQuery还定义了许多处理元素内容的方法，详见表8-6。

表8-6 处理元素内容的方法

方 法	描 述
text()	得到jQuery对象内所有元素及其后代元素的文本内容
text(value)	设置jQuery对象内每个元素的内容
html()	得到jQuery对象内第一个元素的HTML内容
html(value)	设置jQuery对象内每个元素的HTML内容
text(function)、html(function)	使用回调函数设置jQuery对象内每个元素的文本或HTML内容

jQuery的text方法非常特别，无参数调用时其返回值由jQuery结果集中所有选中元素而非第一个元素生成。如代码清单8-25所示，html及后面的方法都仅返回jQuery结果集中第一个元素的内容。

代码清单8-25 使用html方法读取元素的内容

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var html = $("div.dcell").html();
        console.log(html);
    });
</script>
...
```

这段脚本使用html方法读取div.dcell选择器匹配元素中第一个元素的HTML内容，并把读到的内容输出到控制台，得到下面的结果。注意，输出结果并不包括div.dcell元素本身。

```

<label for="aster">Aster:</label>
<input name="aster" value="0" required="">
```



### 8.4.1 设置元素内容

html或text方法都可用来设置元素内容。由于花店例子页面缺少文本内容，代码清单8-26中只演示了html方法的用法。

代码清单8-26 使用html方法设置元素内容

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#row2 div.dcell").html($("#div.dcell").html());
    });
</script>
...
```

这段脚本把#row2元素的所有div.dcell元素后代元素的HTML内容设置为第一个div.dcell元素的HTML内容。如图8-17所示，这导致页面中下面一行的所有产品的内容都变成了未分类产品（aster）。

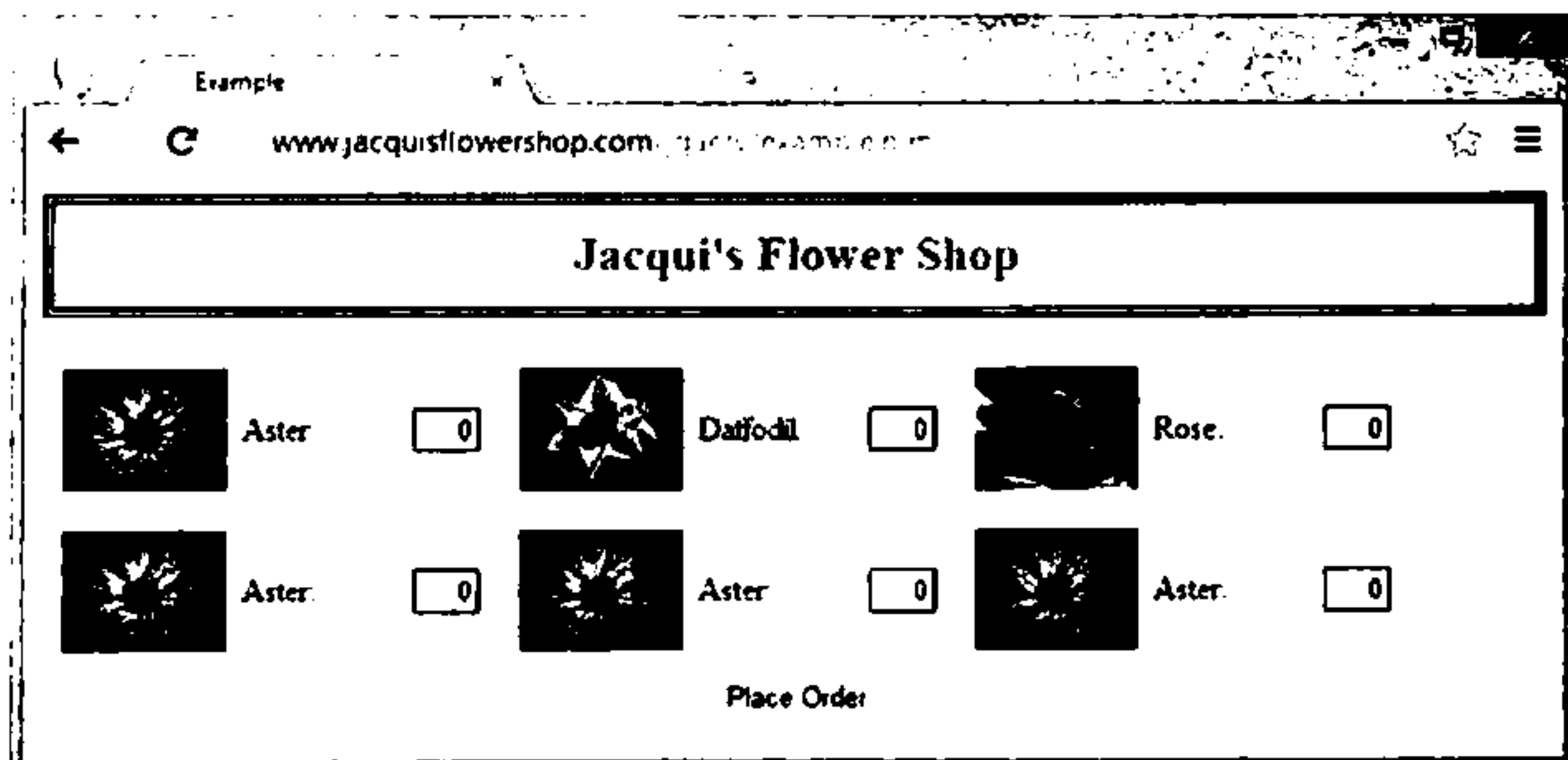


图8-17 使用html方法设置元素的内容

### 8.4.2 使用回调函数设置元素内容

和本章中的许多方法类似，html和text方法也支持使用回调函数动态设置元素内容。其中，回调函数的两个参数分别是当前元素的索引和元素的当前文本内容或HTML内容。函数内的this变量设置为元素对应的HTMLElement对象，而函数的返回值则会被当成要设置的内容。代码清单8-27演示了text方法的这种用法。

代码清单8-27 使用回调函数设置元素文本

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("label").css("border", "thick solid red").text(function(index, currentValue) {
            return "Index " + index;
        });
    });

```

```

    });
</script>
...

```

在这段脚本中，我把label元素的文本内容设置成当前元素的index参数值，并使用css方法为这些元素加了红色边框。脚本的实际效果见图8-18。

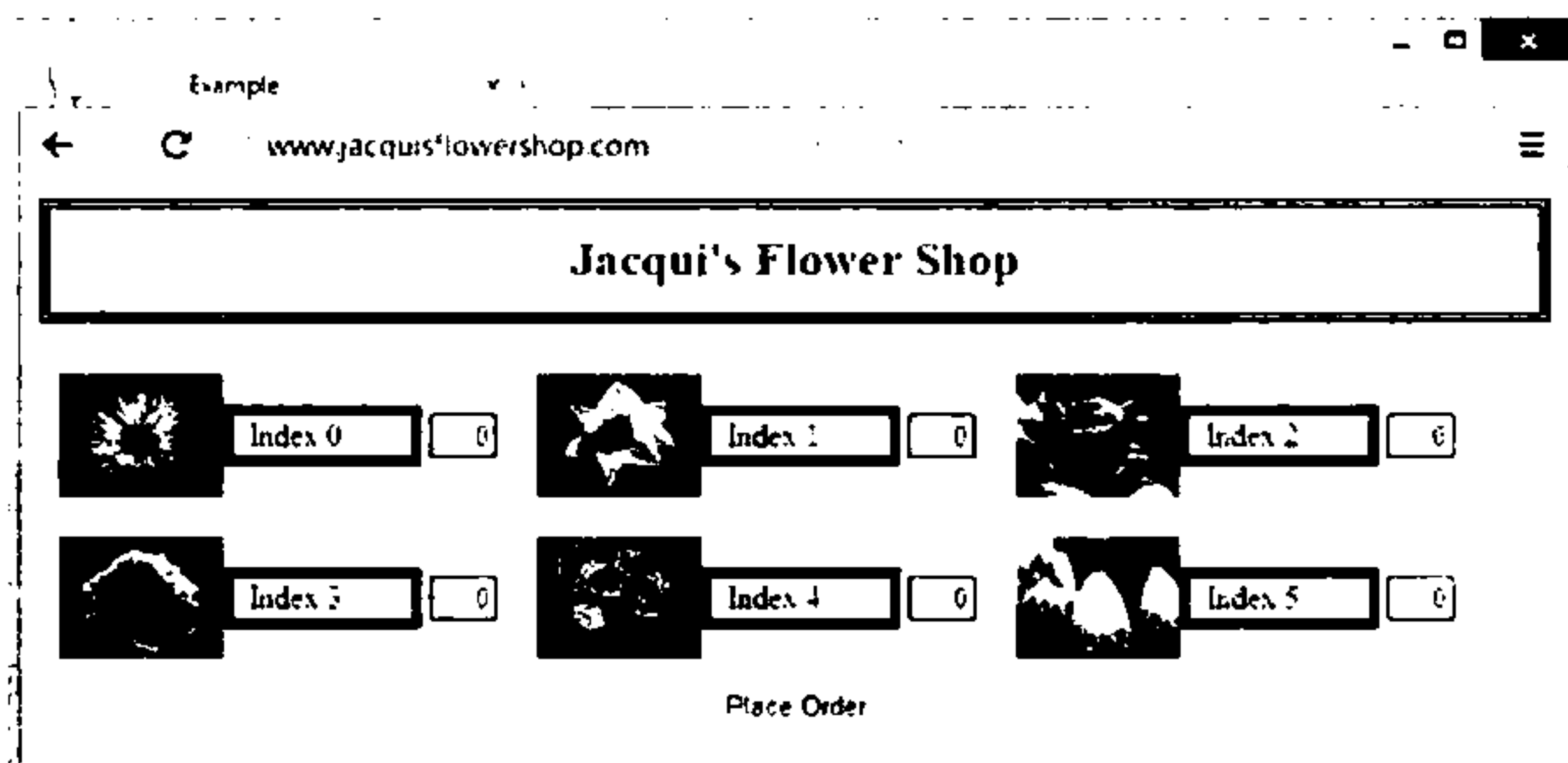


图8-18 使用回调函数设置文本内容

## 8.5 处理表单元素

表8-7列出的val方法用来获取和设置表单元素（input元素之类）的值。

表8-7 val方法

方 法	描 述
val()	得到jQuery对象内第一个元素的值
val(value)	设置jQuery对象内所有元素的值
val(function)	使用回调函数动态设置元素的值

代码清单8-28演示了使用val方法得到jQuery对象内第一个元素值的方法。在这个脚本中，我使用了each方法来保证脚本遍历页面中所有的input元素。

代码清单8-28 使用val方法得到input元素的值

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $("input").each(function(index, elem) {
            console.log("Name: " + elem.name + " Val: " + $(elem).val());
        });
    });
</script>
...

```

我把值输出到了控制台，得到以下内容：

```
Name: aster Val: 0
Name: daffodil Val: 0
Name: rose Val: 0
Name: peony Val: 0
Name: primula Val: 0
Name: snowdrop Val: 0
```

### 8.5.1 设置表单元素的值

只需把值作为参数传递给val方法，我们就能设置jQuery对象内所有元素的值。代码清单8-29给出了一个这样的例子。

代码清单8-29 使用val方法设置元素的值

```
...
<script type="text/javascript">
  $(document).ready(function() {
    $("<button>Set Values</button>").appendTo("#buttonDiv")
      .click(function (e) {
        $("input").val(100);
        e.preventDefault();
      })
  });
</script>
...
```

在这段脚本中，我在页面中添加了一个按钮Set Values（设置值）并为它设置了事件处理函数。点击该按钮会导致立即调用setValues函数，选中页面中所有的input元素并使用val方法把它们的值设置为100。这个例子的实际效果见图8-19。（上面的代码末尾调用了preventDefault方法，这一调用阻止了浏览器把HTML表单内容发送到Web服务器，我将在第9章详细介绍jQuery对事件的支持。）

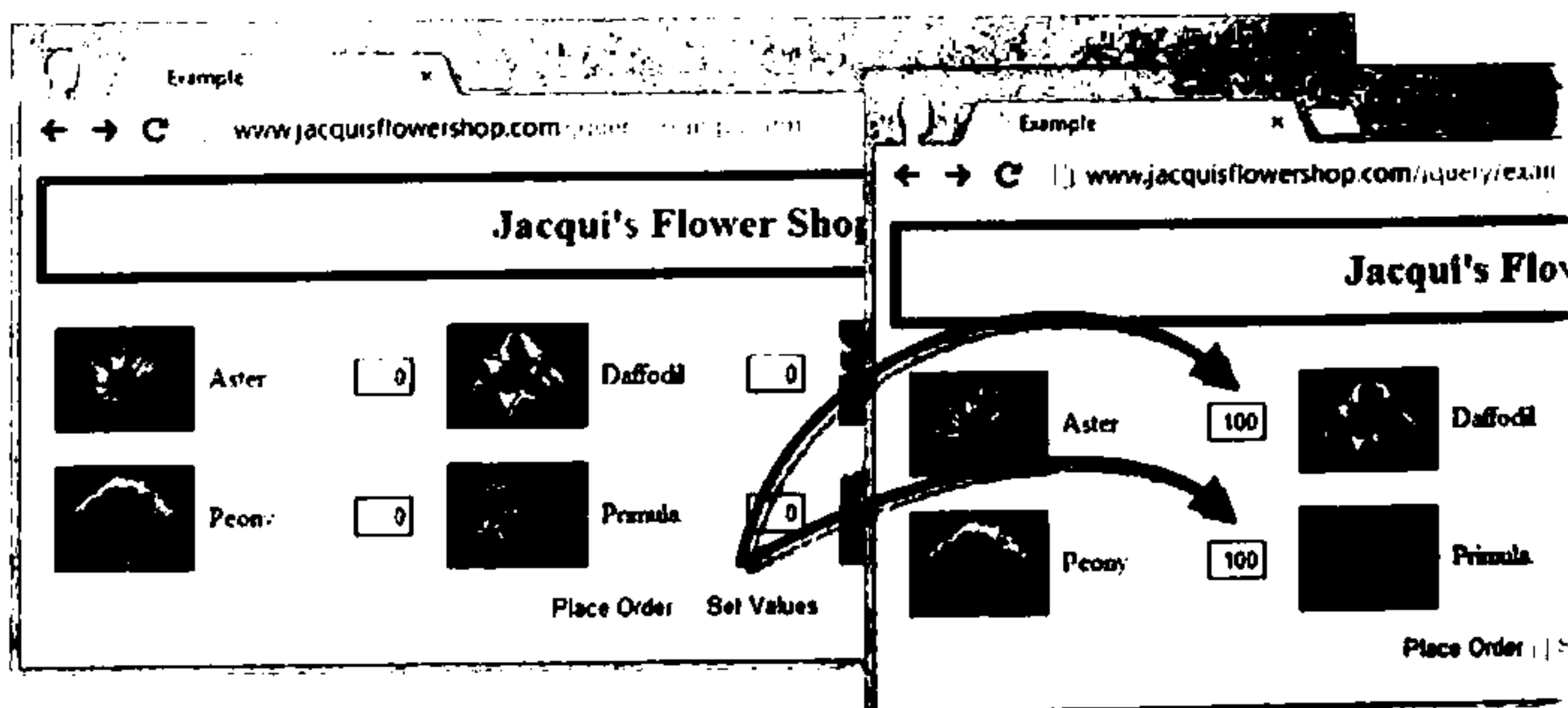


图8-19 使用val方法设置input元素的值

## 8.5.2 使用回调函数设置表单元素的值

你或许已经在想，是不是val方法也支持使用回调函数动态设置元素的值？没错。回调函数的两个参数分别是当前元素的索引和元素的当前值。函数内的this变量设置为元素对应的HTMLElement对象。如代码清单8-30所示，以这种方式调用val方法，我们可以动态设置元素的值。

代码清单8-30 以回调函数为参数调用val方法

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("input").val(function(index, currentVal) {
            return (index + 1) * 100;
        });
    });
</script>
...
```

在这个例子中，我基于元素的索引值设置它的新值。例子的实际效果见图8-20。

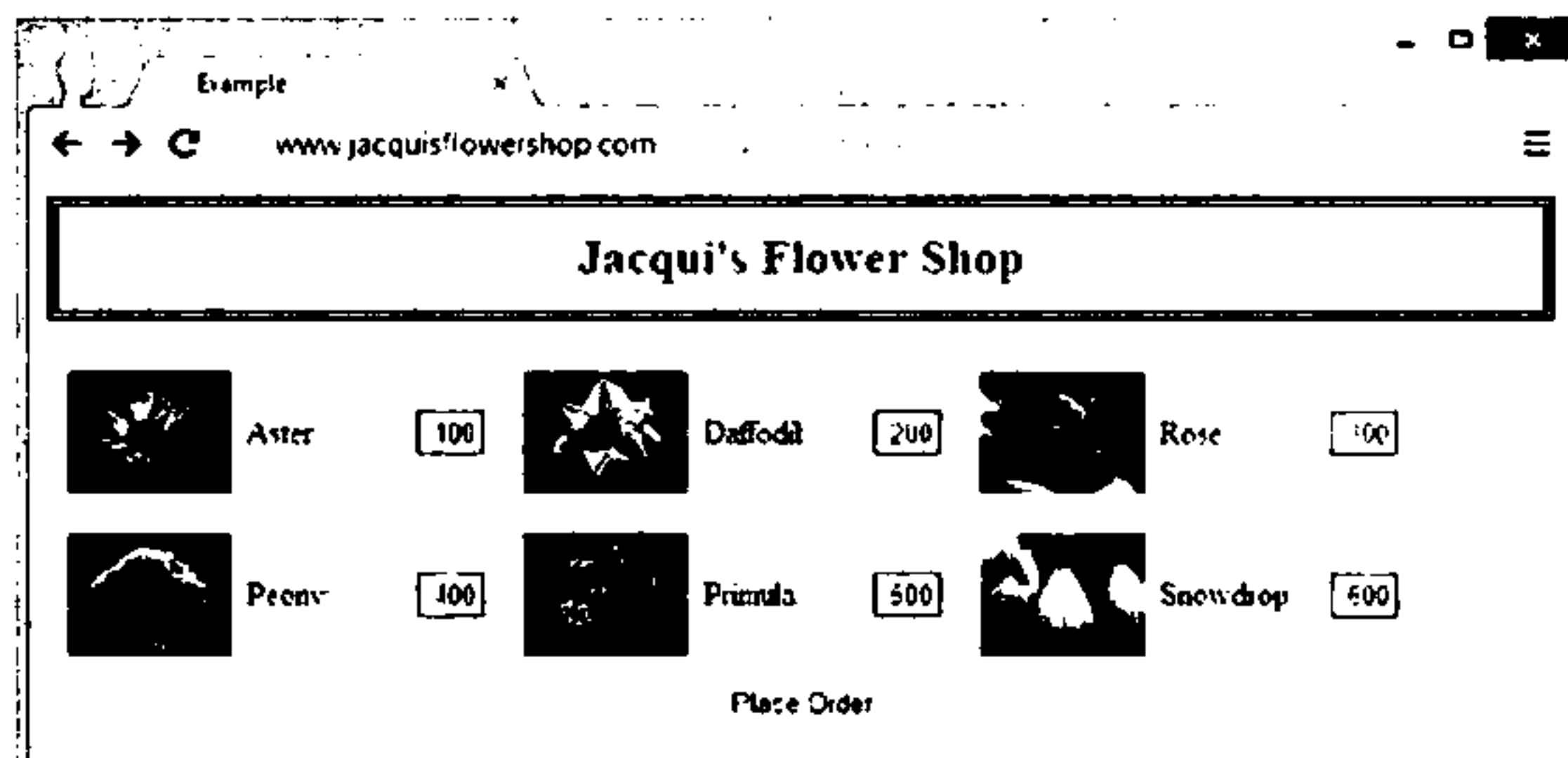


图8-20 使用回调函数动态地设置值

## 8.6 为元素关联数据

jQuery支持为一个元素关联任意的数据、测试数据是否存在，或者检索这些数据。表8-8列出了与数据关联有关的方法。

表8-8 处理任意元素数据的方法

方 法	描 述
data(key, value)、data(map)	为jQuery内含元素关联一个或多个键值对数据
data(key)	得到jQuery对象中第一个元素的与指定的键相关联的数据值
data()	得到jQuery对象中第一个元素关联的所有键值对数据
removeData(key)	删除jQuery对象内所有元素的与key键对应的值
removeData()	删除jQuery对象中所有元素的所有关联数据

代码清单8-31演示了设置、检测、读取和删除数据值的方法。

代码清单8-31 处理元素数据

```
...
<script type="text/javascript">
    $(document).ready(function() {

        // 设置数据
        $("img").each(function () {
            $(this).data("product", $(this).siblings("input[name]").attr("name"));
        });

        // 查找带有相关数据的元素并读取其值
        $("*").filter(function() {
            return $(this).data("product") != null;
        }).each(function() {
            console.log("Elem: " + this.tagName + " " + $(this).data("product"));
        });

        // 删除所有数据
        $("img").removeData();

    });
</script>
...
```

8

**注意** 当我们使用clone方法时，关联在源元素上的数据不会被自动关联到新元素，除非我们明确告诉jQuery自己需要这些关联数据。如果需要详细了解clone方法的使用以及如何保留关联数据到新元素，请阅读第7章。

这个脚本共分3步。第一步，我使用data方法为img元素关联了一个product数据项。我使用each方法遍历每个img元素，把product键设置为当前图片兄弟元素中input元素的name属性值。

第二步，我选中页面中所有的元素，然后过滤出那些关联了product键数据的元素。接着，我使用each方法遍历这些元素，并把data属性值输出到控制台。没错，尽管只是重复那些数据，这一例子却演示了能够选出关联了数据的元素的最佳技术。由于缺少合适的选择器和专门的方法，我们必须使用filter方法和一个回调函数以达到目的。

最后，我使用removeData方法删除所有img元素关联的数据。这个脚本的控制台输出如下：

```
Elem: IMG aster
Elem: IMG daffodil
Elem: IMG rose
Elem: IMG peony
Elem: IMG primula
Elem: IMG snowdrop
```

## 8.7 小结

我在本章中演示了许多在DOM中处理元素的方法,包括获取及设置属性(jQuery提供的处理class和样式属性的各种便捷方法)。我还演示了获取和设置元素文本或HTML内容的方法,以及使用jQuery为元素关联任意数据的两种方式。

本章的主题是jQuery事件处理。如果你对事件处理非常陌生,请参阅第2章中一个关于事件在DOM中如何传播的简明教程。jQuery提供了非常实用的事件处理功能,其中我最喜欢的一项功能是,当把元素动态地添加到DOM中去时,能够让这些元素自动绑定事件处理函数。本章概要见表9-1。

表9-1 本章概要

问 题	解决方法	代码清单
如何注册(绑定)一个函数来处理一个或多个事件	使用bind方法或某个快捷方法	1~4、19、20、23
如何阻止(抑制)事件的默认行为	使用Event.preventDefault方法或不提供事件处理函数参数的bind方法	5、6
如何删除某个元素的事件处理函数	使用unbind方法	7~9
如何创建每个绑定元素只触发一次的事件处理函数	使用one方法	10
如何为动态创建的元素自动绑定事件处理函数	使用on方法	11~13
如何撤销用live方法绑定的事件处理函数	使用off方法	14
如何为动态创建的DOM中某个元素的子元素绑定事件处理函数	使用delegate和undelegate方法	15
如何主动触发元素的事件处理函数	使用trigger、triggerHandler方法或某个快捷方法	16~18、21、22

## 新版jQuery中与本章有关的变化

jQuery 1.9/2.0移除了live和die方法,使用on和off方法可以实现同样的功能,详见9.2节。

使用trigger方法时, focus和blur事件的发送顺序有了一些变化,这两个事件的发送时机更合理,更符合用户的视觉感受。若希望详细了解trigger方法的用法,可阅读9.3节。

由此相关的另一个变化是,当input元素的单选钮或者复选框的click事件发生时,事件处理函数会得到元素最新的状态,即用户希望到达的状态。在旧版jQuery中,事件处理函数只能得到旧的状态(点击之前的状态)。

在本书的上一版中,我有意略去了一些jQuery团队明确声明已过时的功能,因此最新版本的jQuery所做的变化并不会影响到本章的内容。不过仍有一个变化值得一提:一些属性或者方法已经从新版jQuery中删除,它们曾出现在jquery.com网站上,你也有可能把它们用到了自己的项目中。它们是jQuery Event对象的attrChange、attrName、relatedNode和srcElement属性,伪hover事件(但不会影响到9.4节中讲到的hover方法),以及使用同一事件交替执行不同事件处理函数的toggle方法。

## 9.1 事件处理

利用jQuery提供的一整套事件注册方法，我们能够对各种事件注册处理函数，在目标元素触发指定的事件时，这些函数就会被调用。表9-2描述了这些方法。

表9-2 jQuery事件处理方法一览表

方 法	描 述
<code>bind(eventType, function)</code>	为jQuery对象内含元素 <sup>①</sup> 添加一个事件处理函数，支持可选的data参数
<code>bind(eventType, data, function)</code>	
<code>bind(eventType, boolean)</code>	(可用于)生成一个总是返回false的事件处理函数以阻止事件的默认行为。这个布尔型参数决定是否允许事件冒泡
<code>bind(map)</code>	使用JavaScript对象字面量map为jQuery对象内含元素一次添加多个事件处理函数
<code>one(eventType, function)</code>	为jQuery对象内含元素添加一个事件处理函数，支持可选的data参数；该事件处理函数一旦被触发执行就会自动撤销绑定(即至多执行一次)
<code>one(eventType, data, function)</code>	
<code>unbind()</code>	撤销之前绑定在jQuery对象内含元素上的所有事件处理函数
<code>unbind(eventType)</code>	撤销之前绑定在jQuery对象内含元素上的某一事件处理函数
<code>unbind(eventType, boolean)</code>	解除(为抑制事件默认行为而)绑定在jQuery对象内含元素上的总是返回false的事件处理函数
<code>unbind(Event)</code>	移除Event对象对应的事件处理函数

如上所示，`bind()`方法提供了多种绑定事件处理函数的方法，而且jQuery会把事件处理函数绑定到(调用`.bind()`方法的)jQuery对象内封装的所有元素上。代码清单9-1展示了一个简单的例子。

代码清单9-1 使用`.bind()`方法绑定事件处理函数

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script type="text/javascript">
    $(document).ready(function() {
      function handleMouseEnter(e) {
        $(this).css({
          "border": "thick solid red",
          "opacity": "0.5"
        });
      }

      function handleMouseOut(e) {
        $(this).css({
```

① 即封装在jQuery对象内的一个或多个元素。——译者注



```

        "border": "",
        "opacity": ""
    });
}

$("img").bind("mouseenter", handleMouseEnter)
    .bind("mouseout", handleMouseOut);
});
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <label for="aster">Aster:</label>
                        <input name="aster" value="0" required />
                    </div>
                    <div class="dcell">
                        <label for="daffodil">Daffodil:</label>
                        <input name="daffodil" value="0" required />
                    </div>
                    <div class="dcell">
                        <label for="rose">Rose:</label>
                        <input name="rose" value="0" required />
                    </div>
                </div>
                <div id="row2" class="drow">
                    <div class="dcell">
                        <label for="peony">Peony:</label>
                        <input name="peony" value="0" required />
                    </div>
                    <div class="dcell">
                        <label for="primula">Primula:</label>
                        <input name="primula" value="0" required />
                    </div>
                    <div class="dcell">
                        <label for="snowdrop">Snowdrop:</label>
                        <input name="snowdrop" value="0" required />
                    </div>
                </div>
            </div>
            <div id="buttonDiv"><button type="submit">Place Order</button></div>
        </form>
    </body>
</html>

```

在本例中，我先选取页面中所有的img元素，然后使用.bind()方法为这些img元素绑定mouseenter和mouseout事件处理函数。这两个处理函数使用.css()方法改变了图片的边框和透明度属性。当用户把鼠标移动到某个img元素之上时，处理函数就为它画上边框并使之更透明；当鼠标移开时，恢复该

img元素到其初始状态。

当jQuery调用事件处理函数时，this变量被设置为事件绑定的元素。传递给事件处理函数的Event对象是jQuery加工之后的Event对象，并非原生DOM标准中定义的Event对象。表9-3列出了jQuery Event对象支持的属性和方法。

表9-3 jQuery Event对象成员一览表

名 字	描 述	返 回 值
currentTarget	返回正在处理（响应）该事件的元素	HTMLElement
Data	返回绑定事件时传递给.bind()方法的可选data对象，9.1.1节中会详细介绍这个属性	Object
isDefaultPrevented()	若已经调用过preventDefault()方法，返回true	Boolean
isImmediatePropagationStopped()	若已经调用过stopImmediatePropagation()方法，返回true	Boolean
isPropagationStopped()	若已经调用过stopPropagation()方法，返回true	Boolean
originalEvent	返回未经jQuery加工的原始DOM Event对象	Event
pageX	返回相对于页面左上角的鼠标位置	number
pageY		
preventDefault()	阻止当前事件的默认行为	void
relatedTarget	仅对鼠标事件有效，返回该鼠标事件有关的元素（随鼠标事件的类型变化而变化）	HTMLElement
Result	返回处理该事件的最后一个事件处理函数的返回值	Object
stopImmediatePropagation()	立即阻止调用当前事件的其他事件处理函数	void
stopPropagation()	阻止事件冒泡，但允许正在响应事件的元素处理该事件	void
Target	返回触发事件的元素	HTMLElement
timestamp	返回事件发生的时间	number
Type	返回事件类型	string
Which	在键盘和鼠标事件中，返回用户按下的键或者鼠标按钮	number

jQuery的Event对象几乎定义了标准DOM Event对象的所有属性。因此，几乎在所有的场合，你都可以把jQuery Event对象看成是一个具有更多功能的标准DOM Event对象。

### 9.1.1 用一个函数处理多种事件

我们经常会使用一个事件处理函数处理多种事件。这些事件通常具有某种程度的相关性，比如mouseenter和mouseout事件。我们在使用bind方法时，可以在第一个参数中采用空格分隔的方式指定多种事件。代码清单9-2演示了这种用法。

代码清单9-2 为多种事件绑定同一个事件处理函数

```
...
<script type="text/javascript">
    $(document).ready(function() {

        function handleMouse(e) {
            var cssData = {
                "border": "thick solid red",
```

```

        "opacity": "0.5"
    }
    if (event.type == "mouseout") {
        cssData.border = "";
        cssData.opacity = "";
    }
    $(this).css(cssData);
}

$("img").bind("mouseenter mouseout", handleMouse);

});
</script>
...

```

在上面的脚本里，我只调用bind方法一次就为页面中所有的img元素指定了mouseenter和mouseout事件的处理函数handleMouse。当然也可以使用链式调用的方式绑定同一个事件处理函数，就像下面这样：

```

...
$("img").bind("mouseenter", handleMouse).bind("mouseout", handleMouse);
...

```

我们也可以使用映射对象在一次bind方法调用中为多个事件绑定处理函数。该对象的属性是事件名，属性的值则是事件对应的处理函数。代码清单9-3展示了使用映射对象绑定事件的方法。

代码清单9-3 使用映射对象为事件绑定处理函数

```

...
<script type="text/javascript">
    $(document).ready(function() {

        $("img").bind({
            mouseenter: function() {
                $(this).css("border", "thick solid red");
            },
            mouseout: function() {
                $(this).css("border", "");
            }
        });

    });
</script>
...

```

在代码清单9-3中，我直接使用函数字面量在映射对象内定义事件处理函数。bind方法会自动把映射对象中指定的函数绑定到对应的事件（由相应的键定义）上。

### 9.1.2 为事件处理函数提供数据

在使用bind方法绑定事件时，我们可以传递一个数据对象给bind方法（作为它的第二个参数）。jQuery则会通过Event.data属性把该对象传递给事件处理函数。在我们只用一个函数处理一堆元素的一堆事件时，这个数据对象就派上用场了。它可以有效帮助处理函数决定应该如何响应事件。代码清单9-4展示了应该如何定义和使用数据对象的值。

## 代码清单9-4 通过bind方法传递数据给事件处理函数

```

...
<script type="text/javascript">
    $(document).ready(function() {

        function handleMouse(e) {
            var cssData = {
                "border": "thick solid " + e.data,
            }
            if (event.type == "mouseout") {
                cssData.border = "";
            }
            $(this).css(cssData);
        }

        $("img:odd").bind("mouseenter mouseout", "red", handleMouse);
        $("img:even").bind("mouseenter mouseout", "blue", handleMouse);
    });
</script>
...

```

在这个脚本中，我使用bind方法的可选参数指定了在mouseenter事件发生时元素边界应该显示的颜色。当鼠标进入第奇数个图片时，边界将变成红色，反之边界变成蓝色。<sup>①</sup>在事件处理函数中，我使用Event.data属性读取绑定事件时传入的数据，并把它用于css方法设定元素的边界。上面脚本的实际效果见图9-1。

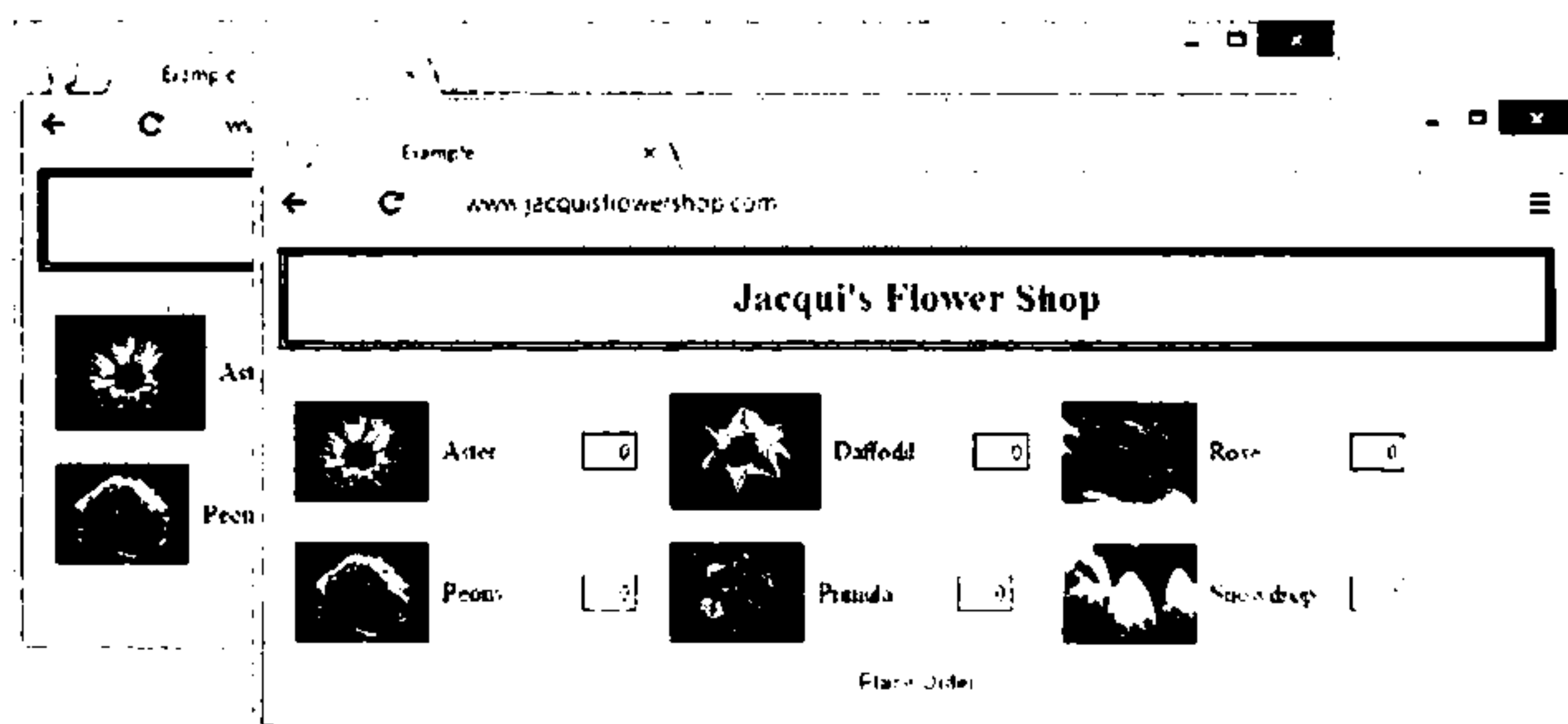


图9-1 通过bind方法传递数据给事件处理函数

### 9.1.3 阻止事件的默认行为

第2章提到过，一些元素的某些事件具有默认行为。点击type属性为submit的按钮就是一个典型的

① 细心的你或许会奇怪，怎么第一张图片（奇数）的边界变成了蓝色，第二张（偶数）变成红色？这与作者描述的奇偶顺序正好相反。是作者搞错了吗？当然没有，这是因为:odd和:even选择器总是从0开始计数（而不是从1开始计数）。——译者注

例子。如果这个按钮被包在一个form元素中，浏览器的默认行为是提交这个表单。要阻止该默认行为的实施，我们可以调用Event对象的preventDefault方法（参见代码清单9-5）。

代码清单9-5 阻止事件的默认行为

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("button:submit").bind("click", function(e) {
            e.preventDefault();
        });

    });
</script>
...
```

这段脚本为所有的type属性值为submit的button元素定义了事件处理函数。这个函数体只包含一条语句，即调用事件对象的preventDefault方法。这条语句阻止了按钮的默认行为（提交表单），也没有定义任何替代行为。因此当我们点击这些按钮时，它们就像未被点击过一样，无动于衷。

一般来说，之所以阻止事件的默认行为是为了做另外一些事情。比如刚才的例子，我们之所以阻止浏览器提交表单，是为了使用Ajax（见第14章和第15章）方式提交表单。其实不必像代码清单9-5那样定义一个只有一行代码的事件处理函数，即像代码清单9-6那样，使用一种另类的方式调用bind方法。

代码清单9-6 使用bind方法创建阻止事件默认行为的事件处理函数

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("button:submit").bind("click", false);
    });
</script>
...
```

bind方法的第一个参数是我们打算阻止默认行为的一个或多个事件的名字，第二个参数控制是否阻止事件在DOM中冒泡（关于事件冒泡，请参考第2章）。

#### 9.1.4 撤销事件处理函数

unbind方法用来撤销之前绑定在元素上的事件处理函数。如果像代码清单9-7那样不提供任何参数调用unbind方法，就会撤销之前绑定在jQuery对象中所有元素上的事件处理函数。

代码清单9-7 删除所有事件处理函数

```
...
<script type="text/javascript">
    $(document).ready(function() {

        function handleMouse(e) {
            var cssData = {
                "border": "thick solid red",
```

```

        "opacity": "0.5"
    }
    if (event.type == "mouseout") {
        cssData.border = "";
        cssData.opacity = "";
    }
    $(this).css(cssData);
}

$("img").bind("mouseenter mouseout", handleMouse);

$("img[src*=rose]").unbind();

});
</script>
...

```

在代码清单9-7中,我先为页面中的所有img元素绑定了mouseenter和mouseout事件处理函数,接着又使用unbind方法删除了所有src属性中包含rose单词的图片上绑定的所有事件处理函数。如代码清单9-8所示,我们也可以有选择地只撤销指定的事件。

**代码清单9-8 有选择地撤销事件处理函数**

```

...
<script type="text/javascript">
    $(document).ready(function() {

        function handleMouse(e) {
            var cssData = {
                "border": "thick solid red",
                "opacity": "0.5"
            }
            if (event.type == "mouseout") {
                cssData.border = "";
                cssData.opacity = "";
            }
            $(this).css(cssData);
        }

        $("img").bind("mouseenter mouseout", handleMouse);

        $("img[src*=rose]").unbind("mouseout");

    });
</script>
...

```

在上面的脚本中,我只撤销了mouseout事件处理函数,保留了mouseenter事件处理函数。

#### 在事件处理函数内撤销事件处理函数

撤销事件处理函数的最后一种方法是在事件处理函数内执行撤销动作。如果希望事件仅仅被处理特定的次数,这个方法很有用。代码清单9-9演示了这种情况。

代码清单9-9 在事件处理函数内撤销事件处理函数

```

...
<script type="text/javascript">
    $(document).ready(function() {

        var handledCount = 0;

        function handleMouseEnter(e) {
            $(this).css("border", "thick solid red");
        }
        function handleMouseExit(e) {
            $(this).css("border", "");
            handledCount ++;
            if (handledCount == 2) {
                $(this).unbind(e);
            }
        }
        $("img").bind("mouseenter", handleMouseEnter).bind("mouseout", handleMouseExit)
    });
</script>
...

```

在handleMouseExit函数中，每处理一次mouseout事件，计数器handledCount就加1。当处理mouseout事件两次之后（handledCount等于2），我们把Event对象作为unbind方法的参数传给它，从而撤销了之前的绑定。jQuery会自动分析Event对象，撤销正确的事件处理函数。

### 9.1.5 仅执行一次事件处理函数

one方法用来绑定至多执行一次的事件处理函数。代码清单9-10展示了one方法的使用法。

代码清单9-10 使用one方法绑定至多执行一次的事件处理函数

```

...
<script type="text/javascript">
    $(document).ready(function() {

        function handleMouseEnter(e) {
            $(this).css("border", "thick solid red");
        };

        function handleMouseOut(e) {
            $(this).css("border", "");
        };

        $("img").one("mouseenter", handleMouseEnter).one("mouseout", handleMouseOut);

    });
</script>
...

```

上面的脚本中，我使用one方法（为页面中的img元素）绑定了mouseenter和mouseout事件处理函数。当用户第一次把鼠标挪进（或挪出）一张图片时便会触发事件，执行绑定的事件处理函数，在事件处理函数执行过一次之后立即撤销之前的绑定（仅撤销当前元素的绑定，当鼠标挪进或挪出其他图

片时，那些图片仍会响应事件至多一次)。

## 9.2 动态绑定事件处理函数

bind方法有一个限制，即使用bind方法绑定的事件处理函数不会应用到后来添加到DOM中的新元素上。代码清单9-11演示了这种情况。

代码清单9-11 绑定事件处理函数之后再添加元素到DOM

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("img").bind({
            mouseenter: function() {
                $(this).css("border", "thick solid red");
            },
            mouseout: function() {
                $(this).css("border", "");
            }
        });

        $("#row1").append($("<div class='dcell' />")
            .append("<img src='lily.png' />")
            .append("<label for='lily'>Lily:</label>")
            .append("<input name='lily' value='0' required />"));
    });
</script>
...
```

在上面的脚本中，我使用bind方法为所有img元素绑定了mouseenter和mouseout事件的处理函数。然后，使用append方法向页面中插入了几个新元素，其中包括一个img元素。在使用bind方法绑定事件处理函数时，这个img元素并不存在，因此我的事件处理函数不会被绑定到这个新添加的img元素上。结果，页面中有6个img元素在鼠标移到图片上时会显示边框，剩下的那个img元素不会显示边框。

对于上面这样的例子，一个简单的解决方案是再调用一次bind方法，但要弄清楚不同类型的元素都需要哪些事件处理函数非常困难。多亏jQuery提供了一系列方法，让匹配相应选择器的元素在动态添加到DOM之后，自动绑定事先定义好的事件处理函数。表9-4介绍了这些方法的使用法。

表9-4 支持自动绑定事件处理函数的方法

方 法	描 述
on(events, selector, data, function) on(map, selector, data)	为匹配选择器的元素（现在匹配或将来匹配）定义事件处理函数jQuery
off(events, selector, function) off(map, selector)	撤销使用on方法绑定的所有事件处理函数
delegate(selector, eventType, function) delegate(selector, eventType, data, function) delegate(selector, map)	如果jQuery对象包含元素内有匹配（现在匹配或将来匹配）指定选择器的元素，则为其绑定事件处理函数
undelegate() undelegate(selector, eventType)	撤销由delegate方法绑定且匹配选择器与事件类型的事件处理函数



代码清单9-12修改了前一个例子，改用on方法。改动非常之小，却有着重要意义。添加到DOM中的任意一个匹配选择器img的元素，都会自动绑定映射对象中定义的mouseenter和mouseout事件处理函数。

代码清单9-12 使用on方法

```
...
<script type="text/javascript">
  $(document).ready(function() {

    $(document).on({
      mouseenter: function() {
        $(this).css("border", "thick solid red");
      },
      mouseout: function() {
        $(this).css("border", "");
      }
    }, "img");

    $("#row1").append($("#<div class='dcell' />")
      .append("<img src='lily.png' />")
      .append("<label for='lily'>Lily:</label>")
      .append("<input name='lily' value='0' required />"));

  });
</script>
...
```

注意我是在\$(document)对象上调用的on方法，这样才能保证DOM内任意位置的img元素都能绑定我指定的事件处理函数。也可以缩小事件处理函数的影响范围。比如，如果只想让事件处理绑定到#row1元素内的img元素上，就可以像这样调用on方法：

```
...
$("#row1").on(...map..., "img");
...
```

**提示** on方法并非直接把事件处理函数绑定到目标元素。实际上，它是在document对象上绑定了一个事件处理函数，该函数在事件发生时检查触发事件的元素是否匹配选择器。一旦事件与元素匹配成功，就调用绑定的事件处理函数。然而，从实用的角度来说，可以把on方法想象成它努力地把事件处理函数添加到新元素上。

使用on方法可以一次为多个事件指定事件处理函数。这时事件名由空格分隔的多个事件组成，要么用作参数对象的属性名，要么像代码清单9-13那样，不使用参数对象，而是作为on方法的第一个参数。

代码清单9-13 使用on方法为多个事件指定事件处理函数

```
...
<script type="text/javascript">
  $(document).ready(function () {
```

```
function handleMouse(event) {
    if (event.type == "mouseenter") {
        $(this).css("border", "thick solid red");
    } else if (event.type == "mouseout") {
        $(this).css("border", "");
    }
}

$("#row1").on("mouseenter mouseout", "img", handleMouse);

$("#row1").append($("#<div class='dcell' />")
    .append("<img src='lily.png' />")
    .append("<label for='lily'>Lily:</label>")
    .append("<input name='lily' value='0' required />"));
});
</script>
...
```

在代码清单9-13中, on方法的第一个参数没有使用映射对象, 它把#row1元素的后代元素中所有img元素的mouseenter和mouseout事件的处理函数都指定为handleMouse函数。

off方法是on方法的反方法, 它负责从已有元素上删除早先绑定的事件处理函数, 以便阻止它们和新创建的元素响应这些事件。清单9-14演示了off方法的使用法。

#### 代码清单9-14 off方法

```
...
<script type="text/javascript">
    $(document).ready(function () {

        function handleMouse(event) {
            if (event.type == "mouseenter") {
                $(this).css("border", "thick solid red");
            } else if (event.type == "mouseout") {
                $(this).css("border", "");
            }
        }

        $("#row1").on("mouseenter mouseout", "img", handleMouse);

        $("#row1").off("mouseout", "img");

        $("#row1").append($("#<div class='dcell' />")
            .append("<img src='lily.png' />")
            .append("<label for='lily'>Lily:</label>")
            .append("<input name='lily' value='0' required />"));
    });
</script>
...
```

---

**警告** 使用off方法撤销on方法绑定的事件处理函数时, 一定要确保off方法使用的选择器与on方法相同, 否则就不能撤销之前的绑定。

---

## 限制“实时”事件处理函数在DOM中的影响范围

on方法绑定事件处理函数有一个问题，在事件处理函数执行之前，我们不得不耐心等待事件传播到document元素。与on方法相比，delegate方法更加直接，它允许我们在页面中指定任意一个元素作为监听事件的元素。代码清单9-15演示了这种用法。

代码清单9-15 使用delegate方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('#row1').delegate("img", {
            mouseenter: function() {
                $(this).css("border", "thick solid red");
            },
            mouseout: function() {
                $(this).css("border", "");
            }
        });

        $('#row1').append($("<div class='dcell' />")
            .append("<img src='carnation.png' />")
            .append("<label for='carnation'>Carnation:</label>")
            .append("<input name='carnation' value='0' required />"));

        $('#row2').append($("<div class='dcell' />")
            .append("<img src='lily.png' />")
            .append("<label for='lily'>Lily:</label>")
            .append("<input name='lily' value='0' required />"));

    });

</script>
...
```

在代码清单9-15中，我使用delegate方法在元素#row1上添加了一个事件监听器，并指定了一个匹配img元素的选择器。结果是，当img元素触发mouseenter或mouseout事件时，若事件成功传播到#row1元素，我之前指定的事件处理函数就会执行。当添加新img元素到#row1元素时，它能自动绑定之前我指定的事件处理函数；当添加新img元素到#row2元素时，却不会自动绑定。

使用delegate方法最大的优点是性能更好。如果页面相当大并且复杂，又绑定了一大堆事件处理函数，使用on方法绑定事件处理函数就可能遭遇性能问题。通过指定页面中由哪个元素来解释事件处理函数，我们缩短了从事件发生到事件处理函数执行这个过程中事件在DOM中传播的路径。

---

**提示** 要撤销通过delegate方法绑定的事件处理函数，我们需要使用undelegate方法。off方法只能撤销由on方法绑定的事件处理函数。

---

## 9.3 人工调用事件处理函数

使用表9-5中列出的方法，我们可以人工调用绑定在元素上的事件处理函数。

表9-5 用于人工调用事件处理函数的方法

方 法	描 述
trigger(eventType)	触发绑定在jQuery对象内含元素特定事件类型（由eventType指定）上的事件处理函数
trigger(Event)	触发绑定在jQuery对象内含元素特定事件（由Event事件指定）上的事件处理函数
triggerHandler(eventType)	触发绑定在jQuery对象内含的第一个元素上的事件处理函数，既不冒泡，也不实施事件的默认行为

代码清单9-16展示了如何人工调用事件处理函数。

代码清单9-16 人工调用事件处理函数

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("img").bind({mouseenter: function() {
            $(this).css("border", "thick solid red");
        },
        mouseout: function() {
            $(this).css("border", "");
        }
        });

        $("<button>Trigger</button>").appendTo("#buttonDiv").bind("click", function (e) {
            $("#row1 img").trigger("mouseenter");
            e.preventDefault();
        });

    });

</script>
...
```

在这个脚本中，我先使用bind方法为页面中的img元素绑定了两个事件处理函数，然后使用appendTo方法在#buttonDiv元素中插入一个button元素，并使用bind方法为它绑定一个click事件处理函数。

当按下这个按钮时执行之前绑定的事件处理函数：选中#row1元素的所有后代img元素，并调用它们的mouseenter事件处理函数。脚本运行效果见图9-2，就像鼠标同时移动到了3个img元素上那样。

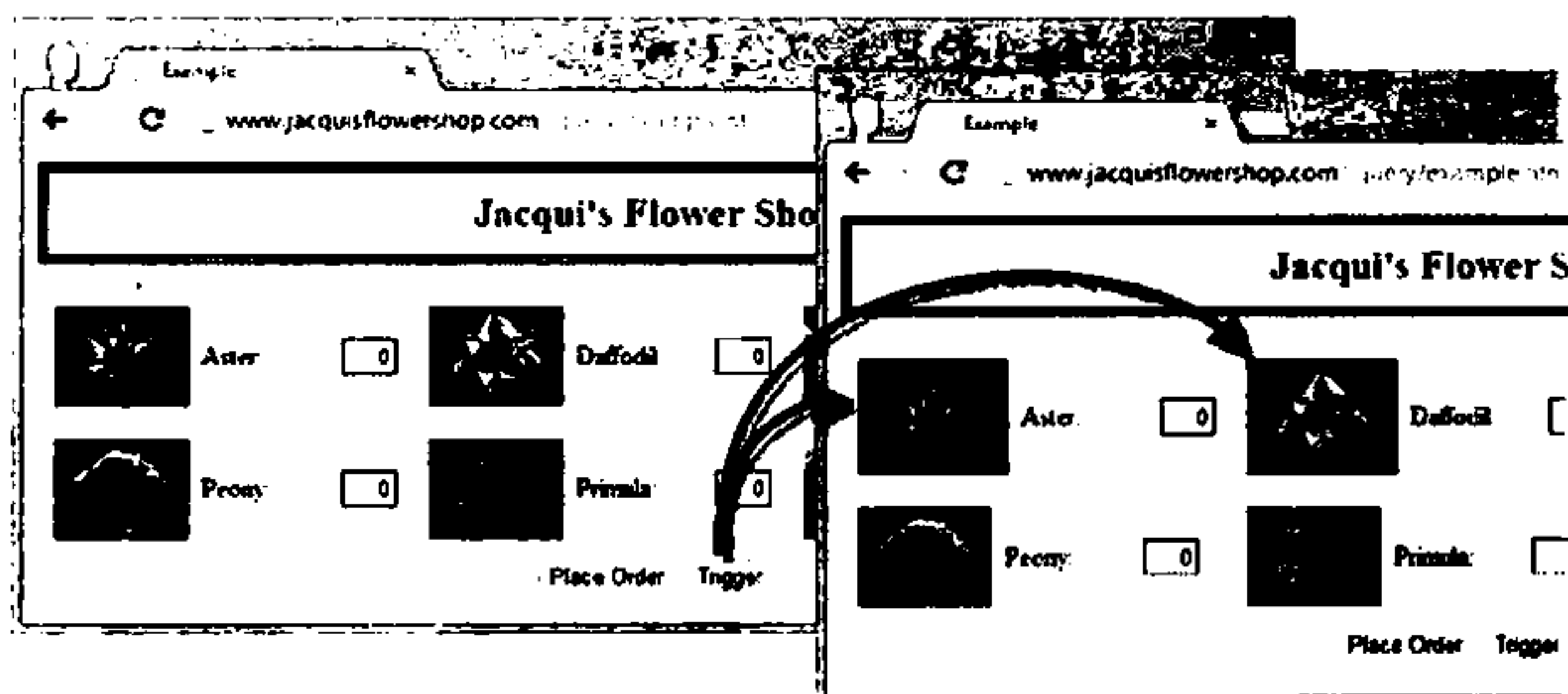


图9-2 人工调用事件处理函数

### 9.3.1 使用Event对象

我们也可以使用Event对象指定要人工调用的其他元素的事件处理函数。如果需要一个事件处理函数内部调用其他元素的（同类型）事件处理函数，采用这个技术就很方便。代码清单9-17演示的就是这种情况。

代码清单9-17 利用Event对象人工调用事件处理函数

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("#row1 img").bind("mouseenter", function() {
            $(this).css("border", "thick solid red");
        });

        $("#row2 img").bind("mouseenter", function(e) {
            $(this).css("border", "thick solid blue");
            $("#row1 img").trigger(e);
        });

    });
</script>
...
```

在代码清单9-17中，我首先用bind方法为#row1元素的所有img后代元素绑定了mouseenter事件处理函数，在事件发生时为图片添加一个红色的边框。接着，我为#row2元素的所有img后代元素也绑定mouseenter事件处理函数，在事件发生时为图片添加一个蓝色的边框。除此之外，我还在这个事件处理函数中添加了下面这条语句：

```
...
$("#row1 img").trigger(e);
...
```

这条额外的语句造成这样的效果（见图9-3）：当我们把鼠标移动到#row2元素的后代img元素上时，

#row1元素的后代img元素的mouseenter事件处理函数也被调用。

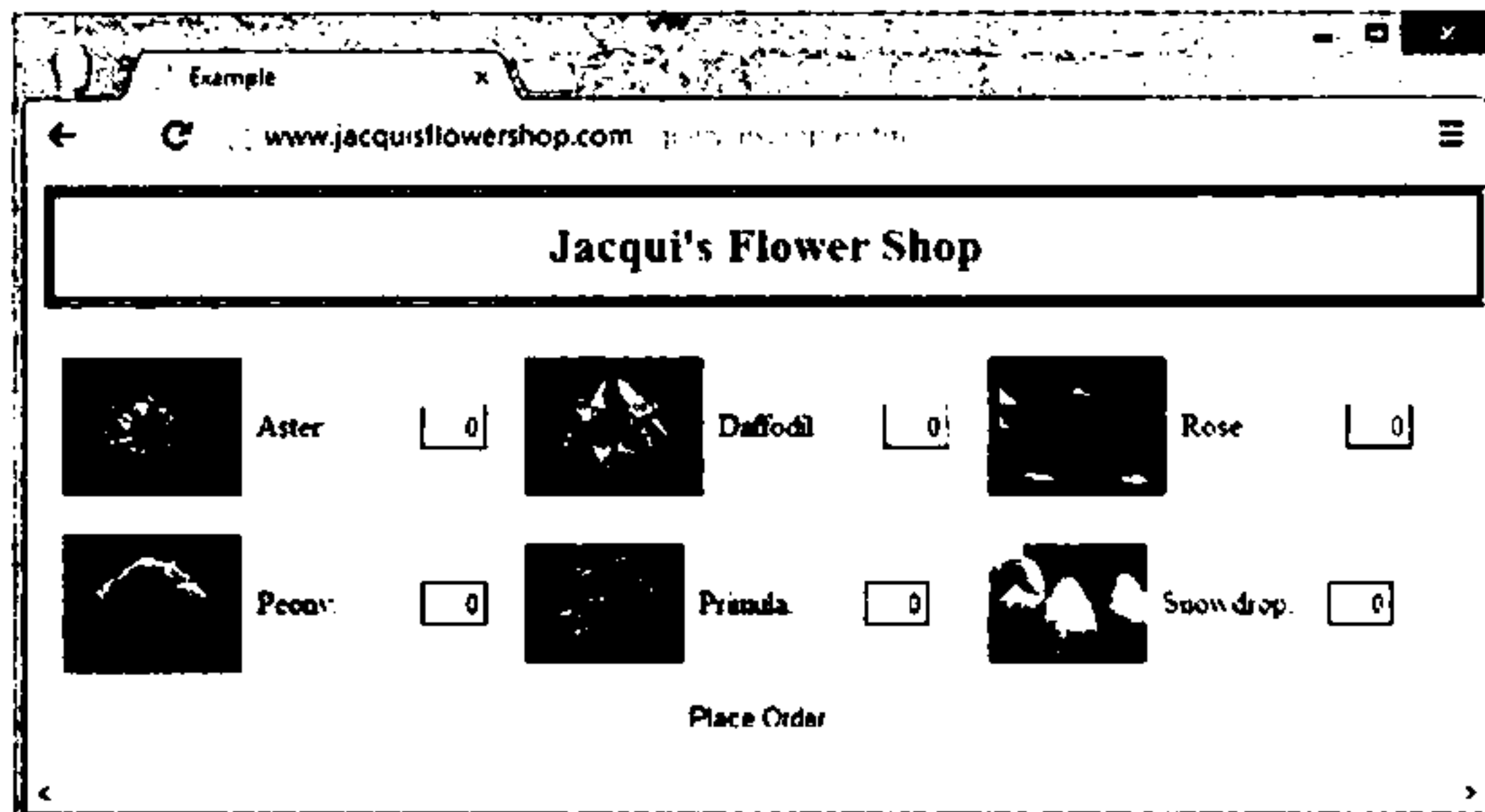


图9-3 使用Event对象指定要人工调用的事件处理函数

如果我们正在处理某一事件，这时希望触发其他元素的同样事件时，使用Event对象指定要触发的事件非常方便。其实我们也可以采取同样简单的指定事件类型方案，效果一模一样。

### 9.3.2 triggerHandler方法

triggerHandler方法不会执行事件的默认行为，也不允许事件沿DOM树向上冒泡。而且，不同于trigger方法，triggerHandler方法只针对jQuery对象中包含的第一个元素调用事件处理函数。代码清单9-18展示了该方法的用法。

代码清单9-18 triggerHandler方法

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#row1 img").bind("mouseenter", function() {
            $(this).css("border", "thick solid red");
        });

        $("#row2 img").bind("mouseenter", function(e) {
            $(this).css("border", "thick solid blue");
            $("#row1 img").triggerHandler("mouseenter");
        });
    });
</script>
...
```

**提示** triggerHandler方法的返回值是事件处理函数的返回值（而不是jQuery对象），这意味着我们不能在调用triggerHandler方法之后再链式调用jQuery方法。

上面脚本的运行结果见图9-4。

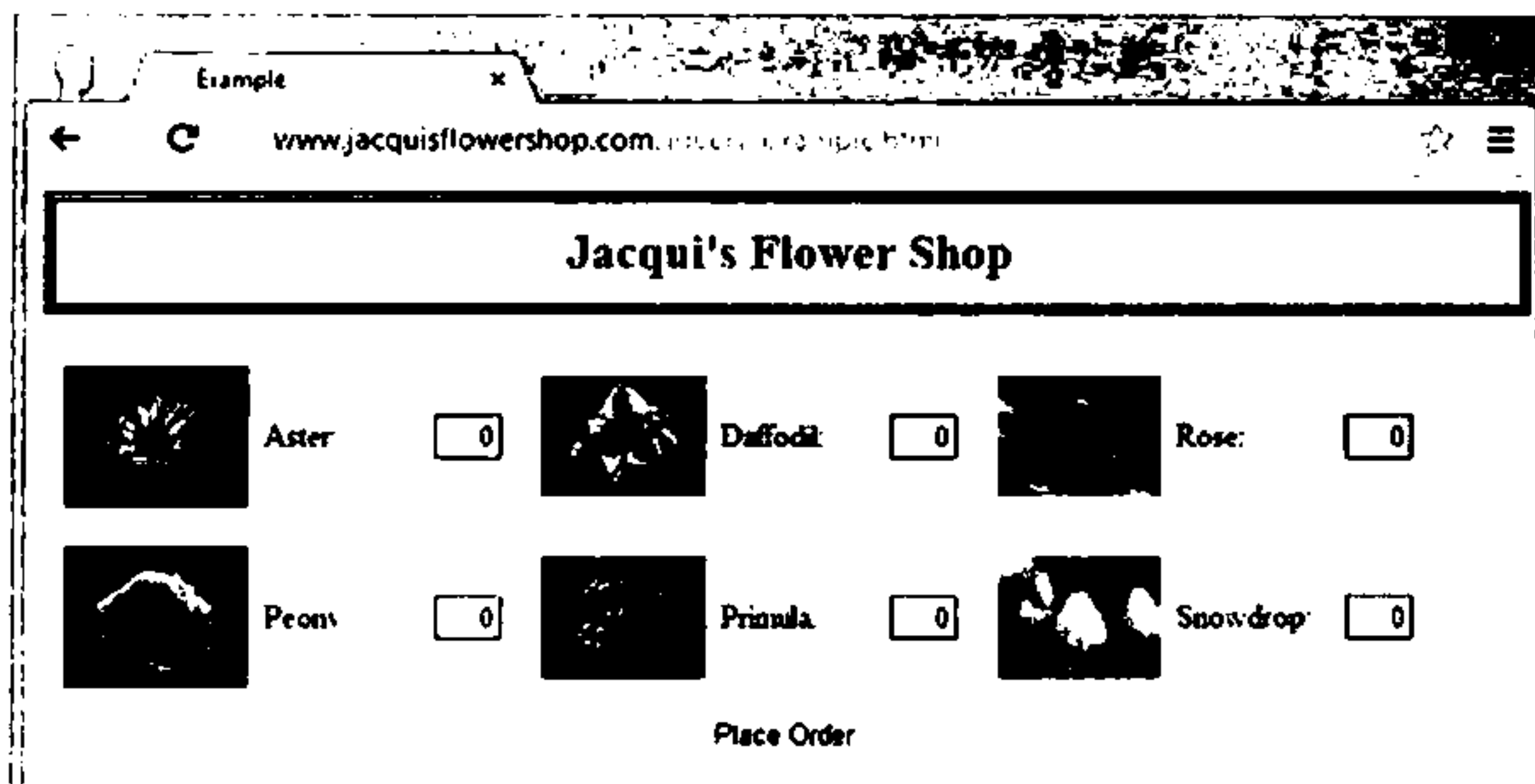


图9-4 使用triggerHandler方法

## 9.4 事件快捷方法

jQuery定义了一些快捷方法，方便我们为那些常用事件绑定事件处理函数。下面的各个表格中，我列出了这些支持函数参数的快捷方法。这都是些最常用的事件，使用这些快捷方法与使用bind方法效果完全相同，不但有效地减少了击键次数，而且（至少对我来说是这样）绑定事件处理函数这件事更清晰容易。代码清单9-19演示了快捷方法的用法。

代码清单9-19 使用事件快捷方法绑定事件处理函数

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("img").mouseenter(function() {
            $(this).css("border", "thick solid red");
        });

    });
</script>
...
```

上面这种写法完全等同于代码清单9-20所示代码的效果，后者使用bind方法绑定mouseenter事件。

代码清单9-20 使用bind方法绑定mouseenter事件处理函数

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("img").bind("mouseenter", function() {
            $(this).css("border", "thick solid red");
        });

    });
```

```
</script>
...
```

真的不错。现在我们已经搞清楚了这些例子的工作原理，真是神清气爽。其实我们还可以用这些快捷方法模拟trigger方法。只要调用事件快捷方法时不带任何参数，这就等同于调用相应的trigger方法。代码清单9-21演示了这种做法。

**代码清单9-21 使用事件快捷方法触发事件处理函数**

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("img").bind("mouseenter", function() {
            $(this).css("border", "thick solid red");
        });

        $("<button>Trigger</button>").appendTo("#buttonDiv").click(function (e) {
            $("img").mouseenter();
            e.preventDefault();
        });
    });
</script>
...
```

我在页面中添加了一个按钮，若点击这个按钮，它就选取页面中的img元素并调用它们的mouseenter事件处理函数。为便于比较，代码清单9-22列出了使用trigger方法实现与代码清单9-20相同效果的脚本。

**代码清单9-22 使用trigger方法**

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("img").bind("mouseenter", function() {
            $(this).css("border", "thick solid red");
        });

        $("<button>Trigger</button>").appendTo("#buttonDiv").click(function (e) {
            $("img").trigger("mouseenter");
            e.preventDefault();
        });
    });
</script>
...
```

在接下来的几节中，我分门别类地列出了jQuery定义的事件快捷方法及其对应的事件。

### 9.4.1 document对象事件快捷方法

表9-6描述了jQuery提供的一些处理document对象事件的快捷方法。



表9-6 document对象事件快捷方法

方 法	描 述
load(function)	即load事件，在页面中的子元素及资源文件载入完成时触发
ready(function)	在页面中的元素已经处理完成，DOM就绪时触发
unload(function)	即unload事件，当用户离开当前页面时触发

ready方法值得特别关注。它并不直接对应某个具体的DOM事件，但在jQuery中分外有用。在第5章我们见识过ready方法的各种用法，在那里我详细解释了如何让脚本在DOM就绪之后才执行以及如何控制ready事件的执行。

## 9.4.2 浏览器事件快捷方法

表9-7描述了浏览器事件，这些事件通常针对的是window对象（不过error事件和scroll事件也适用于页面元素）。

表9-7 浏览器事件快捷方法

方 法	描 述
error(function)	即error事件，在载入外部资源文件出错时触发（如载入图片出错）
resize(function)	即resize事件，当浏览器窗口大小发生变化时触发
scroll(function)	即scroll事件，当用户拖动滚动条时触发

## 9.4.3 鼠标事件快捷方法

表9-8描述了jQuery提供的处理鼠标事件的一系列方法。

表9-8 鼠标事件快捷方法

方 法	描 述
click(function)	即click事件，在用户单击鼠标按钮时触发
dblclick(function)	即dblclick事件，在用户双击鼠标按钮时触发
focusin(function)	即focusin事件，在元素得到焦点时触发
focusout(function)	即focusout事件，在元素失去焦点时触发
hover(function)	在鼠标进入或离开元素时触发。若只指定一个事件处理函数，则鼠标进入或离开元素时都会触发该函数的执行
mousedown(function)	即mousedown事件，当在某元素上按下鼠标时触发
mouseenter(function)	即mouseenter事件，在鼠标进入某元素显示区域时触发
mouseleave(function)	即mouseleave事件，当鼠标离开某元素显示区域时触发
mousemove(function)	即mousemove事件，当鼠标在某元素显示区域内移动时触发
mouseout(function)	即mouseout事件，当鼠标离开某元素显示区域时触发
mouseover(function)	即mouseover事件，当鼠标进入某元素显示区域时触发
mouseup(function)	即mouseup事件，当释放鼠标按钮时触发

hover方法可方便地同时绑定mouseenter和mouseleave事件处理函数。如果我们提供两个函数作为它的参数，那么第一个函数响应mouseenter事件，第二个函数响应mouseleave事件。如果我们只提供一个函数作为它的参数，那么mouseenter和mouseleave事件发生时该函数均会被触发。代码清单9-23展示了hover方法的用法。

代码清单9-23 使用hover方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("img").hover(handleMouseEnter, handleMouseLeave);

        function handleMouseEnter(e) {
            $(this).css("border", "thick solid red");
        };

        function handleMouseLeave(e) {
            $(this).css("border", "");
        }

    });
</script>
...
```

#### 9.4.4 表单事件快捷方法

表9-9描述了jQuery提供的一些处理表单事件的快捷方法。

表9-9 表单事件快捷方法

方 法	描 述
blur(function)	即blur事件，在元素失去焦点时触发
change(function)	即change事件，当元素的值发生变化时触发
focus(function)	即focus事件，在元素得到焦点时触发
select(function)	即select事件，在用户选中某个可选值时触发
submit(function)	即submit事件，当用户提交表单时触发

#### 9.4.5 键盘事件快捷方法

表9-10描述了jQuery提供的一些处理键盘事件的快捷方法。

表9-10 键盘事件的快捷方法

方 法	描 述
keydown(function)	即keydown事件，当用户按下一个键后触发
keypress(function)	即keypress事件，当用户按下一个键并释放（弹起）后触发
keyup(function)	即keyup事件，当用户释放一个（被按下的）键时触发

## 9.5 小结

本章讲解了jQuery对事件的支持情况。由于有了jQuery的强力支持，事件处理才如此简洁而优美。我们能轻易地创建和管理事件处理函数。我酷爱jQuery支持的即时生效事件处理器，只要匹配特定选择器的元素被添加到DOM中，它就自动绑定（事先定义好的）事件处理器。在Web项目里，这极大地减少了我调试事件处理方面问题的时间。下一章的主题是jQuery特效。

## 第 10 章

# jQuery特效

绝大部分用户界面功能都是由jQuery UI库提供的，不过jQuery核心库也包含了一些基础的特效和动画功能，而这正是本章的主题。

尽管我把它说成只是一些“基础的”特效，但它们却可用于实现一些相当高级的动画。本章的重点是动态改变元素的视觉效果，然而我们可以利用这一功能通过多种方式针对一系列CSS属性生成动画。表10-1列出了本章概要。

表10-1 本章概要

问 题	解决方法	代码清单
如何显示与隐藏元素	使用show方法显示元素，使用hide方法隐藏元素	1
如何切换元素的显示状态	使用toggle方法	2、3
如何以动画形式改变元素的可见状态	为show、hide和toggle方法提供一个动画持续时间参数	4
如何在动画完成之后调用一个函数	为show、hide和toggle方法额外提供一个回调函数参数	5~7
如何在垂直方向上以动画方式让元素显示或者隐藏	使用slideDown向下展开元素，使用slideUp向上收起元素，或者使用slideToggle方法切换元素的显示与隐藏	8
如何实现淡入淡出效果	使用fadeIn方法淡入，使用fadeOut方法淡出，使用fadeToggle方法以淡入淡出的方式切换显示，使用fadeTo方法以动画形式变换到某个具体的透明度	9~11
如何生成自定义动画	使用animate方法	12~14
如何管理动画队列	使用queue方法	15、16
如何停止并清理动画队列	使用stop或finish方法	17、18
如何在动画队列之间插入一段延时	使用delay方法	19
如何往动画队列中插入自定义函数	把自定义函数作为queue方法的参数并确保队列中的下一个函数会执行	20、21
如何禁用动画效果	把\$.fx.off属性设置为true	22

### 新版jQuery中与本章有关的变化

jQuery 1.9/2.0定义了新方法finish，用来完成当前特效并清理动画队列。要了解此方法的细节，请参阅10.5.2节。

## 10.1 基础特效

直接显示或者隐藏元素是最简单的特效。表10-2列出了用来显示或者隐藏元素的方法。

表10-2 基础特效方法

方 法	描 述
hide()	立即隐藏jQuery对象内的所有元素
hide(time)、hide(time, easing)	在指定的时间内以动画方式隐藏jQuery对象内的所有元素，并可选一种缓动风格
hide(time, function)、hide(time, easing, function)	在指定的时间内以动画方式隐藏jQuery对象内的所有元素，并可选一种缓动风格，且指定一个回调函数在动画完成之后执行
show()	让jQuery对象内所有元素立即可见
show(time)、show(time, easing)	指定的时间内以动画方式显示出jQuery对象内的所有元素，并可选一种缓动风格
show(time, function)、show(time, easing, function)	在指定的时间内以动画方式隐藏jQuery对象内所有元素，并可选一种缓动风格，且指定一个回调函数在动画完成之后执行
toggle()	立即切换jQuery对象内所有元素的显示状态
toggle(time)、toggle(time, easing)	在指定的时间内以动画方式切换jQuery对象内元素的显示状态，并可选一种缓动风格
toggle(time, function)、toggle(time, easing, function)	在指定的时间内以动画方式切换jQuery对象内元素的显示状态，并可选一种缓动风格，且指定一个回调函数在动画完成之后执行
toggle(boolean)	单向切换jQuery对象内元素的显示状态

代码清单10-1展示了这些效果的最简版本：不带任何参数调用show和hide方法。

代码清单10-1 不带任何参数调用show和hide方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script type="text/javascript">
    $(document).ready(function() {
      $("<button>Hide</button><button>Show</button>").appendTo("#buttonDiv")
        .click(function(e) {
          if ($(e.target).text() == "Hide") {
            $("#row1 div.dcell").hide();
          } else {
            $("#row1 div.dcell").show();
          }
          e.preventDefault();
        });
    });
  </script>
</head>
</html>
```

```

    </script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                    <div class="dcell">
                        <label for="aster">Aster:</label>
                        <input name="aster" value="0" required />
                    </div>
                    <div class="dcell">
                        <label for="daffodil">Daffodil:</label>
                        <input name="daffodil" value="0" required />
                    </div>
                    <div class="dcell">
                        <label for="rose">Rose:</label>
                        <input name="rose" value="0" required />
                    </div>
                </div>
                <div id="row2" class="drow">
                    <div class="dcell">
                        <label for="peony">Peony:</label>
                        <input name="peony" value="0" required />
                    </div>
                    <div class="dcell">
                        <label for="primula">Primula:</label>
                        <input name="primula" value="0" required />
                    </div>
                    <div class="dcell">
                        <label for="snowdrop">Snowdrop:</label>
                        <input name="snowdrop" value="0" required />
                    </div>
                </div>
            </div>
            <div id="buttonDiv"><button type="submit">Place Order</button></div>
        </form>
    </body>
</html>

```

我在DOM中添加了两个button（按钮）元素，并且为这两个按钮的点击事件准备了一个处理函数。这个函数使用text方法读取被点击按钮上的文本，从而判定点击的是哪个按钮，调用的是hide还是show。不论点击哪个按钮，我使用的选择器都是#row1 div.dcell，也就是说点击按钮将改变#row1元素的后代元素中具有dcell类的div元素的显示状态。图10-1展示了我点击Hide按钮前后发生的变化。

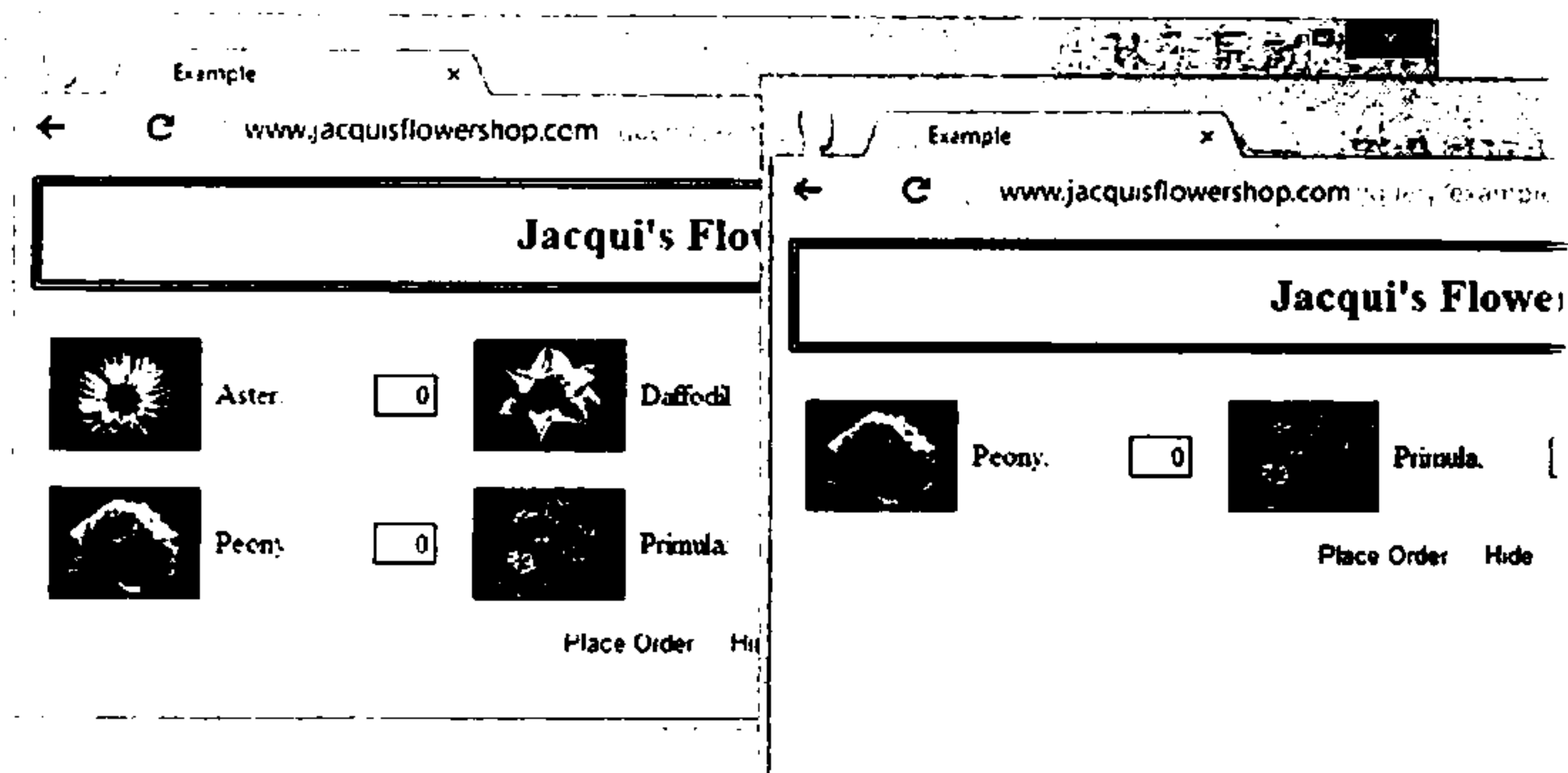


图10-1 使用hide方法隐藏元素

如图10-2所示，点击Show按钮则调用show方法，恢复显示那些隐藏的元素。

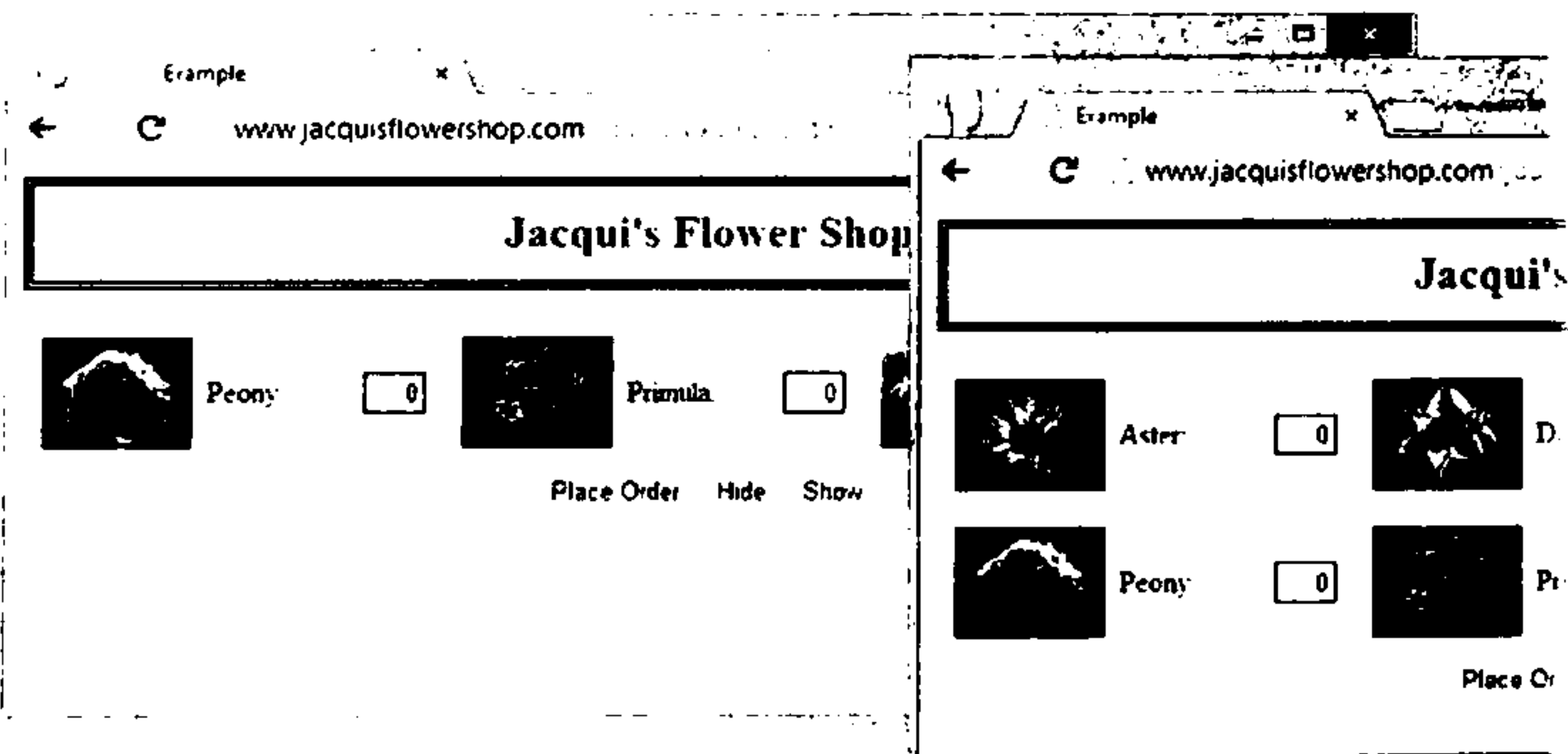


图10-2 使用show方法显示元素

我们很难使用插图表现元素的显示状态变化过程。有几个地方值得注意：首先，变化是立即发生的，被选中的元素立即显示出来或者立即消失，中间没有任何延迟或者动画效果。其次，对已经隐藏的元素调用hide方法或者对已经显示的元素调用show方法，就像这些方法从未被调用一样，不会产生任何效果。最后，如果我们让一个元素隐藏或者显示，那么该元素的所有后代元素也同时隐藏或者显示。

**提示** 我们可以使用:visible和:hidden选择器选择元素。如果需要详细了解jQuery定义的CSS扩展选择器，请阅读第5章。

### 10.1.1 切换元素的显示状态

我们也可以使用toggle方法反转元素的显示状态。代码清单10-2就是一个这样的例子。

代码清单10-2 使用toggle方法反转元素的显示状态

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("<button>Toggle</button>").appendTo("#buttonDiv")
        .click(function(e) {
            $("div.dcell:first-child").toggle();
            e.preventDefault();
        });
    });
</script>
...
```

在这个例子里，我在页面中只添加了一个按钮，点击它的时候，我就在div.dcell:first-child上调用toggle方法，以改变这个元素的显示状态。这个例子产生的效果见图10-3。

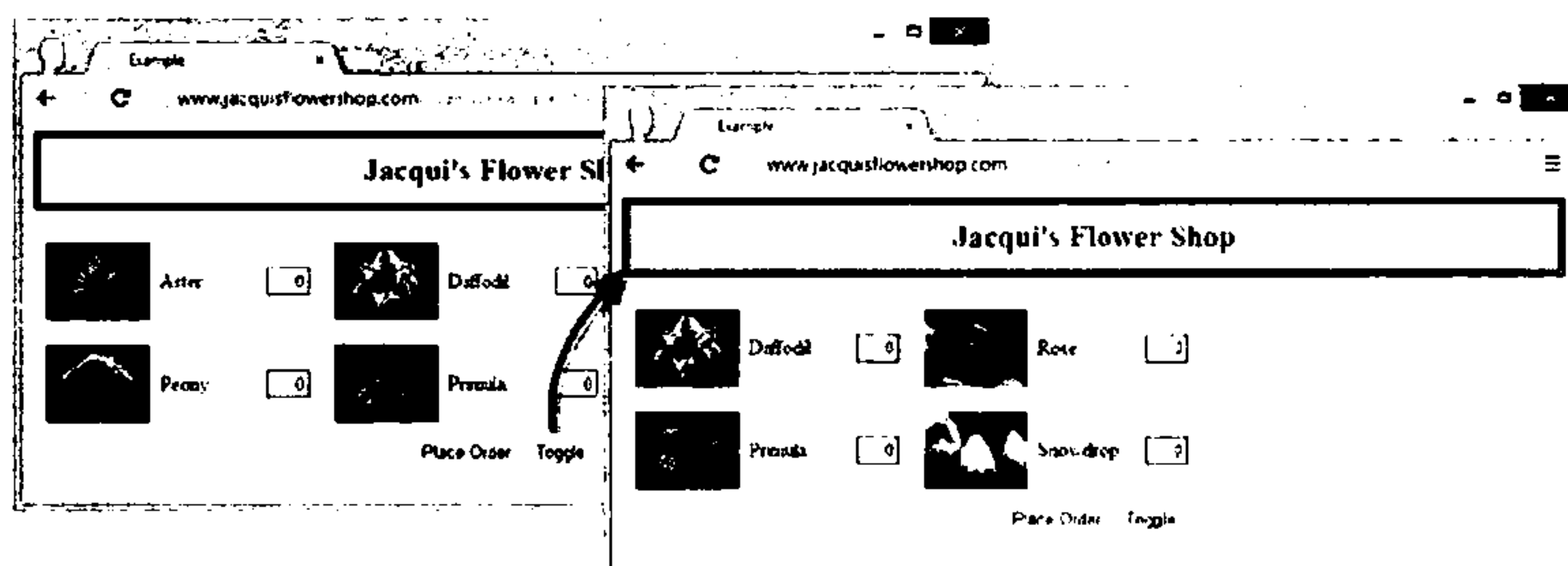


图10-3 反转元素的显示状态

**提示** 当元素隐藏时，这些元素原先占据的空间就立即被旁边的元素挤占。如果你希望隐藏元素并且保留原先占用的空间，可以设置CSS的visibility属性为hidden。

### 10.1.2 单向变换

给toggle方法传递一个布尔值参数，这样就可以限制toggle方法的变换方向。如果传递true，那么jQuery对象中原本隐藏的元素就会显示出来（原本显示的元素不会隐藏起来）。如果传递false，那么就会得到相反的效果，原来显示的元素会隐藏起来，原本隐藏的元素却不会显示出来。代码清单10-3展示了toggle方法的这种用法。该代码清单的效果如图10-3所示。



## 代码清单10-3 使用toggle方法实现单向变换

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $("<button>Toggle</button>").appendTo("#buttonDiv")
        .click(function(e) {
            $("div.dcell:first-child").toggle(false);
            e.preventDefault();
        });
    });
</script>
...

```

## 10.1.3 以动画方式改变元素的显示状态

传递一个持续时间参数给show、hide或者toggle方法，我们就能以动画的方式显示或者隐藏选中元素。在参数指定的时间内，显示状态的变化是逐渐完成的。表10-3列出了这些方法支持的持续时间参数。

表10-3 动画持续时间参数

参 数 值	描 述
<number>	一个十进制整数，表示毫秒（ms）单位的动画持续时间
slow	600 ms的速记写法
fast	200 ms的速记写法

代码清单10-4展示了如何使用动画方式显示和隐藏元素。

## 代码清单10-4 以动画方式改变元素的显示状态

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $("<button>Toggle</button>").appendTo("#buttonDiv")
        .click(function(e) {
            $("img").toggle("fast", "linear");
            e.preventDefault();
        });
    });
</script>
...

```

在代码清单10-4中，持续时间参数使用了fast值，表示页面中的img元素将在600 ms的时间内完成显示状态的变化。

**提示** 当我们使用数值参数表达一段时间时，一定注意不要给参数加引号。也就是说，我们应该使用\$("img").toggle(500)而不是\$("img").toggle("500")。如果我们使用了引号，参数将被直接忽略。

除了持续时间参数，我还多加了一个参数`linear`，它用来指定动画方式，又叫缓动风格或者缓动函数。jQuery内建了两种缓动风格，即`swing`和`linear`。当以`swing`方式播放动画时，动画起始比较慢，逐渐加快，在接近动画终点时又逐渐变慢。`linear`方式播放动画时则匀速播完整个动画。如果我们省略这个参数，则代码默认使用`swing`方式。图10-4展示了以动画方式隐藏img元素的效果。尽管以下面这种方式呈现动画不是十分直观，却也能表达出动画背后发生的事情。

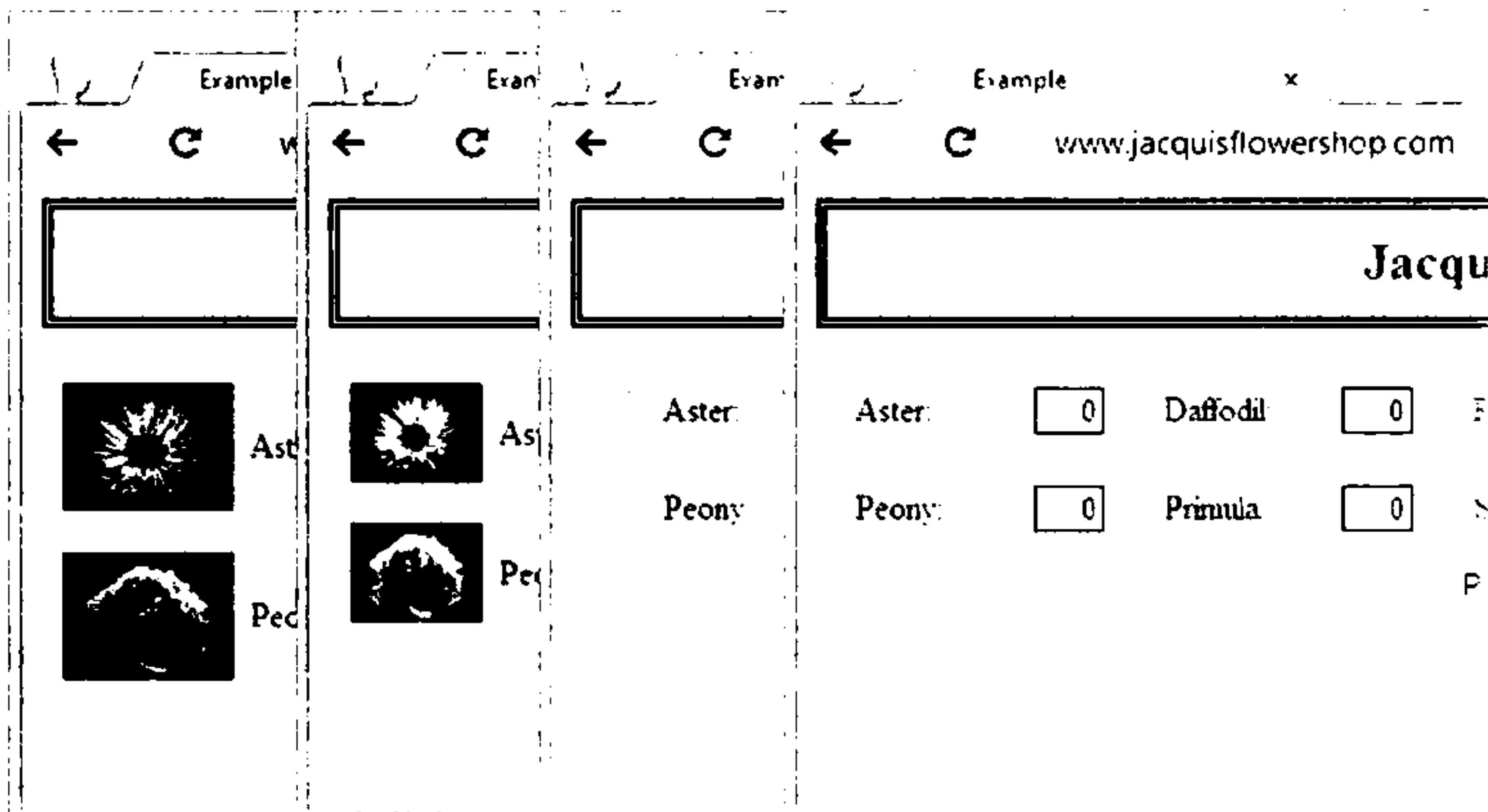


图10-4 以动画方式隐藏元素

如图所示，所谓动画效果就是在水平和垂直方向上减小图片的尺寸，并同时提高透明度。动画结束时，img元素消失。当我们再次点击Toggle按钮时，img元素又都重新出现了（参见图10-5）。

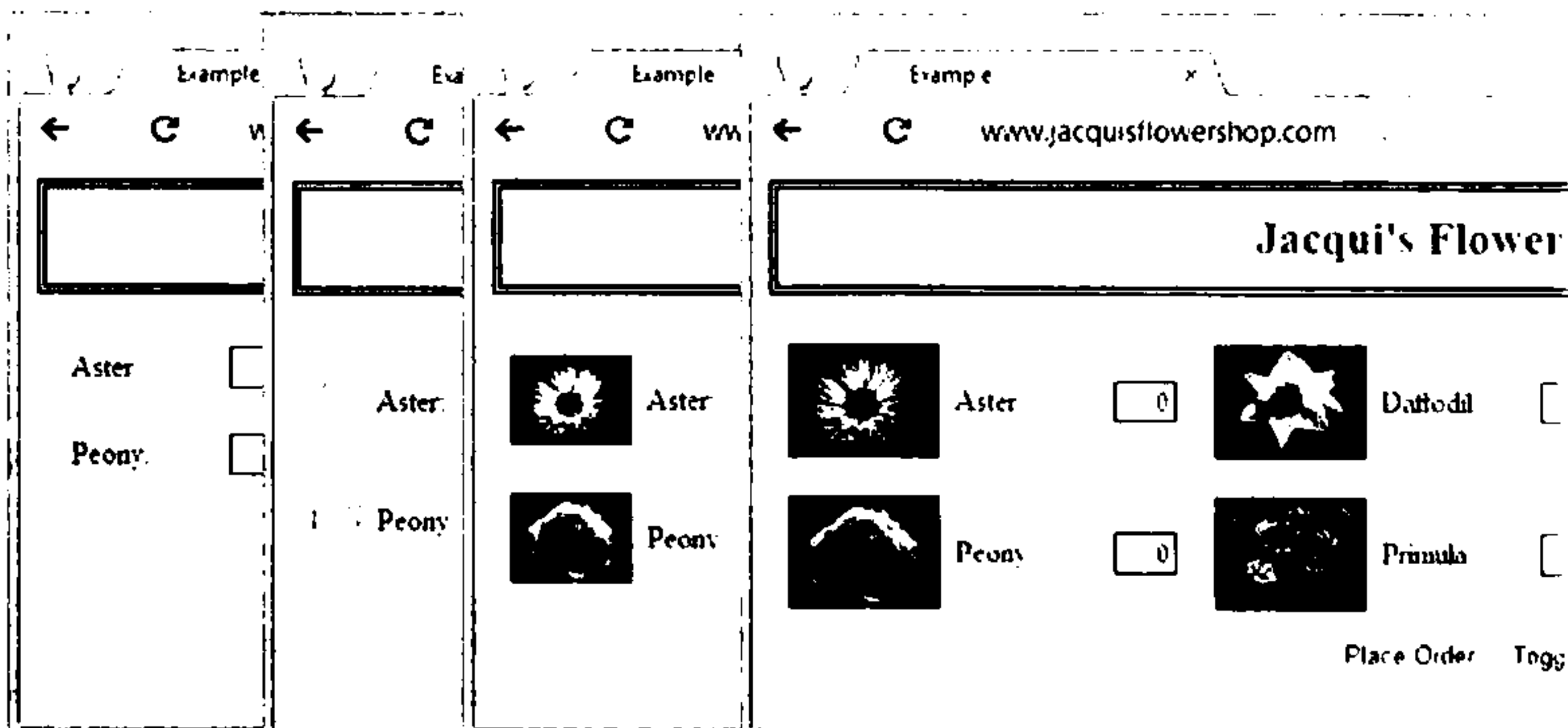


图10-5 以动画方式让元素显示出来

### 10.1.4 使用动画回调函数

我们可以为show、hide和toggle方法额外提供一个函数参数。当动画结束时，这个函数就会被自动调用。如代码清单10-5所示，当我们需要在动画结束时修改其他元素以表示状态变化时，这个功能非常有用。

代码清单10-5 使用事件回调函数

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var hiddenRow = "#row2";
        var visibleRow = "#row1";

        $(hiddenRow).hide();

        function switchRowVariables() {
            var temp = hiddenRow;
            hiddenRow = visibleRow;
            visibleRow = temp;
        }

        function hideVisibleElement() {
            $(visibleRow).hide("fast", showHiddenElement);
        }

        function showHiddenElement() {
            $(hiddenRow).show("fast", switchRowVariables);
        }

        $("<button>Switch</button>").insertAfter("#buttonDiv button")
            .click(function(e) {
                hideVisibleElement();
                e.preventDefault();
            });

    });
</script>
...
```

**提示** 如果你希望在同一元素上执行多段动画，可以使用jQuery标准的链式调用。请参阅10.5节以详细了解这一技术。

为了使这个例子更清楚，我把动画行为分解为几个独立的函数。首先，我隐藏起其中一行（表格布局中的一个div元素），并定义了两个用来跟踪哪一行可见哪一行不可见的变量。我还在页面中添加了一个按钮，点击时将调用hideVisibleElement函数。在这个函数中，我使用了hide方法以动画方式隐藏当前可见的行：

```
...
$(visibleRow).hide("fast", showHiddenElement);
...
```

在上面这行代码中，我指定动画结束后要立即执行showHiddenElement函数。

**提示** 回调函数本身没有任何参数，不过this变量被设置为DOM中动画进行中的元素。如果有多个元素同时变化（动画中），那么每个元素在动画结束后都会执行一次这个回调函数。

这个函数使用show方法以动画方法恢复元素的显示：

```
...
$(hiddenRow).show("fast", switchRowVariables);
...
```

又一次，我指定动画结束之后立即执行一个函数。在上面这行代码中，我指定的是switchRowVariables函数，它调换了跟踪元素可见状态的变量的值，这样当我们再次点击按钮时才能得到正确的结果（动画效果施加到正确的元素上）。于是我们得到这样的结果，点击按钮，当前显示的行被原先隐藏的行以动画方式平稳地替换掉。图10-6展示了这一动画效果。（不过，我再说一次，只有在浏览器里运行这个例子才能更好地体验动画效果。）

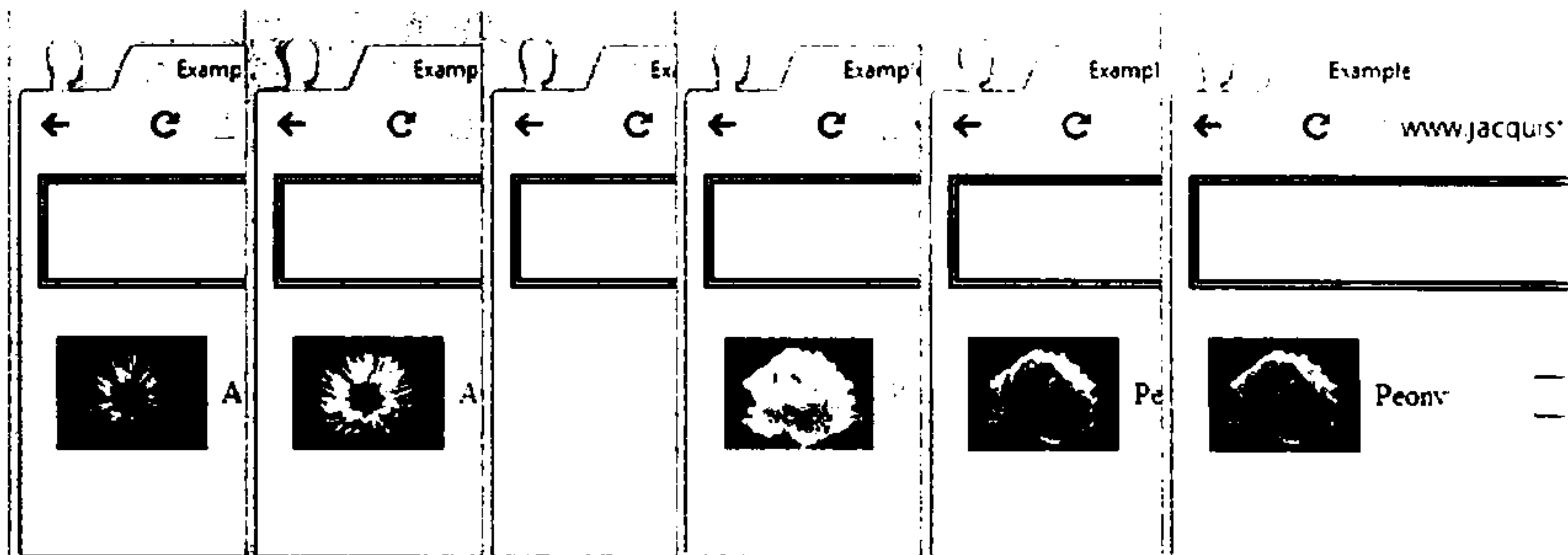


图10-6 用回调函数链接动画特效

通常，我们并不需要像上面那样把代码拆成一些独立的函数。在代码清单10-6中，我们改用匿名函数实现上一个例子。

#### 代码清单10-6 使用内联回调函数

```
...
<script type="text/javascript">
  $(document).ready(function() {

    var hiddenRow = "#row2";
    var visibleRow = "#row1";

    $(hiddenRow).hide();

    $("<button>Switch</button>").insertAfter("#buttonDiv button")
```

```

        .click(function(e) {
            $(visibleRow).hide("fast", function() {
                $(hiddenRow).show("fast", function() {
                    var temp = hiddenRow;
                    hiddenRow = visibleRow;
                    visibleRow = temp;
                });
            });
            e.preventDefault();
        });
    });
</script>
...

```

### 10.1.5 创建循环动画

我们可以使用回调函数生成循环动画，具体示例见代码清单10-7。

代码清单10-7 使用回调函数生成循环动画

```

...
<script type="text/javascript">
    $(document).ready(function() {

        $("<button>Toggle</button>").insertAfter("#buttonDiv button")
            .click(function(e) {
                performEffect();
                e.preventDefault();
            });

        function performEffect() {
            $("h1").toggle("slow", performEffect)
        }
    });
</script>
...

```

在这个例子中，点击按钮则执行performEffect函数。这个函数使用toggle方法改变页面中h1元素的显示状态，并把自身作为回调参数传递给自己。最终，我们得到h1元素不断重复可见、隐藏、可见、隐藏的过程，如图10-7所示。

**提示** 让函数自己调用自己会带来一个问题，这样很快就会耗光JavaScript的调用栈，从而导致脚本停止运行。解决这个问题最简单的方式是使用setTimeout函数，比如\$('h1').toggle("slow", setTimeout(performEffect, 1))。实际上这个方法避免了递归调用，根本不可能耗光JavaScript的调用栈；即使把这个页面扔到一边再不管它，这个动画也会一直运行。但请记住，一定要谨慎使用finish方法（参阅10.5.2节）。

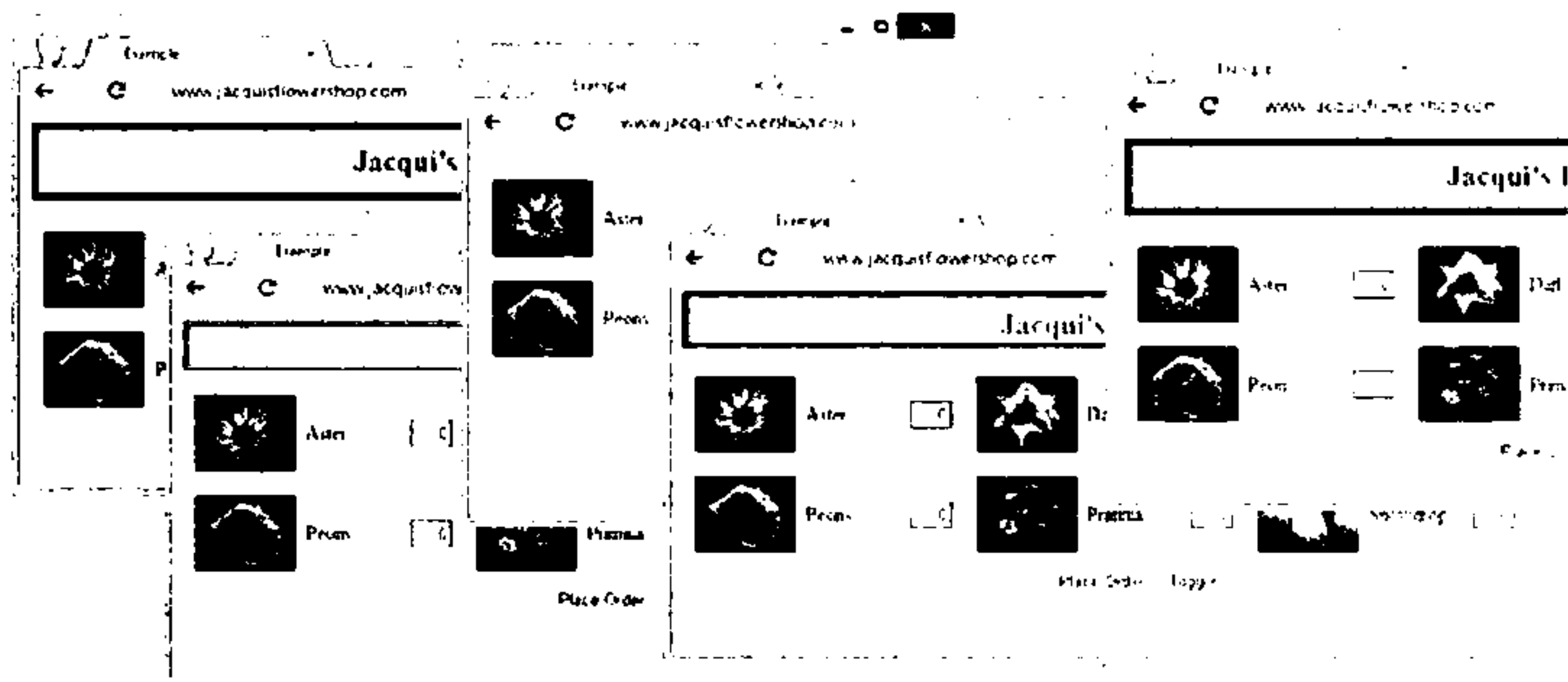


图10-7 使用循环实现特效

### 谨慎地使用特效

在我看来，应该尽量少使用循环特效。也就是说，我们应该仅在为了满足某种需要时使用，这种需要应该是来自用户的需要，而不是你卖弄jQuery特效技巧的需要。一般来说，我们应该慎重使用每一种特效。特效很容易让开发人员觉得很棒，然而，特效的不当使用很容易摧毁用户对应用程序的好印象，特别是那些用户每天都会使用的程序。

举一个简单的例子，我是一个跑步爱好者（喜欢跑而已，跑得并不好）。我有一块跑步专用的手表，它能收集一些类似于我的心率、速度、距离以及消耗的卡路里等数据。跑完之后，我会把这些数据上传到手表制造商的网站，它们会替我保存并分析这些数据。

接下来就是痛苦的经历了。每一次点击页面中的按钮，页面都要先演示一个长长的淡入特效然后才显示我想看内容。我知道浏览器早已接收到我需要的数据，因为数据是一点点淡入显示出来的，从开始动画到我能阅读，足足需要两秒钟。两秒钟听起来不长，然而事实并非如此，特别是在我随时都需要在5到10项数据之间来回查看的时候。

我确信这个程序的设计者一定觉得这种特效非常棒，而且能增强用户体验。然而，并没有。这些特效使我恼怒，这个程序的体验简直糟透了——后来我便买了另一家公司的产品。

这个网站为这块手表（我已经把它扔掉了）提供了一些很有用的数据分析工具，然而由于这些特效如此烦人，我宁愿多花点几百美元寻找替代品。如果不是那该死的特效，我今天很可能已经是一名马拉松长跑冠军，而且以惊人的频率消费着啤酒和匹萨了！

你可能认为我言过其辞（指特效……不是匹萨），那就请在本章随便找一个例子，把动画持续时间设置为2 s（2000 ms），感受一下等待动画结束的时长吧。

我的建议是尽量少用特效。我倾向于仅在修改DOM并且过于突兀时（比如元素突然从页面上消失）使用特效。使用特效时，动画持续时间我总是使用short参数，也就是200 ms。我从来不使用无限循环的特效，因为那会让用户头痛。我竭力主张大家花时间思考如何让应用程序或者网站吸引用户使用，删除那些不能简化手头任务的一切东西。让网站美观并没什么错，然而只有那些有用并且美观的网站才是最棒的。

## 10.2 滑动特效

jQuery支持一套滑动效果，元素可以由上而下逐渐滑出，也可以由下而上逐渐卷起消失。表10-4列出了这些特效对应的方法。

表10-4 滑动特效方法

方 法	描 述
slideDown()、slideDown((time, function)、slideDown(time, easing, function)	让元素内容自上而下逐渐显示出来
slideUp()、slideUp(time, function)、slideUp(time, easing, function)	让元素自下而上逐渐消失
slideToggle()、slideToggle(time, function)、slideToggle(time, easing, function)	使用滑动特效反转元素的显示状态

这些方法都是在垂直方向上绘制动画，其参数的含义与那些基本特效方法相同。我们可以指定动画的持续时间、缓动风格和回调函数。代码清单10-8展示了滑动特效的用法。

代码清单10-8 滑动特效

```
...
<script type="text/javascript">
  $(document).ready(function() {
    $("<button>Toggle</button>").insertAfter("#buttonDiv button")
      .click(function(e) {
        $("h1").slideToggle("fast");
        e.preventDefault();
      });
  });
</script>
...
```

在这个脚本中，我使用slideToggle方法反转h1元素的可见状态（参见图10-8）。

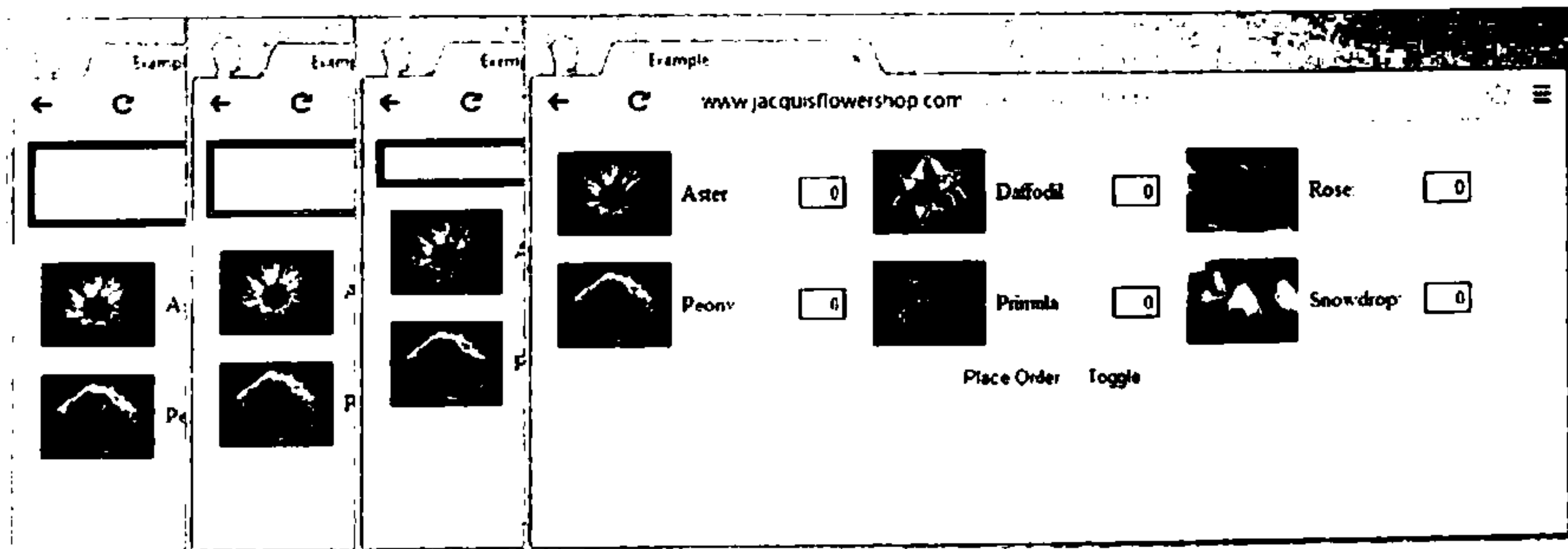


图10-8 利用滑动特效使元素逐渐显示出来

这张图展示了h1元素是如何一步步显示出来的。由于jQuery通过控制元素的高度实现动画，因此h1元素是被剪裁，而不是缩放。图10-9可以为你解释这一过程。

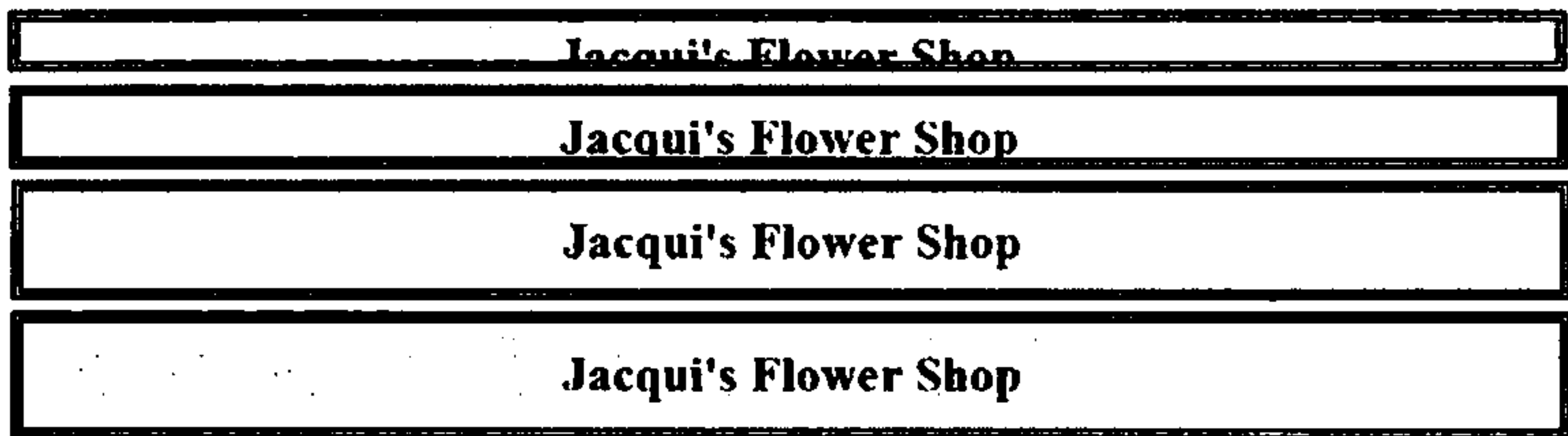


图10-9 jQuery通过改变元素的高度生成特效

这张图以特写的方式展示了h1元素是如何一步步变得可见的。注意，h1文本的大小并没有发生变化，变化的只是暴露出来的尺寸。不过，这一规则并不适用于图片，浏览器总是会缩放图片。如果你观察得足够仔细，会发现整个背景图片已经显示出来，它在不断地放大以适应高度的变化。

### 10.3 淡入淡出特效

淡入淡出特效是指通过减小或者增加元素的不透明度（或者反过来说，即通过增加或者减少元素的透明度）来隐藏或者显示元素。表10-5列出了用来实现淡入淡出特效的那些方法。

表10-5 淡入淡出特效方法

方 法	描 述
<code>fadeOut()</code> 、 <code>fadeOut(timespan)</code> 、 <code>fadeOut(timespan, function)</code> 、 <code>fadeOut(timespan, easing, function)</code>	以淡出（增加透明度）的方式隐藏元素
<code>fadeIn()</code> 、 <code>fadeIn(timespan)</code> 、 <code>fadeIn(timespan, function)</code> 、 <code>fadeIn(timespan, easing, function)</code>	以淡入（减小透明度）的方式把元素显示出来
<code>fadeTo(timespan, opacity)</code> 、 <code>fadeTo(timespan, opacity, easing, function)</code>	增加或者减少透明度到指定大小
<code>fadeToggle()</code> 、 <code>fadeToggle(timespan)</code> 、 <code>fadeToggle(timespan, function)</code> 、 <code>fadeToggle(timespan, easing, function)</code>	以改变透明度的方式反转元素的显示状态

`fadeOut`、`fadeIn`和`fadeToggle`方法的使用方法和其他特效方法一样。就像前面的例子一样，我们可以指定持续时间、缓动风格以及回调函数。代码清单10-9演示了如何淡入淡出。

代码清单10-9 以淡入淡出方式显示及隐藏元素

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("<button>Toggle</button>").insertAfter("#buttonDiv button")
            .click(function(e) {
                $("img").fadeToggle();
                e.preventDefault();
            });
    });
</script>
...
```



我先选中页面上所有的img元素，然后调用fadeToggle方法，顺便演示一下这个特效的局限性。图10-10展示了隐藏这些元素时发生的事情。

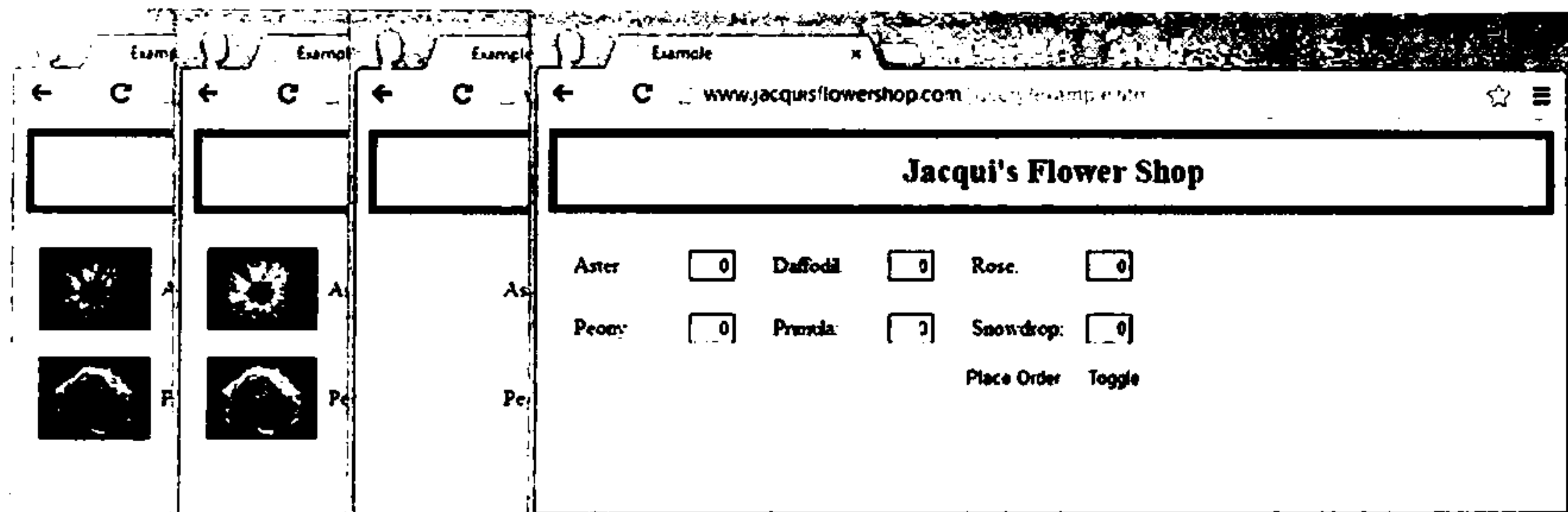


图10-10 淡入淡出特效

淡入淡出特效仅处理不透明度属性，不像别的特效那样会改变选中元素的尺寸。这意味着在元素完全透明之前的渐隐渐现效果非常棒，而变得完全透明之时jQuery隐藏这些元素，将页面切换到新布局。如果我们不小心使用这一功能的话，这最后的一步会显然有些突兀。

## 淡入或淡出至指定的不透明度

利用fadeTo方法，我们可以让元素淡入或淡出至指定的不透明度。不透明度的取值范围为0（完全透明）~1（完全不透明）。这个方法不会改变元素的可见性（即使完全透明状态），因此就避免了前面提到的页面布局的变化。代码清单10-10演示了这个方法的使用。

### 代码清单10-10 淡出至指定的不透明度

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("<button>Fade</button>").insertAfter("#buttonDiv button")
        .click(function(e) {
            $("img").fadeTo("fast", 0);
            e.preventDefault();
        });
    });
</script>
...
```

在这个例子里，我让所有的img元素淡出至完全透明。与fadeOut方法相比，除了不会隐藏img元素，示例所达到的效果完全相同（参见图10-11）。

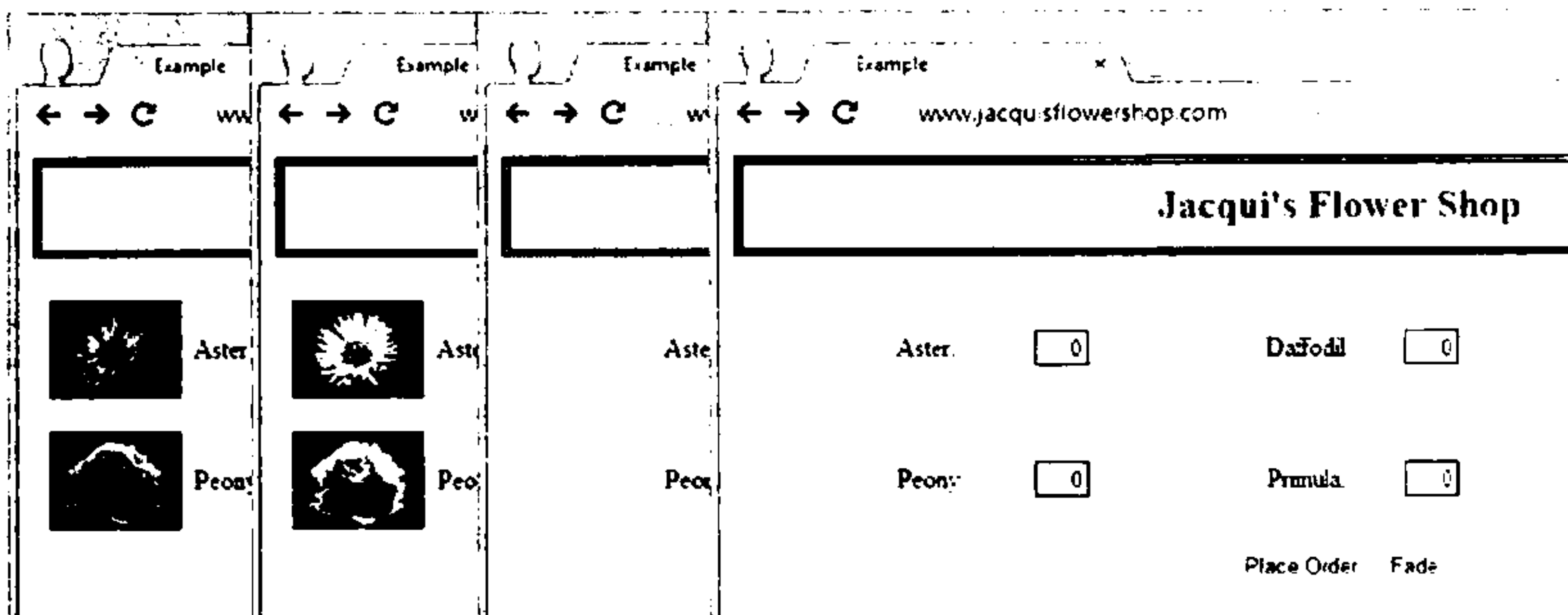


图10-11 使用fadeTo方法让元素淡出

我们不是必须让元素完全淡入或者完全淡出。如代码清单10-11所示，我们可以指定一个中间值。

#### 代码清单10-11 淡出至指定的透明度

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("<button>Fade</button>").insertAfter("#buttonDiv button")
        .click(function(e) {
            $("img").fadeTo("fast", 0.4);
            e.preventDefault();
        });
    });
</script>
...
```

这个例子的效果见图10-12。

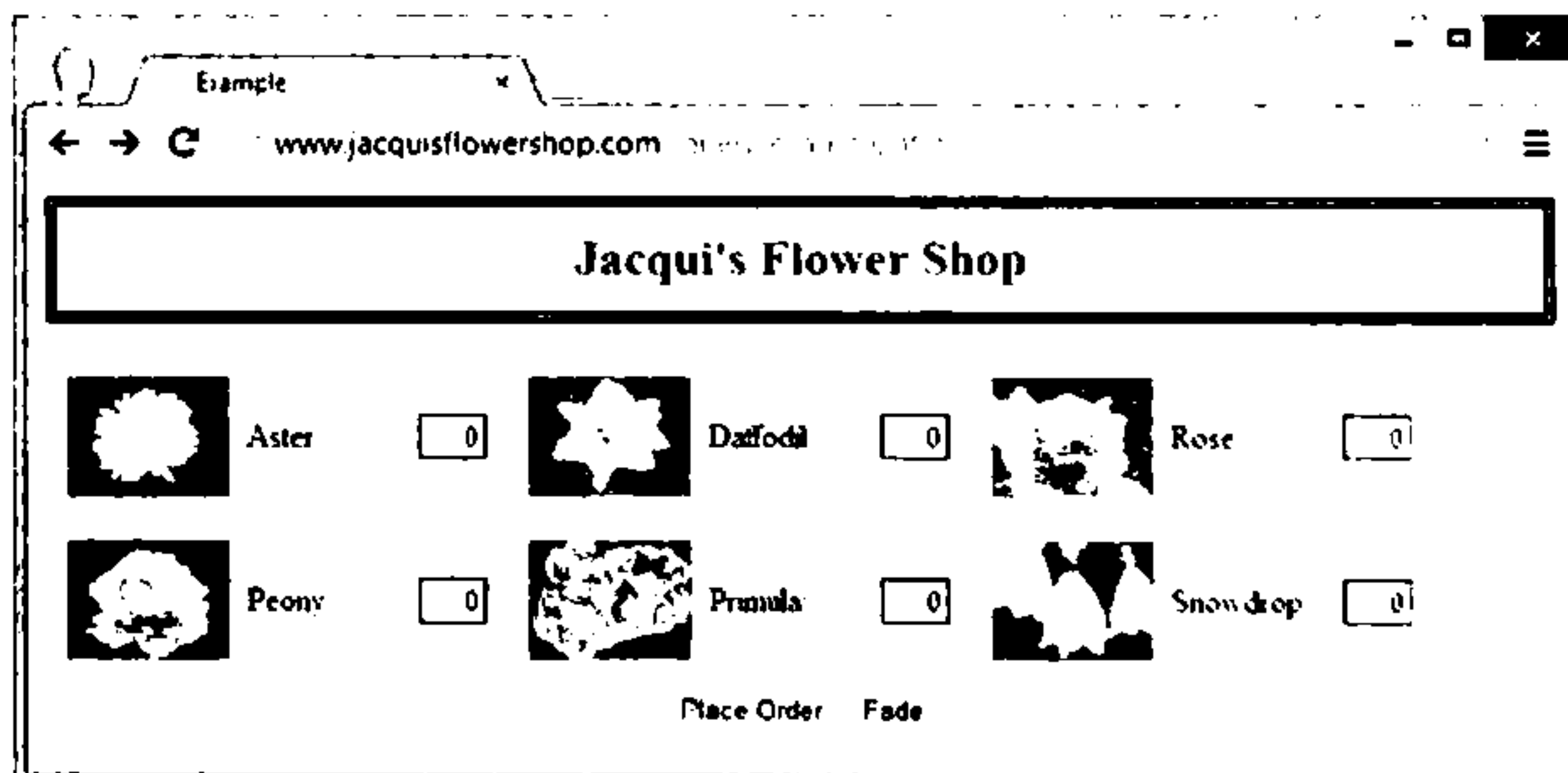


图10-12 淡出至指定的不透明度

## 10.4 实现定制特效

除了基本的滑动及淡入淡出特效，我们也可以利用jQuery实现定制特效。表10-6列出了用来实现定制特效的方法。

表10-6 定制特效的方法

方 法	描 述
<code>animate(properties)</code> 、 <code>animate(properties,time)</code> 、 <code>animate(properties,time,function)</code> 、 <code>animate(properties,time,easing,function)</code>	利用一个或多个CSS属性实现动画，支持可选的动画持续时间、缓动风格以及回调函数
<code>animate(properties,options)</code>	利用一个或多个CSS属性实现动画，使用映射对象指定各个选项

只要一个CSS属性接受纯数值（比如height属性），jQuery就能利用它生成动画。

**注意** 只能根据数值CSS属性制作特效意味着无法实现颜色上的动画效果。有几种办法可以解决这个问题。第一种（也是在我看来最好的一种）是使用jQuery UI，详见本书第四部分。如果不愿意使用jQuery UI，你可以试着使用浏览器支持的原生CSS特效。原生特效的性能非常好，只是各种浏览器对CSS特效的支持情况有好有坏，而较老的浏览器则完全不支持。如果打算详细了解CSS特效，请参阅我的另一本书《HTML5权威指南》。最后一种方法，也是我最不推荐使用的一种方法，是使用jQuery插件。颜色特效难以直接处理，而我到现在为止找到的最喜欢也最可靠的一个是来自<https://github.com/jquery/jquery-color>的jQuery Color。

我们可以使用映射对象指定那些用于制作动画的属性，而且，如果你喜欢，也同样可以用映射对象指定制作动画的种种选项。代码清单10-12展示了如何生成定制动画。

代码清单10-12 定制动画

```
...
<script type="text/javascript">
  $(document).ready(function() {

    $("form").css({"position": "fixed", "top": "70px", "z-index": "2"});
    $("h1").css({"position": "fixed", "z-index": "1", "min-width": "0"});

    $("<button>Animate</button>").insertAfter("#buttonDiv button")
      .click(function(e) {

        $("h1").animate({
          height: $("h1").height() + $("form").height() + 10,
          width: ($("form").width())
        });

        e.preventDefault();
      });

  });
</script>
...
```

在代码清单10-12中,我打算让h1元素的尺寸变大,这样form元素的背景图片也就会随之变大。在开始这个任务之前,我需要先调整一下受动画影响元素的样式。我当然可以使用第3章中提到的样式表来做这件事,不过这既然是一本讲解jQuery的书,我还是选择了使用JavaScript来修改样式。为了让动画容易管理,我把form和h1元素的position属性都设置成了fixed,并通过z-index属性来保证h1元素显示在form元素的下方。

我还在页面中添加了一个按钮,点击它则调用animate方法生成动画。这个动画会改变元素的height和width属性,它们的目标值都是使用jQuery方法获得的。这个动画的效果见图10-13。

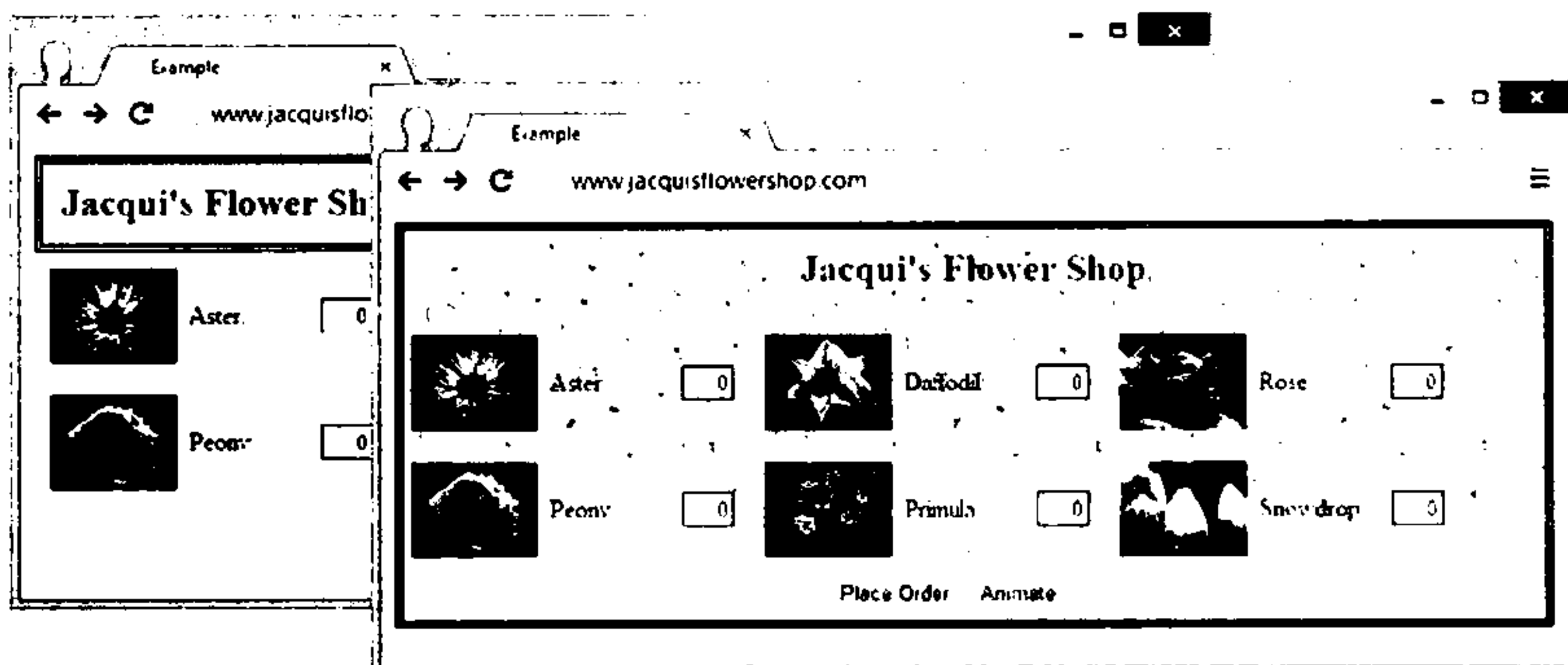


图10-13 定制动画

图中只呈现了动画的开始和结束画面,实际上这个定制动画相当流畅平滑,就像前面讲过的其他特效一样。当然,我们还可以使用持续时间参数和缓动风格参数控制动画的行为。

#### 10.4.1 使用绝对值设置动画属性

注意,我们仅在程序中指定了参与动画的属性的目标值(动画结束时值)。这些属性的起始值就是动画开始前的当前值。在上面这个例子里,这些值来自于一些别的jQuery方法,不过你也可以采用方式。首先,如代码清单10-13所示,也是最直白不过的方式,你可以直接指定一个绝对数值。

代码清单10-13 使用样式属性的绝对值生成定制动画

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("form").css({"position": "fixed", "top": "70px", "z-index": "2"});
        $("h1").css({"position": "fixed", "z-index": "1", "min-width": "0"});

        $("<button>Animate</button>").insertAfter("#buttonDiv button")
            .click(function(e) {
                $("h1").animate({
                    left: 50,
```

```

        height: $("h1").height() + $("form").height() + 10,
        width: ($("form").width())
    });

    e.preventDefault();
});

});
</script>
...

```

在代码清单10-13中, 我为动画中增加了left属性, 指定left的绝对值为50 (表示50像素)。这会使h1元素右移 (参见图10-14)。

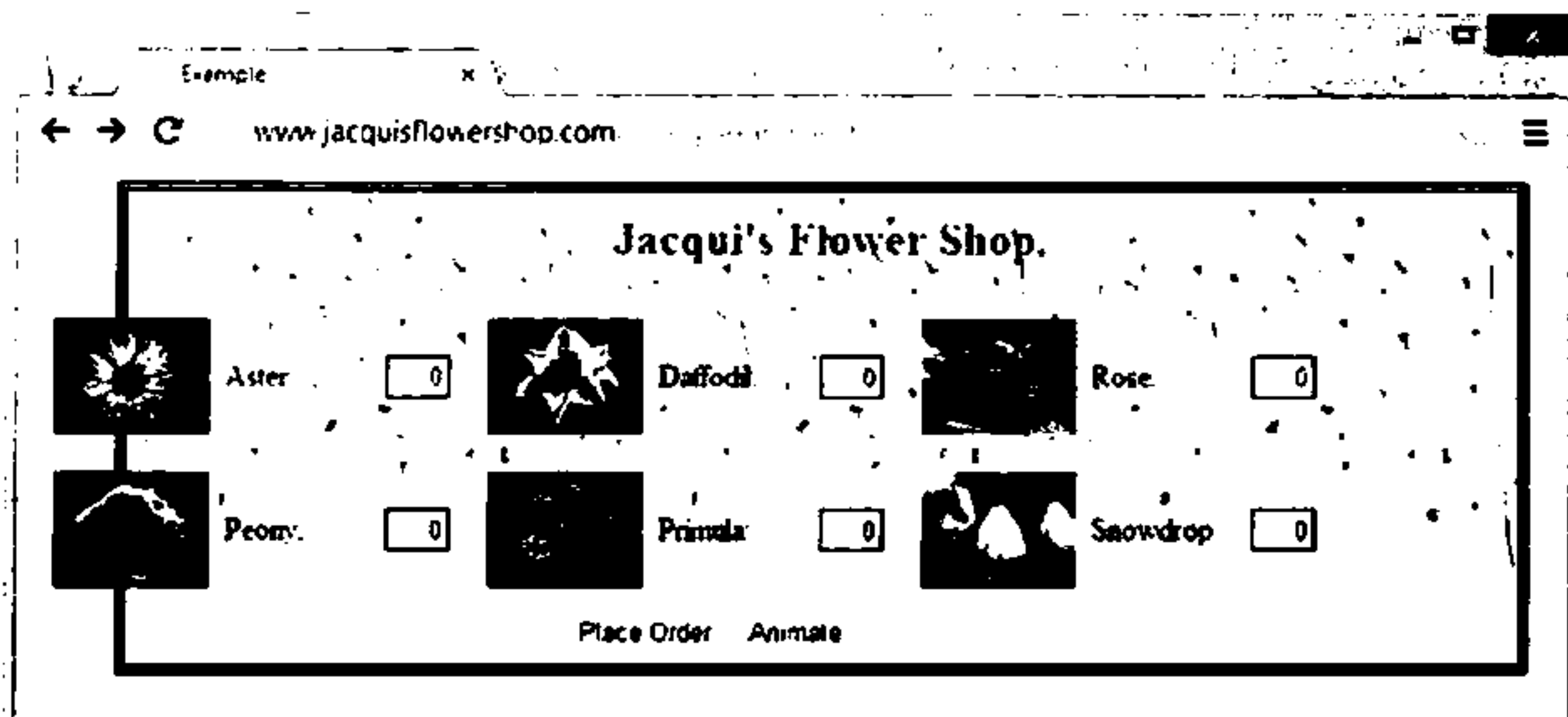


图10-14 使用一个固定的属性值生成定制动画

### 10.4.2 使用相对值设置动画属性

我们也可以使用相对值设置动画属性, 可以使用+=符号设置相对于当前值的增量, 还可以使用-=符号设置相对于当前值的减小量。使用相对值设置属性的具体例子见代码清单10-14。

代码清单10-14 使用相对值设置定制动画属性

```

...
<script type="text/javascript">
    $(document).ready(function() {

        $("form").css({"position": "fixed", "top": "70px", "z-index": "2"});
        $("h1").css({"position": "fixed", "z-index": "1", "min-width": "0"});

        $("<button>Animate</button>").insertAfter("#buttonDiv button")
            .click(function(e) {
                $("h1").animate({
                    height: "+=100",
                    width: "-=700"
                });

                e.preventDefault();
            });
    });

```

```

    });
  });
</script>
...

```

## 10.5 创建并管理动画队列

所谓特效，就是jQuery创建并维护着的一个动画队列。表10-7列出了jQuery提供的一整套方法，它们可以用来获取队列信息或者控制动画队列。

表10-7 动画队列方法

方 法	描 述
queue()	返回针对jQuery对象内元素的正在执行的动画队列
queue(function)	在动画队列的末尾添加一个动画（函数）
dequeue()	从针对jQuery对象内元素的正在执行的动画队列中取出第一个动画并执行这个动画
stop()、stop(clear) stop(clear, jumpToEnd)	停止当前动画
finish()	停止当前动画并清理动画队列
delay(time)	在两个动画之间插入一段延时

如代码清单10-15所示，把一串动画方法链接在一起就创建了一个动画队列。

代码清单10-15 生成动画队列

```

...
<script type="text/javascript">
  $(document).ready(function() {

    $("form").css({"position": "fixed", "top": "70px", "z-index": "2"});
    $("h1").css({"position": "fixed", "z-index": "1", "min-width": "0"});

    var timespan = "slow";

    cycleEffects();

    function cycleEffects() {
      $("h1")
        .animate({left: "+=100"}, timespan)
        .animate({left: "-=100"}, timespan)
        .animate({height: 223,width: 700}, timespan)
        .animate({height: 30,width: 500}, timespan)
        .slideUp(timespan)
        .slideDown(timespan, cycleEffects);
    }
  });
</script>
...

```

代码清单10-15中的脚本使用标准jQuery链式调用方式把施加于h1元素的一系列特效串联起来。最后一个特效使用了cycleEffects回调函数，这个回调函数会让动画不断重放。这确实是一种相当闹心

的做法。起初它有一点儿蛊惑性，渐渐地开始让你恼怒，最后头痛不已。不过这个例子确实创建了一个动画队列，达到了我的演示目的。

**注意** 使用回调函数也能达到同样的目标，不过不会创建动画队列。因为那样做只有等到第一个动画完成之后，第二个动画才会开始。然而，如果我们像上面这个例子那样使用链式调用，所有的方法都在被“求值”之后放入队列。链式队列的限制在于只能在当前选择结果上创建动画，而使用回调函数则可以使用毫不相干的元素（在其他位置）生成动画。

### 10.5.1 显示动画队列中的内容

我们可以使用queue方法来检视动画队列中的内容。由于队列中包含两种数据对象，这个方法不是特别有用。如果一个动画正在进行，那么它在队列中是字符串值inprogress。如果一个动画尚未开始，那么它是队列中一个即将被调用的函数（但不能通过这个函数知道哪个动画将被执行）。不管怎么说，检视队列的内容是了解队列的一个很好的起点。代码清单10-16演示了如何检视一个队列。

代码清单10-16 检视动画队列的内容

```
...
<script type="text/javascript">
  $(document).ready(function() {

    $("h1").css({"position": "fixed", "z-index": "1", "min-width": "0"});
    $("form").remove();
    $("<table border=1></table>")
      .appendTo("body").css({
        position: "fixed", "z-index": "2",
        "border-collapse": "collapse", top: 100
      });

    var timespan = "slow";

    cycleEffects();
    printQueue();

    function cycleEffects() {
      $("h1")
        .animate({left: "+=100"}, timespan)
        .animate({left: "-=100"}, timespan)
        .animate({height: 223,width: 700}, timespan)
        .animate({height: 30,width: 500}, timespan)
        .slideUp(timespan)
        .slideDown(timespan, cycleEffects);
    }

    function printQueue() {
      var q = $("h1").queue();
      var qtable = $("table");
```

```

qtable.html("<tr><th>Queue Length:</th><td>" + q.length + "</td></tr>");
for (var i = 0; i < q.length; i++) {
    var baseString = "<tr><th>" + i + ":</th><td>";
    if (q[i] == "inprogress") {
        $("table").append(baseString + "In Progress</td></tr>");
    } else {
        $("table").append(baseString + q[i] + "</td></tr>");
    }
}
setTimeout(printQueue, 500);
}
});
</script>
...

```

这个例子不需要form元素，因此我把它从DOM中删除，并代之以一个简单的表格。这个表格用来显示动画队列的内容。我还增加了一个重复执行的函数printQueue，它负责调用queue方法，在表格中显示当前动画的序号以及当前处理动画的一点详细信息。我已经说过，队列内容本身不是特别有用，我们只能对正在发生的事情有个大概的了解。图10-15尽可能详细地展示了jQuery是如何处理动画队列的。

Queue Length:	Queue Length:	Queue Length:	Queue Length:	Queue Length:	Queue Length:	Queue Length:	Queue Length:
0: In Progress	0: In Progress	0: In Progress	0: In Progress	0: In Progress	0: In Progress	0: In Progress	0: In Progress
1: function() { opacity: 0; }	1: function() { opacity: 0; }	1: function() { opacity: 0; }	1: function() { opacity: 0; }	1: function() { opacity: 0; }	1: function() { opacity: 0; }	1: function() { opacity: 0; }	1: function() { opacity: 0; }
2: function() { opacity: 1; }	2: function() { opacity: 1; }	2: function() { opacity: 1; }	2: function() { opacity: 1; }	2: function() { opacity: 1; }	2: function() { opacity: 1; }	2: function() { opacity: 1; }	2: function() { opacity: 1; }
3: function() { opacity: 0; }	3: function() { opacity: 0; }	3: function() { opacity: 0; }	3: function() { opacity: 0; }	3: function() { opacity: 0; }	3: function() { opacity: 0; }	3: function() { opacity: 0; }	3: function() { opacity: 0; }
4: function() { opacity: 1; }	4: function() { opacity: 1; }	4: function() { opacity: 1; }	4: function() { opacity: 1; }	4: function() { opacity: 1; }	4: function() { opacity: 1; }	4: function() { opacity: 1; }	4: function() { opacity: 1; }
5: function() { opacity: 0; }	5: function() { opacity: 0; }	5: function() { opacity: 0; }	5: function() { opacity: 0; }	5: function() { opacity: 0; }	5: function() { opacity: 0; }	5: function() { opacity: 0; }	5: function() { opacity: 0; }
6: function() { opacity: 1; }	6: function() { opacity: 1; }	6: function() { opacity: 1; }	6: function() { opacity: 1; }	6: function() { opacity: 1; }	6: function() { opacity: 1; }	6: function() { opacity: 1; }	6: function() { opacity: 1; }

图10-15 检视动画队列的内容

这个例子很难使用静态图片表达。我建议你在浏览器中载入这个示例页面，看看究竟会发生什么事情。当cycleEffects函数第一次执行时，动画队列中共有6个动画，其中第一个动画正在播放，另外5个都是doAnimation函数实例。一旦动画播放完成，jQuery就把它从队列中删除。在最后一个动画播放完成之后，cycleEffects函数会被再次调用，在队列中重新放入6个动画实例。

### 10.5.2 停止当前动画并清理动画队列

我们可以使用stop和finish方法中断正在播放的动画。其中stop方法支持两个可选的布尔型参数。



如果把第一个参数的值设置为true, 就会删除队列中的所有其他动画。如果把第二个参数设置为true, 那么正在播放的动画会瞬间抵达动画终点(各CSS属性的目标值)。

这两个参数的默认值都是false, 也就是说只有当前正在播放的动画被从队列中删除, 并且各个CSS属性的值会保持为动画被打断瞬间的值。如果你没有清理动画队列, jQuery就会跳到下一个动画, 并正常执行它。代码清单10-17演示了stop方法的使用。

代码清单10-17 使用stop方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("h1").css({"position": "fixed", "z-index": "1", "min-width": "0"});
        $("form").remove();

        $("<table border=1></table>")
            .appendTo("body").css({
                position: "fixed", "z-index": "2",
                "border-collapse": "collapse", top: 100
            });

        $("<button>Stop</button><button>Start</button>")
            .appendTo($("<div/>")).appendTo("body")
            .css({
                position: "fixed", "z-index": "2",
                "border-collapse": "collapse", top: 100, left: 200
            }).click(function (e) {
                $(this).text() == "Stop" ? $("h1").stop(true, true) : cycleEffects();
            });

        var timespan = "slow";

        cycleEffects();
        printQueue();

        function cycleEffects() {
            $("h1")
                .animate({left: "+=100"}, timespan)
                .animate({left: "-=100"}, timespan)
                .animate({height: 223, width: 700}, timespan)
                .animate({height: 30, width: 500}, timespan)
                .slideUp(timespan)
                .slideDown(timespan, cycleEffects);
        }

        function printQueue() {
            var q = $("h1").queue();
            var qtable = $("table");
            qtable.html("<tr><th>Queue Length:</th><td>" + q.length + "</td></tr>");

            for (var i = 0; i < q.length; i++) {
                var baseString = "<tr><th>" + i + ":</th><td>";
            }
        }
    });
</script>
```

```

        if (q[i] == "inprogress") {
            $("table").append(baseString + "In Progress</td></tr>");
        } else {
            $("table").append(baseString + q[i] + "</td></tr>");
        }
    }
    setTimeout(printQueue, 500);
}
});
</script>
...

```

---

**提示** 如果我们使用stop方法中断当前动画，当前动画的回调函数就不会执行。如果我们使用stop方法清理整个队列，那么每个动画的回调函数都不会执行。

---

为了演示stop方法，我在页面中添加了两个按钮。点击Stop按钮时，脚本调用stop方法并传入两个true参数。这会清理掉剩余的动画，并且快播当前动画到终点，因此动画元素的CSS属性会一下子变换为动画结束时的目标值。由于stop方法阻止了回调函数的执行，cycleEffects方法的调用循环被打断，因此动画会彻底终止。点击Start按钮时，脚本会调用cycleEffects方法，动画又一次开始。

---

**提示** 即使动画正在播放，点击Start按钮也不会把jQuery搞“糊涂”。jQuery只是把cycleEffects方法创建的动画放入动画队列。尽管回调函数的使用导致队列瞬间变长，然而从动画的角度来看，一切都在正常进行。

---

finish方法的功能类似stop(true, true)，不过二者处理CSS属性的方式有所不同。当我们调用stop(true, true)时，CSS属性会从当前动画一下子蹦到当前动画的终点。而当使用finish方法时，所有正在参与当前动画的CSS属性以及所有队列中等待执行的动画效果都会一步执行到终点。代码清单10-18演示了finish方法的使用。

代码清单10-18 使用finish方法

```

...
<script type="text/javascript">
    $(document).ready(function() {

        $("h1").css({ "position": "fixed", "z-index": "1", "min-width": "0" });
        $("form").remove();

        $("<table border=1></table>")
            .appendTo("body").css({
                position: "fixed", "z-index": "2",
                "border-collapse": "collapse", top: 100
            });

        var finishAnimations = false;

        $("<button>Stop</button><button>Start</button>")

```

```

.appendTo($("#div/").appendTo("body")
.css({
    position: "fixed", "z-index": "2",
    "border-collapse": "collapse", top: 100, left:200
})).click(function(e) {
    if ($(this).text() == "Stop" ) {
        finishAnimations = true;
        $("#h1").finish();
    } else {
        finishAnimations = false;
        cycleEffects();
    }
});

var timespan = "slow";

cycleEffects();
printQueue();

function cycleEffects() {
    $("#h1")
    .animate({left: "+=100"}, timespan)
    .animate({left: "-=100"}, timespan)
    .animate({height: 223,width: 700}, timespan)
    .animate({height: 30,width: 500}, timespan)
    .slideUp(timespan)
    .slideDown(timespan, function() {
        if (!finishAnimations) {
            cycleEffects();
        }
    });
}

function printQueue() {
    var q = $("#h1").queue();
    var qtable = $("#table");
    qtable.html("<tr><th>Queue Length:</th><td>" + q.length + "</td></tr>");

    for (var i = 0; i < q.length; i++) {
        var baseString = "<tr><th>" + i + ":</th><td>";
        if (q[i] == "inprogress") {
            $("#table").append(baseString + "In Progress</td></tr>");
        } else {
            $("#table").append(baseString + q[i] + "</td></tr>");
        }
    }
    setTimeout(printQueue, 500);
}

});
</script>
...

```

使用finish方法时，要小心使用类似代码清单10-18中的循环特效。为了得到所有参与动画CSS属性的终点值，finish方法需要执行所有的特效，这意味着它将没有任何延时地执行所有的回调函数。

在代码清单10-18中，`cycleEffects`函数内定义的最后一个特效启动了下一个特效循环：

```
...
.slideDown(timespan, cycleEffects);
...
```

`finish`方法执行时，并不能阻止新动画添加到队列，因此它无法跟踪动画队列的状态。这就是说，`finish`方法会调用`cycleEffects`函数，这个函数会把动画添加到队列，紧接着`finish`方法又去执行队列里的这些动画，而动画的执行又触发了添加动画到队列的回调函数……如此这般，JavaScript的函数调用栈瞬间就会耗光。

为避免发生这种问题，我增加了一个变量`finishAnimations`，点击按钮时，设置该变量到适当的值。如下面的代码所示，当我需要添加动画到队列时，先检查`finishAnimations`变量的值：

```
...
.slideDown(timespan, function () {
    if (!finishAnimations) {
        cycleEffects();
    }
});
...
```

### 10.5.3 在动画队列中插入延时

我们可以使用`delay`方法在两个动画之间引入一段延时。`delay`方法的参数是一个以毫秒（ms）为单位的数据，表示应该延迟的时间。代码清单10-19演示了如何使用`delay`方法在动画队列中引入1秒钟的延时。

代码清单10-19 使用`delay`方法

```
...
function cycleEffects() {
    $('h1')
        .animate({left: "+=100"}, timespan)
        .animate({left: "-=100"}, timespan)
        .delay(1000)
        .animate({height: 223,width: 700}, timespan)
        .animate({height: 30,width: 500}, timespan)
        .delay(1000)
        .slideUp(timespan)
        .slideDown(timespan, function () {
            if (!finishAnimations) {
                cycleEffects();
            }
        });
}
...
```

### 10.5.4 在动画队列中插入自定义函数

我们也可以使用`queue`方法把自己编写的函数添加到动画队列中。jQuery会把你的函数当成标准的

动画方法来执行。我们可以利用这一特性启动别的动画，或者根据一个外部变量的值优雅地退出动画链，甚至做任何想做的其它事情。具体示例见代码清单10-20。

代码清单10-20 在动画队列中插入自定义函数

```
...
function cycleEffects() {
    $("h1")
        .animate({left: "+=100"}, timespan)
        .animate({left: "-=100"}, timespan)
        .queue(function() {
            $("body").fadeTo(timespan, 0).fadeTo(timespan, 1);
            $(this).dequeue();
        })
        .delay(1000)
        .animate({height: 223,width: 700}, timespan)
        .animate({height: 30,width: 500}, timespan)
        .delay(1000)
        .slideUp(timespan)
        .slideDown(timespan, function () {
            if (!finishAnimations) {
                cycleEffects();
            }
        });
});
...

```

在自定义函数内，this变量被设置成调用queue方法的jQuery对象。这非常有用，因为在某些时候，我们需要在自定义函数内调用dequeue方法以便启动下一个动画或者函数。在这个例子中，我使用queue方法添加了一个函数，这个函数会把body元素淡出至完全透明，然后再淡入至完全不透明。

**提示** 在自定义函数中针对body元素生成的动画，也会被添加到动画队列中。每个元素都有着自己的动画队列，因此我们可以独立地管理这些队列。如果希望使用同一些元素（我们正在管理的队列）的多个CSS属性生成动画，那么就直接调用animate方法，否则动画就会被顺序添加到动画队列中。

另外，自定义函数可接受一个唯一的参数，即队列中的下一个函数。在这种情况下，我们必须像代码清单10-21中那样直接在函数中调用这个参数函数。

代码清单10-21 利用参数传入自定义函数

```
...
function cycleEffects(nextFunction) {
    $("h1")
        .animate({left: "+=100"}, timespan)
        .animate({left: "-=100"}, timespan)
        .queue(function() {
            $("body").fadeTo(timespan, 0).fadeTo(timespan, 1);
            nextFunction();
        });
};
...

```

```

    })
    .delay(1000)
    .animate({height: 223,width: 700}, timespan)
    .animate({height: 30,width: 500}, timespan)
    .delay(1000)
    .slideUp(timespan)
    .slideDown(timespan, function () {
        if (!finishAnimations) {
            cycleEffects();
        }
    });
}
...

```

---

**提示** 如果你忘记调用下一个函数，或者`dequeue`方法，这个动画队列就会停止运行。

---

## 10.6 启用或者禁用动画特效

如代码清单10-22所示，我们也可以把`$.fx.off`属性设置为`true`，从而禁用动画特效。

**代码清单10-22 禁用动画特效**

```

...
<script type="text/javascript">
    $(document).ready(function() {

        $.fx.off = true;

        $("hi").css({"position": "fixed", "z-index": "1", "min-width": "0" });
        $("form").remove();

        // .....为了简便起见，省略下面的语句.....

    });
</script>
...

```

如果禁用了动画特效，调用特效方法将会导致元素瞬间变换为参与动画的各CSS属性的目标值。持续时间参数将被忽略，因而也就没有了任何中间动画。在禁用动画的状态下，递归调用特效方法将很快耗光JavaScript的调用栈。为避免这个问题，请使用本章前面讲过的`setTimeout`方法代替递归调用。

## 10.7 小结

本章重点讲解了jQuery提供的各种特效功能。jQuery内建的特效方法主要是以各种各样的方法让元素可见或者消失。不过，我们可以使用任意的数值CSS属性生成更个性的动画。另外，我们还深入了解了动画队列，以及如何控制施加在目标元素上的动画队列。在下一章中，我重构了示例页面，并展示如何把本章及前面几章中讲到的jQuery基础功能集成到一起。

本书前面各章相对独立地讲解了每一个功能，包括事件处理、DOM操控等。只有学会综合使用这些功能，才能发挥出jQuery真正的威力和灵活性。本章将通过重构花店页面演示这些功能的综合应用。

本章所有的变更都发生于script元素内，没有修改示例文档的HTML。绝大部分jQuery特性都有多种方法实现同一目的。我在本章采用的解决办法反映出了jQuery中我最喜欢的功能以及DOM操控方式。你的解决方案完全可以与我的不同，采用更喜欢的方法组合。这完全不是问题，条条大路通罗马，使用jQuery没有成规。

## 11.1 回顾示例文档

本书从一个非常简单的花店示例文档开始讲解。在接下来的章节中，我们会使用jQuery在这个页面上选择元素、检索重排DOM、侦听事件，并对一些元素施加特效。在我们重构这个页面之前，先回顾一下最初的页面。代码清单11-1展示了这个简单页面的代码。

代码清单11-1 简单的示例文档

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script type="text/javascript">
    $(document).ready(function() {
      // jQuery语句
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
        <div id="row1" class="drow">
          <div class="dcell">
            <label for="aster">Aster:</label>
            <input name="aster" value="0" required />
          </div>
```

```

        <div class="dcell">
            <label for="daffodil">Daffodil:</label>
            <input name="daffodil" value="0" required />
        </div>
        <div class="dcell">
            <label for="rose">Rose:</label>
            <input name="rose" value="0" required />
        </div>
    </div>
    <div id="row2" class="drow">
        <div class="dcell">
            <label for="peony">Peony:</label>
            <input name="peony" value="0" required />
        </div>
        <div class="dcell">
            <label for="primula">Primula:</label>
            <input name="primula" value="0" required />
        </div>
        <div class="dcell">
            <label for="snowdrop">Snowdrop:</label>
            <input name="snowdrop" value="0" required />
        </div>
    </div>
</div>
</div>
<div id="buttonDiv"><button type="submit">Place Order</button></div>
</form>
</body>
</html>

```

代码中的script元素做了加粗处理，因为那是我们在本书中挥洒汗水的地方。我在script标签里放上了无处不在的jQuery ready事件处理函数，不过也就这些，除此之外再无其他JavaScript语句。图11-1是这个未经修饰的“素面朝天”的页面在浏览器中的实际呈现效果。

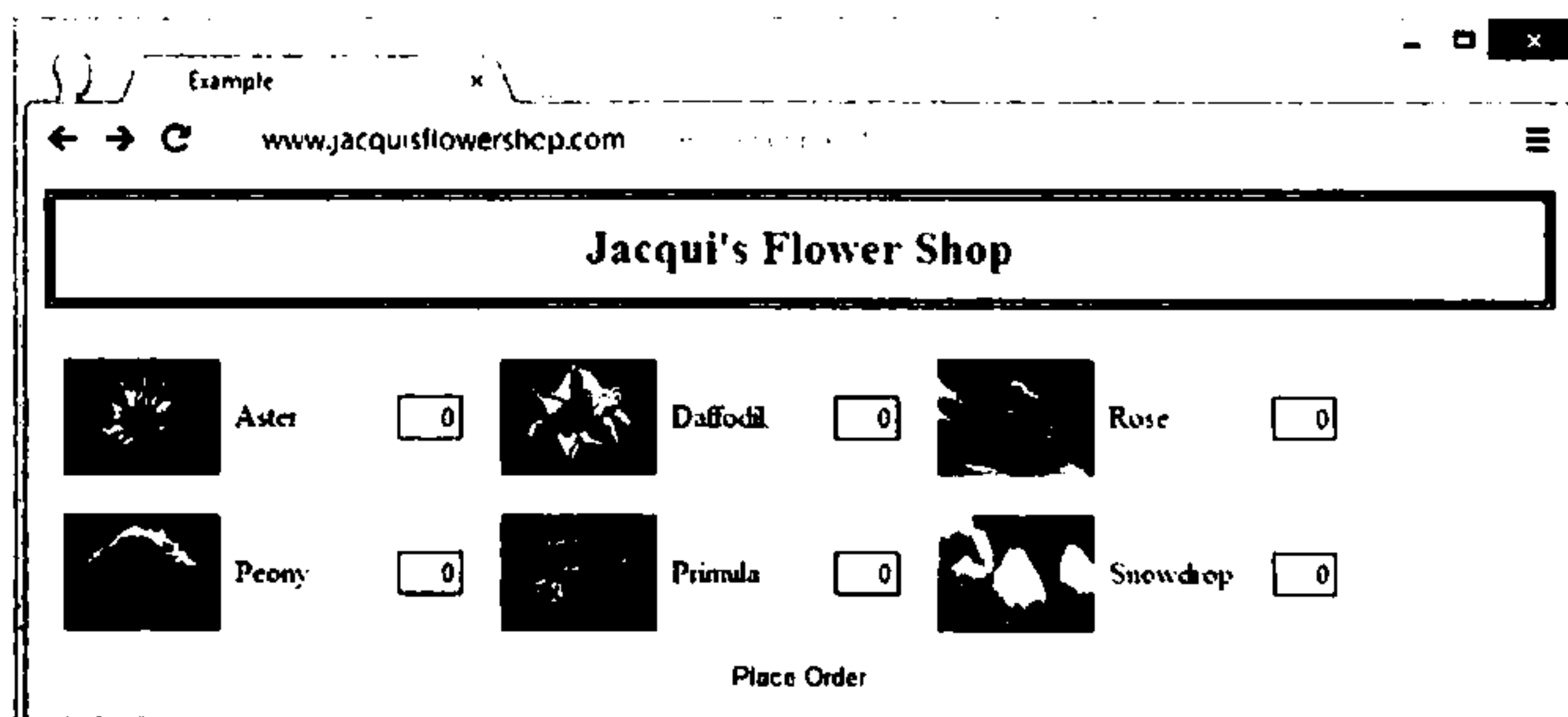


图11-1 简单的示例页面



## 11.2 添加更多的花卉产品

首先我要在花店中添加更多的花卉。我希望在这个修改中演示如何在循环中创建一些元素。代码清单11-2展示了变更之后的script元素。

代码清单11-2 往页面中添加更多的花卉产品

```
...
<script type="text/javascript">
  $(document).ready(function() {
    var fNames = ["Carnation", "Lily", "Orchid"];
    var fRow = $("<div id=row3 class=drow/>").appendTo('div.dtable');
    var fTemplate = $("<div class=dcell><img/><label/><input/></div>");
    for (var i = 0; i < fNames.length; i++) {
      fTemplate.clone().appendTo(fRow).children()
        .filter("img").attr("src", fNames[i] + ".png").end()
        .filter("label").attr("for", fNames[i]).text(fNames[i]).end()
        .filter("input").attr({name: fNames[i], value: 0, required: "required"})
    }
  });
</script>
...
```

我定义了3种新花卉(Carnation、Lily和Orchid,即康乃馨、百合和兰花),创建了一个新的div.drow元素并把它追加到在CSS表格布局模型中扮演表格角色的div.dtable中:

```
...
var fNames = ["Carnation", "Lily", "Orchid"];
var fRow = $("<div id=row3 class=drow/>").appendTo('div.dtable');
...
```

接下来定义“骨架”,也就是产品元素的结构。这个“骨架”不包含任何具体产品的具体属性:

```
...
var fTemplate = $("<div class=dcell><img/><label/><input/></div>");
...
```

我把这个“骨架”元素当成一个简单的模板。为计划添加的每一种花卉克隆一份“骨架”并使用花卉的名字数据设置具体的属性和值:

```
...
for (var i = 0; i < fNames.length; i++) {
  fTemplate.clone().appendTo(fRow).children()
    .filter("img").attr("src", fNames[i] + ".png").end()
    .filter("label").attr("for", fNames[i]).text(fNames[i]).end()
    .filter("input").attr({name: fNames[i], value: 0, required: "required"})
}
...
```

我使用filter方法和end方法来过滤及恢复结果集,使用attr方法设置元素属性。最后,每一种花都得到了一套完整的元素,并被插入到了相应的行一级div元素中,紧接着行一级元素又被插入到表格一级的元素。具体的效果见图11-2。

这个例子证实了jQuery一个很棒的特性,那就是可以选择并处理那些尚未添加到页面的元素。当

我克隆模板元素时，模板元素还不是页面的一部分，但仍然可以使用children方法和filter方法过滤选择结果。

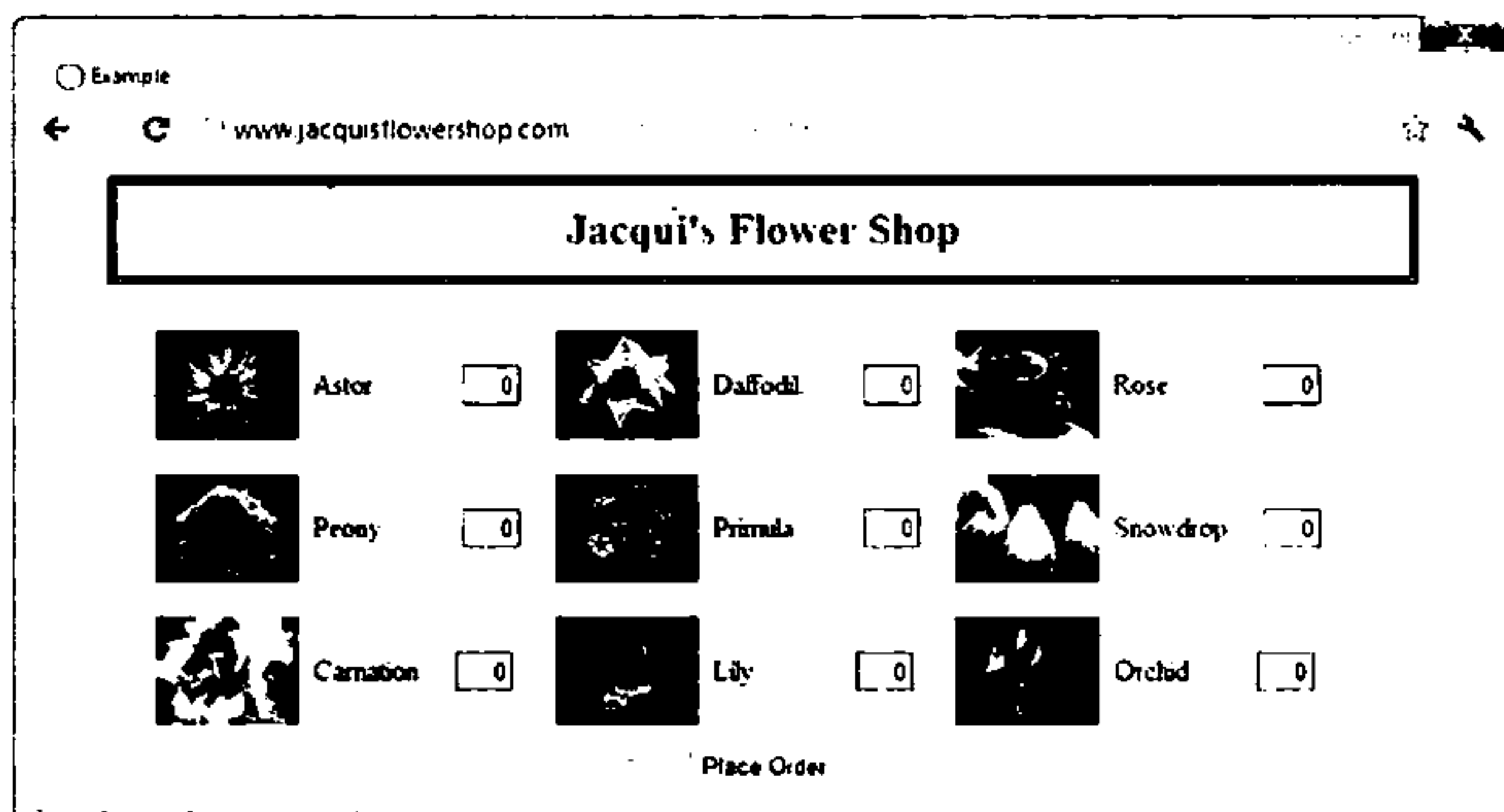


图11-2 在页面中添加新的花卉

### 11.3 添加翻页按钮

接下来添加一个简单的翻页按钮，让用户能够分页浏览花卉产品。首先，我们需要两个按钮，一个用于翻到上一页（向左翻），一个用于翻到下一页（向右翻）。代码清单11-3演示了如何把这两个按钮添加到页面。

代码清单11-3 添加翻页按钮

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var fNames = ["Carnation", "Lily", "Orchid"];
        var fRow = $("<div id=row3 class=drow/>").appendTo("div.dtable");
        var fTemplate = $("<div class=dcell><img/><label/><input/></div>");
        for (var i = 0; i < fNames.length; i++) {
            fTemplate.clone().appendTo(fRow).children()
                .filter("img").attr("src", fNames[i] + ".png").end()
                .filter("label").attr("for", fNames[i]).text(fNames[i]).end()
                .filter("input").attr({name: fNames[i], value: 0, required: "required"});
        }

        $("<a id=left></a><a id=right></a>").prependTo("form")
            .css({
                "background-image": "url(leftarrows.png)",
                "float": "left",
                "margin-top": "15px",
                display: "block", width: 50, height: 50
            }).click(handleArrowPress).hover(handleArrowMouse);
    });
</script>
```

```

$('#right').css("background-image", "url(rightarrows.png)").appendTo('form');

$("#oblock").css({float: "left", display: "inline", border: "thin black solid"});
$("#form").css({"margin-left": "auto", "margin-right": "auto", width: 885});

function handleArrowMouse(e) {
}

function handleArrowPress(e) {
}

});
</script>
...

```

首先，我定义了两个a元素，把它们作为第一、第二子元素插入到form元素中，然后使用css方法为它们设置了一系列样式属性。

```

...
$("<a id=left></a><a id=right></a>").prependTo("form")
.css({
  "background-image": "url(leftarrows.png)",
  "float": "left",
  "margin-top": "15px",
  display: "block", width: 50, height: 50
}).click(handleArrowPress).hover(handleArrowMouse)
...

```

background-image属性相当关键，图11-3展示的就是我们在这里使用的背景图片。



图11-3 leftarrows.png图片

这张图中有3个不同的箭头，每个箭头都是50像素宽。通过把a元素的宽度和高度都设置为50像素，我们就能确保任何时候都仅显示其中的一个箭头。我使用click和hover方法来定义click、mouseenter及mouseleave事件的处理函数。

```

...
$("<a id=left></a><a id=right></a>").prependTo("form")
.css({
  "background-image": "url(leftarrows.png)",
  "float": "left",
  "margin-top": "15px",
  display: "block", width: 50, height: 50
}).click(handleArrowPress).hover(handleArrowMouse);
...

```

handleArrowPress和handleArrowMouse函数现在还是空的，但我们马上就会把它们补全。现在有两个箭头元素在表单中紧挨着，而且都是左箭头。由于这两个箭头大部分属性相同，我才把它们一起创建并一起配置。不过，现在到了把它们分开的时候了，请看下面的代码：

```
...
$("#right").css("background-image", "url(rightarrows.png)").appendTo("form");
...
```

我使用appendTo方法把右箭头元素挪到form元素的末尾，并使用css方法把背景图片修改为rightarrows.png（参见图11-4）。

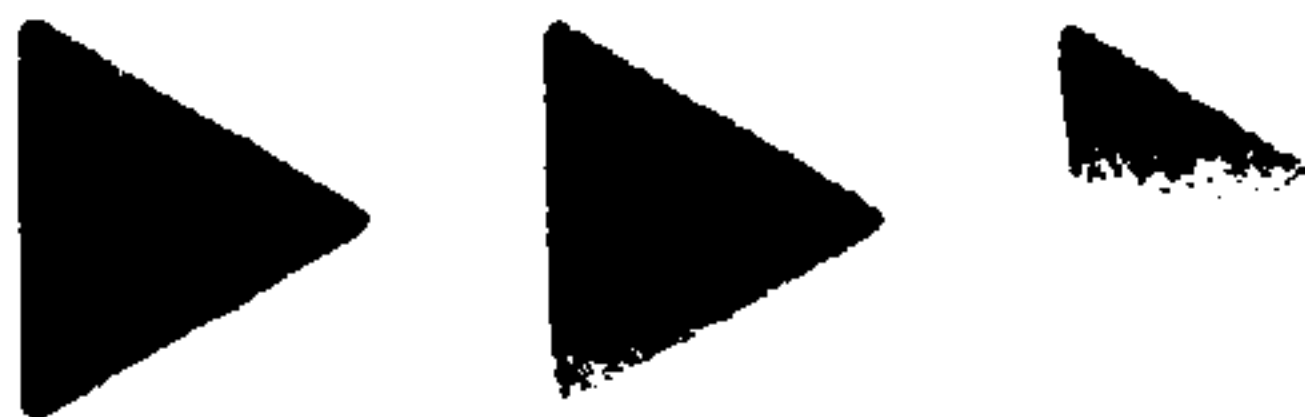


图11-4 rightarrows.png图片

这里使用了组合图片，这是一项很常用的技术。利用这项技术，我们只需要一次而不是3次请求，就能得到3张密切相关的图片。这能够有效地降低服务器的请求数，避免服务器过载。接下来，当我们补全handleArrowMouse函数时，你就会看到我将如何使用这样的图片。页面现在的样子见图11-5。

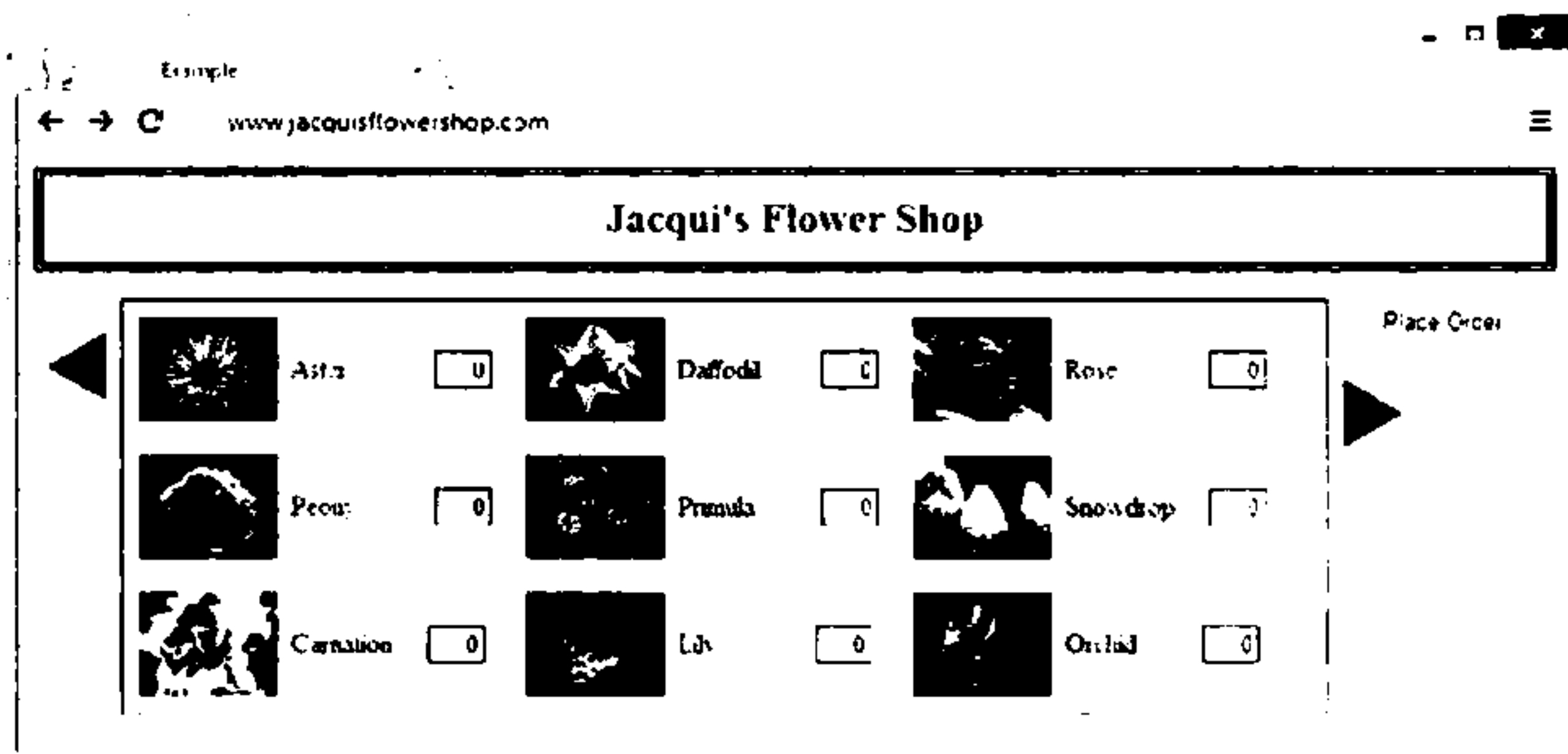


图11-5 半成品的示例文档

## 11.4 处理提交按钮

如图11-5所示，现在的页面还只是一个半成品。我们看到了新功能，但还欠缺对部分已有元素的适当处理，其中最明显的就是用来提交表单的Place Order按钮。代码清单11-4展示了处理这一按钮的新增脚本（还增加了一个新功能）。

### 代码清单11-4 处理提交按钮

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var fNames = ["Carnation", "Lily", "Orchid"];
        var fRow = $("<div id=row3 class=drow/>").appendTo("div.dtable");
```

```

var fTemplate = $("<div class=dcell><img/><label/><input/></div>");
for (var i = 0; i < fNames.length; i++) {
    fTemplate.clone().appendTo(fRow).children()
        .filter("img").attr("src", fNames[i] + ".png").end()
        .filter("label").attr("for", fNames[i]).text(fNames[i]).end()
        .filter("input").attr({name: fNames[i], value: 0, required: "required"});
}

$("<a id=left></a><a id=right></a>").prependTo("form")
    .css({
        "background-image": "url(leftarrows.png)",
        "float": "left",
        "margin-top": "15px",
        display: "block", width: 50, height: 50
    }).click(handleArrowPress).hover(handleArrowMouse)

$("#right").css("background-image", "url(rightarrows.png)").appendTo("form");

$("h1").css({"min-width": "0", width: "95%",});
$("#row2, #row3").hide();
$("#oblock").css({float: "left", display: "inline", border: "thin black solid"});
$("form").css({"margin-left": "auto", "margin-right": "auto", width: 885});

var total = $("#buttonDiv")
    .prepend("<div>Total Items: <span id=total>0</span></div>")
    .css({clear: "both", padding: "5px"});
$("<div id=bbox />").appendTo("body").append(total).css("clear: left");

function handleArrowMouse(e) {
}

function handleArrowPress(e) {
}

});
</script>
...

```

为了协调翻页按钮对页面布局的影响，我把包含着按钮元素的div元素（即#buttonDiv）挪到了一个新div元素里面，并把这个新div元素追加到body元素内。这样就把按钮的位置挪到了页面的底部。我还增加了一个div元素和一个span元素，用于显示用户选中产品的总数。

```

...
var total = $("#buttonDiv")
    .prepend("<div>Total Items: <span id=total>0</span></div>")
    .css({clear: "both", padding: "5px"});
$("<div id=bbox />").appendTo("body").append(total).css("clear: left");
...

```

为了实现翻页功能，我还让#row2和#row3两行默认隐藏起来，以便用户点击翻页按钮时再显示它们。

```

...
$("#row2, #row3").hide();
...

```

我还调整了h1元素的样式以适合修正之后的页面布局：

```
...
$("h1").css({"min-width": "0", width: "95%",});
...
```

经过这番修正，页面的效果见图11-6。

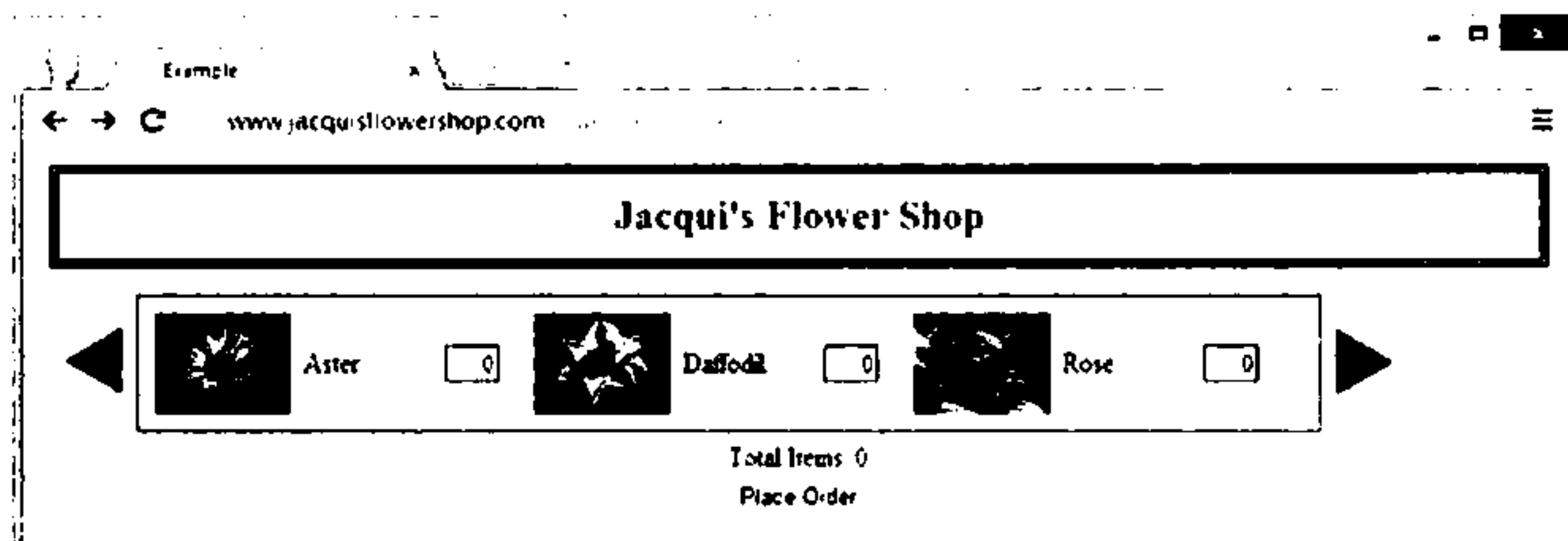


图11-6 处理提交按钮并整理样式

## 11.5 实现翻页事件处理函数

接下来实现翻页按钮的事件处理函数。首先处理mouseenter和mouseleave事件，这两个事件由handleArrowMouse函数处理。代码清单11-5展示了这个函数的具体实现。

代码清单11-5 处理箭头按钮鼠标事件

```
...
function handleArrowMouse(e) {
    var propValue = e.type == "mouseenter" ? "-50px 0px" : "0px 0px";
    $(this).css("background-position", propValue);
}
...
```

处理组合图片的秘密在于使用background-position属性移动图片位置，只让需要显示的部分可见。虽然箭头图片中一共有3张图片，但是我只会使用其中的两个。一般情况下显示颜色最深（最黑）的箭头，当鼠标悬停在箭头上时显示颜色较浅的箭头。剩下的那个箭头可以表示正在被点击的按钮或者禁用的按钮。不过，我没打算搞那么复杂，图11-7展示了箭头图片的两种状态。

handleArrowPress函数负责生成翻页效果，允许用户翻页查看花卉产品。代码清单11-6展示了这个函数的具体实现。

代码清单11-6 实现handleArrowPress函数

```
...
function handleArrowPress(e) {
    var elemSequence = ["row1", "row2", "row3"];
    var visibleRow = $("div.drow:visible");
    var visibleRowIndex = jQuery.inArray(visibleRow.attr("id"), elemSequence);

    var targetRowIndex;
```

```

if (e.target.id == "left") {
    targetRowIndex = visibleRowIndex - 1;
    if (targetRowIndex < 0) {
        targetRowIndex = elemSequence.length - 1;
    };
} else {
    targetRowIndex = (visibleRowIndex + 1) % elemSequence.length;
}

visibleRow.fadeOut("fast", function() {
    $("#"+elemSequence[targetRowIndex]).fadeIn("fast"));
}
...

```

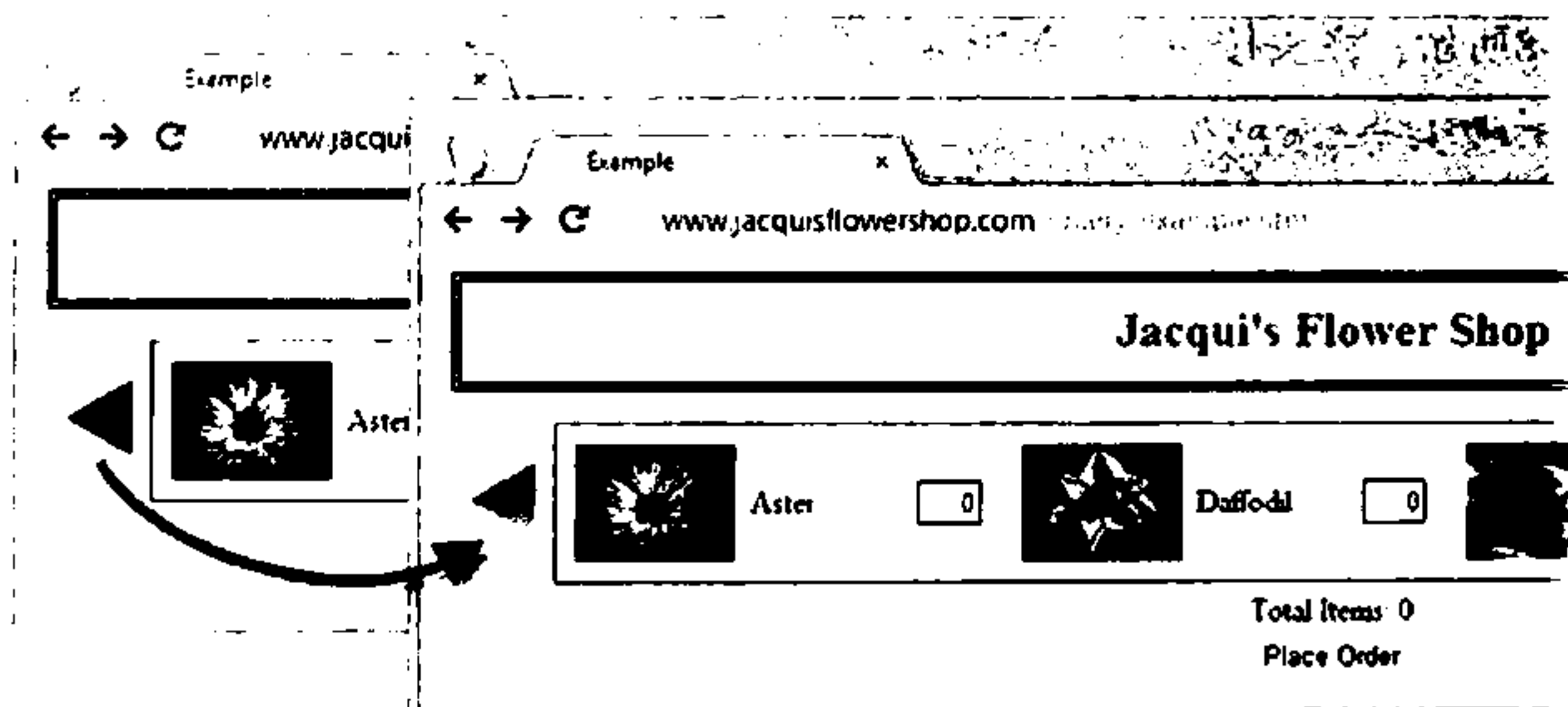


图11-7 箭头按钮的两种状态

函数的前3行代码用来准备翻页所需的基础数据:

```

...
var elemSequence = ["row1", "row2", "row3"];
var visibleRow = $("#div.drow:visible");
var visibleRowIndex = jQuery.inArray(visibleRow.attr("id"), elemSequence);
...

```

第一条语句定义了一个由行级元素的id属性组成的id数组。第二条语句使用jQuery得到当前可见的行。接下来,第三条语句用来得到可见行在id数组中的索引。在这里我使用了jQuery工具方法inArray(参见第34章)。这样我们就知道哪一行现在可见,以及这一行在id数组中的位置。接下来,我们解决下一页应该显示哪一行的问题:

```

...
var targetRowIndex;
if (e.target.id == "left") {
    targetRowIndex = visibleRowIndex - 1;
    if (targetRowIndex < 0) {
        targetRowIndex = elemSequence.length - 1;
    };
} else {
    targetRowIndex = (visibleRowIndex + 1) % elemSequence.length;
}
...

```

几乎所有编程语言都支持使用取余运算得出下一个要显示的行。然而，JavaScript语言的取余运算有一个bug，即它不支持负数。因此，当用户点击向左的箭头时，我需要手工检查数组的边界；而在用户点击向右的箭头时，使用%运算符得出要显示的行。在确认当前显示的元素以及下一页要显示的元素之后，我使用了jQuery动画特效展示从一页到另一页的过程：

```
...
visibleRow.fadeOut("fast", function() {
    $('#'+ elemSequence[targetRowIndex]).fadeIn("fast"));
...

```

由于fadeOut和fadeIn方法配合CSS表格样式布局甚佳，因此我使用了这两个方法。第一个特效使用了回调函数，由回调函数实施第二个特效。两个特效都使用了fast参数。我并未改变页面的静态布局，然而如图11-8所示，箭头按钮却能够引导用户从一行花卉到另一行。

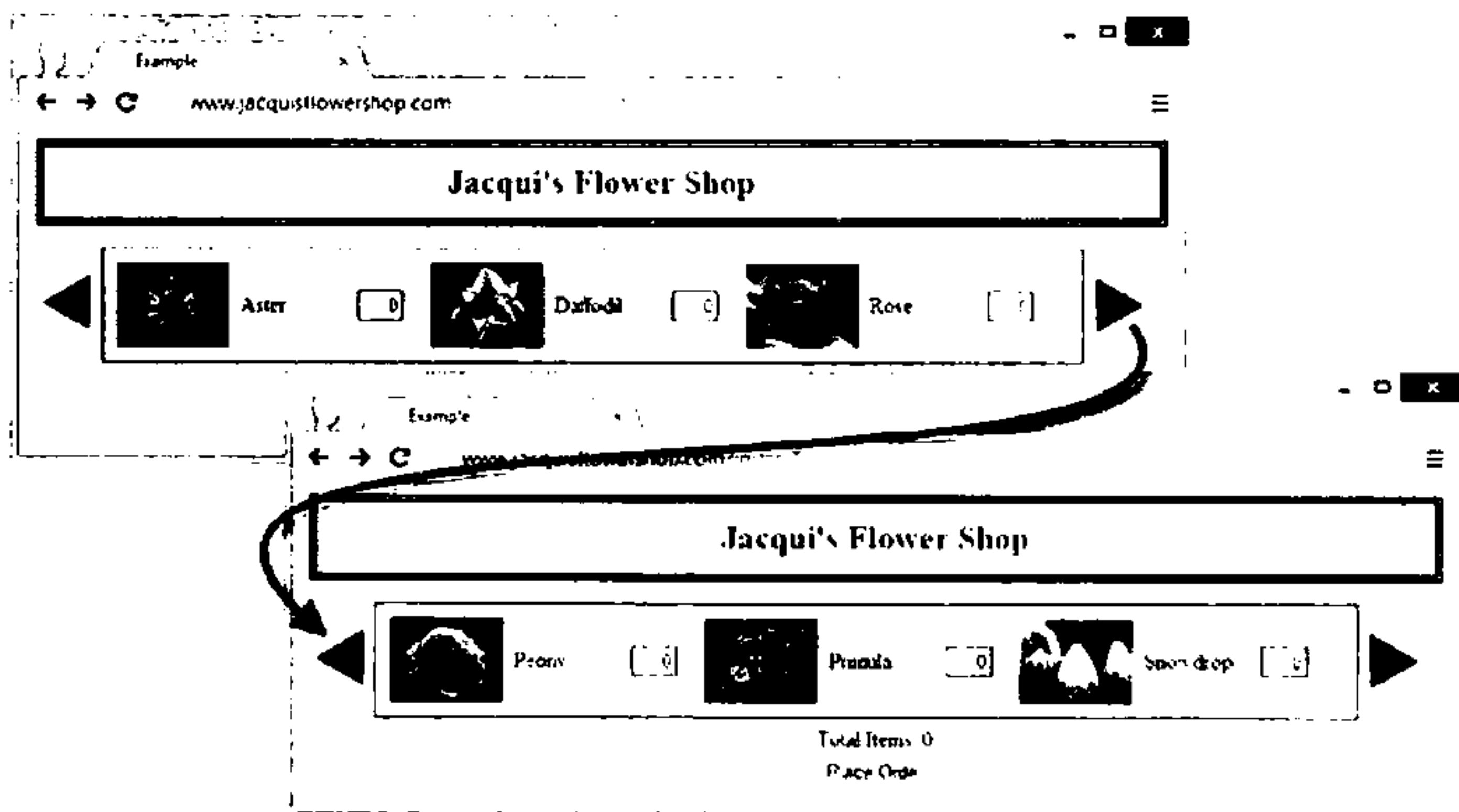


图11-8 为花卉产品提供翻页功能

## 11.6 计算订购产品总数

最后一项修改是实现显示用户订购产品总数的功能，把订购总数显示在花卉下方的输入框里。代码清单11-7展示了脚本中相关的修改。

代码清单11-7 实现显示订购总数的功能

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var fNamees = ["Carnation", "Lily", "Orchid"];
        var fRow = $("<div id=row3 class=drow/>").appendTo("div.dtable");
        var fTemplate = $("<div class=dcell><img/><label/><input/></div>");
        for (var i = 0; i < fNamees.length; i++) {

```



```

    fTemplate.clone().appendTo(fRow).children()
        .filter("img").attr("src", fNames[i] + ".png").end()
        .filter("label").attr("for", fNames[i]).text(fNames[i]).end()
        .filter("input").attr({name: fNames[i], value: 0, required: "required"});
}

$("<a id=left></a><a id=right></a>").prependTo("form")
    .css({
        "background-image": "url(leftarrows.png)",
        "float": "left",
        "margin-top": "15px",
        display: "block", width: 50, height: 50
    }).click(handleArrowPress).hover(handleArrowMouse);

$("#right").css("background-image", "url(rightarrows.png)").appendTo("form");

$("h1").css({min-width: "0", width: "95%",});
$("#row2, #row3").hide();
$("#oblock").css({float: "left", display: "inline", border: "thin black solid"});
$("form").css({margin-left: "auto", "margin-right": "auto", width: 885});

var total = $("#buttonDiv")
    .prepend("<div>Total Items: <span id=total>0</span></div>")
    .css({clear: "both", padding: "5px"});
$("<div id=bbox />").appendTo("body").append(total).css("clear: left");

$("input").change(function(e) {
    var total = 0;
    $("input").each(function(index, elem) {
        total += Number($(elem).val());
    });
    $("#total").text(total);
});

function handleArrowMouse(e) {
    var propValue = e.type == "mouseenter" ? "-50px 0px" : "0px 0px";
    $(this).css("background-position", propValue);
}

function handleArrowPress(e) {
    var elemSequence = ["row1", "row2", "row3"];

    var visibleRow = $("div.drow:visible");
    var visibleRowIndex = jQuery.inArray(visibleRow.attr("id"), elemSequence);

    var targetRowIndex;

    if (e.target.id == "left") {
        targetRowIndex = visibleRowIndex - 1;
        if (targetRowIndex < 0) {targetRowIndex = elemSequence.length - 1;}
    } else {
        targetRowIndex = (visibleRowIndex + 1) % elemSequence.length;
    }
}

```

```

        visibleRow.fadeOut("fast", function() {
            $("#"+elemSequence[targetRowIndex]).fadeIn("fast");
        });
    }

    });
</script>
...

```

在这一部分，我选中页面中所有的input元素，注册了change事件处理函数。它负责得到每个input元素的值并求和，然后把总数放到之前添加的span元素中。最终效果见图11-9。

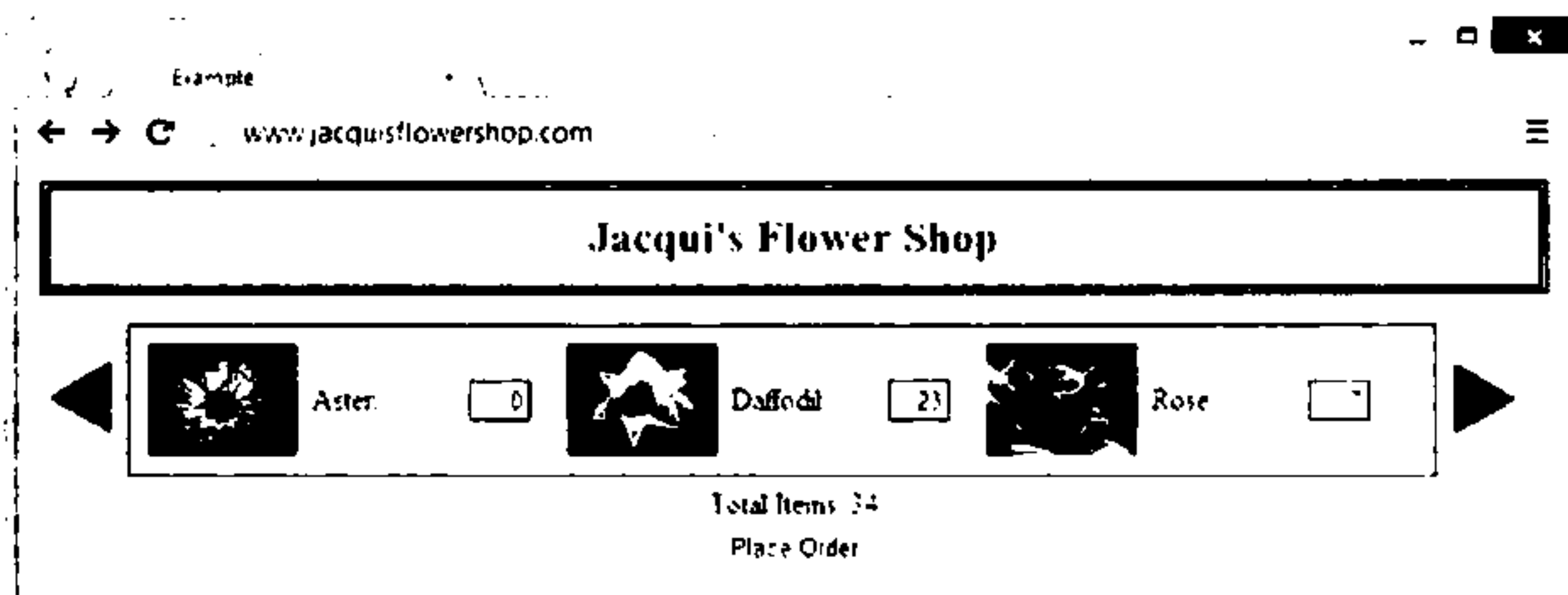


图11-9 显示产品订购总数

这里显示的总数是所有input元素的总和，而不是当前可见的花卉订购数之和。这是一个小问题，很容易解决。

## 11.7 禁用 JavaScript

我们全面改进了示例文档，只不过所有的改变都是使用jQuery完成的。这意味着我们已经高效地实现了两个层面的文档，一层用于支持JavaScript的浏览器，另一层用于不支持JavaScript的浏览器。图11-10演示了禁用JavaScript之后示例页面在浏览器中的样子。

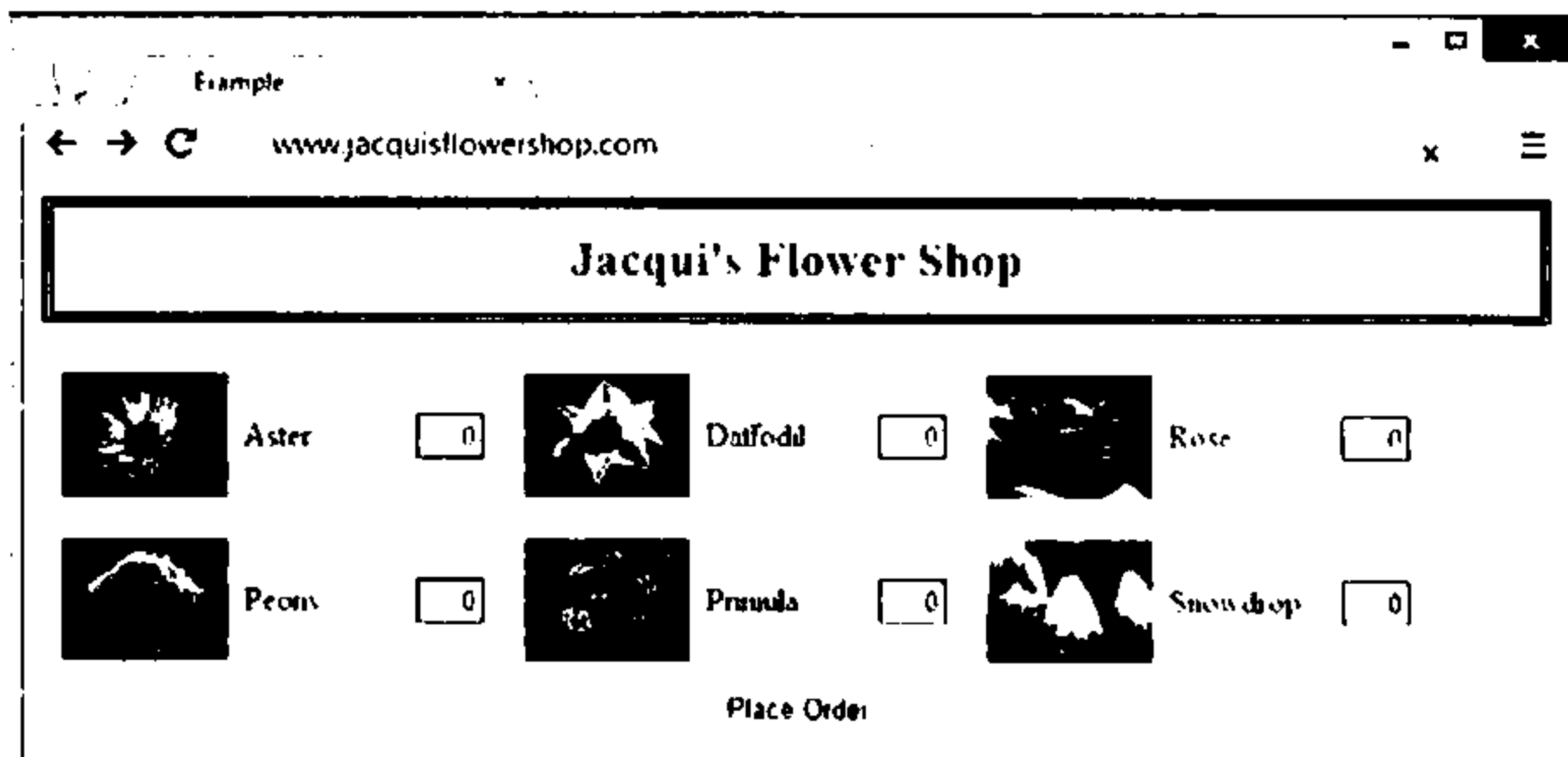


图11-10 禁用JavaScript之后页面的显示效果

我们又回到了起点。多一点计划和远见，我们就能为不支持JavaScript的客户端提供一些功能，让它们仍然能够与我们的页面或者应用程序交互。由于很多集中管理IT的大公司网管出于安全方面的考虑会禁用JavaScript，这通常都是一个好主意。好了，多少有这么点意思。当我在这种公司工作了很多年之后，开始相信这些“警察”们并没有真正阻止同事们使用JavaScript，这些人只是为同事们去寻找漏洞和散播解决办法创造了机会。

## 11.8 小结

本章演示了如何综合利用前面各章的技术重构示例页面。我使用程序添加了新的内容，添加了简单的产品分页功能，还添加了用来显示选中产品总数的元素。在这个过程中，我还微调了DOM和样式以适应这些改变。总地来说，我所做的这一切都不会影响那些不支持JavaScript的浏览器，这些浏览器仍会得到一个可用的页面。

在本书第三部分，我将继续完善这个例子，引入更多的jQuery特性并实现更多的新功能。大多数情况下，我都是在最初的示例页面上使用这些新特性，以便依次讲解这些新功能。不过到第16章时，为了引入更多的新特性，我会再次重构这个页面。

在本书上一版中，我介绍了jQuery模板插件。这个插件的历史相当传奇。当年Microsoft公司和jQuery团队共同宣布：由Microsoft开发的三款插件被jQuery接受为“官方”插件。以前从来没有一款插件享此殊荣，然而好景不长。其后不久，jQuery团队宣布这些插件已经过时，摘掉了这些插件头上的“官方”帽子，并且制订了替代这些插件功能的开发计划。新开发的（用来替代这些插件的）组件则并入jQuery UI（本书第四部分将详细介绍jQuery UI）。

不久之前，jQuery团队发布了新的jQuery官方模板引擎，也就是jsViews，其实直到今天，它依然没有彻底完成。目前有一个beta版本可用，但它还很粗糙，API也不太稳定。我对jsViews谈不上喜欢，如果你有兴趣，可以访问<https://github.com/BorisMoore/jsviews>获取最新的beta版本。

与此同时，上一版中使用的已经过时的jQuery模板插件尚未成熟。不久前，在我的个人项目里，我不再使用jQuery模板插件，转而采用新的模板库Handlebars（官方站点：<http://handlebarsjs.com>）。它与jQuery未做任何整合，事实上它根本不是一个jQuery插件，不过，我们只需添加很少的代码，就能做到让模板支持jQuery语法。下面将告诉你应该如何实现。

和前面提到的jQuery模板插件一样，Handlebars支持“胡子”（mustache）模板，即在HTML中夹杂各种模板指令。我会解释这些指令，也许你会奇怪为什么使用“胡子”这个术语，这是因为所有的模板指令都包括在一对大括号（{和}）当中，而大括号看起来有点像胡子的两边。表12-1列出了本章概要。

表12-1 本章概要

问 题	解决方法	代码清单
如何使用模板生成HTML	安装Handlebars库，创作一个jQuery插件，调用template方法	1~6
如何把模板生成的HTML赋予不同的父元素	把源数据分成多份，调用template方法两次，或者使用slice过滤器和end方法分开生成的HTML	7~10
如果定义了data属性，且data属性不为null，如何改变模板的输出	使用内建的模板助手#if或#unless	11、12
如何在模板中枚举数组数据或者对象属性	使用模板助手#each	13、14
如何在模板中引用data对象的其他部分	使用模板助手#with或者使用../路径	15~17
如何自定义模板助手	使用Handlebars.registerHelper方法登记助手方法的名字，助手方法应返回渲染之后的模板内容	18~22
如何在助手函数中使用可选参数	使用options.hash属性	23、24
如何在一个自定义模板助手块中定义专有属性	使用options.data属性	25、26

## 12.1 模板解决什么问题

模板解决了一个非常实际的问题：利用模板我们能够以JavaScript对象为数据源，按照设计好的逻辑生成HTML。这个问题不使用模板也可以解决，在第11章的例子中，需要生成一些元素来表示新增花卉，我们已经解决过这类问题。代码清单12-1列出了从那一章中摘抄的有关语句。

代码清单12-1 用脚本生成HTML

```
...
<script type="text/javascript">
  $(document).ready(function() {
    var fNames = ["Carnation", "Lily", "Orchid"];
    var fRow = $("<div id=row3 class=drow/>").appendTo("div.dtable");
    var fTemplate = $("<div class=dcell><img/><label/><input/></div>");
    for (var i = 0; i < fNames.length; i++) {
      fTemplate.clone().appendTo(fRow).children()
        .filter("img").attr("src", fNames[i] + ".png").end()
        .filter("label").attr("for", fNames[i]).text(fNames[i]).end()
        .filter("input").attr({name: fNames[i], value: 0, required: "required"})
    }
  });
</script>
...
```

这些示例代码的可读性很差，而且随着陆续添加更复杂的元素，代码的复杂度还会显著增加。jQuery数据模板库却非常巧妙地把重心移到了HTML层，让你能用最少的代码得到所需的HTML。

若看得更远一点，你会发现把数据整合到页面乃是一个常见问题。在我的项目中有两种情况。第一种情况是因为由遗留系统提供数据驱动Web应用程序。我能够获取数据并把这些数据在服务器端整合到页面当中（很多好用的技术能做这件事），不过这也意味着服务器集群要花费大量时间处理那些本来可以交给浏览器做的事情。如果你曾经构建并运营过很高负载的Web应用程序，一定知道这种花费是巨大的，而且任何一种能够降低负载的机会都会得到认真对待。

需要整合数据到页面还有一个原因，即我的Web应用是由响应用户行为的Ajax操作获取数据的。我会在第14章和第15章全面讲解jQuery对于Ajax的支持情况。长话短说，Ajax是一种无需在浏览器中刷新整个页面就能从服务器端获取并显示数据的技术。这是一项被广泛使用的强大技术，并且数据模板库与Ajax技术配合得极好，天衣无缝。

## 12.2 使用模板库

在使用模板库之前，我们需要先下载这个库，并把它链接到页面中。可以从<https://handlebarsjs.com>下载模板库。我把下载下来的JavaScript代码保存到名为handlebars.js的文件中，与jQuery库文件放在一起。

handlebars是一个独立模板库，与jQuery未做任何整合。不过，写个jQuery插件以支持使用jQuery语法渲染模板也不是件多么麻烦的事。我编写的插件保存在handlebars-jquery.js文件内，代码清单12-2列出了该文件的内容。

代码清单12-2 利用handlebars-jquery.js使handlebars.js支持jQuery语法

```

(function ($) {
    var compiled = {};
    $.fn.template = function (data) {
        var template = $.trim($(this).first().html());
        if (compiled[template] == undefined) {
            compiled[template] = Handlebars.compile(template);
        }
        return $(compiled[template](data));
    };
})(jQuery);

```

### 自己动手编写jQuery插件

代码清单12-2演示了如何编写一个简单的jQuery插件，非常简单，特别是当你只需要封装一个现成的库时。本书不会详细介绍插件的工作原理，因为只有少数开发者才需要了解它。如果你对此有兴趣，请访问 <http://learn.jquery.com/plugins>。

以上代码定义了一个可在jQuery对象上直接调用的template方法，它负责把符合handlebars格式的数据对象应用于（当前jQuery对象对应的）模板之上。它的返回值是包含着模板成功渲染之后的HTML元素的jQuery对象。如代码清单12-3所示，要使模板库正常工作，需要在示例文档中为handlebars.js和handlebars-jquery.js增加两个script标签。

代码清单12-3 在示例文档中添加模板库

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="handlebars.js" type="text/javascript"></script>
    <script src="handlebars-jquery.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            // 这里放示例代码

        });
    </script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2" class="drow"></div>
            </div>

```

```

    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
</html>

```

本章会一直使用这个文档。除了添加模板库和简单的插件，你一定还注意到我删除了展示花的具体代码，这样就可以使用模板库来探索一些新技术，并利用它们把花儿加回来。图12-1展示了浏览器中初始的HTML文档。



图12-1 初始示例文档

## 12.3 第一个数据模板示例

学习数据模板的最好办法是边用边学。代码清单12-4演示了模板的基本功能。因为使用一个script元素保存模板内容，我在这个代码清单中包含了完整的HTML文档，其中包括一些现在用不到的元素。在本章后面的例子里，我会讲解这些元素。

### 代码清单12-4 第一个数据模板示例

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js" type="text/javascript"></script>
  <script src="handlebars-jquery.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script id="flowerImpl" type="text/x-handlebars-template">
    {{#each flowers}}
      <div class="dcell">
        
        <label for="{{product}}">{{name}}:</label>
        <input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
          value="0" required />
      </div>
    {{/each}}
  </script>
  <script type="text/javascript">
    $(document).ready(function () {
      var data = {
        flowers: [
          { name: "Aster", product: "aster", stock: "10", price: 2.99 },

```



```

        { name: "Daffodil", product: "daffodil", stock: "12", price: 1.99 },
        { name: "Rose", product: "rose", stock: "2", price: 4.99 },
        { name: "Peony", product: "peony", stock: "0", price: 1.50 },
        { name: "Primula", product: "primula", stock: "1", price: 3.12 },
        { name: "Snowdrop", product: "snowdrop", stock: "15", price: 0.99 }
    ];
    var template = $("#flowerTpl").template(data).appendTo("#row1");
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2" class="drow"></div>
            </div>
        </div>
        <div id="buttonDiv"><button type="submit">Place Order</button></div>
    </form>
</body>
</html>

```

在接下来的几节中，我会分解这个示例并针对每个具体的部分进行解析。

---

**提示** 如果数据是页面的一部分，我们称它为内置数据。与内置数据对应的是远程数据，即我们从服务器端获取的独立于页面的纯粹数据。我会在本章后面讲解远程数据，而这会涉及jQuery对Ajax的支持（第14章和第15章的主题）。

---

### 12.3.1 定义数据

该示例的起点是数据，在本例中数据是一个对象，它只有一个属性（flowers），该属性的值是一个由对象组成的数组。每一个对象描述了一种花卉。示例HTML文档中与数据有关的语句见代码清单12-5。

代码清单12-5 定义花卉的数据

```

...
var data = {
    flowers: [
        { name: "Aster", product: "aster", stock: "10", price: 2.99 },
        { name: "Daffodil", product: "daffodil", stock: "12", price: 1.99 },
        { name: "Rose", product: "rose", stock: "2", price: 4.99 },
        { name: "Peony", product: "peony", stock: "0", price: 1.50 },
        { name: "Primula", product: "primula", stock: "1", price: 3.12 },
        { name: "Snowdrop", product: "snowdrop", stock: "15", price: 0.99 }
    ]
};
...

```

Handlebars模板基于对象和属性运作，因此，我必须把花卉对象数组封装在data对象里。这个例子中，数组包含6个对象，每个对象都有一套描述花店产品的属性：显示名（name）、产品名（product）、



库存量 (stocklevel) 和价格 (price)。

### 12.3.2 定义模板

如你所想, 数据模板库的核心是数据模板。这是一些包含着占位符的HTML元素, 其中的占位符对应着数据对象的各个部分。代码清单12-6列出了示例文档中模板有关的部分。

代码清单12-6 定义数据模板

```
...
<script id="flowerTpl" type="text/x-handlebars-template">
  {{#flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}:</label>
      <input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
        value="0" required />
    </div>
  {{/flowers}}
</script>
...
```

模板中需要注意的第一点是, 模板内容包含在一个type为text/x-handlebars-template的script元素中。这么做是为了避免浏览器对模板内容进行解析。第二点是我们给定义了模板的script元素设置了id属性。在本例中, 模板script元素的id属性值 (即模板的名字) 为flowerTpl。这个名字非常重要, 因为使用数据渲染模板时会用到它。

要为对象数组中的每个对象生成HTML标签, 就得把模板应用到数组中的每一个对象上。我们看到, 这个模板的结构与之前章节中使用的花店产品的结构相似。

最大的不同, 我已经在代码清单中加粗突出显示, 也就是双胡子指令 (之所以这么说, 是因为包含着指令的两个大括号就像人的胡子, “胡子模板”即由此得来)。在这个例子里有两种类型的指令, 在本章后面的部分, 我还会介绍更多的指令类型。

第一种指令是区域指令, 区域指令的名称与data对象中对应数据的属性名相同, 它在模板中定义一个区域, 存放数据属性中每个对象渲染得来的HTML标签。区域指令的起始指令由#字符开头, 如本例中的 ({{#flowers}}), 区域指令的结束指令则由/字符开头 ({{/flowers}})。我使用区域指令为flowers属性对应的每个对象生成HTML内容。

第二种指令是变量指令, 它的使命就是被data对象中对应名称的属性值替掉。举例来说, 模板库一旦遇到{{product}}变量就会把它替成当前对象的product属性, 比如下面这部分模板:

```
...

...
```

就会被翻译成:

```
...

...
```

### 12.3.3 使用模板

在代码清单12-2中，我把data对象作为参数传递给我在jQuery插件中定义的template方法。下面列出的template方法的具体调用代码摘自示例文档：

```
...
var template = $("#flowerTpl").template(data).appendTo("#row1");
...
```

我们使用jQuery的\$函数选中包含模板的script元素，然后把希望处理的数据对象作为参数传递给template方法。

template方法返回的是jQuery对象格式的模板渲染结果。在本例中，我们得到一系列div元素，每个div都包含着按照数据数组中对应对象“订做”的img、label和input元素。接着，使用appendTo方法把它们整个插入到#row1元素中，这些div元素就成为#row1元素的子元素。最终结果见图12-2。

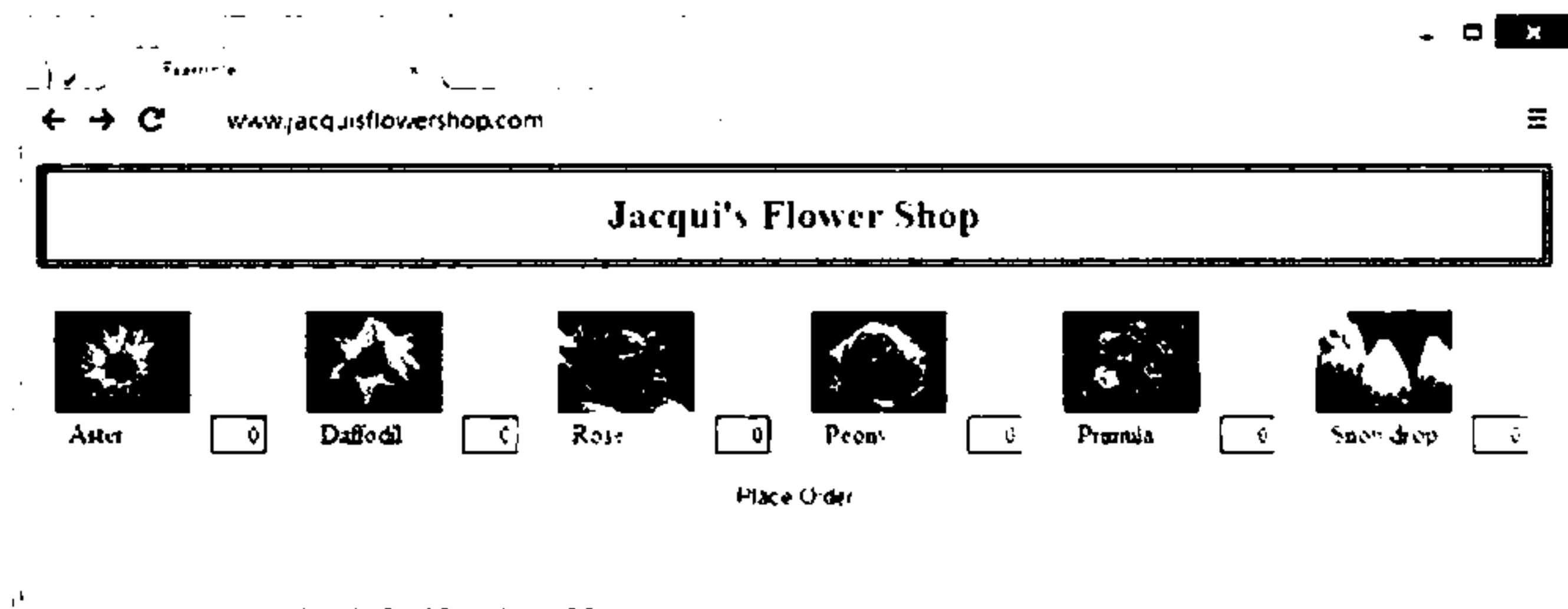


图12-2 使用数据模板

#### 1. 调整模板结果

结果并不十分理想，因为所有的花卉都排成了一行，而不是像前面的章节那样分成两行。不过，因为处理的是jQuery对象，我们可以使用第二部分介绍的那些方法对元素做适当的裁切。代码清单12-7展示了如何裁切template方法的返回结果以达到目标。

#### 代码清单12-7 处理模板的返回结果

```
...
<script type="text/javascript">
  $(document).ready(function () {
    var data = {
      flowers: [
        { name: "Aster", product: "aster", stocklevel: "10", price: 2.99 },
        { name: "Daffodil", product: "daffodil", stocklevel: "12", price: 1.99 },
        { name: "Rose", product: "rose", stocklevel: "2", price: 4.99 },
        { name: "Peony", product: "peony", stocklevel: "0", price: 1.50 },
        { name: "Primula", product: "primula", stocklevel: "1", price: 3.12 },
        { name: "Snowdrop", product: "snowdrop", stocklevel: "15", price: 0.99 }
      ]
    };

    $("#flowerTpl").template(data)
```

```

        .slice(0, 3).appendTo("#row1").end().end().slice(3).appendTo("#row2")
    });
</script>
...

```

在这个例子中，我使用了slice方法限制选择结果，使用end方法来还原选择结果，并使用appendTo方法把模板生成的原始结果的两个子集分别插入到不同的行。

注意，上例中我不得不连续两次调用end方法才回退到最初选择结果（第一次回退到调用appendTo方法之前的结果集，第二次回退到调用slice方法之前的结果集）。图12-3展示的是这个例子的最终结果，我们确实接近了目标，但仍然未能达成。

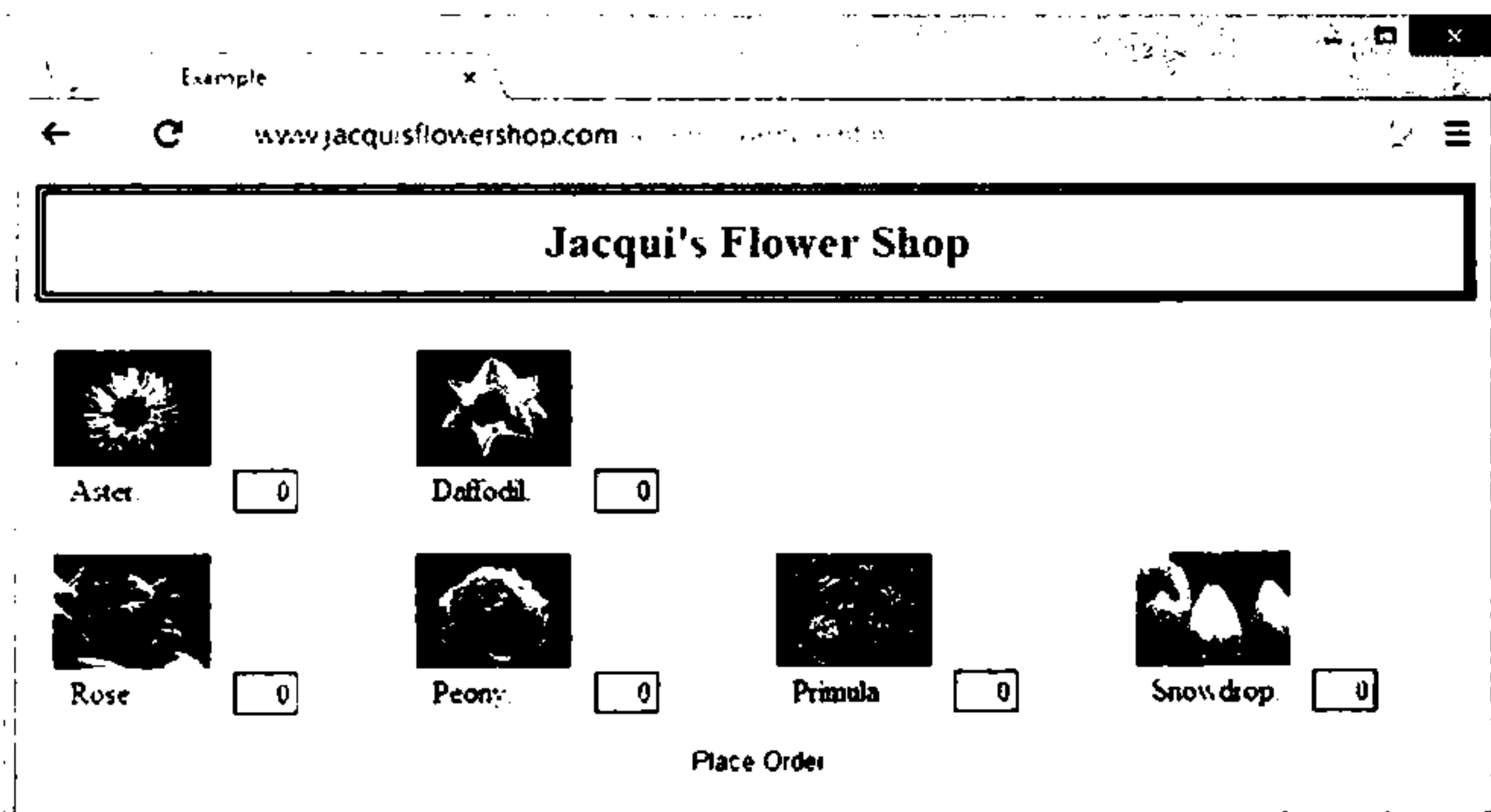


图12-3 尝试调整模板渲染结果的布局

问题是，Handlebars向我们打算切割的模板渲染结果里添加了文本节点。而使用slice方法的切割目标是所有元素（包括文本节点），这导致了我们在错误的位置切割内容。

有好几种方法修正这个问题。第一种，我们调整模板，让所有内容都显示成一行。但我不喜欢这么做，我更希望让模板保持尽可能好的可读性。

另一种方法是调整slice方法的起始索引，把handlebars模板库添加的文本节点考虑进去。然而我也不喜欢这个方案，因为不是所有文本编辑器的换行符都会导致产生多余的文本节点，这意味不同编辑器修改出来HTML文件可能导致JavaScript行为不同，这太糟糕了。

理想的方案是，在把模板渲染结果插入到DOM之前，使用jQuery滤掉其中的文本节点。然而，jQuery库里并没有特别适合这个任务的方法，因此最佳的选择就是以\*过滤器为参数调用filter方法，这个选择器匹配任意的HTML标签，不包括文本节点。在代码清单12-8里，我已经加上了filter方法。

#### 代码清单12-8 使用filter方法滤掉文本节点

```

...
$("#flowerTpl").template(data).filter("*")
    .slice(0, 3).appendTo("#row1").end().end().slice(3).appendTo("#row2")
...

```

结果见图12-4：这些花儿被恰到好处地分为两行。

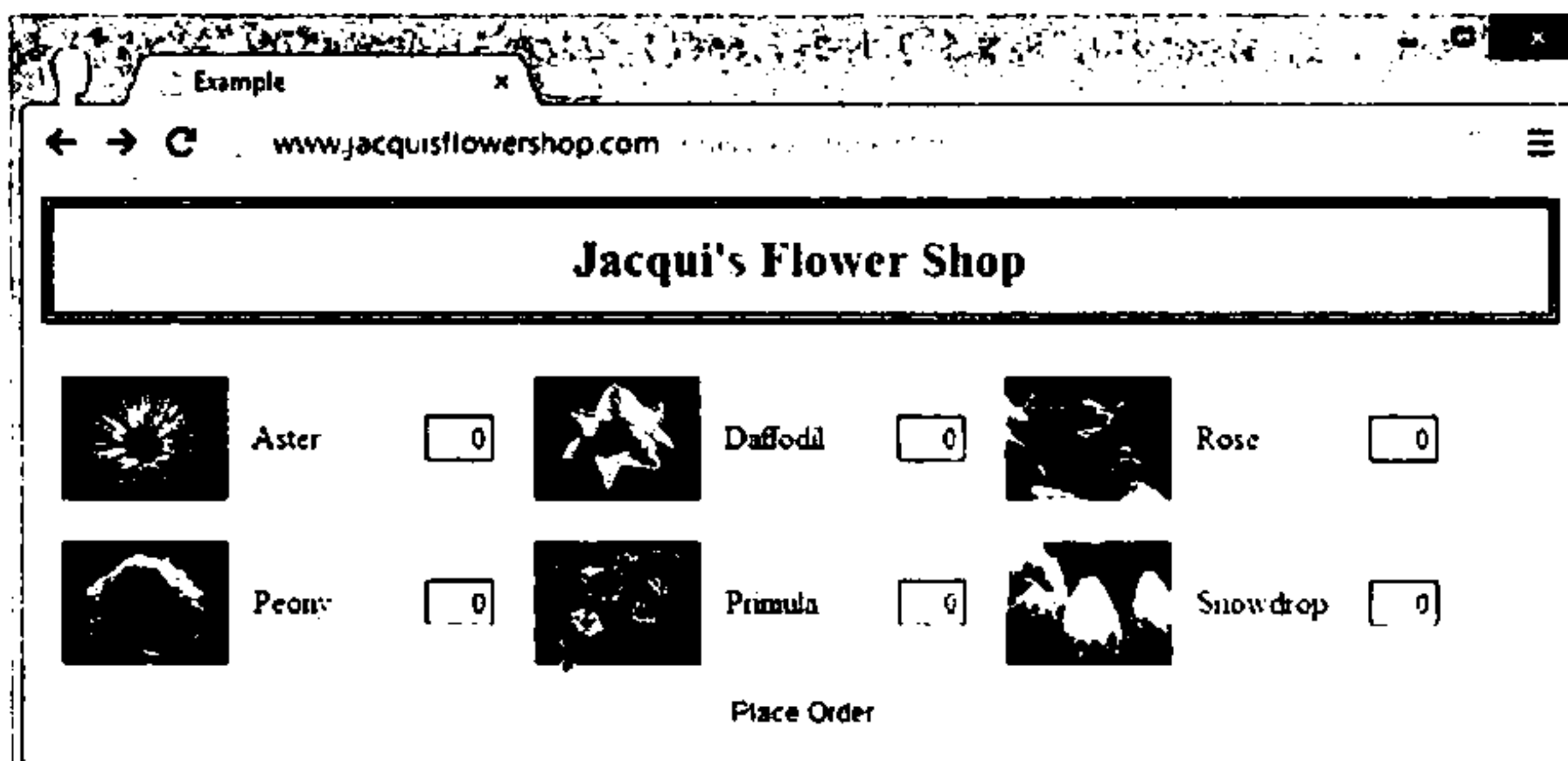


图12-4 使用filter方法滤掉文本节点

用这种方法处理模板渲染的结果，我仍然不太满意。通常我喜欢在一条语句里使用end方法链式调用处理各种操作，但end().end()的写法却不怎么讨喜。这时，我更倾向于把这几步操作写成独立的步骤。如代码清单12-9所示，它的输出结果与代码清单12-8完全相同，但更容易理解。

#### 代码清单12-9 使用多个语句分离元素

```
...
var templateHtml = $("#flowerTpl").template(data).filter("*");
templateHtml.slice(0, 3).appendTo("#row1");
templateHtml.slice(3).appendTo("#row2");
...
```

#### 2. 调整输入数据

另一个解决方案是调整提供给template方法的输入数据。代码清单12-10演示了这种做法。

#### 代码清单12-10 调整输入数据以改变模板输出

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var data = {
            flowers: [
                { name: "Aster", product: "aster", stocklevel: "10", price: 2.99 },
                { name: "Daffodil", product: "daffodil", stocklevel: "12", price: 1.99 },
                { name: "Rose", product: "rose", stocklevel: "2", price: 4.99 },
                { name: "Peony", product: "peony", stocklevel: "0", price: 1.50 },
                { name: "Primula", product: "primula", stocklevel: "1", price: 3.12 },
                { name: "Snowdrop", product: "snowdrop", stocklevel: "15", price: 0.99 }
            ]
        };

        var tElem = $("#flowerTpl");
        tElem.template({ flowers: data.flowers.slice(0, 3) }).appendTo("#row1");
        tElem.template({ flowers: data.flowers.slice(3) }).appendTo("#row2");
    });
</script>
...
```

在这个脚本中，我通过针对每一行单独调用一次template方法解决了花卉的分配问题。使用split方法得到适当的数据以提供给template方法。如图12-4所示，这个脚本采用了不同的方法，但得出的结果是相同的。注意，我们必须小心维护传递给template方法的对象格式，以确保它符合模板中声明的区域指令——它必须是一个对象，并且有一个名为flowers的属性，而且这个属性的值是由我们打算处理的对象组成的数组。

## 12.4 模板逻辑

辨别众多JavaScript模板引擎的一个方法，就是观察模板如何处理各种数据类型，如何输出数据。

有一种“极品”模板，它不包括任何逻辑。使用这种模板，意味着要改变模板的输出，就要在应用模板前仔细准备数据。另一种“极品”则支持完备的逻辑，它就像一种专注于定义并执行模板指令的简单编程语言，支持内建的逻辑语句、循环语句、数组处理，还支持对数据集的管理。

模板之间的区别，无非谁支持的逻辑多些，谁支持的少些。模板引擎到底应该支持多少逻辑，永远不可能达成一致。我倾向于两个极端之间的某个标准，能在模板里使用逻辑固然很好，但我仍然希望模板足够简单，而不是在HTML文档里再引入一种新语言。在这一章里，我之所以选择Handlebars库，是因为它允许你根据需要，或多或少地使用逻辑。而且，在本章后面你会看到，在需要解决特定问题时，能自定义逻辑这一特性使得问题更容易解决。如表12-2所示，Handlebars库包含一些内建助手指令，这都是些简单的逻辑运算符，可以用来根据输入数据的值调整模板的输出。

表12-2 Handlebars内建助手指令

助手指令	描 述
#if	if/then/else条件指令，对属性求值，若指定属性存在且不是null，则返回true
#unless	#if指令的反指令，若指定属性不存在或者值为null，返回true
#each	迭代处理对象数组或一个对象的属性
#with	为模板的一个区块设定上下文

### 12.4.1 依据条件生成内容

为了演示如何在Handlebars模板中使用逻辑，我会根据data对象里花卉数组中每个对象stock属性的值设置input元素的value属性。我的目的在于若stock属性的值大于零，我就把value属性设置成1。在代码清单12-11中，你可以看到我是如何在模板中使用#if助手指令的。

代码清单12-11 使用模板逻辑改变模板输出

```
...
<script id="flowerTpl" type="text/x-handlebars-template">
  {{#flowers}}
  <div class="dcell">
    
    <label for="{{product}}">{{name}}:
    <input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
      value="{{#if stock}}1{{else}}0{{/if}}" required />
  </div>
```

```

    {{/flowers}}
</script>
<script type="text/javascript">
    $(document).ready(function () {
        var data = {
            flowers: [
                { name: "Aster", product: "aster", stock: "10", price: 2.99 },
                { name: "Daffodil", product: "daffodil", stock: "12", price: 1.99 },
                { name: "Rose", product: "rose", stock: "2", price: 4.99 },
                { name: "Peony", product: "peony", stock: "0", price: 1.50 },
                { name: "Primula", product: "primula", stock: "1", price: 3.12 },
                { name: "Snowdrop", product: "snowdrop", stock: "15", price: 0.99 }
            ];
            for (var i = 0; i < data.flowers.length; i++) {
                if (data.flowers[i].stock == 0) {
                    data.flowers[i].stock = null;
                }
            }
            var tElem = $("#flowerTpl");
            tElem.template({ flowers: data.flowers.slice(0, 3) }).appendTo("#row1");
            tElem.template({ flowers: data.flowers.slice(3) }).appendTo("#row2");
        });
    </script>
    ...

```

#if指令的每个部分都使用一对双大括号包裹：第一部分，#if后面紧跟着我打算检查的属性，在本例中就是stock。若stock属性由#flowers区块指令当前正在处理的对象所有且值不为null，则该指令返回true。如果是这样，模板引擎就会输出条件{{#if stock}}之后的内容为HTML，在这个例子中就是字符串1。

可选的else部分，它的工作方式与JavaScript中完全一样。利用它我们可以为求值不为true的情况，即当前对象没有stock属性或者属性值为null时，提供一个替代值。在本例中，模板引擎会输出字符串0。最后一部分是/if，用来标识逻辑块的结束。

#if助手指令背后的逻辑是只要属性存在且定义了有效的值就会返回true，这就迫使我在传递给template方法之前重新处理相关数据。使用JavaScript循环处理每个花卉对象，若stock属性值为0，就把它设置为null。结果如图12-5所示，除了Peony（牡丹花），其他花卉的input元素的模板输出都是1。

---

**提示** 我不喜欢为了适应模板引擎而去处理数据，在后面我会介绍创建自定义逻辑的方法，这样就不需要处理数据了。

---

#unless助手指令与#if指令的工作方式相同，只是逻辑正好相反。当属性不存在，或者属性值为null时，它会返回true。在代码清单12-12中，你将看到如何做到无需else指令，通过#if和#unless指令在模板中设置input元素的value属性。

#### 代码清单12-12 在模板中使用#if和#unless助手指令

```

...
<script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}

```



```

<div class="dcell">
  
  <label for="{{product}}">{{name}}:
  <input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
    value="{{#if stock}}1{/if}}{{#unless stock}}0{/unless}}" required />
</div>
{{/flowers}}
</script>
...

```

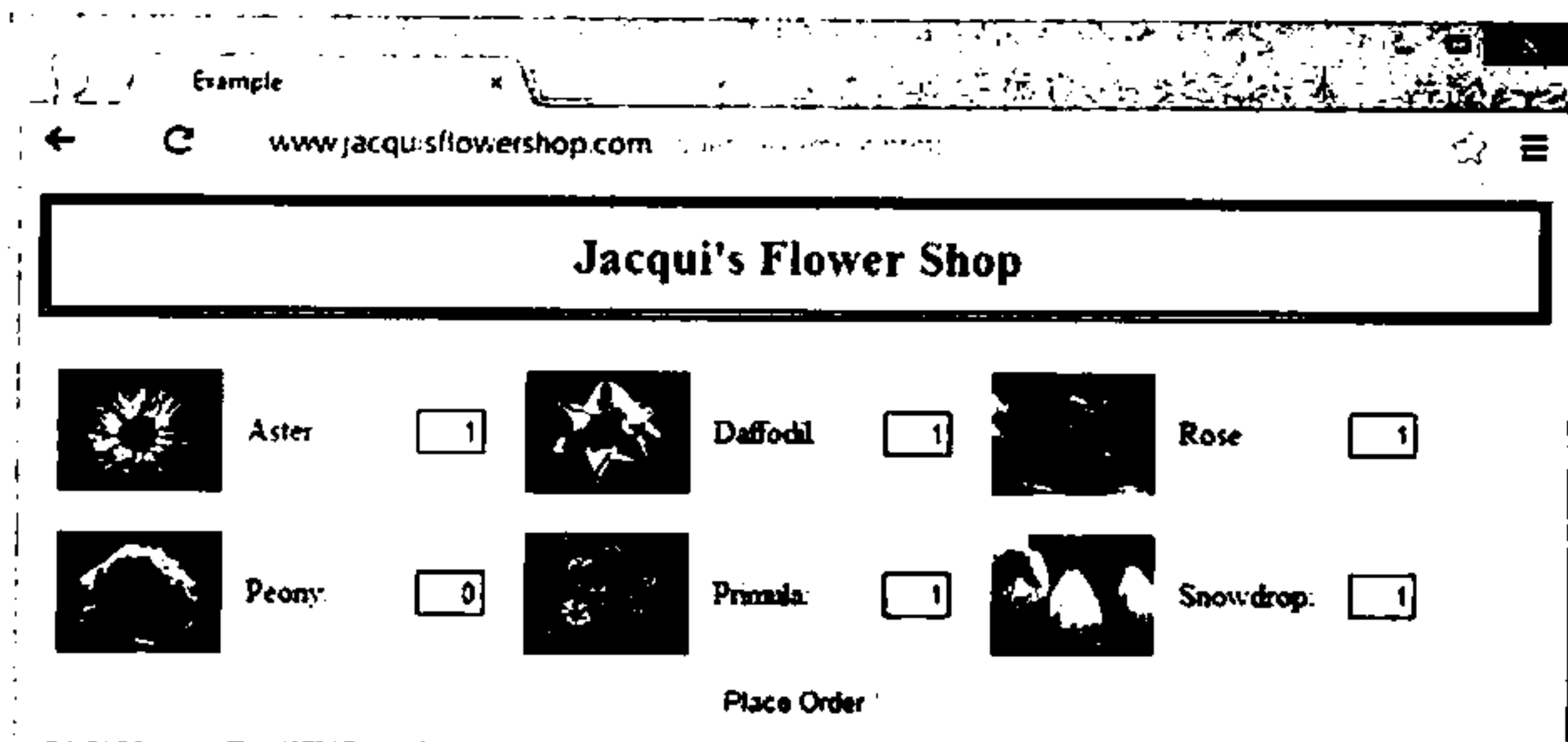


图12-5 使用模板逻辑改变模板输出

#if和#unless指令完全相互独立，之所以以这种方式使用它们只是为了演示在使用else指令的情况下，如何测试一个属性是否缺失。如图12-5所示，这个代码清单的输出结果与清单12-11完全相同。

## 12.4.2 遍历数组和对象属性

在代码清单12-12中使用的区块指令，作为Handlebars库的一部分，在胡子模板中被广泛使用。而#each指令则是区块指令的一个更复杂的替代品，它提供了一些可以在模板中使用的特殊属性。表12-3列出了这些属性。

表12-3 #each助手指令提供的特殊属性

属 性	描 述
this	返回当前正在处理的对象
@index	若#each指令正在处理一个数组，则它就是当前元素的数组索引
@key	若#each指令正在处理一个对象，则返回当前属性的名字

在代码清单12-13中，可以看到如何对现有数据应用#each助手指令，以及@index属性。

### 代码清单12-13 #each助手指令与@index属性的用法

```

...
<script id="flowerTpl" type="text/x-handlebars-template">
  {{#each flowers}}
    <div class="dcell">

```

```

<label>Position: {{@index}}</label>

<label for="{{product}}">{{name}}:</label>
<input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
      value="{{#if stock}}1{{/if}}{{#unless stock}}0{{/unless}}" required />
</div>
{{/each}}
</script>
...

```

我把#each助手指令的参数指定为将被遍历的对象数据源,在本例中也就是传递给template方法的data对象的flowers属性。本例的执行结果见图12-6(每一行的索引都是从0开始的,这是因为我在代码清单12-11中把数据切割成了两部分,调用了template方法两次)。

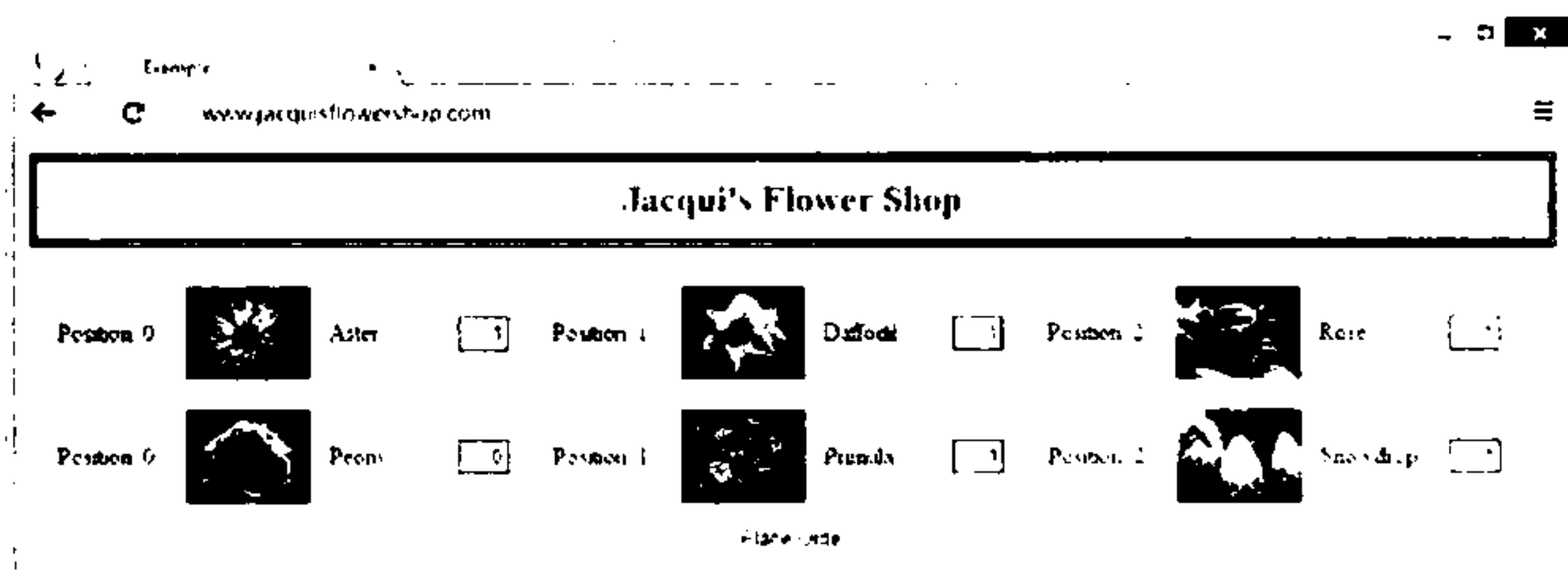


图12-6 #each助手指令与@index属性的用法

与数组相比,当处理对象时,this和@key属性更有用。Handlebars库会遍历对象的属性,也就是说,@key属性里保存着当前属性的名字,this属性里则保存着当前属性的值。在代码清单12-14中,同时用到了这两个属性。

#### 代码清单12-14 使用#each助手指令遍历对象属性

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js" type="text/javascript"></script>
  <script src="handlebars-jquery.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script id="flowerListTpl" type="text/x-handlebars-template">
    <ul>
      {{#each stockData}}
        <li>{{@key}} ({{this}} in stock)</li>
      {{/each}}
    </ul>
  </script>
  <script type="text/javascript">
    $(document).ready(function () {
      var data = {

```



```

        stockData: {
            Aster: 10, Daffodil: 12, Rose: 2,
            Peony: 0, Primula: 1, Snowdrop: 15
        }
    };
    $("#flowerListTpl").template(data).appendTo("form");
});
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post">
        </form>
    </body>
</html>

```

在这个例子里我将data对象替换为更简单的结构——data.stockData属性返回一个对象，它的属性是花卉的名称，而属性的值则是相应花卉的存货数量。对这个模板，#each助手指令的参数是stockData属性（记住我是把一个对象传递给template方法，所有的助手指令都将应用到该对象的属性上）。示例中的模板生成了一个列表，通过this属性获取每种花的存货数，并使用@key属性得到了每种花的名称。图12-7是这个例子的运行结果。

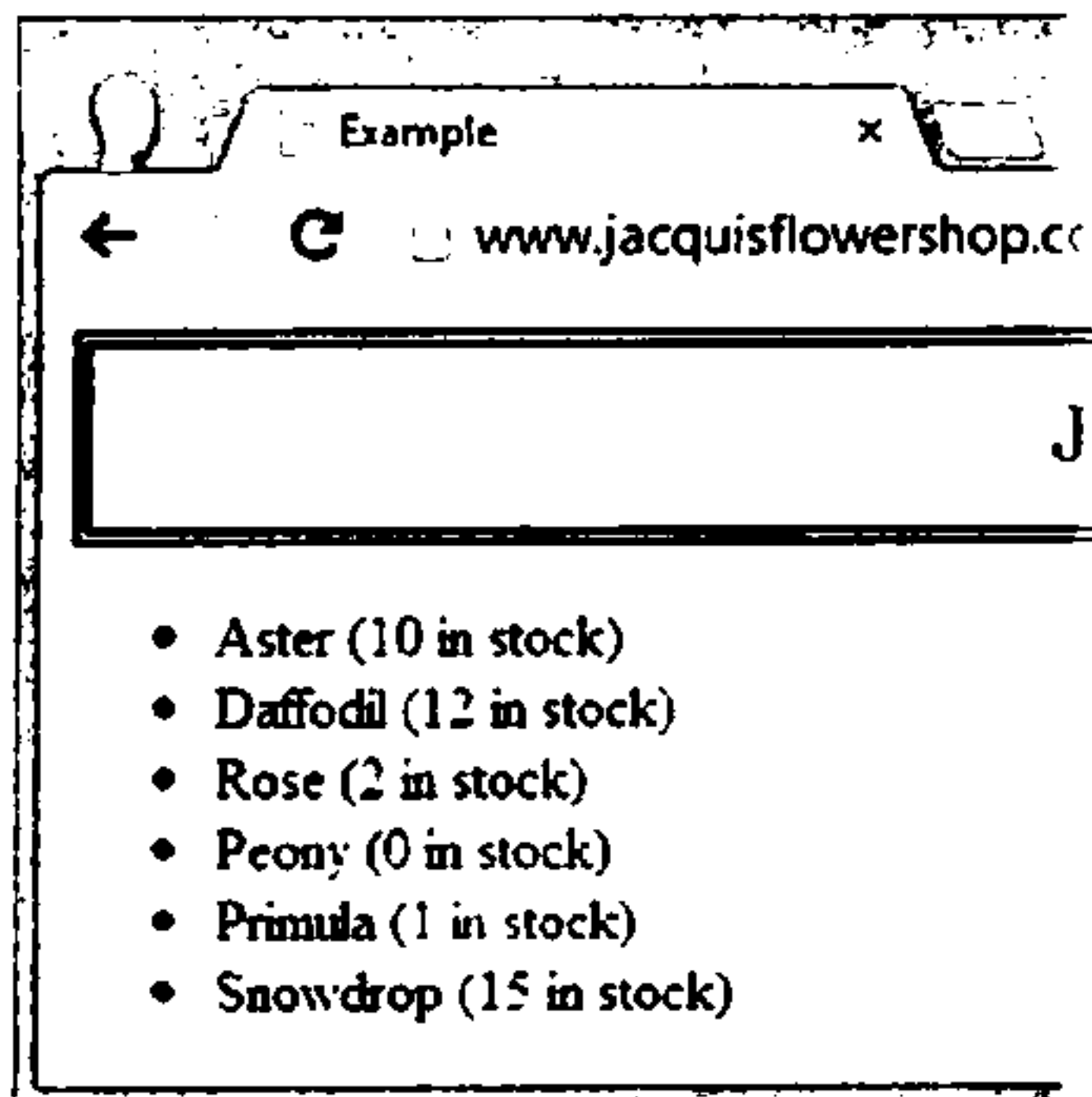


图12-7 在#each助手指令中使用@key和this属性

### 12.4.3 改变数据上下文

数据上下文就是助手指令或变量所依赖的数据对象。一旦模板处理开始，上下文就是整个数据对象，不过为了方便编写模板，如下面的模板（来自上一个例子）所示，它可以被助手指令或者区块指令改变。

```

...
{{#each stockData}}
    <li>{{@key}} ({{this}} in stock)</li>
{{/each}}
...

```

#each助手指令每循环一次就顺序更换一个对象属性，也就是说#each块中定义的变量和助手指令会以当前对象为上下文求值。在代码清单12-15中，我修改了数据对象和模板，从而使这一点更加清楚。

代码清单12-15 强调模板中的上下文角色

```
...
<script id="flowerListTpl" type="text/x-handlebars-template">
  <ul>
    <h3>{{title}}</h3>
    {{#each stockData}}
      <li>{{description.Name}} ({{description.Stock}} in stock)</li>
    {{/each}}
  </ul>
</script>
<script type="text/javascript">
  $(document).ready(function () {

    var data = {
      title: "Stock List",
      stockData: {
        aster: {
          description: { Name: "Aster", Stock: 10 }
        },
        daffodil: {
          description: { Name: "Daffodil", Stock: 12 }
        },
        rose: {
          description: { Name: "Rose", Stock: 2 }
        }
      }
    };
    $("#flowerListTpl").template(data).appendTo("form");
  });
</script>
...
```

我在数据对象里增加了一个title属性，并使每一种花的数据结构化。模板中的第一个指令依赖默认的数据上下文，即整个数据对象：

```
...
<h3>{{title}}</h3>
...
```

显然这个指令的求值与上下文有关，只需要指定属性名称，就能拿到数据对象中title属性的值，无需任何限制。模板中的下一个指令则改变了上下文：

```
...
{{#each stockData}}
...
```

#each指令遍历stockData所对应的对象的属性，并顺序地把每一个属性作为模板指令的上下文。在模板的下一行中，你可以看到实际效果：

```
...
<li>{{description.Name}} ({{description.Stock}} in stock)</li>
...
```

根据当前上下文访问Name和Stock属性，也就是说跟随数据对象的结构，可直接访问description对象。执行结果见图12-8。

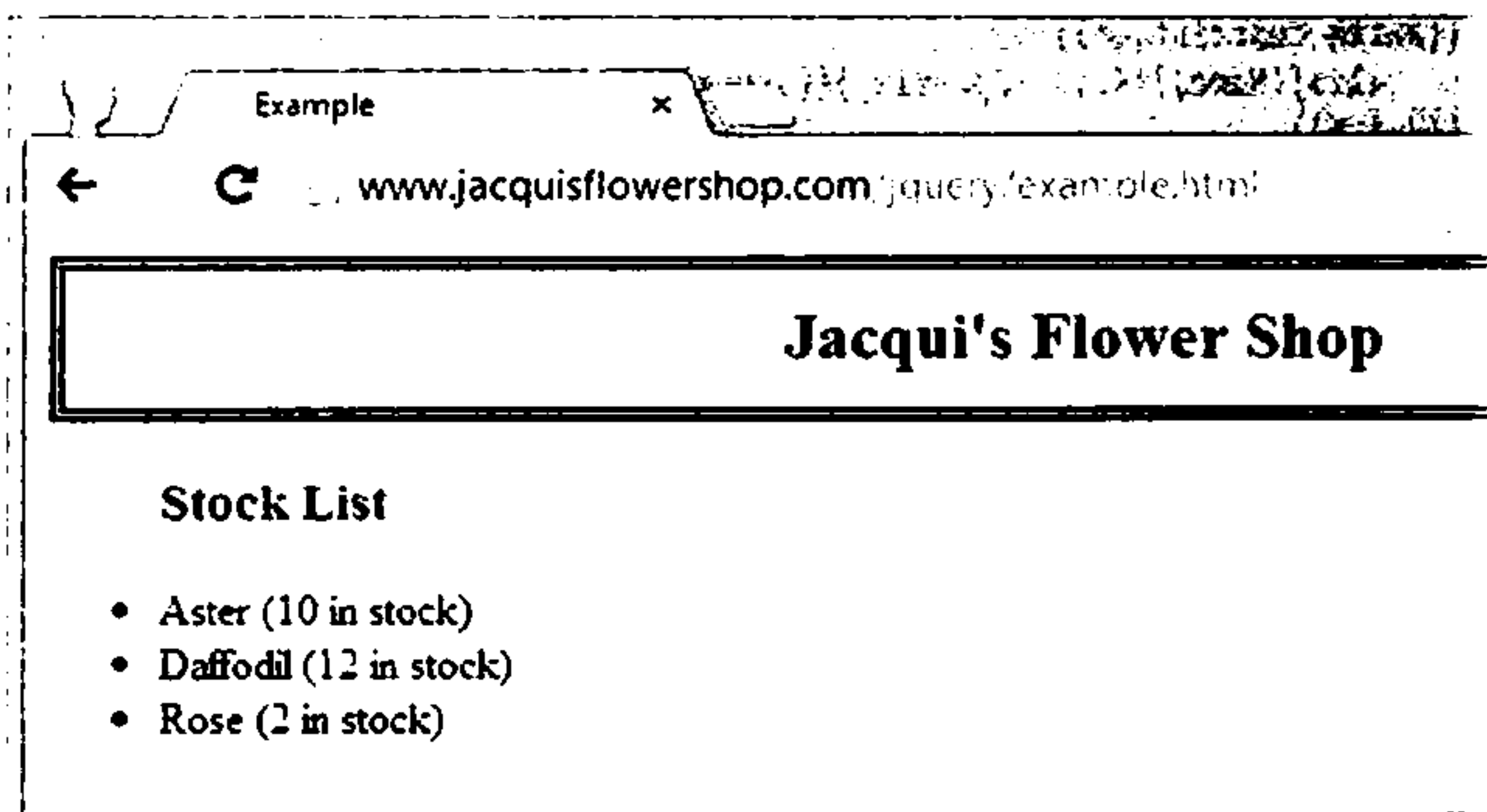


图12-8 相对于上下文访问属性

### 1. #with助手指令

在上一个例子中，我只能使用description.Name和description.Stock来访问所需属性的值。如代码清单12-16所示，#with指令可以改变数据上下文，从而帮助我们消灭每次都要重复书写的公共属性名。

#### 代码清单12-16 使用#with指令改变数据上下文

```
...
<script id="flowerListTpl" type="text/x-handlebars-template">
  <ul>
    <h3>{{title}}</h3>
    {{#each stockData}}
      {{#with description}}
        <li>{{Name}} ({{Stock}} in stock)</li>
      {{/with}}
    {{/each}}
  </ul>
</script>
...
```

在#with指令块内，上下文被进一步缩小为description属性。在#each指令与#with指令的双重作用之下，可以直接访问每一种花的Name与Stock属性，无需指定属性名description。

### 2. 访问上级数据节点

改变数据上下文带来的并不总是好处，有时我们需要在模板中访问数据对象的另一部分，如代码清单12-17所示，可以在变量名字之前添加 ../前缀来访问上一级上下文。

代码清单12-17 访问上级上下文

```

...
<script id="flowerListTpl" type="text/x-handlebars-template">
  <ul>
    <h3>{{title}}</h3>
    {{#each stockData}}
      {{#with description}}
        <li>{{Name}}{{../../prefix}}{{Stock}}{{../../suffix}}</li>
      {{/with}}
    {{/each}}
  </ul>
</script>
<script type="text/javascript">
  $(document).ready(function () {

    var data = {
      title: "Stock List",
      prefix: " (",
      suffix: " in stock)",
      stockData: {
        aster: {
          description: { Name: "Aster", Stock: 10 }
        },
        daffodil: {
          description: { Name: "Daffodil", Stock: 12 }
        },
        rose: {
          description: { Name: "Rose", Stock: 2 }
        }
      }
    };
    $("#flowerListTpl").template(data).appendTo("form");
  });
</script>
...

```

我在数据对象的顶层定义了prefix和suffix属性。如下面代码所示，要从模板里访问这两个属性，需要访问上两级上下文：

```

...
{{../../prefix}}
...

```

由于这条指令出现在#with指令块内，因此一次../只能把上下文切换到模板的上一层，也就是#each指令块。#each指令定义的上下文是stockData属性，因此还需要向上提高一级上下文才能拿到prefix属性，也就是说必须使用../../prefix才能拿到我想要的值。

---

**提示** ../指令所指的是模板的上一级，而不是数据对象的上一级。

---

## 12.5 自定义助手指令

有些助手指令的逻辑很简单，这意味着通常需要预先对数据做一些处理以满足助手指令的要求。代码清单12-18就是这样实现的，它使用了本章前面讲解#if指令时所用的那个例子。

代码清单12-18 为适应模板助手指令而处理数据

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js" type="text/javascript"></script>
  <script src="handlebars-jquery.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>

  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}:</label>
      <input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
        value="{{#if stock}}1{{else}}0{{/if}}" required />
    </div>
    {{/flowers}}
  </script>
  <script type="text/javascript">
    $(document).ready(function () {
      var data = {
        flowers: [
          { name: "Aster", product: "aster", stock: "10", price: 2.99 },
          { name: "Daffodil", product: "daffodil", stock: "12", price: 1.99 },
          { name: "Rose", product: "rose", stock: "2", price: 4.99 },
          { name: "Peony", product: "peony", stock: "0", price: 1.50 },
          { name: "Primula", product: "primula", stock: "1", price: 3.12 },
          { name: "Snowdrop", product: "snowdrop", stock: "15", price: 0.99 }
        ]
      };
      for (var i = 0; i < data.flowers.length; i++) {
        if (data.flowers[i].stock == 0) {
          data.flowers[i].stock = null;
        }
      }
      var tElem = $("#flowerTpl");
      tElem.template({ flowers: data.flowers.slice(0, 3) }).appendTo("#row1");
      tElem.template({ flowers: data.flowers.slice(3) }).appendTo("#row2");
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post">
    <div id="oblock">
      <div class="dtable">
```

```

        <div id="row1" class="drow"></div>
        <div id="row2" class="drow"></div>
    </div>
</div>
<div id="buttonDiv"><button type="submit">Place Order</button></div>
</form>
</body>
</html>

```

我不喜欢这种把数据传递给模板之前要先对数据做一些处理的工作模式，因为这意味着我的显示层逻辑分散在两个地方：模板处和for循环处。

有两种方法可以保证模板生成内容的逻辑只在一处。一种是从模板中删除逻辑，用JavaScript代码定义所有的逻辑。另一种是删除JavaScript代码中的for循环，在模板里添加额外的逻辑。模板应该如何使用？两种方案各有所长，为此人们意见不一，争论不休。不过这归根结底与人的使用习惯及所处理数据的性质相关。我个人喜欢在模板里添加逻辑，而且使用Handlebars库来实现既简单又轻松。

### 12.5.1 编写条件助手指令

在代码清单12-19中，我在handlebars-jquery.js文件里加了些东西，为模板定义了一个自定义助手指令。

---

**提示** 可以在任意的script标签或者JavaScript文件里编写自定义逻辑。我个人喜欢把加强Handlebars.js的代码都放在同一个地方。

---

代码清单12-19 在handlebars-jquery.js文件中为Handlebars库添加自定义条件

```

(function ($) {

    Handlebars.registerHelper('gt', function (a, b, options) {
        return (a > b) ? options.fn(this) : options.inverse(this);
    });

    var compiled = {};
    $.fn.template = function (data) {
        var template = $.trim($(this).first().html());
        if (compiled[template] == undefined) {
            compiled[template] = Handlebars.compile(template);
        }
        return $(compiled[template](data));
    };
})(jQuery);

```

Handlebars库定义了一个名为Handlebars的全局对象，它定义有一个方法registerHelper。该方法可接受两个参数：助手指令的名称，以及一个函数，当我们在模板中使用该助手指令时，就会调用这个函数。我定义的助手指令名为gt（即greater than的缩写）。展示一下这个助手指令在模板里如何应用，是弄清该助手指令工作原理最简单的方法。代码清单12-20展示了这个助手指令在示例页面中的用法。

## 代码清单12-20 应用自定义助手指令

```

...
<script id="flowerTpl" type="text/x-handlebars-template">
  {{#flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}:</label>
      <input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
        value="{{#gt stock 0}}1{{else}}0{{/gt}}" required />
    </div>
  {{/flowers}}
</script>
<script type="text/javascript">
  $(document).ready(function () {
    var data = {
      flowers: [
        { name: "Aster", product: "aster", stock: "10", price: 2.99 },
        { name: "Daffodil", product: "daffodil", stock: "12", price: 1.99 },
        { name: "Rose", product: "rose", stock: "2", price: 4.99 },
        { name: "Peony", product: "peony", stock: "0", price: 1.50 },
        { name: "Primula", product: "primula", stock: "1", price: 3.12 },
        { name: "Snowdrop", product: "snowdrop", stock: "15", price: 0.99 }
      ];
    };
    var tElem = $("#flowerTpl");
    tElem.template({ flowers: data.flowers.slice(0, 3) }).appendTo("#row1");
    tElem.template({ flowers: data.flowers.slice(3) }).appendTo("#row2");
  });
</script>
...

```

**提示** 浏览器有时候会主动调用缓存中的JavaScript代码，这将导致更改无效。如果你看到的结果与预期不符，不妨刷新一下页面。

在这个模板中，我的#gt助手指令检查stock属性是否大于0。如果大于0，就把字符串1插入模板，否则插入字符串0。

把模板渲染为HTML之前，必须先调用Handlebars.registerHelper方法，来通知Handlebars库定义了新指令。当模板遇到该助手指令时，Handlebars会对#gt之后的所有值在当前上下文中求值，并把结果作为参数传递给我定义的指令处理函数。

在这个代码清单中，我的#gt指令位于一个区块指令之中，也就是说Handlebars会遍历data.flowers数组，把当前花卉stock属性的值赋给#gt指令的stock属性。例如，对于Aster来说，助手指令gt的stock属性值就是10，stock属性之后的数字0无需求值，因此会原样传递给助手指令处理函数。

**提示** 你需要提供几个参数给助手指令处理函数，数量不限。对这个例子来说，我需要两个参数完成基本的数值比较运算。

这样，我的助手指令处理函数定义的两个参数a和b分别拿到值10和0。除这两个参数之外，我的

处理函数还会额外得到一个选项对象，这个对象由Handlebars库提供。

```
...
Handlebars.registerHelper('gt', function (a, b, options) {
  return (a > b) ? options.fn(this) : options.inverse(this);
});
...
```

表12-4列出了options对象中对编写自定义指令有用的功能。

表12-4 options对象定义的属性与方法

属 性	描 述
fn(data)	若条件表达式值为true，则返回指令定义的内容，或者没有else指令时定义的惟一内容
inverse(data)	返回助手指令{{else}}子句定义的内容
Hash	把可选参数传递给助手指令处理函数
Data	为模板提供特别属性支持

在#gt助手指令中，我调用options.fn方法来得到a大于b时应该插入模板的内容（在示例中，即当前花卉的stock属性大于0时）。传递给这个方法的this参数，是由Handlebars自动设定的当前数据上下文。若a不大于b，就调用options.inverse方法。在这个例子里，options.fn方法会返回1，而options.inverse方法则返回0。

## 12.5.2 返回更复杂的内容

插入到模板中的HTML内容，就是助手指令的返回结果。可以任意增加助手指令返回结果的复杂度，插入更大的HTML片段，从而简化模板。代码清单12-21展示了我在handlebars-jquery.js中定义的助手指令#gtValAttr。

代码清单12-21 在handlebars-jquery.js中定义一个复杂的助手指令

```
(function ($) {

  Handlebars.registerHelper('gt', function (a, b, options) {
    return (a > b) ? options.fn(this) : options.inverse(this);
  });

  Handlebars.registerHelper("gtValAttr", function () {
    return "value='" + (this.stock > 0 ? "1" : "0") + "'";
  });

  var compiled = {};
  $.fn.template = function (data) {
    var template = $.trim($(this).first().html());
    if (compiled[template] == undefined) {
      compiled[template] = Handlebars.compile(template);
    }
    return $(compiled[template](data));
  };
})(jQuery);
```



这个助手指令根本不需要参数，它从this（前面说过，this的值即当前数据上下文）属性里拿到它需要的值。这个助手指令的返回值是针对当前花卉为模板定制的完整value属性定义。代码清单12-22展示了该助手指令的用法。

代码清单12-22 在模板中应用#gtValAttr

```
...
<script id="flowerTpl" type="text/x-handlebars-template">
  {{#flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}:</label>
      <input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
        {{#gtValAttr}}{{/gtValAttr}} required />
    </div>
  {{/flowers}}
</script>
...
```

**注意** 之所以编写这个助手指令只是为了告诉你Handlebars库有多么灵活，不过我并不会在真实项目中使用这种指令。它过于依赖数据和模板的结构，任意一方改变都会造成代码出错。我更喜欢创建那些小巧的、像#gt那样专注解决某个通用问题的指令，这种指令只生成尽可能少的内容，若能完全由参数提供会更加完美。

### 12.5.3 在助手指令处理函数中使用可选参数

可以在模板中定义传递什么参数给助手指令处理函数。可选参数的用法完全取决于助手指令的作者，不过最常见的用法还是传递一些属性值给助手指令处理函数，以便生成完整的HTML标签。在代码清单12-23中，我定义了一个助手指令，它负责根据花卉数据对象生成一个完整的input元素。我再重申一次，也许有读者会喜欢在模板中使用这种复杂的助手指令，但在我自己的项目中，我并不喜欢这种助手指令。

代码清单12-23 在handlebars-jquery.js中定义使用可选参数的助手指令

```
(function ($) {

  Handlebars.registerHelper('gt', function (a, b, options) {
    return (a > b) ? options.fn(this) : options.inverse(this);
  });

  Handlebars.registerHelper("gtValAttr", function () {
    return "value='" + (this.stock > 0 ? "1" : "0") + "'";
  });

  Handlebars.registerHelper("inputElem", function (product, stock, options) {
    options.hash.name = product;
    options.hash.value = stock > 0 ? "1" : "0";
  });
})
```

```

        options.hash.required = "required";
        return $("<input>", options.hash)[0].outerHTML;
    });

    var compiled = {};
    $.fn.template = function (data) {
        var template = $.trim($(this).first().html());
        if (compiled[template] == undefined) {
            compiled[template] = Handlebars.compile(template);
        }
        return $(compiled[template](data));
    };
})(jQuery);

```

这个#inputElem助手指令为花卉生成了一个完整的input元素。如代码清单12-24所示，在实际的模板中观察如何使用这个助手指令，会更容易理解它的工作原理。

代码清单12-24 在模板中使用#inputElem助手指令

```

...
<script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
    <div class="dcell">
        
        <label for="{{product}}">{{name}}:</label>
        {{#inputElem product stock data-stock=stock data-price=price}}{{/inputElem}}
    </div>
    {{/flowers}}
</script>
...

```

在代码清单12-23中可以看到，除可选参数之外，#inputElem助手指令的处理函数可接受两个参数，在模板中这两个参数负责把product和stock属性的值传递给处理函数。除此之外的其他参数都是key=value这种形式，在当前上下文环境中，Handlebars库会对这些参数先求值，然后以options.hash对象属性的方式传递给助手指令函数。在这个例子中，对花卉Aster来说，options.hash属性会返回如下内容的对象：

---

```

{"data-stock": 10, "data-price": 2.99}

```

---

本书第二部分提到了\$函数的一种用法，即传入一个HTML字符串和一个映射对象，它会生成一个jQuery对象。option.hash的格式是创建input元素所必需的格式。不过，现在option.hash对象的属性还不能满足input元素的需要，因此我使用下面的代码进一步完善该对象。

```

...
options.hash.name = product;
options.hash.value = stock > 0 ? "1" : "0";
options.hash.required = "required";
...

```

下面的代码使用\$函数生成所需的input元素，添加各个属性，然后返回模板需要的HTML字符串：

```

...
return $("<input>", options.hash)[0].outerHTML;
...

```

为了得到HTML字符串，我在jQuery对象上使用数组索引值0得到input元素对应的HTMLElement对象，然后使用outerHTML属性得到如下字符串：

---

```
<input data-stock="10" data-price="2.99" name="aster" value="1" required="required">
```

---

如#inputElem助手指令示例所示，在利用助手指令生成HTML标签方面，Handlebars与jQuery配合得相当完美。

---

**提示** 由于jQuery缺少一个便捷的方法从一个HTML元素得到对应的HTML字符串，我只能使用更底层的HTMLElement对象的outerHTML属性。jQuery虽然提供了一个功能接近的html方法，但这个方法返回的是一个元素的HTML内容而非代表元素本身的HTML。这意味着我必须先把input元素追加到另一个元素内才可以使用html方法，如：`$("<div ></div>").append($("<input>", options.hash)).html();`，这实在太绕，不如直接使用HTMLElement对象简单易懂。

---

## 12.5.4 自定义模板属性

在本章前面部分，我说过#each助手指令定义了一些特别的属性，在#each块范围内可以使用这些属性。在我们自己编写的助手指令里，同样可以这样做。利用自定义模板属性，能很容易简化模板的结构。为了演示这一功能，在代码清单12-25中，我编写了#stockValue助手指令。

**代码清单12-25** 在handlebars-jquery.js文件中增加#stockValue助手指令

```
(function ($) {

    Handlebars.registerHelper('gt', function (a, b, options) {
        return (a > b) ? options.fn(this) : options.inverse(this);
    });

    Handlebars.registerHelper("gtValAttr", function () {
        return "value='" + (this.stock > 0 ? "1" : "0") + "'";
    });

    Handlebars.registerHelper("inputElem", function (product, stock, options) {
        options.hash.name = product;
        options.hash.value = stock > 0 ? "1" : "0";
        options.hash.required = "required";
        return $("", options.hash)[0].outerHTML;
    });

    Handlebars.registerHelper("stockValue", function (options) {
        options.data.attributeValue = this.stock > 0 ? "1" : "0";
        return options.fn(this);
    });

    var compiled = {};
    $.fn.template = function (data) {
```

```

        var template = $.trim($(this).first().html());
        if (compiled[template] == undefined) {
            compiled[template] = Handlebars.compile(template);
        }
        return $(compiled[template](data));
    });
})(jQuery);

```

这是一个相当简单的助手指令，它仅仅在options.data对象里添加了一个attributeValue的属性，并把它值设置为我需要的input元素的value值。如代码清单12-26所示，在#stockValue指令定义的模板块内，可以使用@attributeValue来访问这个值。

代码清单12-26 在模板内访问特殊属性

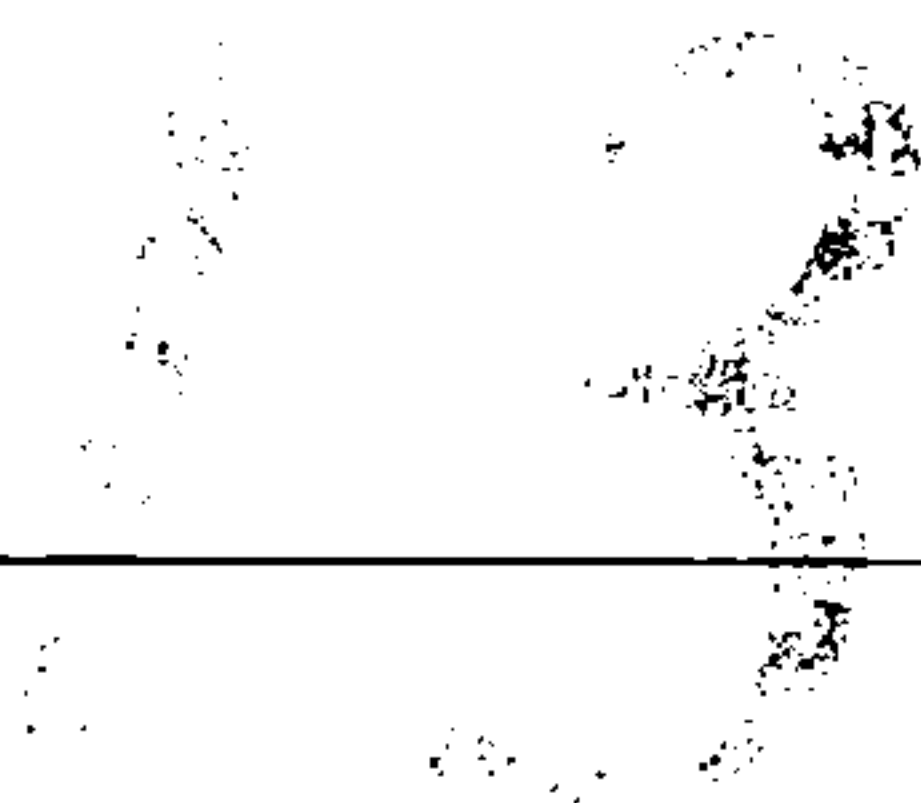
```

...
<script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
    <div class="dcell">
        
        <label for="{{product}}">{{name}}:</label>
        {{#stockValue}}
            <input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
                value="{{@attributeValue}}" required />
        {{/stockValue}}
    </div>
    {{/flowers}}
</script>
...

```

## 12.6 小结

本章的主题是Handlebars模板库。它提供了恰到好处的功能集合，从此在把JavaScript数据转化成HTML标签时，再也不必编写冗长繁杂的代码。我喜欢这个库的一个原因是：它相当灵活，在模板里定义多少逻辑，对数据做多少处理，在指令处理函数里隐藏多少细节，完全由你来定。下一章的主题是如何使用jQuery处理HTML表单，以及如何通过一个被广泛使用的jQuery插件来验证用户输入数据。



本章，我们一起研究jQuery提供的表单处理功能。其中，我们会重温表单有关的事件以及jQuery处理这些事件的方法。不过，本章最大的篇幅奉献给了一个卓越的表单验证插件，它负责在把用户输入的数据提交给服务之前验证用户输入是否有效。如果你曾经写过任何基于表单的Web应用，就会知道用户会往表单里填写各种各样的数据，因此验证用户输入是一个非常重要的过程。

本书的这一部分内容要用到服务器端脚本Node.js，因此我会首先介绍它。对于本章来说，这个脚本除了显示用户输入到表单中的数据之外，并不做更多事。不过在后面的章节中，我们将会更多地依赖Node.js。表13-1列出了本章概要。

表13-1 本章概要

问 题	解决方法	代码清单
如何安装Node.js服务器	使用本章列出的脚本（这些脚本也包括在随书的源代码内）	1、2
如何响应一个表单元素得到焦点和失去焦点事件	使用focus和blur方法	3
如何响应用户改变一个表单元素的值	使用change方法	4
如何响应及中断用户提交表单	使用submit方法	5、6
如何验证表单中的值	使用验证插件	7
如何配置验证插件	传入一个映射对象参数给validate方法	8
如何通过class定义和施加验证规则	使用addClassRules和addClass方法	9~12
如何添加直接适用于表单元素的规则	使用rules方法	13、14
如何添加适用于元素名的验证规则	在options对象中添加rules属性	15
如何添加适用于元素属性的验证规则	定义与验证检查相对应的属性	16
如何为基于元素名和属性的验证规则自定义提示信息	在选项对象中添加message属性，或者设置一个有自定义提示信息的映射对象	17、18
如何为直接适用于表单元素的规则自定义提示信息	将定义了message属性的选项对象用作rules方法的第二个参数	19
如何添加自定义验证方法	使用addMethod方法	20、21
如何格式化验证信息	使用选项对象的highlight、unhighlight、errorElement和errorClass属性	22~26
如何使用验证概要信息	使用errorContainer和errorLabelContainer属性	27
如何使用模板编写出错信息	使用\$.validator.format方法	28

## 13.1 准备 Node.js 服务器

本章使用Node.js接收和处理表单数据。我不想深入讲解Node.js的工作原理，不过Node.js使用JavaScript语言构建应用，这是本书选用它的重要原因。换言之，Node.js使我们能够利用客户端（JavaScript）编程技术进行服务器端编程。

---

**提示** 如果希望在自己的机器上再现本章的例子，可以参阅第1章了解如何获得Node.js。服务器端用到的formserver.js脚本连同本章的所有示例都可以从Apress.com下载。

---

代码清单13-1展示了本章将要用到的服务器端脚本，它位于formserver.js文件夹中。我把它当成一个“黑盒”，只讲解输入和输出部分。

代码清单13-1 Node.js脚本formserver.js

```
var http = require("http");
var querystring = require("querystring");

var port = 80;

http.createServer(function (req, res) {
    console.log("[200 OK] " + req.method + " to " + req.url);

    if (req.method == "POST") {
        var dataObj = new Object();
        var cType = req.headers["content-type"];
        var fullBody = "";

        if (cType && cType.indexOf("application/x-www-form-urlencoded") > -1) {
            req.on("data", function(chunk) { fullBody += chunk.toString();});
            req.on("end", function() {
                res.writeHead(200, "OK", {"Content-Type": "text/html"});
                res.write("<html><head><title>Post data</title></head><body>");
                res.write("<style>th, td {text-align:left; padding:5px; color:black}\n");
                res.write("th {background-color:grey; color:white; min-width:10em}\n");
                res.write("td {background-color:lightgrey}\n");
                res.write("caption {font-weight:bold}</style>");
                res.write("<table border='1'><caption>Form Data</caption>");
                res.write("<tr><th>Name</th><th>Value</th>");
                var dBody = querystring.parse(fullBody);
                for (var prop in dBody) {
                    res.write("<tr><td>" + prop + "</td><td>"
                        + dBody[prop] + "</td></tr>");
                }
                res.write("</table></body></html>");
                res.end();
            });
        }
    }
}).listen(port);
console.log("Ready on port " + port);
```

在命令行输入下面的命令运行这个脚本：

```
node.exe formserver.js
```

如果你使用的不是Windows操作系统，使用的命令就会与上面不同。具体细节请阅读Node.js的文档。为演示Node.js功能，我将使用代码清单13-2所示的示例文档，保存为example.html。

代码清单13-2 本章使用的示例文档

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js" type="text/javascript"></script>
  <script src="handlebars-jquery.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#each flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}: </label>
      <input name="{{product}}" value="0" required />
    </div>
    {{/each}}
  </script>
  <script type="text/javascript">
    $(document).ready(function() {

      var data = [
        { name: "Aster", product: "aster", stock: "10", price: "2.99"},
        { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
        { name: "Rose", product: "rose", stock: "2", price: "4.99"},
        { name: "Peony", product: "peony", stock: "0", price: "1.50"},
        { name: "Primula", product: "primula", stock: "1", price: "3.12"},
        { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}
      ];

      var templResult = $("#flowerTpl").template(data).filter("");
      templResult.slice(0, 3).appendTo("#row1");
      templResult.slice(3).appendTo("#row2");
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="oblock">
      <div class="dtable">
        <div id="row1" class="drow">
          </div>
        <div id="row2" class="drow">
          </div>
        </div>
      </div>
    </div>
  </form>
</body>
</html>
```



```

        </div>
      </div>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
</html>

```

这个例子使用数据模板生成产品的HTML标签（参阅第12章）。我还为form元素的act属性指定了一个值，这意味着表单数据将会被发送到以下URL地址：

```
http://node.jacquisflowershop.com/order
```

我使用两台不同种类的服务器。第一台服务器（[www.jacquisflowershop.com](http://www.jacquisflowershop.com)）是从开始到现在一直提供HTML内容的那个。它负责传输静态内容，如HTML文档、脚本文件和图片文件。它使用的Web服务器软件是微软的IIS。之所以使用IIS，这是因为我写的书大都与微软Web编程技术有关，而且这一台服务器早就安装好了，随时可以使用。你尽可以使用适合自己的服务器。

第二台服务器（[node.jacquisflowershop.com](http://node.jacquisflowershop.com)）运行着Node.js（运行着前面展示的formserver.js脚本），当我们在示例文档中提交表单时，数据就会被发送到这里。在本章中，对于服务器如何处理接收到的数据我们不需要了解很多。这里重点研究表单本身。来看图13-1，我已经在页面中输入了一些数据。

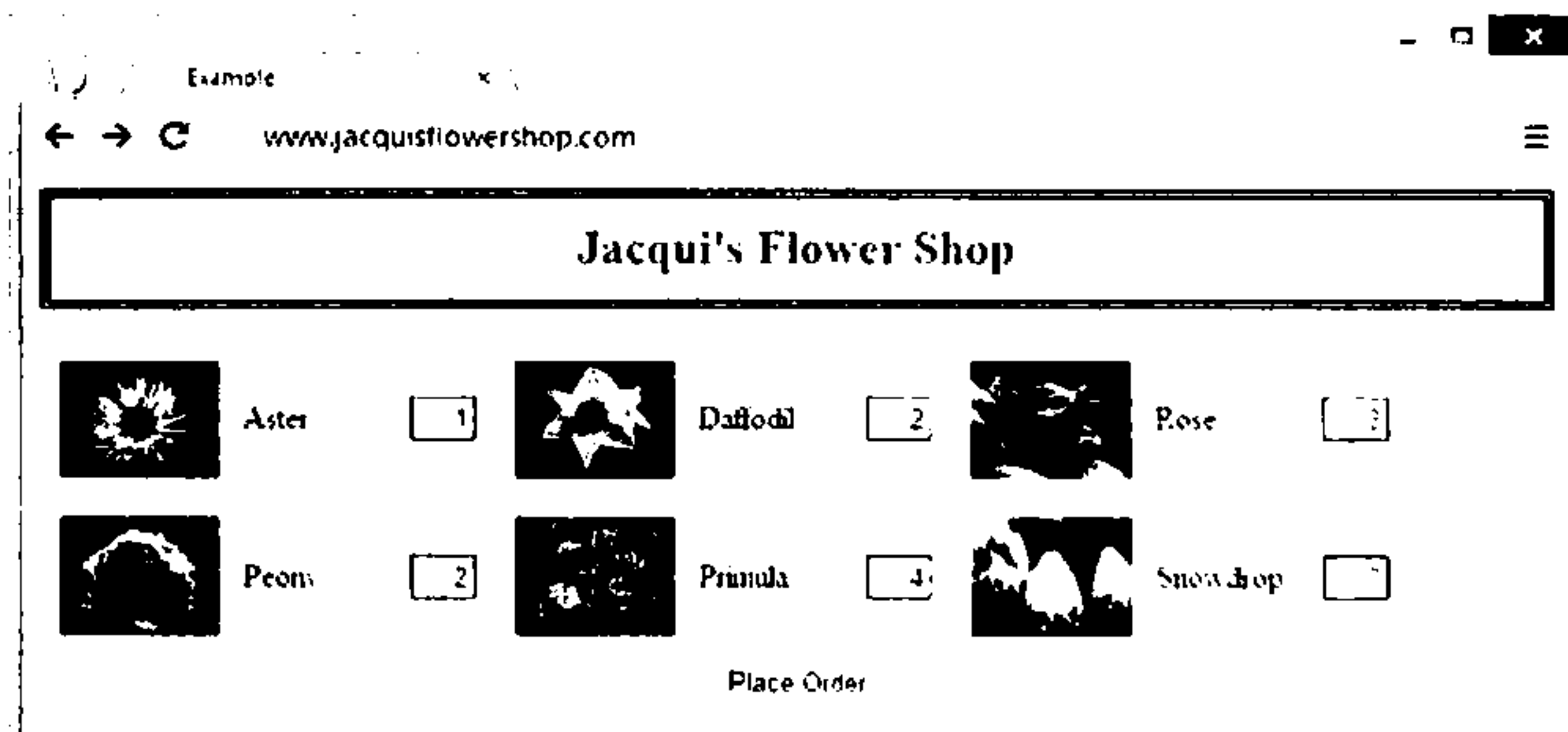
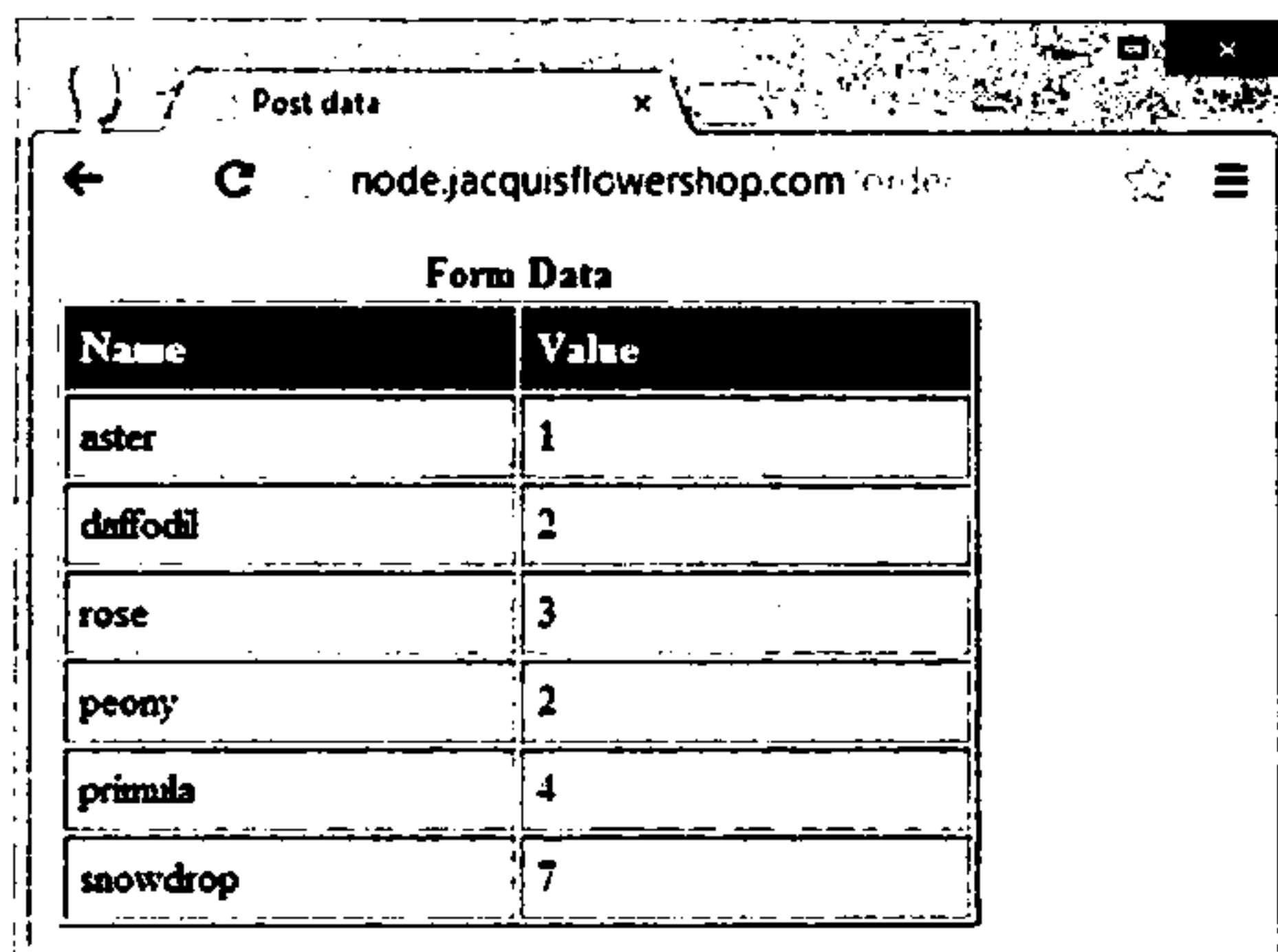


图13-1 输入数据到input元素

如果点击Place Order按钮，表单数据就会被提交到Node.js服务器，如图13-2所示。接着，浏览器就会接收到一个简单响应。

确实不是什么有趣的响应，不过我们现在只是需要一个发送数据的目的地，而且我确实也没有信马由缰，深入到服务器端开发世界的打算。





Name	Value
aster	1
daffodil	2
rose	3
peony	2
primula	4
snowdrop	7

图13-2 Node.js服务器的响应

## 13.2 回顾表单事件方法

jQuery提供了许多处理表单有关事件的方法。现在很有必要有针对性地复习一下这些方法。表13-2列出了这些方法以及它们对应的事件。

表13-2 jQuery提供的表单事件方法

方 法	事 件	描 述
<code>blur(function)</code>	Blur	当表单元素失去焦点时触发
<code>change(function)</code>	Change	当表单元素的值发生变化时触发
<code>focus(function)</code>	Focus	当表单元素得到焦点时触发
<code>select(function)</code>	Select	当用户选中表单元素中的文本时触发
<code>submit(function)</code>	Submit	当用户提交表单时触发

**提示** 别忘了，jQuery还定义了许多匹配表单元素的扩展选择器，详见第5章。

### 13.2.1 处理表单焦点

`blur`和`focus`方法用来响应焦点变化。这两个方法最大的用途是提醒用户焦点现在的位置（即哪个元素会接收键盘输入）。代码清单13-3给出了一个示例。

代码清单13-3 管理表单元素的焦点

```
...
<script type="text/javascript">
  $(document).ready(function() {
```

```

var data = { flowers: [
  { name: "Aster", product: "aster", stock: "10", price: "2.99"},
  { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
  { name: "Rose", product: "rose", stock: "2", price: "4.99"},
  { name: "Peony", product: "peony", stock: "0", price: "1.50"},
  { name: "Primula", product: "primula", stock: "1", price: "3.12"},
  { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}]
};

var templResult = $("#flowerTpl").template(data).filter("*");
templResult.slice(0, 3).appendTo("#row1");
templResult.slice(3).appendTo("#row2");

function handleFormFocus(e) {
  var borderVal = e.type == "focus" ? "medium solid green" : "";
  $(this).css("border", borderVal);
}
$("input").focus(handleFormFocus).blur(handleFormFocus);
});
</script>
...

```

在这个例子中，我选中所有的input元素并给它们的focus和blur事件都绑定了事件处理函数handleFormFocus。这个方法会在元素得到焦点时为元素添加一个绿色边框，而在失去焦点时移除此边框。这个脚本的具体效果见图13-3。

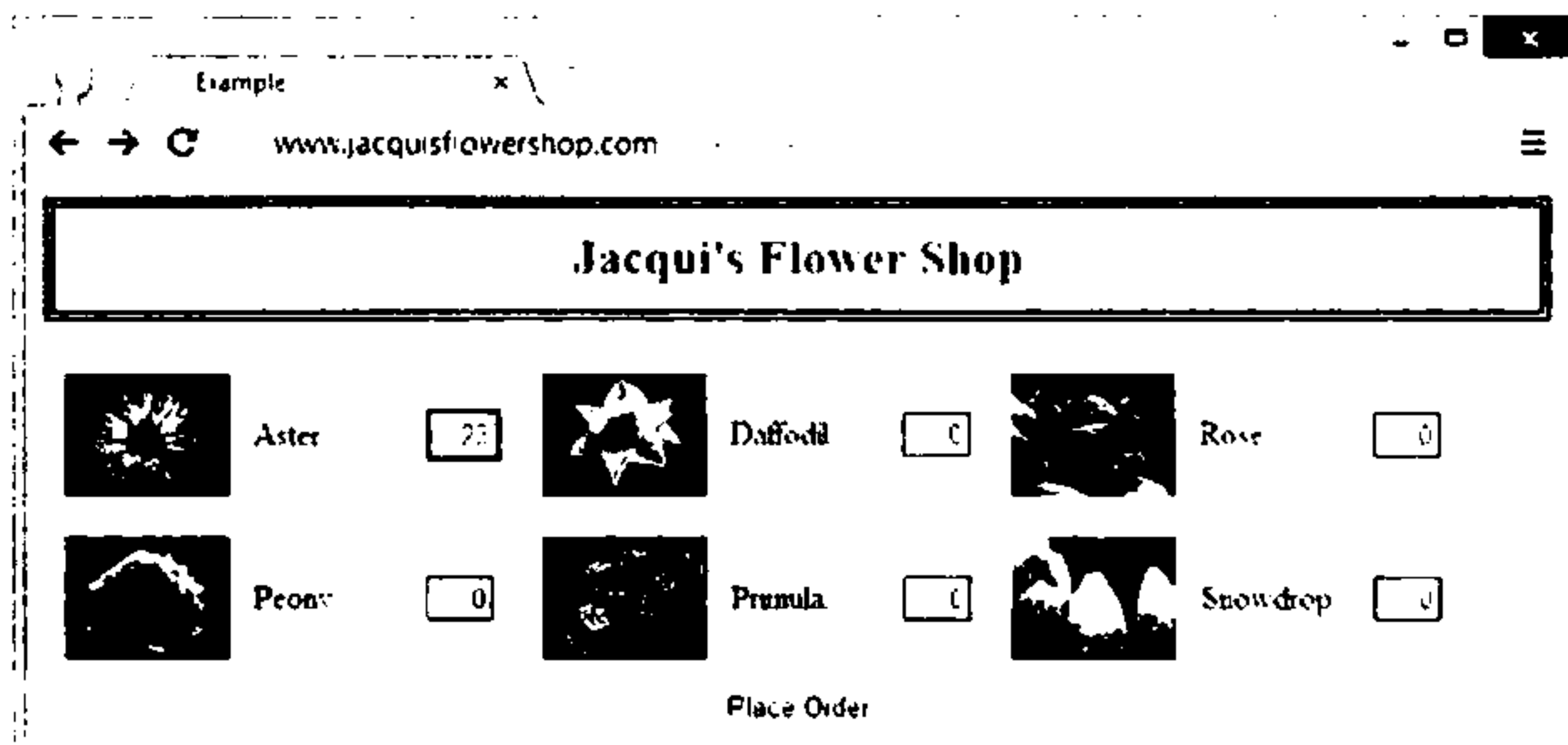


图13-3 突出显示得到焦点的元素

注意，我使用的是input选择器。也就是说，我使用标签来选择元素。jQuery还提供了另一个扩展选择器:input（关于扩展选择器详见第5章），这个扩展选择器匹配的元素范围更广，还会匹配那些有能力提交表单的按钮。这意味着，如果我们使用扩展选择器，绿色边框将不仅仅应用于input元素，还将应用于button元素。当button得到焦点时，你就会看到两种选择器之间的区别（参见图13-4）。应该选用哪种选择器？这纯属个人喜好问题，毕竟“萝卜白菜，各有所爱”。不过，知道两者的区别总是有用的。

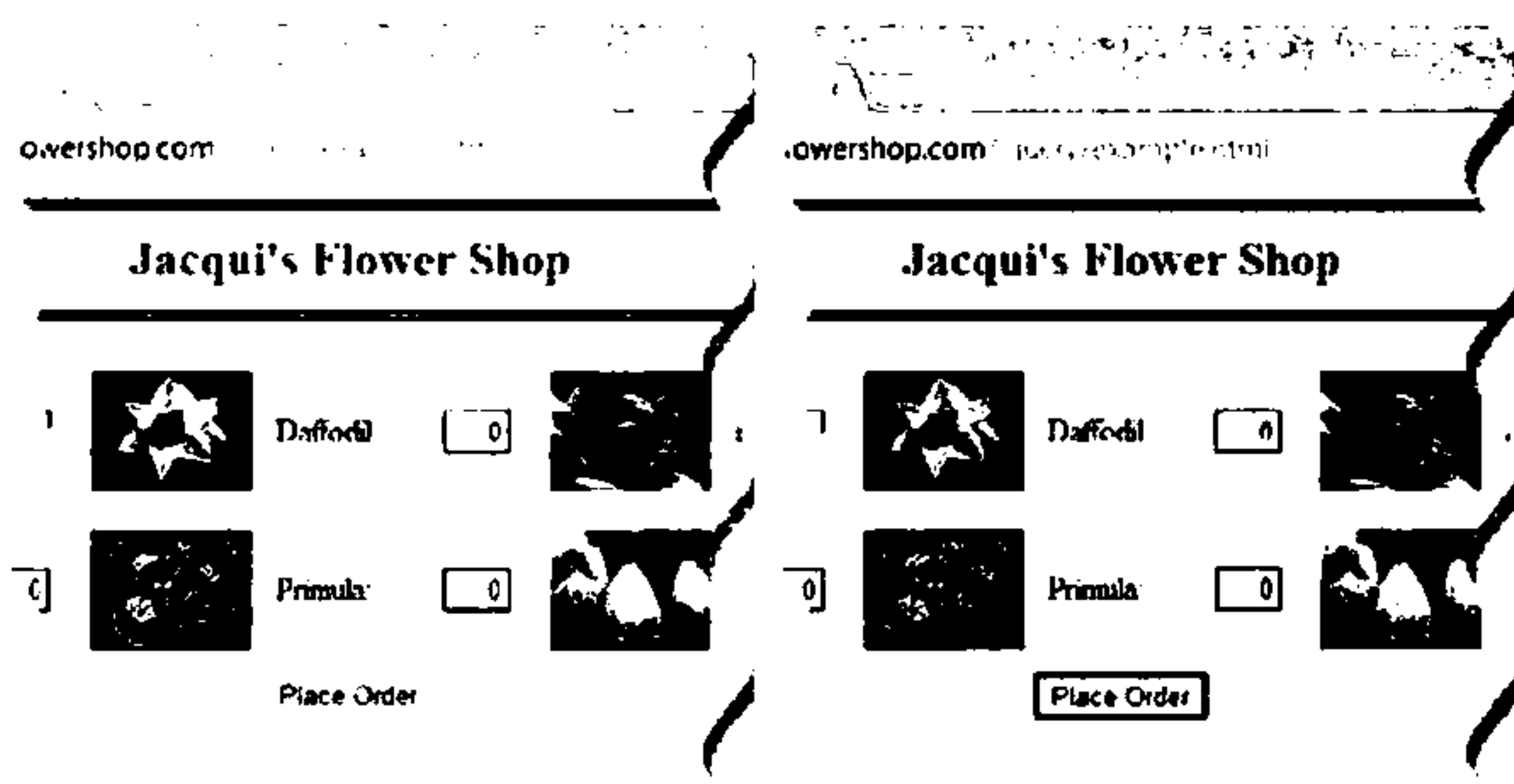


图13-4 input与:input选择器的区别

### 13.2.2 处理值的变化

当用户改变表单元素值时就会发生change事件。当我们需要根据表单中的值提供总和时，这个事件非常有用。代码清单13-4演示了如何利用这一事件在花店页面中跟踪订货总数。这也是我在本书第二部分末尾重构示例页面时所采用的方法。

#### 代码清单13-4 响应change事件

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = { flowers: [
            { name: "Aster", product: "aster", stock: "10", price: "2.99"},
            { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
            { name: "Rose", product: "rose", stock: "2", price: "4.99"},
            { name: "Peony", product: "peony", stock: "0", price: "1.50"},
            { name: "Primula", product: "primula", stock: "1", price: "3.12"},
            { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}]
        };

        var templResult = $("#flowerTpl").template(data).filter("*");
        templResult.slice(0, 3).appendTo("#row1");
        templResult.slice(3).appendTo("#row2");

        function handleFormFocus(e) {
            var borderVal = e.type == "focus" ? "medium solid green" : "";
            $(this).css("border", borderVal);
        }
        $("input").focus(handleFormFocus).blur(handleFormFocus);

        var total = $("#buttonDiv")
            .prepend("<div>Total Items: <span id=total>0</span></div>")
            .css({clear: "both", padding: "5px"});
    });
</script>
```

```

$("<div id=bbox />").appendTo("body").append(total).css("clear: left");

$("input").change(function(e) {
    var total = 0;
    $("input").each(function(index, elem) {
        total += Number($(elem).val());
    });
    $("#total").text(total);
});
});
</script>
...

```

在这个例子中，我在change事件处理函数中把所有input元素的值相加得出计算结果，并把结果显示在不久前添加的span元素中。

---

**提示** 注意，我使用val方法来获取input元素的值。

---

### 13.2.3 处理表单提交

当我们能够阻止浏览器表单机制的默认行为时，就能做很多“高级”的事情。代码清单13-5是一个简单的演示。

**代码清单13-5 拦截表单提交**

```

...
<script type="text/javascript">
    $(document).ready(function() {

        var data = { flowers: [
            { name: "Aster", product: "aster", stock: "10", price: "2.99"},
            { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
            { name: "Rose", product: "rose", stock: "2", price: "4.99"},
            { name: "Peony", product: "peony", stock: "0", price: "1.50"},
            { name: "Primula", product: "primula", stock: "1", price: "3.12"},
            { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}]
        };

        var templResult = $("#flowerTpl").template(data).filter("*");
        templResult.slice(0, 3).appendTo("#row1");
        templResult.slice(3).appendTo("#row2");

        $("form").submit(function (e) {
            if ($("#input").val() == 0) {
                e.preventDefault();
            }
        });
    });
</script>
...

```

在这个脚本中，我为submit事件注册了一个匿名函数。当用户点击Place Order按钮时，就会触发submit事件，如果第一个input元素值为0，我就调用preventDefault方法来终止表单提交的默认行为，也就是终止把数据提交到服务器的行为。如果不为0，表单就会被顺利提交。

**提示** 作为一种替代方法，我们可以让事件处理函数返回false，这样同样能够起到阻止事件行为的效果。

有两种方式可以让程序提交表单。我们可以不带任何参数调用jQuery的submit方法，也可以直接使用HTML5标准为表单元素定义的submit方法。代码清单13-6演示了这两种方法的使用。

代码清单13-6 明确地提交表单

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = { flowers: [
            { name: "Aster", product: "aster", stocklevel: "10", price: "2.99"},
            { name: "Daffodil", product: "daffodil", stocklevel: "12", price: "1.99"},
            { name: "Rose", product: "rose", stocklevel: "2", price: "4.99"},
            { name: "Peony", product: "peony", stocklevel: "0", price: "1.50"},
            { name: "Primula", product: "primula", stocklevel: "1", price: "3.12"},
            { name: "Snowdrop", product: "snowdrop", stocklevel: "15", price: "0.99"}
        ]};

        var templResult = $("#flowerTpl").tmpl(data).filter("*");
        templResult.slice(0, 3).appendTo("#row1");
        templResult.slice(3).appendTo("#row2");

        $("form").submit(function (e) {
            if ($("#input").val() == 0) {
                e.preventDefault();
            }
        });

        $("<button>jQuery Method</button>").appendTo("#buttonDiv").click(function (e) {
            $("form").submit();
            e.preventDefault();
        });

        $("<button>DOM API</button>").appendTo("#buttonDiv").click(function (e) {
            document.getElementsByTagName("form")[0].submit();
            e.preventDefault();
        });
    });
</script>
...
```

我在页面中添加了两个按钮。一个按钮调用jQuery submit方法，它会触发submit事件。在代码清单13-6中我为该事件设置了一个处理函数。也就是说，如果第一个input元素值为0，代码会放弃提交表单。

另一个按钮使用DOM API直接调用表单的submit方法，由于这个方法并不触发submit事件，它事实上会忽略掉我们定义的事件处理函数。也就是说，表单总是会被提交到服务器，无论第一个input元素的值是什么。

---

**提示** 我建议总是使用jQuery方法。当然，如果执意要使用DOM方法，你至少要明白它的结果。

---

## 13.3 验证表单数据

为什么要中断并阻止浏览器提交数据？最主要的原因是我们希望验证用户在表单中输入的值。在某些情况下，Web开发者应该知道用户会输入各种各样的数据，假定用户会输入有用且有效的数据不是明智之举。我们可能需要处理形形色色的用户输入。不过经验告诉我，只有很少的几个原因导致用户输入我们不需要的东西。

第一个原因是用户不知道我们希望得到什么数据。比如说，你希望用户输入他们信用卡上的名字，用户却输入了信用卡号码。

第二个原因是用户根本不想提供有意义的信息，他们只想尽快结束这个表单。于是他们胡乱填写信息，目的就是走到下一步。如果你发现许多新用户的电子邮件地址都是a@a.com，这就是原因了。

第三个原因是用户无法提供你所要求的信息。比如你要求用户输入他们居住在英国哪个州。（英国根本没有州。用户看着你的表单，可是没法提供这一信息。）

最后一个原因是用户不小心填错了，最常见的就是拼写错误。比如，某些用户就是打字很快但不准确，甚至经常把自己的姓都敲错（比如丢掉一个字母e）。

对于拼写错误我们无计可施。不过我们处理另外3个常见原因的方式决定了产品体验是流畅好用，还是使用户困惑甚至恼怒。

在这里我并不想大谈特谈表单设计，不过还是要指出，解决这类问题的最佳方式在于关注用户的目标。出现问题时，请尽量从用户的角度看待问题和解决方法。用户既不了解我们系统的结构，也不了解我们的业务逻辑，他们只希望解决问题。只要我们始终关注用户要解决的问题，并避免因为他们未能提供我们需要的数据而不必要的惩罚用户，所有人都会满意。

尽管我们完全能够利用jQuery提供的各种工具构建表单验证系统，但是我建议大家采用另一个解决方案。它就是Validation——最受欢迎的jQuery插件之一。从名字你就能够猜到，它专门处理表单验证问题。

---

**警告** 本章讨论的问题是客户端验证。客户端验证是服务器端验证（由服务器检查接收到的数据是否有效）的补充，而非代替品。客户端验证能有效改善用户体验：不需要一遍遍地把数据提交到服务器就能发现和提示用户纠正输入数据的错误。服务器端验证的目标却是确保不良数据不会破坏系统。两种验证方式都是必需的：客户端验证是很容易就能绕过的，仅有客户端验证无法对系统提供有效保护。

---

你可以直接从<http://jqueryvalidation.org>下载Validation插件，也可以使用本书源代码下载包中的Validation插件。代码清单13-7演示了这个插件的用法。（撰写本书时版本为1.1.1。）

#### 代码清单13-7 Validation插件的用法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js" type="text/javascript"></script>
  <script src="handlebars-jquery.js" type="text/javascript"></script>
  <script src="jquery.validate.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    .errorMsg {color: red}
    .invalidElem {border: medium solid red}
  </style>
  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#each flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}: </label>
      <input name="{{product}}" value="0" required />
    </div>
    {{/each}}
  </script>
  <script type="text/javascript">
    $(document).ready(function() {

      var data = {flowers: [
        { name: "Aster", product: "aster", stock: "10", price: "2.99"},
        { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
        { name: "Rose", product: "rose", stock: "2", price: "4.99"},
        { name: "Peony", product: "peony", stock: "0", price: "1.50"},
        { name: "Primula", product: "primula", stock: "1", price: "3.12"},
        { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}
      ]};

      var templResult = $("#flowerTpl").template(data).filter("");
      templResult.slice(0, 3).appendTo("#row1");
      templResult.slice(3).appendTo("#row2");

      $("form").validate({
        highlight: function(element, errorClass) {
          $(element).add($(element).parent()).addClass("invalidElem");
        },
        unhighlight: function(element, errorClass) {
          $(element).add($(element).parent()).removeClass("invalidElem");
        },
        errorElement: "div",
        errorClass: "errorMsg"
      });
```



```

        $.validator.addClassRules({
            flowerValidation: {
                min: 0
            }
        });

        $("input").addClass("flowerValidation").change(function(e) {
            $("form").validate().element($(e.target));
        });
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                </div>
                <div id="row2" class="drow">
                </div>
            </div>
        </div>
        <div id="buttonDiv"><button type="submit">Place Order</button></div>
    </form>
</body>
</html>

```

从网站上下载的插件是一个zip文件，需要先解压缩，然后再把jquery.validate.js文件从解压目录复制到example.html文件所在目录。

---

**提示** Validation插件有许多配置选项。本章重点讲解那些最常用且最常见的情况。如果我讲的这些选项无法解决你的问题，请阅读和插件一并下载下来的插件文档，了解该插件的其他选项。

---



---

**注意** HTML5内建了一些基本的表单验证功能，这很好。不过，HTML提供的这些功能都相当简单，而且各浏览器对这些功能的支持情况也不一致。在HTML5对表单验证支持有限并且实现不一致的前提下，我坚持推荐使用jQuery处理表单验证任务。

---

### 1. 导入插件文件

我们首先使用下面的代码把验证插件代码包含到页面中：

```

...
<script src="jquery-2.0.2.js" type="text/javascript"></script>
<script src="handlebars.js" type="text/javascript"></script>
<script src="handlebars-jquery.js" type="text/javascript"></script>
<script src="jquery.validate.js" type="text/javascript"></script>
...

```



**提示** 我在这里使用的是该插件的开发版本，这个插件当然也有压缩版本。而且由于这个插件使用如此之广，有些CDN也对此文件提供了托管服务。

## 2. 配置插件

接下来是配置插件，对表单元素进行验证。我们先选中需要为其验证数据的表单元素，然后调用 `validate` 方法。如代码清单13-8所示，`validate` 方法的参数是一个包含着各种设置选项的映射对象。

代码清单13-8 配置插件

```
...
$("form").validate({
  highlight: function(element, errorClass) {
    $(element).add($(element).parent()).addClass("invalidElem");
  },
  unhighlight: function(element, errorClass) {
    $(element).add($(element).parent()).removeClass("invalidElem");
  },
  errorElement: "div",
  errorClass: "errorMsg"
});
...
```

我指定了4个选项（`highlight`、`unhighlight`、`errorElement`和`errorClass`）。稍后，我们还会回顾并详细解释这些选项的含义。

## 3. 定义验证规则

验证插件非常灵活，它支持我们快速并且轻松定义检查输入数据有效性的规则。有多种方法可以把规则与表单元素相关联。我喜欢使用 `class`。我们先定义一些对应某个 `class` 的规则，在验证表单时这些规则将被应用于所有拥有这个 `class` 的表单元素。在代码清单13-9所示的例子中，我只定义了一条规则。

代码清单13-9 定义验证规则

```
...
$.validator.addClassRules({
  flowerValidation: {
    min: 0
  }
});
...
```

在本例中，我针对表单中拥有 `flowerValidation` 类的元素创建了一条规则，这条规则要求它的值大于或者等于0。我使用单词 `min` 来表达这条规则。验证插件预先定义了一些使用方便的规则，`min` 只是其中的一条而已。本章后面我会详细讲解所有的预定义规则。

## 4. 实施验证

上一步为那些拥有特定 `class` 的元素指定了验证规则。我们可以使用这种方式为表单中不同类型的元素定制验证方法。对于本例来说，表单中所有的元素都应同等对待，因此我使用 `jQuery` 选中所有的 `input` 元素，并给它们加上 `flowerValidation` 类（参见代码清单13-10）。

代码清单13-10 为input元素添加验证所需的class

```

...
$("input").addClass("flowerValidation").change(function(e) {
    $("form").validate().element($(e.target));
});
...

```

我还为这些input元素绑定了change事件处理函数，当它们的值发生变化时立即实施验证。这就保证了用户能够及时得到输入正确或错误的反馈。这里验证插件的效果见图13-5。为了得到这张图，我在输入框里输入了值-1，然后点击Place Order按钮。

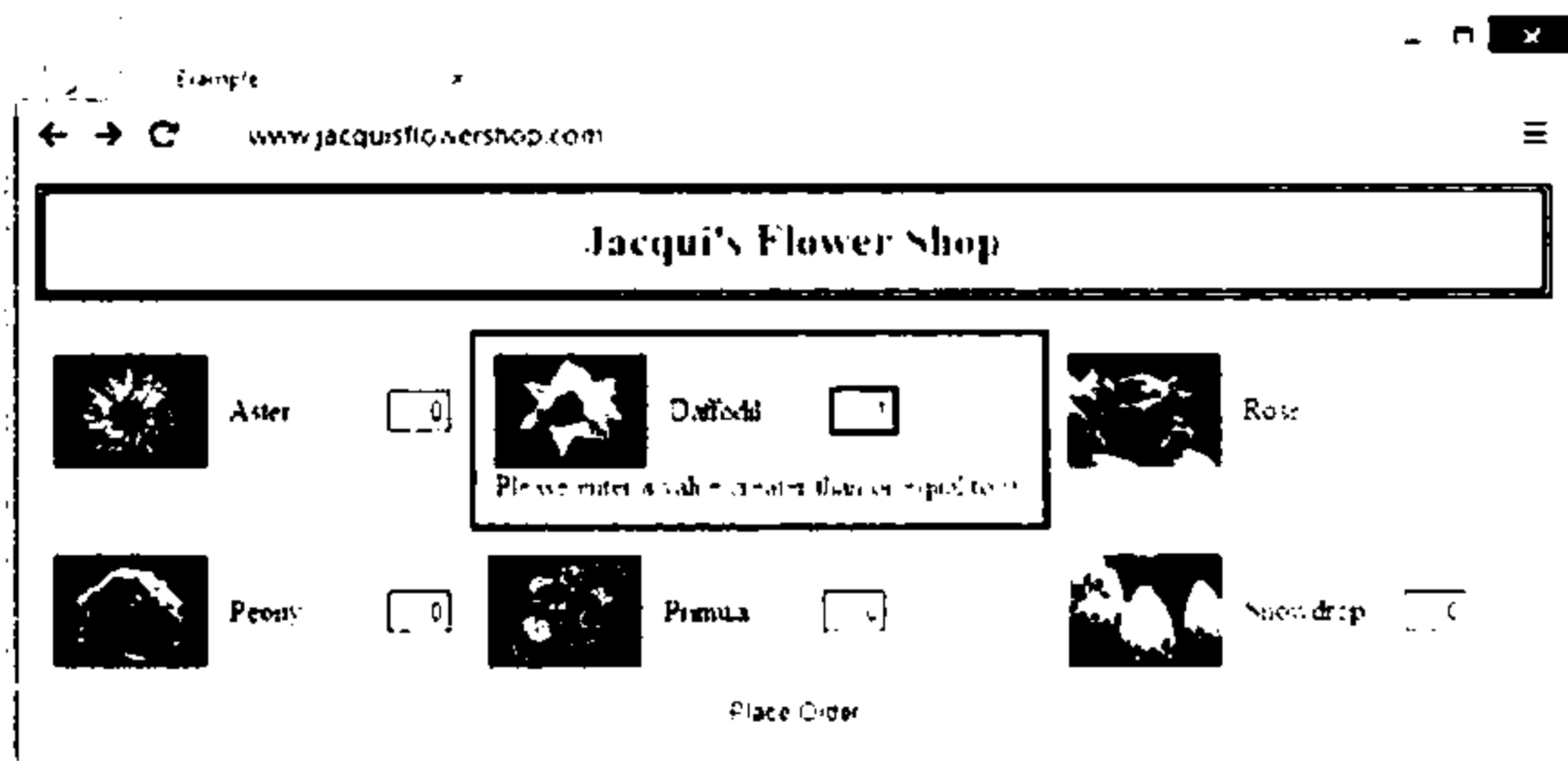


图13-5 使用验证插件

**提示** 图中显示给用户的提示文本是验证插件生成的。我会在本章后面的部分演示定制提示信息的方法。

验证插件显示了一条出错信息，用户将无法提交表单，直到错误输入得以修正。这条出错信息能引导用户解决问题。（插件默认显示的信息多少有点过于通用，如图13-5所示，本章稍后将介绍如何修改提示信息文本。）

### 13.3.1 插件内建的检查功能

Validation插件能够对表单数据进行种种检查。在前面的例子中我们使用了min检查，min检查确保元素的值大于或等于指定的数值。表13-3列出了插件支持的各种检查。

表13-3 验证插件提供的检查功能

检 查	描 述
creditcard: true	值必须是有效的信用卡号码
date: true	值必须是合法的JavaScript日期
digits: true	值必须由且仅由数字组成
email: true	值必须是一个有效的电子邮件地址

(续)

检 查	描 述
max: maxVal	值必须是数字, 并且小于或等于maxVal
maxlength: length	值的长度必须小于等于length个字符
min: minVal	值必须是数字, 并且必须大于或等于minVal
minlength: length;	值至少要有length个字符
number: true	值必须是一个十进制数
range: [minVal, maxVal]	值必须是数字, 大于等于minVal, 并且小于等于maxVal
rangelength: [minLen, maxLen]	值的字符串长度必须大于等于minLen, 并小于等于maxLen
required: true;	必须填写
url: true	必须填写URL

我们可以把多条规则汇总为一条规则, 这样就能够以一种简炼清晰的方式实施复杂的验证。

**提示** 随Validation插件一起发布的, 还有一个additional-methods.js文件。这个文件定义了一些扩展检查, 包括美国及英国的电话号码、IPv4及IPv6地址, 以及一些扩展日期格式、电子邮件格式、URL格式。

我们有多种方法可以对表单元元素实施这些检查, 以下各节逐一讲解这些方法。

**注意** Validation插件也支持远程验证, 即用户输入一个数据, 由远程服务器来检查数据是否有效。当我们需要检查当前数据的可用性时, 由于用户很可能输入不安全或者不可用的数据(比如检查一个用户名是否可用), 这种检查就非常有用。由于远程验证依赖于第14章和第15章中才会讲到的技术, 我将在第16章演示远程验证。

### 1. 通过class实施验证

正如我在前面所说的, 通过class实施验证是我最喜欢使用的技术, 下面的例子就使用了这种方法。不过并没有一次只能实施一项检查的限制。如代码清单13-11所示, 我们完全可以在一条规则中对多个不同部分的值实施多项检查。

代码清单13-11 在一条规则中实施多项检查

```
...
<script type="text/javascript">
    $(document).ready(function () {

        var data = { flowers: [
            { name: "Aster", product: "aster", stock: "10", price: "2.99"},
            { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
            { name: "Rose", product: "rose", stock: "2", price: "4.99"},
            { name: "Peony", product: "peony", stock: "0", price: "1.50"},
            { name: "Primula", product: "primula", stock: "1", price: "3.12"},
            { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}]
```

```

    });

    var templResult = $("#flowerTpl").template(data).filter("");
    templResult.slice(0, 3).appendTo("#row1");
    templResult.slice(3).appendTo("#row2");

    $("#form").validate({
        highlight: function(element, errorClass) {
            $(element).add($(element).parent()).addClass("invalidElem");
        },
        unhighlight: function(element, errorClass) {
            $(element).add($(element).parent()).removeClass("invalidElem");
        },
        errorElement: "div",
        errorClass: "errorMsg"
    });

    $.validator.addClassRules({
        flowerValidation: {
            required: true,
            digits: true,
            min: 0,
            max: 100
        }
    });

    $("#input").addClass("flowerValidation").change(function (e) {
        $("#form").validate().element($(e.target));
    });
});
</script>
...

```

在这个例子中，我综合使用了required、digits、min和max检查，以确保用户输入的值都是数字并且大小位于0至100之间。

注意，我是使用addClassRules方法关联这条规则的。这个方法参数是由目标class与相应检查集合构成的一个对象。如代码所示，我们使用\$.validator.addClassRules方式调用addClassRules方法。

如图13-6所示，验证插件会针对每一个检查逐一验证需要验证的表单元素，这意味着用户很可能会看到针对不同问题的多种错误提示信息。

我输入了几个会导致检测失败的值。值得注意的是，检查将依照规则定义的顺序执行。如果我们看到Rose产品有一条错误信息“Please enter only digits”（数字检查失败），我们就会明白它没有通过“值全部由数字组成”这项检查。如果我们重新排列规则的顺序，就会得到不同的错误信息。代码清单13-12展示了修订顺序之后的代码。

#### 代码清单13-12 修改规则的检查顺序

```

...
$.validator.addClassRules({
    flowerValidation: {
        required: true,
        min: 0,

```

```

    max: 100,
    digits: true
  }
});
...

```

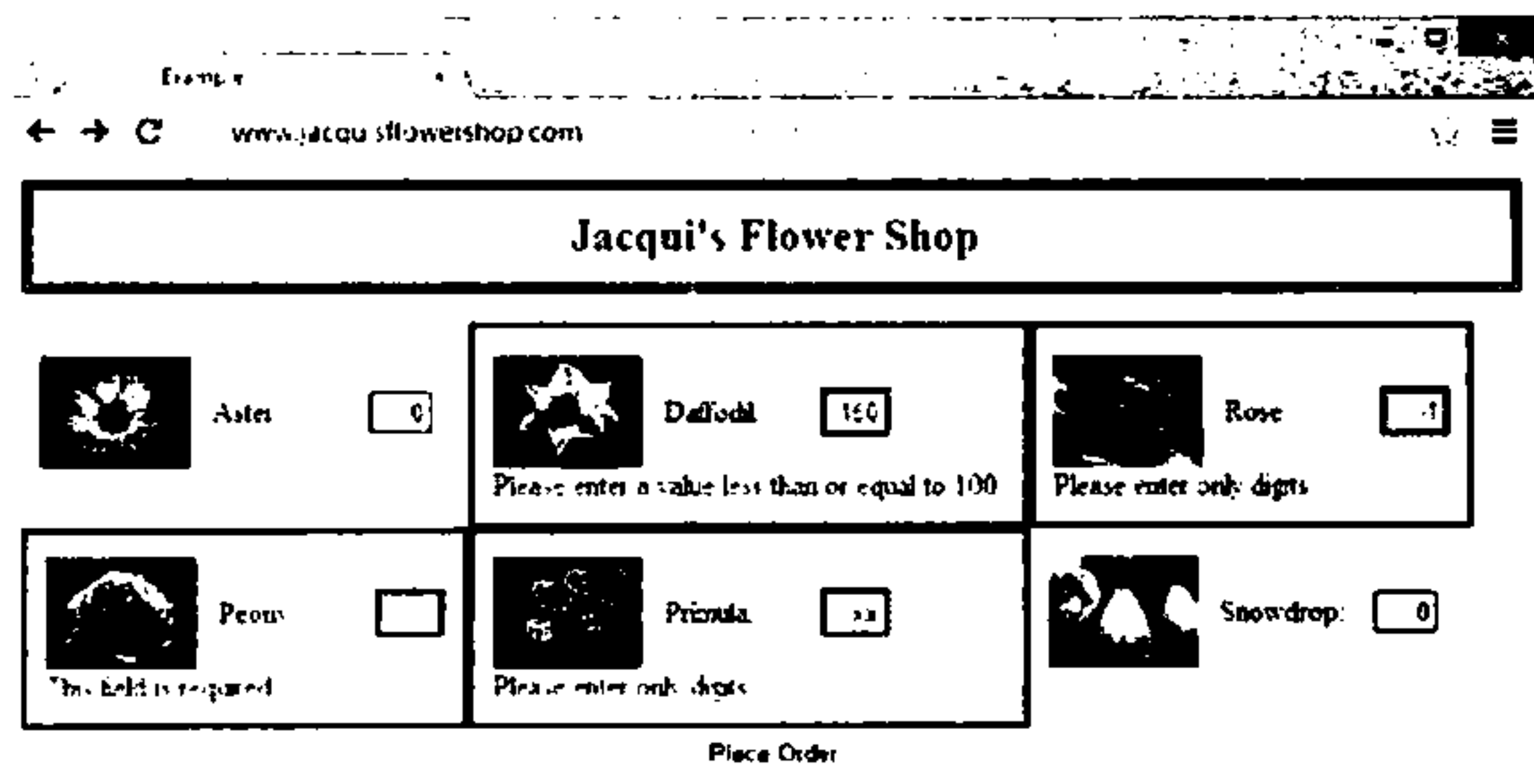


图13-6 对表单元素实施多项检查

在这个例子里，我把digits检查放到了规则的末尾。现在，如果输入-1到一个表单项中，如图13-7所示，我们就会看到错误信息提示min检查失败。

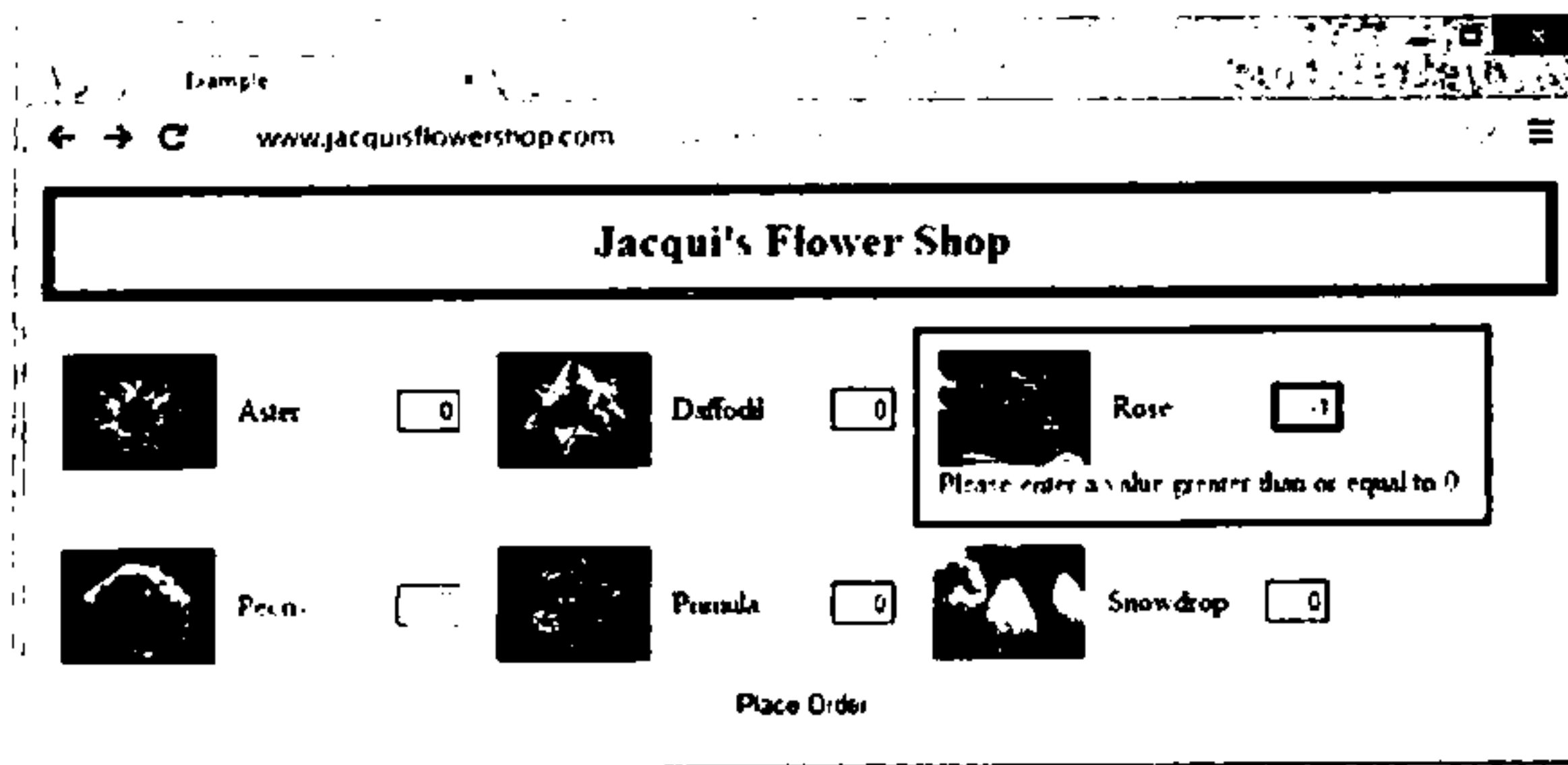


图13-7 修改验证规则的检查顺序

## 2. 直接使用规则检查具体的元素

如代码清单13-13所示，我们可以直接把规则应用于单一具体元素。

### 代码清单13-13 把验证规则直接施加到结果集中的具体元素

```

...
<script type="text/javascript">
  $(document).ready(function () {

    var data = { flowers: [
      { name: "Aster", product: "aster", stock: "10", price: "2.99"},

```

```

        { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
        { name: "Rose", product: "rose", stock: "2", price: "4.99"},
        { name: "Peony", product: "peony", stock: "0", price: "1.50"},
        { name: "Primula", product: "primula", stock: "1", price: "3.12"},
        { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}]
    });

    var templResult = $("#flowerTpl").template(data).filter("*");
    templResult.slice(0, 3).appendTo("#row1");
    templResult.slice(3).appendTo("#row2");

    $("form").validate({
        highlight: function(element, errorClass) {
            $(element).add($(element).parent()).addClass("invalidElem");
        },
        unhighlight: function(element, errorClass) {
            $(element).add($(element).parent()).removeClass("invalidElem");
        },
        errorElement: "div",
        errorClass: "errorMsg"
    });

    $.validator.addClassRules({
        flowerValidation: {
            required: true,
            min: 0,
            max: 100,
            digits: true
        }
    });

    $("#row1 input").each(function(index, elem) {
        $(elem).rules("add", {
            min: 10,
            max: 20
        })
    });

    $("input").addClass("flowerValidation").change(function (e) {
        $("form").validate().element($(e.target));
    });
});
</script>
...

```

注意，这次是在一个jQuery对象上调用rules方法，第一个参数是字符串add，而第二个参数是一个由检查项及其参数组成的映射对象。rules方法仅对结果集中的第一个元素有效，因此如果希望对所有元素进行检查，就必须使用each方法。在本例中，我选中了#row1元素内的所有input元素，然后实施了最大值和最小值检查，以确保用户所输入的值在10~20之间。

---

**提示** 我们也可以把第一个参数由add改为remove，从而删除规则。

---

实施检查时，施加到具体元素的规则优先于通过class施加的规则。在本例中，这意味着顶行元素将检查元素的值是否在10~20之间，而其他input元素则检查元素的值是否在0~100之间。这段脚本的效果见图13-8。

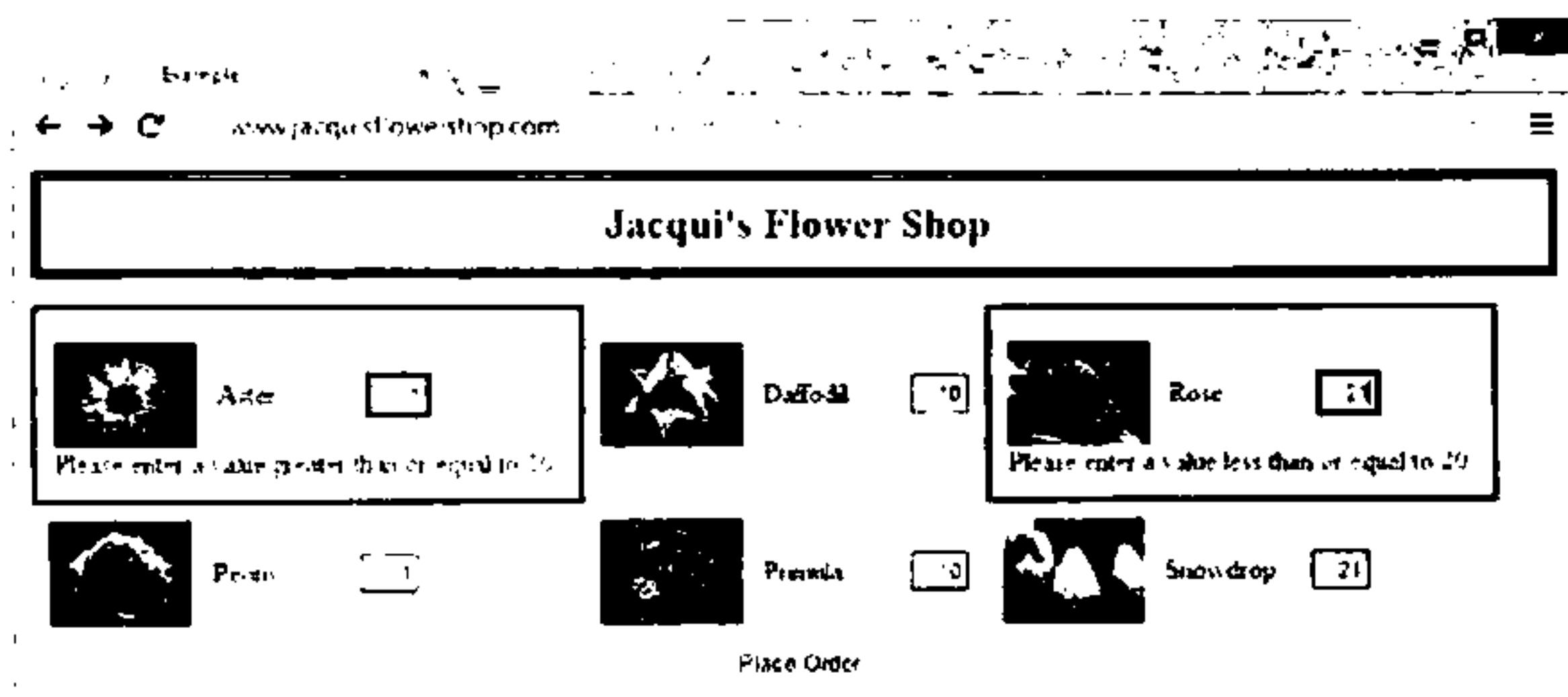


图13-8 直接把规则施加到具体元素

由于我们是单独处理每个元素，因此可以定制检查项。代码清单13-14给出了一个例子。

#### 代码清单13-14 为元素定制检查项

```
...
<script type="text/javascript">
    $(document).ready(function () {

        var data = { flowers: [
            { name: "Aster", product: "aster", stock: "10", price: "2.99"},
            { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
            { name: "Rose", product: "rose", stock: "2", price: "4.99"},
            { name: "Peony", product: "peony", stock: "0", price: "1.50"},
            { name: "Primula", product: "primula", stock: "1", price: "3.12"},
            { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}]
        };

        var templResult = $("#flowerTpl").template(data).filter("*");
        templResult.slice(0, 3).appendTo("#row1");
        templResult.slice(3).appendTo("#row2");

        $("form").validate({
            highlight: function (element, errorClass) {
                $(element).add($(element).parent()).addClass("invalidElem");
            },
            unhighlight: function (element, errorClass) {
                $(element).add($(element).parent()).removeClass("invalidElem");
            },
            errorElement: "div",
            errorClass: "errorMsg"
        });

        $("input").each(function (index, elem) {
            var rules = {
```

```

        required: true,
        min: 0,
        max: data.flowers[index].stock,
        digits: true
    }
    if (Number(data.flowers[index].price) > 3.00) {
        rules.max--;
    }
    $(elem).rules("add", rules);
});

$("input").addClass("flowerValidation").change(function (e) {
    $("form").validate().element($(e.target));
});
});
</script>
...

```

在这个例子中，我利用提供给模板生成页面元素所使用的数据对象定制max的值。如果stock属性的值大于\$3，我就向下调整max的值。如果你使用的数据对象与这个例子相似，还能够实施更实用的检查。这个变化产生的效果见图13-9。

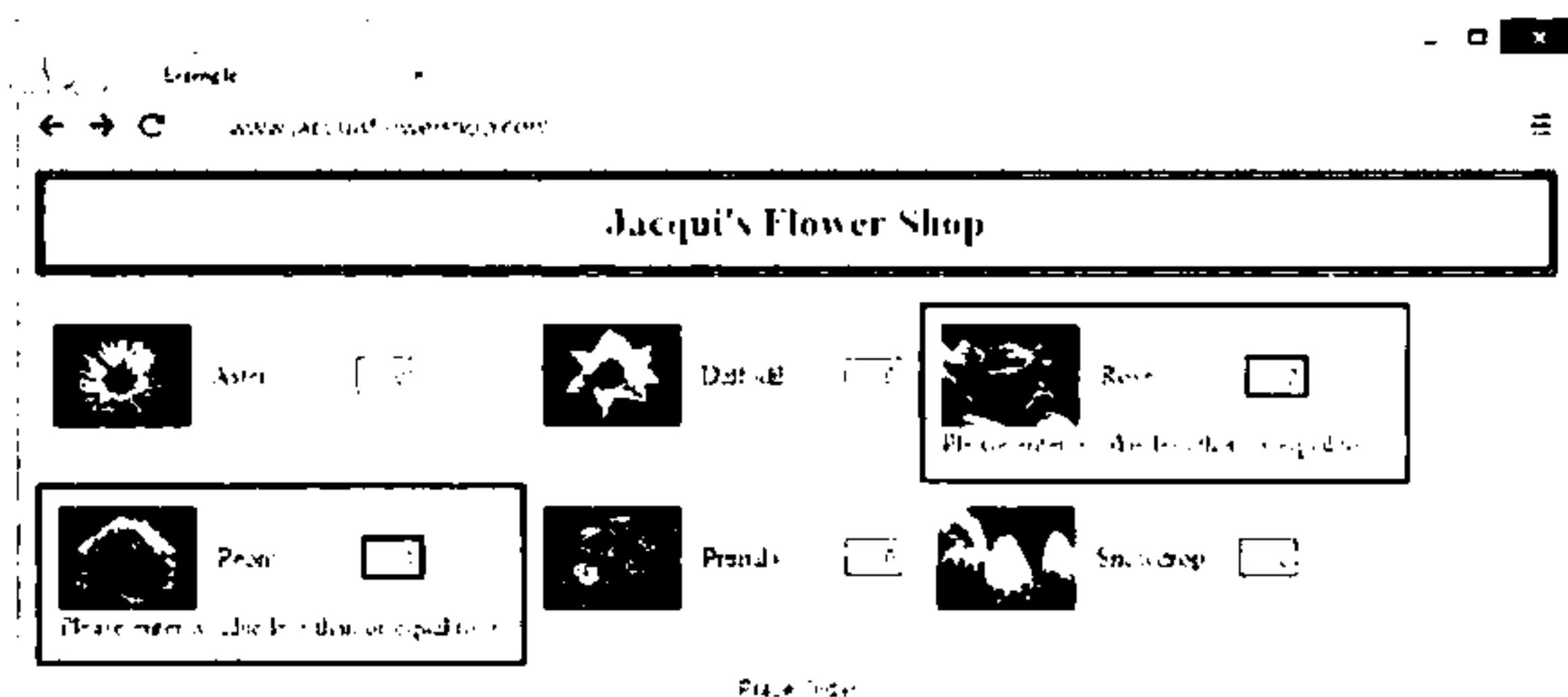


图13-9 基于数据对象为检查设定不同的边界

### 3. 根据元素名实施检查

我们也可以通过元素名实施检查。HTML标准并没有要求元素名必须唯一，不过表单中常常使用唯一的名字来分组表单元素。在花店页面示例中，每个input元素名都不相同，分别对应着一种花卉产品。不管怎样，我们总能够根据name属性值设定规则，而且这些规则会被施加到name属性相同的所有元素上。代码清单13-15演示了这种用法。

#### 代码清单13-15 基于元素名实施检查

```

...
<script type="text/javascript">
    $(document).ready(function () {

        var data = { flowers: [
            { name: "Aster", product: "aster", stock : "10", price: "2.99"},

```



```

    { name: "Daffodil", product: "daffodil", stock : "12", price: "1.99"},
    { name: "Rose", product: "rose", stock : "2", price: "4.99"},
    { name: "Peony", product: "peony", stock : "0", price: "1.50"},
    { name: "Primula", product: "primula", stock : "1", price: "3.12"},
    { name: "Snowdrop", product: "snowdrop", stock : "15", price: "0.99"}}
  ];

  var templResult = $("#flowerTpl").template(data).filter("*");
  templResult.slice(0, 3).appendTo("#row1");
  templResult.slice(3).appendTo("#row2");

  var rulesList = new Object();
  for (var i = 0; i < data.flowers.length; i++) {
    rulesList[data.flowers[i].product] = {
      min: 0,
      max: Number(data.flowers[i].stock),
    }
  }

  $("form").validate({
    highlight: function(element, errorClass) {
      $(element).add($(element).parent()).addClass("invalidElem");
    },
    unhighlight: function(element, errorClass) {
      $(element).add($(element).parent()).removeClass("invalidElem");
    },
    errorElement: "div",
    errorClass: "errorMsg",
    rules: rulesList
  });

  $("input").change(function (e) {
    $("form").validate().element($(e.target));
  });
});
</script>
...

```

在配置验证插件时，我们把根据元素名生成的规则作为配置对象的rules属性传递给validate方法。注意，我使用数据对象生成了一系列规则，并使用数据对象的product属性生成input元素的名字。同时还要注意，为保证验证规则准确，我在这里必须使用Number把字符串类型的数字转换为数值类型。

**提示** 在我自己的项目中，我不会使用这种方式（配置验证规则）。我更愿意直接处理页面中的元素。不过，如果你有一个数据对象，并且希望在表单元素添加到页面之前就设置好表单验证规则，例子里的方法就很合适。

#### 4. 根据元素的属性实施检查

检查属性的最后一种方法是使用元素属性。验证插件逐个查看表单元素，检查是否定义有与内建检查名字相符的属性。因此，如果一个元素定义了required属性，我们就认为它需要检查。具体示例见代码清单13-16。

代码清单13-16 使用元素属性实施验证

```

...
<script id="flowerTpl" type="text/x-handlebars-template">
  {{#each flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}: </label>
      <input name="{{product}}" value="0" required min="0" max="{{stock}}"/>
    </div>
  {{/each}}
</script>
<script type="text/javascript">
  $(document).ready(function () {

    var data = { flowers: [
      { name: "Aster", product: "aster", stock: "10", price: "2.99"},
      { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
      { name: "Rose", product: "rose", stock: "2", price: "4.99"},
      { name: "Peony", product: "peony", stock: "0", price: "1.50"},
      { name: "Primula", product: "primula", stock: "1", price: "3.12"},
      { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}
    ]};

    var templResult = $("#flowerTpl").template(data).filter("*");
    templResult.slice(0, 3).appendTo("#row1");
    templResult.slice(3).appendTo("#row2");

    $("form").validate({
      highlight: function (element, errorClass) {
        $(element).add($(element).parent()).addClass("invalidElem");
      },
      unhighlight: function (element, errorClass) {
        $(element).add($(element).parent()).removeClass("invalidElem");
      },
      errorElement: "div",
      errorClass: "errorMsg",
    });

    $("input").change(function (e) {
      $("form").validate().element($(e.target));
    });
  });
</script>
...

```

如果与数据模板结合使用，我喜欢这一技术。不过我发现，如果把它用于静态定义的元素，就不得不把同样的属性一遍又一遍重复应用于元素，导致页面混乱不堪。

### 13.3.2 指定错误提示信息

验证插件为所有内建检查都定义了默认的错误提示信息，不过这些信息都很笼统，对于用户来说不是十分有用。打个比方，如果我们设置max检查，不允许用户输入超过10的值，而用户却输入了20，

那么他就会看到下面的错误提示：

---

```
Please enter a value less than or equal to 10
```

---

这个信息与我们的规则一致，但它并没有告诉用户为什么会有这样一个限制。好在我们能够改变这些信息，提供更有用的背景信息，甚至根据需要定制信息。改变提示信息的方法依赖于创建验证规则的方式。如果我们使用class应用验证规则，就不能修改错误提示信息。不过在下面几节中，我提供了使用其他技术定义错误提示信息的办法。

#### 1. 为属性及名字检查指定错误提示信息

如果我们使用名字或者属性定义检查规则，就能通过为传递给validate方法的options对象添加messages属性的方法改变提示信息。具体示例见代码清单13-17。

代码清单13-17 使用options对象的messages属性

```
...
<script type="text/javascript">
    $(document).ready(function () {

        var data = { flowers: [
            { name: "Aster", product: "aster", stock: "10", price: "2.99"},
            { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
            { name: "Rose", product: "rose", stock: "2", price: "4.99"},
            { name: "Peony", product: "peony", stock: "0", price: "1.50"},
            { name: "Primula", product: "primula", stock: "1", price: "3.12"},
            { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}]
        };

        var templResult = $("#flowerTpl").template(data).filter("*");
        templResult.slice(0, 3).appendTo("#row1");
        templResult.slice(3).appendTo("#row2");

        $("form").validate({
            highlight: function (element, errorClass) {
                $(element).add($(element).parent()).addClass("invalidElem");
            },
            unhighlight: function (element, errorClass) {
                $(element).add($(element).parent()).removeClass("invalidElem");
            },
            errorElement: "div",
            errorClass: "errorMsg",
            messages: {
                rose: { max: "We don't have that many roses in stock!" },
                primula: { max: "We don't have that many primulas in stock!" }
            }
        });

        $("input").change(function (e) {
            $("form").validate().element($(e.target));
        });
    });
</script>
...
```

在这个例子中，验证插件利用input元素的min和max属性施加验证，在JavaScript代码里，我们可以清楚地了解用来设置messages属性的对象结构。

在messages对象内，我使用元素的name属性定义属性，并把属性的值设为一个映射对象，用这个对象定义相应的检查以及出错时的提示信息。在这个例子中，我修改了rose和primula花卉的max检查的提示信息。示例的具体效果见图13-10，图中显示的出错信息正是我们代码中定制的提示信息。

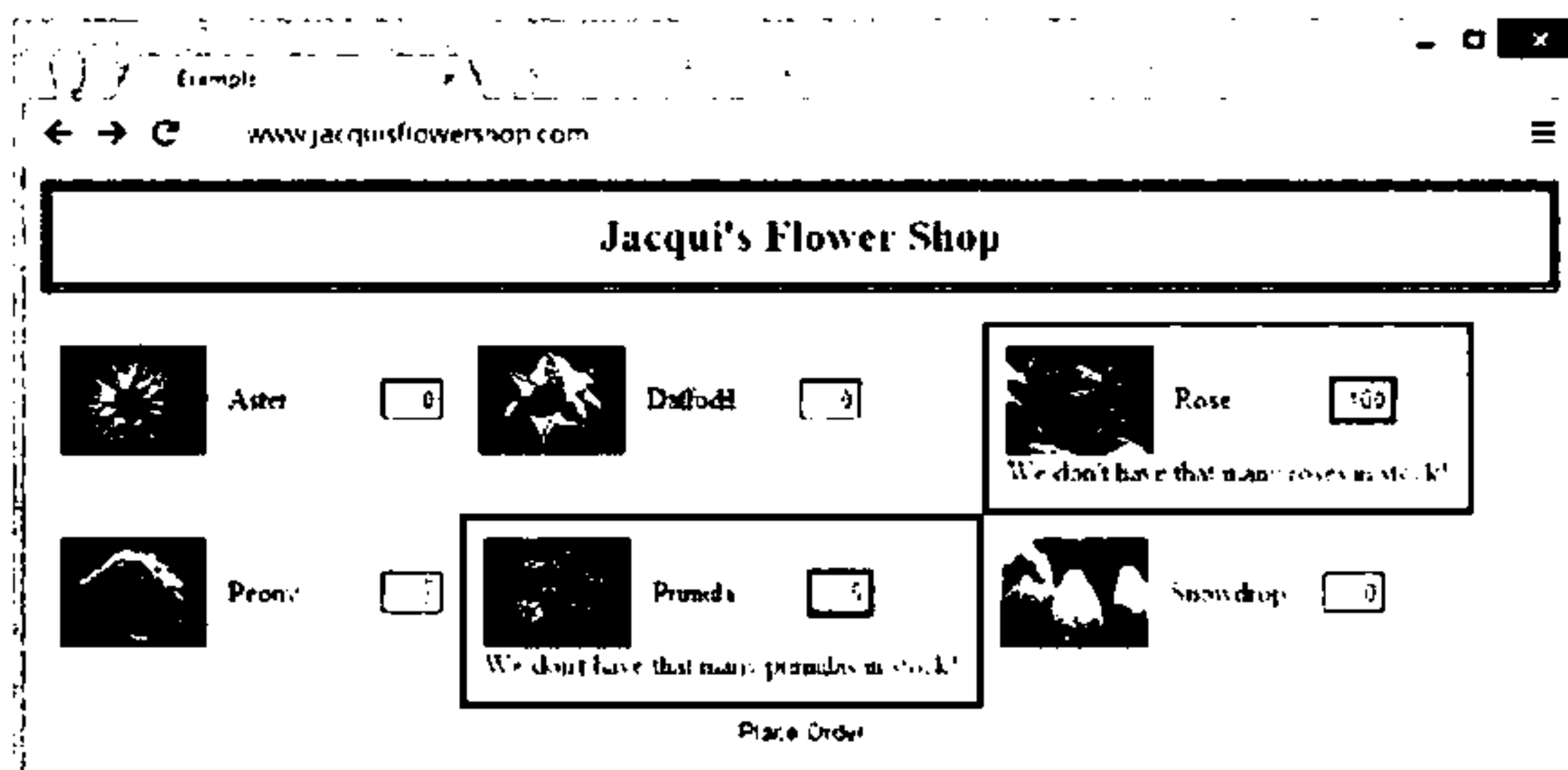


图13-10 利用options对象改变提示信息

这种设置定制提示信息的语法重复度很高，因此我更愿意写程序生成包含提示信息的对象（参见代码清单13-18）。

#### 代码清单13-18 通过一段程序定义出错信息

```
...
<script type="text/javascript">
  $(document).ready(function () {

    var data = { flowers: [
      { name: "Aster", product: "aster", stock: "10", price: "2.99"},
      { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
      { name: "Rose", product: "rose", stock: "2", price: "4.99"},
      { name: "Peony", product: "peony", stock: "0", price: "1.50"},
      { name: "Primula", product: "primula", stock: "1", price: "3.12"},
      { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}
    ]};

    var templResult = $("#flowerTpl").template(data).filter("*");
    templResult.slice(0, 3).appendTo("#row1");
    templResult.slice(3).appendTo("#row2");

    var customMessages = new Object();
    for (var i = 0; i < data.flowers.length; i++) {
      customMessages[data.flowers[i].product] = {
        max: "We only have " + data.flowers[i].stock + " in stock"
      }
    }
  })
}
```

```

$("form").validate({
    highlight: function(element, errorClass) {
        $(element).add($(element).parent()).addClass("invalidElem");
    },
    unhighlight: function(element, errorClass) {
        $(element).add($(element).parent()).removeClass("invalidElem");
    },
    errorElement: "div",
    errorClass: "errorMsg",
    messages: customMessages
});

$("input").change(function (e) {
    $("form").validate().element($(e.target));
});
});
</script>
...

```

在这个例子中,我利用数据对象中stocklevel属性值为用户提供了更有用的提示信息,如图13-11所示。

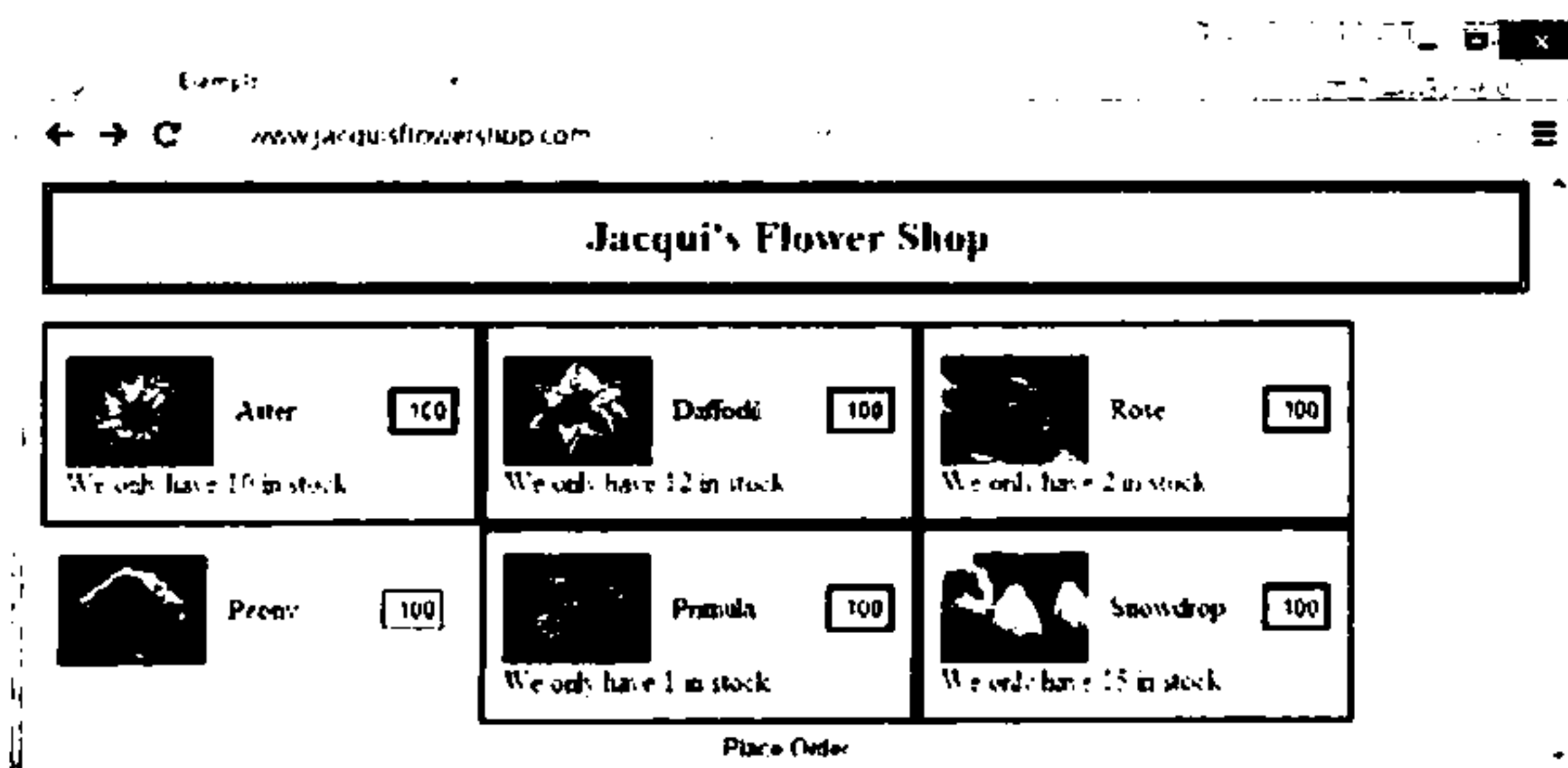


图13-11 使用程序生成自定义验证提示信息

## 2. 为基于具体元素的规则设置自定义提示信息

当我们为单个元素应用规则时,也可以传递一个定义有检查项及提示信息的messages对象作为rules对象的第二个参数。代码清单13-19演示了这种做法。

代码清单13-19 为基于具体元素的规则设置自定义提示信息

```

...
<script type="text/javascript">
    $(document).ready(function () {

        var data = [
            { name: "Aster", product: "aster", stock: "10", price: "2.99"},
            { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
            { name: "Rose", product: "rose", stock: "2", price: "4.99"},
            { name: "Peony", product: "peony", stock: "0", price: "1.50"},
            { name: "Primula", product: "primula", stock: "1", price: "3.12"},
            { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}]
    });

```

```

    });

    var templResult = $("#flowerTpl").template(data).filter("");
    templResult.slice(0, 3).appendTo("#row1");
    templResult.slice(3).appendTo("#row2");

    $("#form").validate({
        highlight: function(element, errorClass) {
            $(element).add($(element).parent()).addClass("invalidElem");
        },
        unhighlight: function(element, errorClass) {
            $(element).add($(element).parent()).removeClass("invalidElem");
        },
        errorElement: "div",
        errorClass: "errorMsg",
    });

    $("#input").change(function (e) {
        $("#form").validate().element($(e.target));
    }).each(function (index, elem) {
        $(elem).rules("add", {
            messages: {
                max: "We only have " + data.flowers[index].stock + " in stock"
            }
        });
    });
});
</script>
...

```

我又一次利用stock属性定义了提示信息。简单起见，我假定这些input元素的显示顺序与它们的数据对象中的顺序一致。这些自定义信息的效果见图13-12。

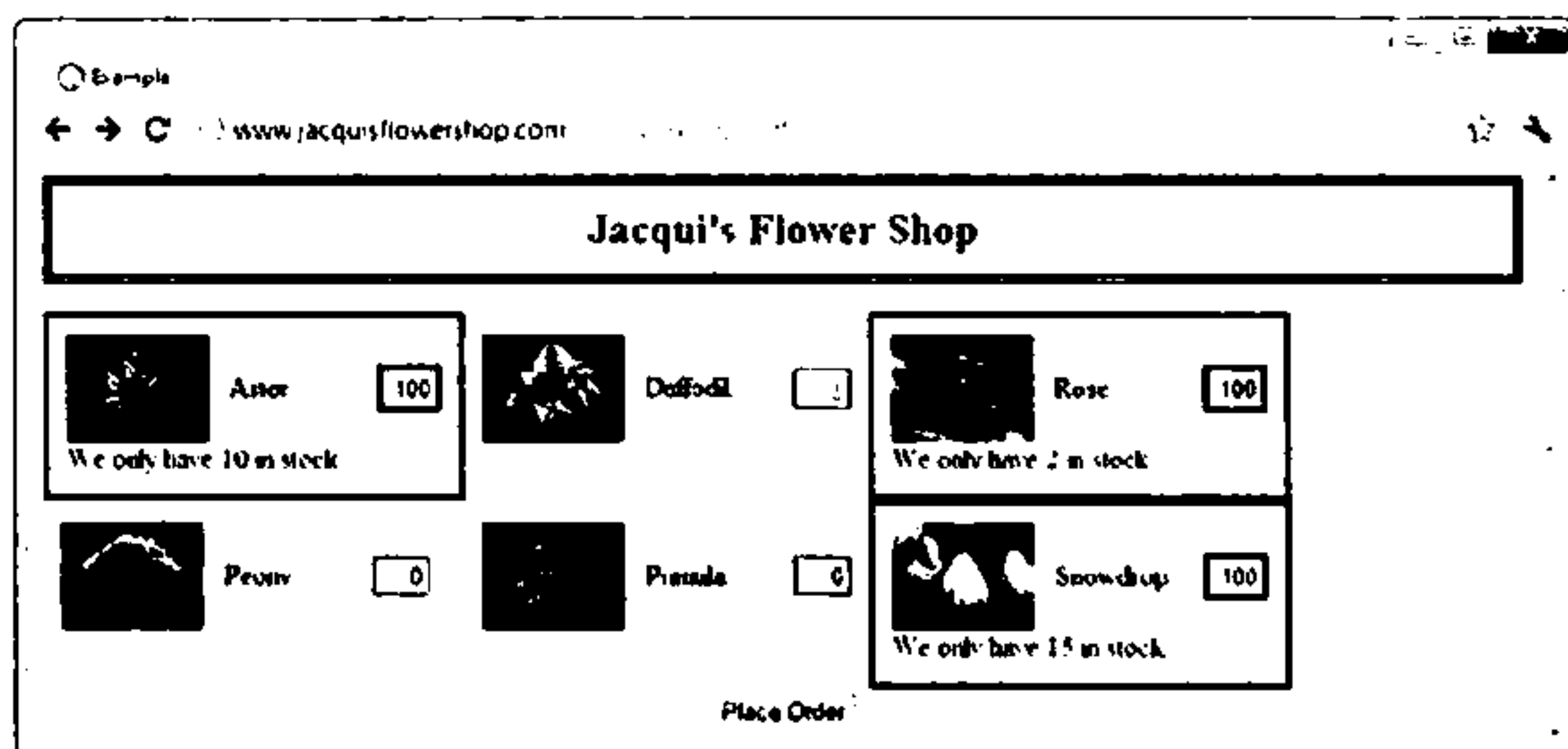


图13-12 利用数据对象生成定制提示信息

**提示** 在这个例子里，我用脚本仅仅定制了验证失败时的提示信息。min和max验证规则仍然是通过模板中的input元素施加的（见代码清单13-17）。

### 13.3.3 自定义检查

如果预定义检查不能满足要求，我们也可以创建自定义检查。这是一个简单的过程，我们借此能够让验证规则与应用更紧密地联系在一起。具体示例见代码清单13-20。

代码清单13-20. 生成自定义检查

```
...
<script id="flowerTpl" type="text/x-handlebars-template">
  {{#each flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}: </label>
      <input name="{{product}}" value="0" required min="0" max="{{stock}}"/>
    </div>
  {{/each}}
</script>
<script type="text/javascript">
  $(document).ready(function () {

    var data = {
      flowers: [
        { name: "Aster", product: "aster", stock: "10", price: "2.99"},
        { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
        { name: "Rose", product: "rose", stock: "2", price: "4.99"},
        { name: "Peony", product: "peony", stock: "0", price: "1.50"},
        { name: "Primula", product: "primula", stock: "1", price: "3.12"},
        { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}
      ]
    };

    var templResult = $("#flowerTpl").template(data).filter("*");
    templResult.slice(0, 3).appendTo("#row1");
    templResult.slice(3).appendTo("#row2");

    $("form").validate({
      highlight: function(element, errorClass) {
        $(element).add($(element).parent()).addClass("invalidElem");
      },
      unhighlight: function(element, errorClass) {
        $(element).add($(element).parent()).removeClass("invalidElem");
      },
      errorElement: "div",
      errorClass: "errorMsg",
    });

    $.validator.addMethod("stock", function (value, elem, args) {
      return Number(value) < Number(args);
    }, "We don't have that many in stock");

    $("input").each(function (index, elem) {
      $(elem).rules("add", {
        stock: data.flowers[index].stock
      })
    })
  })
}
```

```

        }).change(function (e) {
            $("form").validate().element($(e.target));
        });
    });
</script>
...

```

我从模板中删除了input元素的min和max属性，改用JavaScript代码实施自定义验证。（可以混用自定义验证与内建验证，这个例子中的自定义验证只是简单地重复实现了max验证。）

我们使用addMethod方法实现自定义检查，它使用\$.validator.addMethod这种方式进行调用。这个方法第一个参数是自定义检查的名字，第二个函数是实施具体验证的代码，第三个参数是检查失败后的提示信息。在这个例子中，我定义了一个名为stock的检查。

### 1. 定义一个验证函数

自定义验证方法的第一个参数是用户输入的值，第二个参数是对应这个表单元素的HTMLElement对象，剩下的参数在执行检查时指定，比如：

```

...
$(elem).rules("add", {
    min: 0,
    stock: data.flowers[index].stock
});
...

```

应用这条规则时，我把验证参数中的stock属性设置为input元素对应花卉对象的stock属性值。它将成为第三个参数传递给自定义检查函数：

```

...
function(value, elem, args) {
    return Number(value) <= Number(args);
}
...

```

用户输入的值（value）及参数（args）都是字符串类型，这意味着必须使用Number类型转换才能保证JavaScript按数值比较。对这个验证函数来说，如果是可以接受的值，检查函数返回true，否则就返回false。在我的这个函数中，若用户输入的值小于或者等于参数，这个值就是有效的。

### 2. 定义验证提示信息

我们有两种方式指定提示信息。第一种，也就是我在前面示例中用过的，是以字符串的形式提供。另一种是使用回调函数，它能帮助我们提供更确切的提示信息。具体示例见代码清单13-21。

**代码清单13-21 使用回调函数生成自定义提示信息**

```

...
<script type="text/javascript">
    $(document).ready(function () {

        var data = {
            flowers: [
                { name: "Aster", product: "aster", stock: "10", price: "2.99"},
                { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
                { name: "Rose", product: "rose", stock: "2", price: "4.99"},
                { name: "Peony", product: "peony", stock: "0", price: "1.50"},
            ]
        };
    });
</script>

```



```

        { name: "Primula", product: "primula", stock: "1", price: "3.12"},
        { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}]
    };

    var templResult = $("#flowerTpl").template(data).filter("*");
    templResult.slice(0, 3).appendTo("#row1");
    templResult.slice(3).appendTo("#row2");

    $("form").validate({
        highlight: function (element, errorClass) {
            $(element).add($(element).parent()).addClass("invalidElem");
        },
        unhighlight: function (element, errorClass) {
            $(element).add($(element).parent()).removeClass("invalidElem");
        },
        errorElement: "div",
        errorClass: "errorMsg",
    });

    $.validator.addMethod("stock", function (value, elem, args) {
        return Number(value) <= Number(args);
    }, function(args) {
        return "We only have " + args + " in stock"
    });

    $("input").each(function (index, elem) {
        $(elem).rules("add", {
            stock: data.flowers[index].stock
        })
    }).change(function (e) {
        $("form").validate().element($(e.target));
    });
});
</script>
...

```

这个回调函数的参数就是设置规则时使用的参数,在这个例子中指stock属性值。具体效果见图13-13。

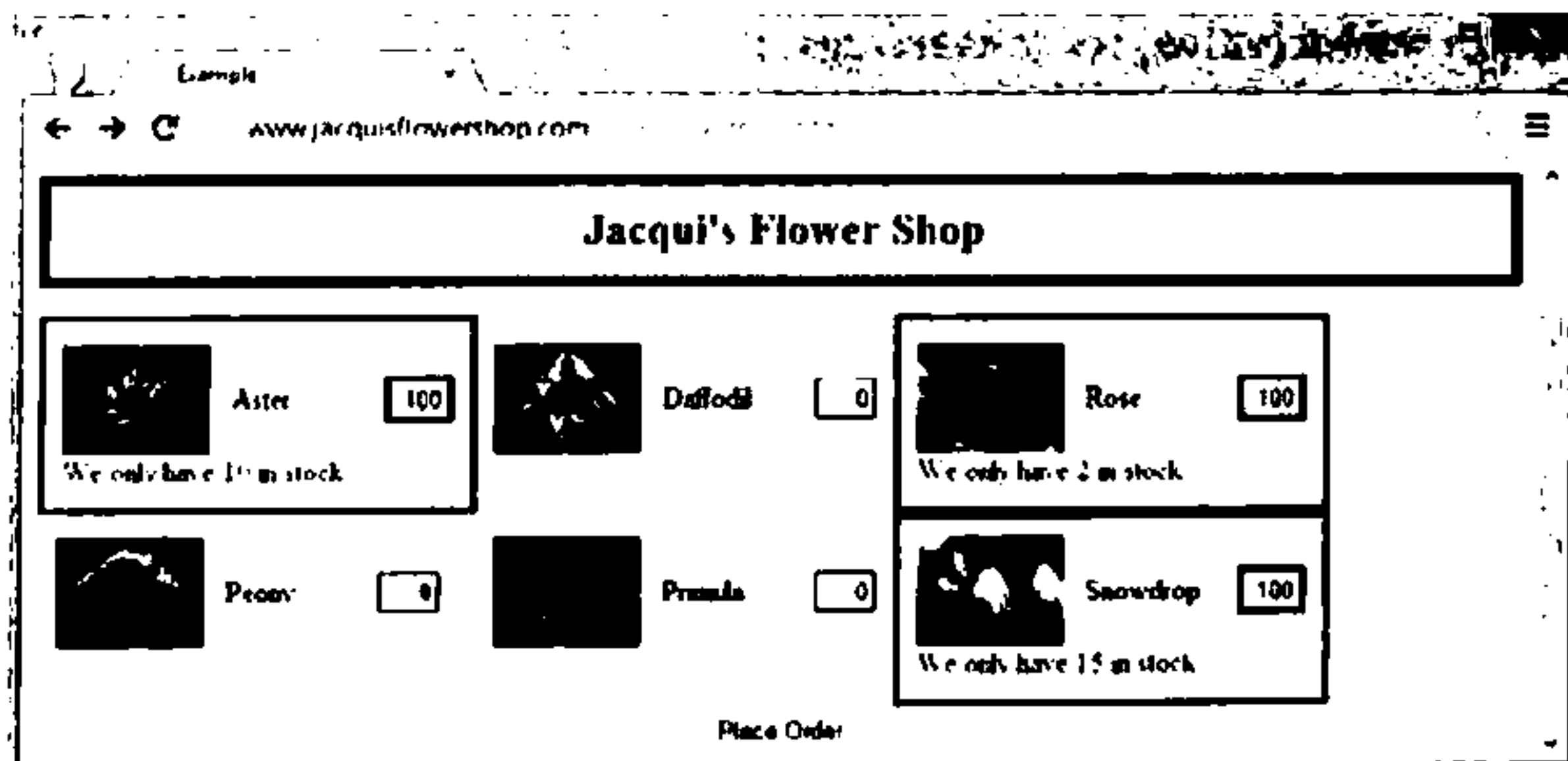


图13-13 使用回调函数自定义提示信息

### 13.3.4 定制错误提示的显示格式

我认为支持配置多种错误信息显示方式是验证插件的最佳功能之一。在本章前面的这些例子里，我们始终信赖代码清单13-22中加粗显示的代码。

代码清单13-22 处理错误提示信息格式的配置选项

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var data = {
            flowers: [
                { name: "Aster", product: "aster", stock: "10", price: "2.99"},
                { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
                { name: "Rose", product: "rose", stock: "2", price: "4.99"},
                { name: "Peony", product: "peony", stock: "0", price: "1.50"},
                { name: "Primula", product: "primula", stock: "1", price: "3.12"},
                { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}]
        };

        var templResult = $("#flowerTpl").template(data).filter("");
        templResult.slice(0, 3).appendTo("#row1");
        templResult.slice(3).appendTo("#row2");

        $("form").validate({
            highlight: function (element, errorClass) {
                $(element).add($(element).parent()).addClass("invalidElem");
            },
            unhighlight: function (element, errorClass) {
                $(element).add($(element).parent()).removeClass("invalidElem");
            },
            errorElement: "div",
            errorClass: "errorMsg"
        });

        $.validator.addMethod("stock", function (value, elem, args) {
            return Number(value) <= Number(args);
        }, function(args) {
            return "We only have " + args + " in stock"
        });

        $("input").each(function (index, elem) {
            $(elem).rules("add", {
                stock: data.flowers[index].stock
            })
        }).change(function (e) {
            $("form").validate().element($(e.target));
        });
    });
</script>
...
```

我们一直依赖于这4个紧密联系的配置选项。在接下来的几节中，我将逐个讲解这些选项的重要性。

### 1. 为问题元素设置class

当用户输入无效值时，验证插件会提示出错信息。我们可以用errorClass选项指定错误信息使用的class。在示例中，我指定这个class为errorMsg。相应地，在代码清单13-23里，我们使用style元素定义该class的样式。这条样式规则把文本设置成红色，以突出显示验证错误。

代码清单13-23 示例页面的style元素

```
...
<style type="text/css">
    .errorMsg {color: red}
    .invalidElem {border: medium solid red}
</style>
...
```

### 2. 设置存放错误信息的容器元素

错误信息文本将作为包含无效数据的表单元素的下一个兄弟元素插入到页面中。插件会默认把提示信息放到一个label元素中。由于我们的样式表里有匹配格子一级div元素内所有label元素的样式，错误提示可能显示异常，因此这个默认设置并不适用于这个例子。为了解决这个问题，如代码清单13-24所示，我使用errorElement选项指定使用div元素而不是默认的label元素存放错误提示。

代码清单13-24 指定存放错误提示信息的容器元素

```
...
$("form").validate({
    highlight: function(element, errorClass) {
        $(element).add($(element).parent()).addClass("invalidElem");
    },
    unhighlight: function(element, errorClass) {
        $(element).add($(element).parent()).removeClass("invalidElem");
    },
    errorElement: "div",
    errorClass: "errorMsg",

});
...
```

### 3. 设置问题元素的高亮效果

highlight及 unhighlight选项允许我们指定高亮显示包含错误值元素的回调函数。回调函数的参数是问题元素以及errorClass选项的值（不论你是否指定这个选项）。

看看代码清单13-25中的加粗部分，你会发现我忽略了第二个参数，而且使用HTMLElement对象生成一个jQuery结果集，通过它找到问题元素的父元素，然后给它加上invalidElem class。

代码清单13-25 控制元素的高亮显示

```
...
$("form").validate({
    highlight: function(element, errorClass) {
        $(element).add($(element).parent()).addClass("invalidElem");
    },
    unhighlight: function(element, errorClass) {
```

```

        $(element).add($(element).parent()).removeClass("invalidElem");
    },
    errorElement: "div",
    errorClass: "errorMsg",

    });
    ...

```

当用户纠正了错误数据，输入有效的数据之后，代码会自动调用unhighlight选项指定的函数。我借这个机会移掉由刚才的函数添加上的invalidElem class。如代码清单13-26所示，invalidElem class对应着页面中style元素中的一个选择器。

代码清单13-26 用来高亮显示元素的样式

```

...
<style type="text/css">
    .errorMsg {color: red}
    .invalidElem {border: medium solid red}
</style>
...

```

我们可以在这两个函数中按照自己喜欢的方式任意选择处理元素。我为问题元素的父元素添加了一个边框，其实，我也可以直接操作问题元素，或者处理页面中我想处理的任何部分。

### 13.3.5 使用问题摘要

我们可以让验证插件显示包含所有问题的摘要，而不是为每个元素单独提示出错信息。当页面结构或者布局无碍做到在每一个问题元素旁边提示出错信息时，这一特性就很有用。代码清单13-27演示了如何生成问题摘要。

代码清单13-27 使用问题摘要

```

...
<script type="text/javascript">
    $(document).ready(function() {

        var data = {
            flowers: [
                { name: "Aster", product: "aster", stock: "10", price: "2.99"},
                { name: "Daffodil", product: "daffodil", stock: "12", price: "1.99"},
                { name: "Rose", product: "rose", stock: "2", price: "4.99"},
                { name: "Peony", product: "peony", stock: "0", price: "1.50"},
                { name: "Primula", product: "primula", stock: "1", price: "3.12"},
                { name: "Snowdrop", product: "snowdrop", stock: "15", price: "0.99"}
            ];

            var plurals = {
                aster: "Asters", daffodil: "Daffodils", rose: "Roses",
                peony: "Peonies", primula: "Primulas", snowdrop: "Snowdrops"
            };

            var templResult = $("#flowerTpl").tmpl(data);
            templResult.slice(0, 3).appendTo("#row1");
        });
    }

```

```

templResult.slice(3).appendTo("#row2");

$("<div id='errorSummary'>Please correct the following errors:</div>")
  .addClass("errorMsg invalidElem")
  .append("<ul id='errorsList'></ul>").hide().insertAfter("h1");

$("form").validate({
  highlight: function(element, errorClass) {
    $(element).addClass("invalidElem");
  },
  unhighlight: function(element, errorClass) {
    $(element).removeClass("invalidElem");
  },
  errorContainer: "#errorSummary",
  errorLabelContainer: "#errorsList",
  wrapper: "li",
  errorElement: "div"
});

$.validator.addMethod("stock", function (value, elem, args) {
  return Number(value) <= Number(args.data.stock);
}, function(args) {
  return "You requested " + $(args.element).val() + " "
    + plurals[args.data.product] + " but we only have "
    + args.data.stocklevel + " in stock";
});

$("input").each(function (index, elem) {
  $(elem).rules("add", {
    stock: {
      index: index,
      data: data.flowers[index],
      element: elem
    }
  });
}).change(function (e) {
  $("form").validate().element($(e.target));
});

});
</script>
...

```

对于这个例子来说，在解释如何实现这种效果之前，我们先看看效果，然后再研究代码。问题信息摘要的显示效果见图13-14。

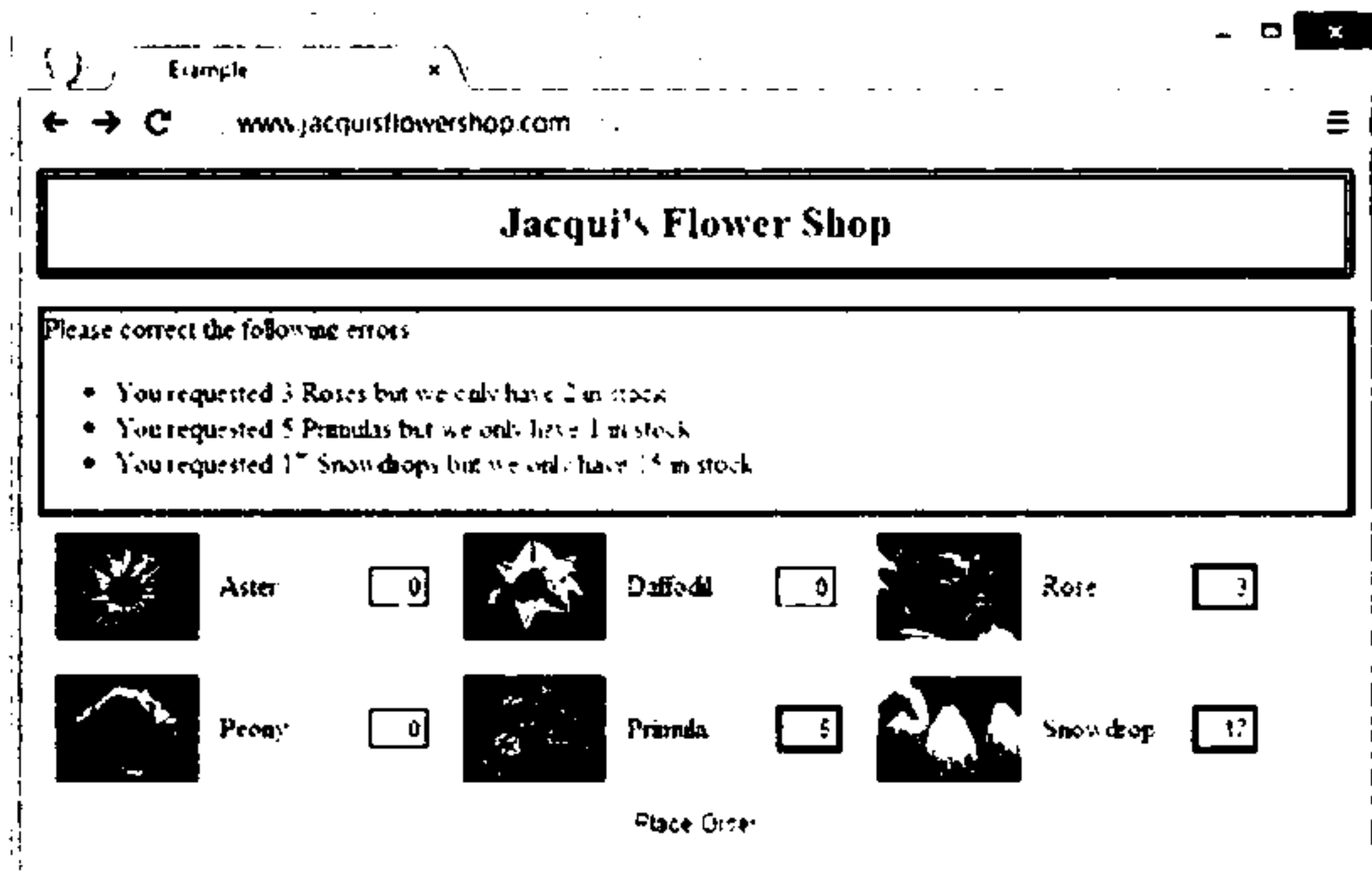


图13-14 问题摘要信息

### 1. 准备验证信息

当使用问题摘要时，需要解决的第一个问题就是：缺少上下文信息（保存在问题元素下一个兄弟元素内的出错信息丢失了）。因此，我们不得不在错误信息上做一些额外的工作，以便解决问题。首先，定义一个包含花卉复数名字的对象：

```
...
var plurals = {
  aster: "Asters", daffodil: "Daffodils", rose: "Roses",
  peony: "Peonies", primula: "Primulas", snowdrop: "Snowdrops"
}
...
```

我使用这些值利用自定义检查生成具体的出错信息，详见以下代码：

```
...
$.validator.addMethod("stock", function (value, elem, args) {
  return Number(value) <= Number(args.data.stocklevel);
}, function(args) {
  return "You requested " + $(args.element).val() + " "
    + plurals[args.data.product] + " but we only have "
    + args.data.stock + " in stock";
});
...
```

连接这两步的关键是对表单元素实施自定义检查时指定的参数。内建检查只支持简单的参数，然而我们可以使用复杂的对象并传递满足需要的任意数据：

```
...
$("input").each(function (index, elem) {
  $(elem).rules("add", {
    stock: {
      index: index,
      data: data.flowers[index],
      element: elem
    }
  })
})
```

```

    });
  }).change(function (e) {
    $("form").validate().element($(e.target));
  });
  ...

```

在本例中，我传入了index、data数组和元素本身，这就是显示问题摘要所需的全部数据。在本章最后，我还会介绍一个能够简化组装出错信息字符串的插件功能。

## 2. 生成问题摘要

创建存放问题摘要的元素，并将其添加到页面中。为此，添加一个包含ul元素的div元素。我的目标是使用一个无序列表显示每一个错误：

```

...
$("<div id='errorSummary'>Please correct the following errors:</div>")
  .addClass("errorMsg invalidElem").append("<ul id='errorsList'></ul>").hide().insertAfter("h1");
...

```

这个div元素中包含的文本信息会显示在问题摘要的上方。注意我在把这些元素追加到DOM之后调用了hide方法。我们不仅负责生成这些元素，还应保证在没有错误发生时，用户就看不到这些元素。hide方法保证了问题摘要在初始状态下不会显示。一旦验证进行开始，验证插件自会根据验证结果来控制问题摘要的可见性。

现在我们已经拥有生成摘要所需的所有零件，接下来如下配置问题摘要：

```

...
$("form").validate({
  highlight: function (element, errorClass) {
    $(element).addClass("invalidElem");
  },
  unhighlight: function (element, errorClass) {
    $(element).removeClass("invalidElem");
  },
  errorContainer: "#errorSummary",
  errorLabelContainer: "#errorsList",
  wrapper: "li",
  errorElement: "div"
});
...

```

我改变了highlight和unhighlight函数中的目标元素，让它们仅仅处理input元素。errorContainer选项指定一个选择器，当需要显示问题摘要时，验证插件会让这个选择器对应的元素可见。在本例中，这个选项的值是#errorSummary（对应着问题摘要的容器div）。errorLabelContainer选项指定存放具体出错提示的容器元素。在本例中，我希望出错信息显示成一个列表，因此这个容器就是ul#errorList元素。

wrapper选项用来指定包住每一条出错信息的元素。如果你希望出错信息显示成一个列表，这个选项就太有用了。最后一个选项errorElement用来指定包住错误提示文本的元素。所有这些选项共同造就了图13-14那样的问题摘要显示效果。

用户每解决一个问题，验证插件就从摘要中删除对应的信息。当完全没有问题时，问题摘要则彻底消失，用户顺利提交表单。图13-15展示了用户解决完3个问题中的2个之后所显示的问题摘要。

选择就近显示错误提示还是问题摘要，这纯属个人喜好问题，而且往往由页面结构决定。好在验证插件足够灵活，而且确实无需花费大量工作就能定义和实施量身订做的表单验证规则。

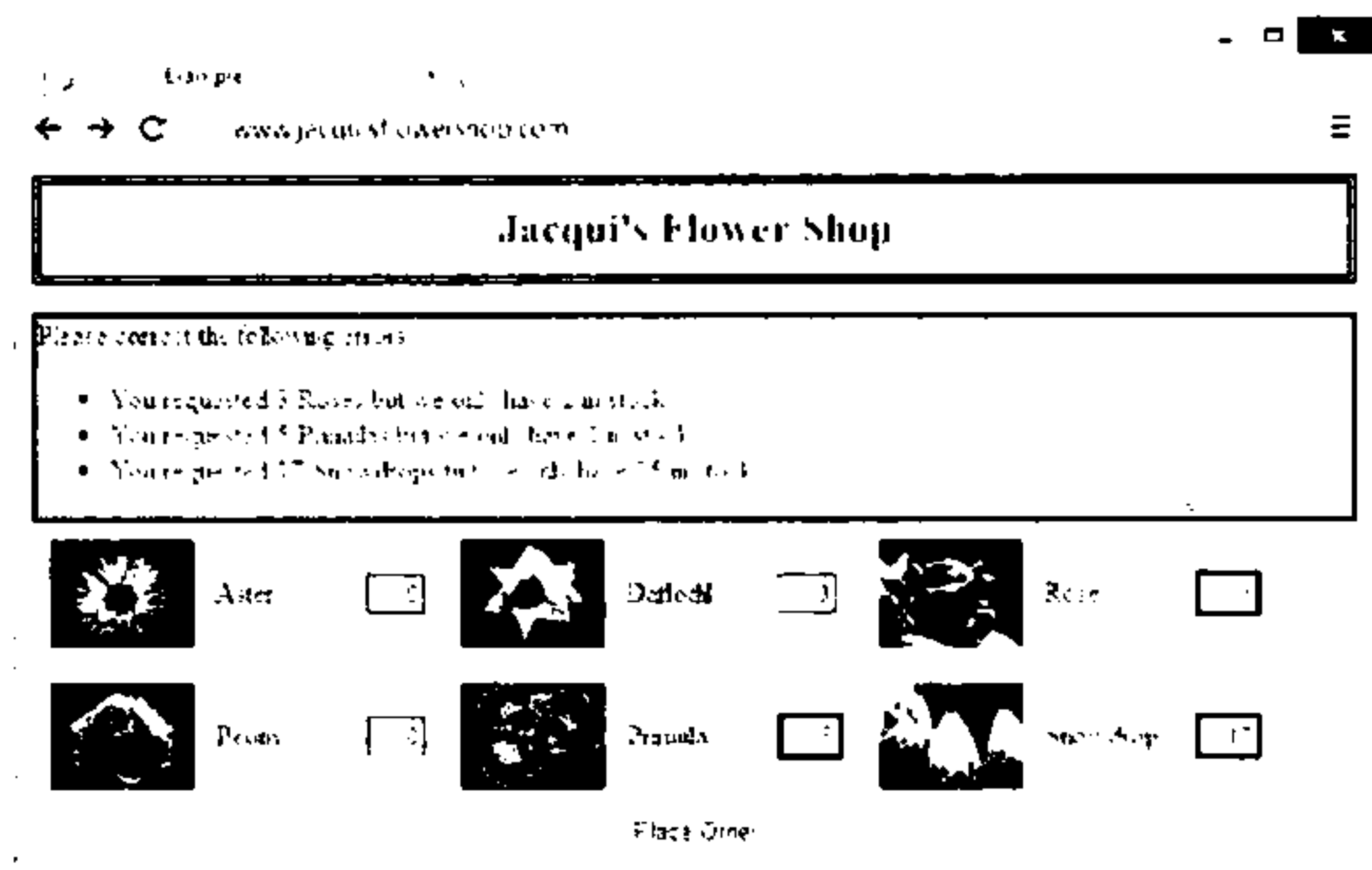


图13-15 显示更少错误信息的问题摘要

### 3. 更规整地组装出错信息

现在来对脚本做最后的改造，演示一个由验证插件提供的与验证数据不直接相关、却很有用的功能。在前面的例子中，当我需要根据上下文生成出错信息时，采用了如下代码来拼接字符串与变量：

```
...
$.validator.addMethod("stock", function (value, elem, args) {
    return "You requested " + $(args.element).val() + " "
        + plurals[args.data.product] + " but we only have "
        + args.data.stock + " in stock";
});
...
```

这么做没问题，只是代码难看又难读。验证插件提供了一个类似C#语言的字符串组装工具，在代码清单13-28中，我演示了这一工具的用法。

#### 代码清单13-28 jQuery验证插件的字符串格式化功能

```
...
$.validator.addMethod("stock", function (value, elem, args) {
    return $.validator.format("You requested {0} {1} but we only have {2} in stock",
        $(args.element).val(), plurals[args.data.product], args.data.stock );
});
...
```

在上面的代码中，字符串组装工作由`$.validator.format`完成，它支持一个模板字符串参数，以及一些表示值的参数。在模板字符串中，大括号括起来的数字如`{0}`，表示一个占位符，在返回结果中，它会被相应的值参数替换掉。第一个值参数对应`{0}`，第二个对应`{1}`，以此类推。`$.validator.format`方法的返回值是一个函数，它会等到需要显示出错信息时才运行，这样就保证了总是使用正确的值组装出错信息。

初次使用这种字符串组装方式，你可能会觉得这么做有点古怪，不过对那些熟悉类似C#语言的字符串组装方式的人说，这真是一个不可或缺的功能。



## 13.4 小结

本章演示了jQuery对表单验证的支持。一开始，我们先回顾了表单有关的事件方法，并讲解了HTML表单世界中最重要几个角色。本章绝大部分内容都用来讲解Validation插件，在验证用户输入的表单数据方面，它提供了足够的灵活性和扩展性，以及在把数据提交到服务器之前提示用户解决输入问题的方法。接下来，我们开始学习在jQuery中使用Ajax。

Ajax本来代表的是异步JavaScript和XML，但现如今已是有着特定含义的通用术语。Ajax允许我们向服务器发起异步请求，这意味着请求发生在幕后，不会阻塞用户与页面内容之间的交互。Ajax最常见的用途是异步提交表单，而这样做的最大优点是浏览器不必重新载入新页面即可接收来自服务器的数据，并且支持使用标准的jQuery函数在页面中平滑地显示这些数据。

本章用到的Ajax功能已经内建于jQuery核心库，不过我还是在本章末尾简要介绍了一个有用的插件。jQuery并没有重新发明Ajax，但让浏览器内建的Ajax API变得更容易使用。本章我会介绍Ajax的快捷方法。它们都是些很简单的方法，使我们能相当快速轻松地使用Ajax。第15章会讲解底层的jQuery Ajax API，它是本章讲到的这些快捷方法的基础。不过，你很快会看到底层API其实没有那么底层，总有一些不适用Ajax快捷方法的场合，这是我们使用底层API的主要原因。表14-1列出了本章概要。

表14-1 本章概要

问 题	解决方法	代码清单
如何发起一个异步HTTP GET请求	使用get方法	1~3
如何处理来自Ajax GET请求的数据	传递一个回调函数参数给get方法	4
如何为响应用户行为而发起Ajax请求	在事件处理函数中调用get方法	5
如何向服务器端请求JSON数据	使用get方法，并在回调函数中接收数据	6、7
如何在发起GET请求的同时发送数据给服务器	作为参数传递一个JavaScript对象给get方法	8
如何发起一个HTTP异步POST请求	使用post方法	9、10
如何在POST请求中发送非表单数据	作为参数传递一个JavaScript对象给post方法	11
如何在Ajax请求中忽略服务器端指定的数据类型	在发起Ajax请求时（给get和post方法）指定期望的数据类型	12~13
如何避免最常见的Ajax陷阱	不要错把Ajax异步请求当成同步请求	14
如何使用特定数据类型的GET快捷方法	使用load、getScript或getJSON方法	15~22
如何为表单元素轻松启用Ajax	使用Ajax Forms插件	23

## 14.1 Ajax 快捷方法

Ajax通常与发送表单数据相关，不过它的适用范围相当广。我将从一些相当简单的任务开始介绍Ajax，首先尝试在完全不使用表单的前提下从服务器端获取数据的几种方法。

jQuery定义了一整套Ajax快捷方法，它们本质上都是核心Ajax函数的某种封装，用于方便快捷地执行常见的Ajax任务。在接下来的几节中，我将介绍几个使用HTTP GET请求从服务器端获取数据的快捷方法。表14-2概括介绍了这些方法。

表14-2 Ajax快捷方法

名 称	描 述
\$.get()	使用 HTTP GET 方法发起Ajax请求
\$.post()	使用 HTTP POST 方法发起Ajax请求

(简要) 理解异步任务

如果你是一个Ajax新手，请听我简要解释一下什么是异步请求。这一点非常重要，因为异步请求在Ajax技术中是核心中的核心。Ajax术语的第一个字母A表示的就是异步。绝大部分时候，我们都是编写异步代码。我们定义一些语句块，执行某个任务，然后等待浏览器执行它们。当最后一行代码执行完毕，任务完成。在这个任务执行期间，浏览器拒绝与用户进行任何形式的交互。

执行一个异步任务时，我们告诉浏览器希望在后台做一些事情。“后台”这个词很抽象，基本上就是“去做这件事情，不要阻止用户与页面的任何交互，当事情完成时通知我”的意思。就Ajax来说，我们告诉浏览器与服务器做一些通信，并让浏览器在请求完成时通知我们。这个通信过程是通过回调函数处理的。我们提供给jQuery一个或多个函数，这些函数在任务完成后被自动调用。事实上，可以有一个函数处理成功的请求，也可以有一些别的函数处理遇到的其他结果，比如错误。

异步请求的优点在于我们得以创建具有更好体验的HTML页面，根据来自服务器的数据平滑地更新页面，既无需打断用户的交互，也不必让用户苦苦等待浏览器载入新页面。

这样做也有缺点：我们必须非常小心地通盘考虑代码。我们无法预知请求在什么时候完成，也无法对请求结果做任何假定。而且，回调函数的广泛使用通常会导致更加复杂的代码，这对于那些喜欢对请求结果或者请求时间做假定的粗心大意的程序员来说是一种惩罚。

14.1.1 发起Ajax GET请求

首先，我们使用Ajax发起一个HTTP GET请求，以获取一个HTML片段并把这个片段添加到浏览器显示的HTML页面中。代码清单14-1展示了我们即将处理的示例页面。

代码清单14-1 示例页面

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js" type="text/javascript"></script>
  <script src="handlebars-jquery.js" type="text/javascript"></script>
  <script src="jquery.validate.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script type="text/javascript">
```

```

        $(document).ready(function() {
            // 在这里添加脚本
        });
    </script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                </div>
                <div id="row2" class="drow">
                </div>
            </div>
        </div>
        <div id="buttonDiv"><button type="submit">Place Order</button></div>
    </form>
</body>
</html>

```

这个页面与我们前面见过的例子相似, 不过它缺少描述产品的元素项, 而且也没有数据或者模板能生成它们。这是怎么回事? 原来我另外准备了一个名为flowers.html的文件, 这个文件和示例页面(示例页面在本书源代码下载包中的名字是example.html)放在一起。代码清单14-2展示了flowers.html的内容。

#### 代码清单14-2 flowers.html文件

```

<div>
    <label for="aster">Aster:</label>
    <input name="aster" value="0" required />
</div>
<div>
    <label for="daffodil">Daffodil:</label>
    <input name="daffodil" value="0" required />
</div>
<div>
    <label for="rose">Rose:</label>
    <input name="rose" value="0" required />
</div>
<div>
    <label for="peony">Peony:</label>
    <input name="peony" value="0" required />
</div>
<div>
    <label for="primula">Primula:</label>
    <input name="primula" value="0" required />
</div>
<div>
    <label for="snowdrop">Snowdrop:</label>
    <input name="snowdrop" value="0" required />
</div>

```

这些元素与我们在前面章节看到过的内容一模一样，只是它们没有按行排列，而且div元素也没有class属性。之所以做这些修改，并非出于技术原因，而是为了演示这些内容在载入之后再进一步处理。注意，这并不是一个完整的HTML页面，它缺少html、head、body等元素，只是一个片段。如图14-1所示，flowers.html页面已完全与主页面分离出来（主页面只剩下标题）。

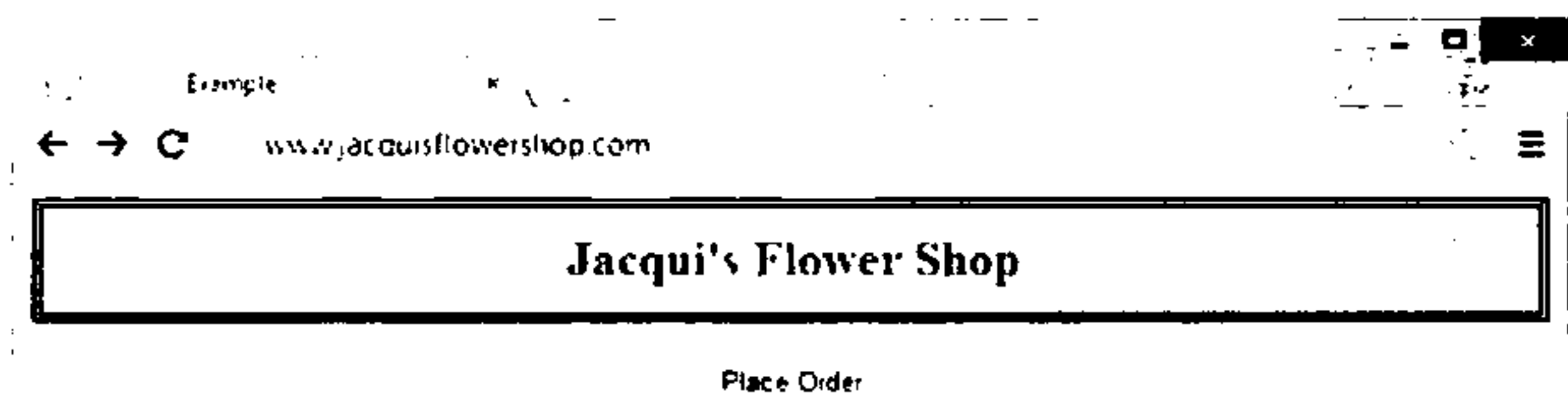


图14-1 初始的示例页面

我打算使用jQuery的Ajax功能载入这个HTML片段，并将它添加到主文档。你也许觉得这么做真是多此一举，不过我们正在模拟一种常见情况：不同片段的内容由不同的系统生成，并且需要组装到一起以便得到一个复杂的页面或者Web应用。简单起见，在这个例子里我只使用了一台服务器，不过很容易想象产品信息来自其他某个地方的情况。事实上，在后面的例子里，我会引入Node.js以便演示多个服务器的处理。不过，那是后面的事。现在把目光放到基础的jQuery Ajax功能上来，使用它处理flowers.html文件。具体代码见代码清单14-3。

#### 代码清单14-3 利用jQuery Ajax功能处理HTML片段

```
...
<script type="text/javascript">
  $(document).ready(function() {
    $.get("flowers.html", function(data) {
      var elems = $(data).filter("div").addClass("dcell");
      elems.slice(0, 3).appendTo("#row1");
      elems.slice(3).appendTo("#row2");
    });
  });
</script>
...
```

我使用了get方法，并提供了两个参数。第一个参数是希望载入的URL，在本例中就是flowers.html，我们会将它解释成一个相对URL，并根据主文件的位置得出其载入位置。

第二个参数是一个函数，用于在请求成功完成之后自动调用。我已经在刚才的“(简要)理解异步任务”中提到过，由于请求是异步的，Ajax相当依赖于回调函数。jQuery会将来自服务器的数据作为参数传递给回调函数。

在浏览器加载包含上面脚本的例子文档时，我的jQuery代码就会向服务器端发出请求，载入flowers.html。当flowers.html加载完成后，它包含的HTML片段会被解析成HTML元素，然后添加到主页面。图14-2展示了最终的结果。

好了，我承认最终得到的结果与前面你看到的（数据直接放到文件中或者通过模板生成）没啥不同，但这是怎么做到的？这是一个值得深究的问题，下面我们就深入了解台前幕后发生的一切。

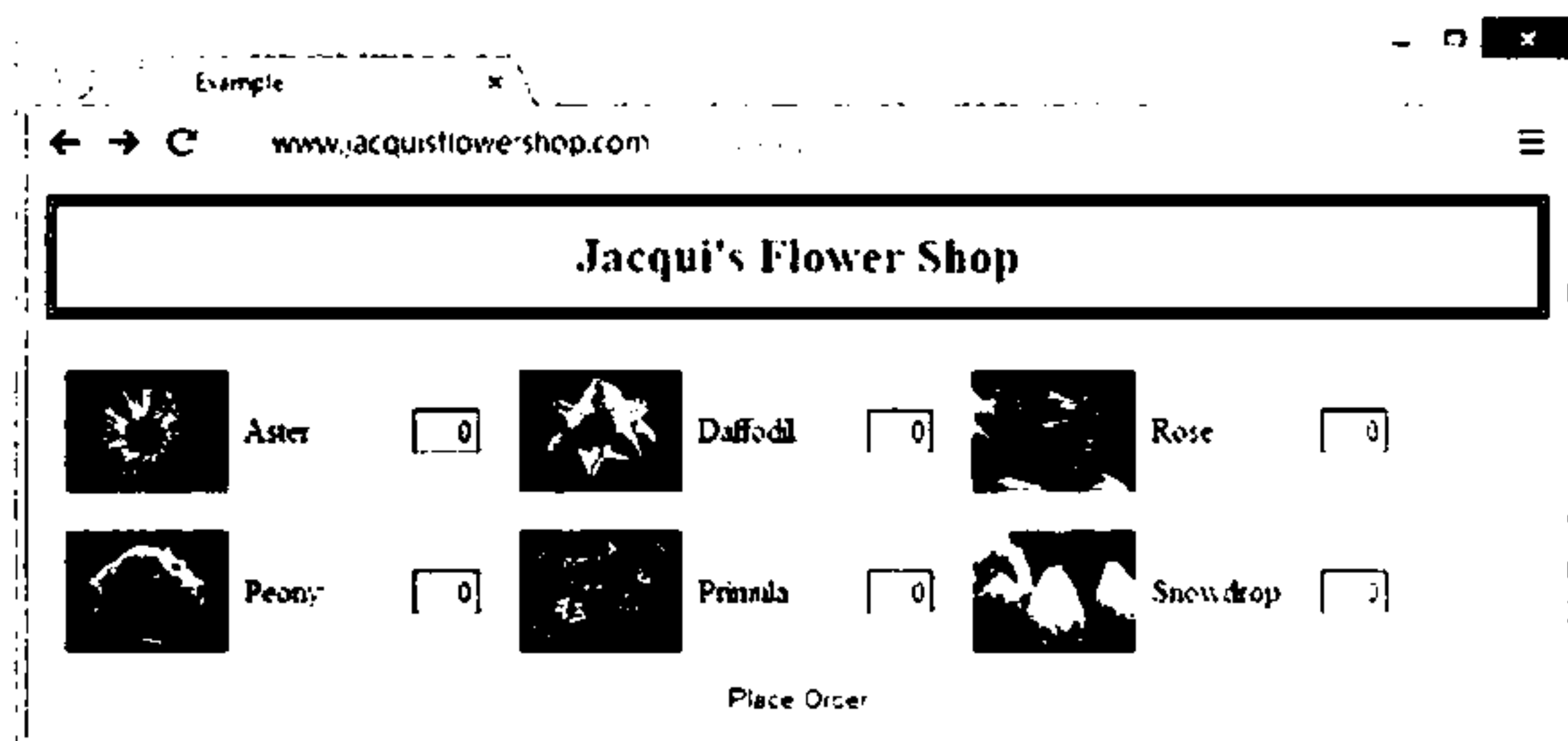


图14-2 Ajax请求的效果

**提示** 虽然在这里get方法载入的是一个HTML片段，但并不限于此，get方法可用来从服务器载入任意类型的数据。

### 1. 处理响应数据

请求成功完成之后，传递给回调函数的data参数正是服务器响应Ajax请求返回的数据。在这个例子中，我们得到了flowers.html的内容，它是一段HTML。

如代码清单14-4所示，为了使jQuery能够利用这段HTML，我把data作为参数传递给jQuery的\$函数，从而解析data并得到层次分明的一系列HTMLElement对象。

#### 代码清单14-4 处理来自服务器的数据

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $.get("flowers.html", function(data) {
            var elems = $(data).filter("div").addClass("dcell");
            elems.slice(0, 3).appendTo("#row1");
            elems.slice(3).appendTo("#row2");
        });
    });
</script>
...
```

前面提到过，我故意去掉了div元素的class属性。现在我已经使用标准的jQuery addClass方法把它们加了回来。一旦把data作为参数传递给\$函数，我们就得到一个标准的jQuery对象。接下来和前面各节一样，我连续使用slice和appendTo方法把这些元素添加到页面。

**提示** 注意，我使用了filter方法得到只包含div元素的结果集。在解析data时，jQuery会把添加在flowers.html文件中各div元素之间的回车换行符号解析为文本内容，并为它们创建文本元素节点。为了避免这种情况，我们既可以删除各div元素之间的回车换行符号，也可以使用filter方法来滤掉这些不必要的元素。这个问题与在上一章遇到的数据模板问题类似。

## 2. 让Ajax效果更明显

触发Ajax请求的语句在ready事件（详见第5章）发生时执行，这让我们很难直观看到Ajax发挥作用，因为flowers.html文件内容会自动载入并显示。为了更容易观察，我在页面中添加了一个按钮，只有点击这个按钮才会发出Ajax请求。代码清单14-5列出了这些变更。

代码清单14-5 按下按钮才触发Ajax请求

```
...
<script type="text/javascript">
  $(document).ready(function() {
    $("<button>Ajax</button>").appendTo("#buttonDiv").click(function (e) {
      $.get("flowers.html",
        function (data) {
          var elems = $(data).filter("div").addClass("dcell");
          elems.slice(0, 3).appendTo("#row1");
          elems.slice(3).appendTo("#row2");
        });
      e.preventDefault();
    });
  });
</script>
...
```

现在flowers.html不会被自动载入了，除非你按下Ajax按钮。而且，如图14-3所示，每按下一次按钮都会添加一些元素到页面。

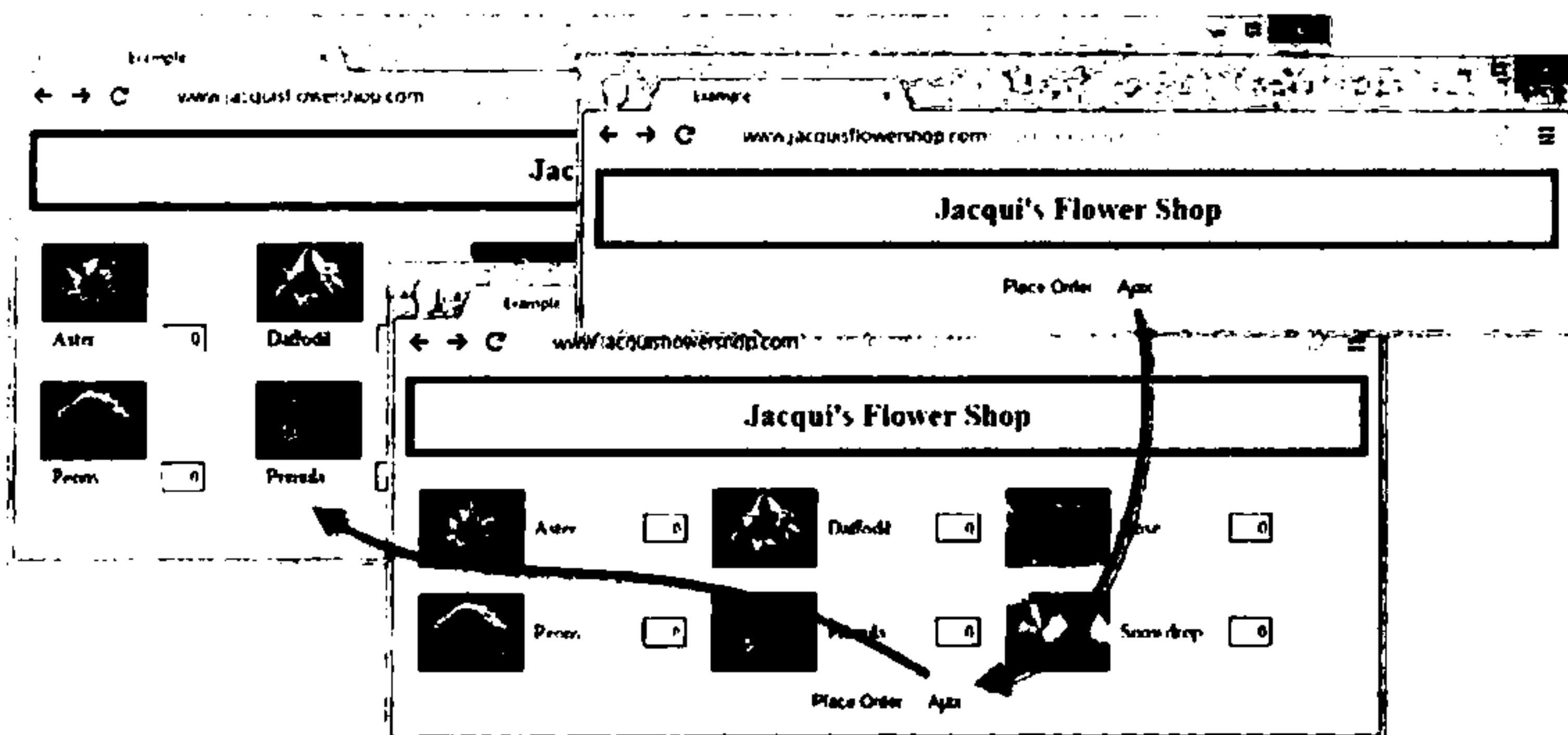


图14-3 按下按钮才触发Ajax请求

**提示** 注意，我在click事件处理函数中调用了传递给回调函数的Event对象的preventDefault方法（以阻止按钮的默认行为）。由于这个button元素位于一个表单内，它具有提交表单到服务器的默认行为。



### 3. 获取其他类型的数据

没有人说我们只能使用get方法获取HTML。我们可以从服务器获取任意类型的数据。特别值得一提的是JSON (JavaScript Object Notation) 数据, 它是一种jQuery容易为我们处理的数据格式。当年Ajax刚刚得到广泛应用时, XML是首选的数据格式, 甚至Ajax中的X就代表着XML。我不打算在这儿深入讲解XML的细节, 那相当冗长, 可读性不好, 而且相对来说其生成与处理都耗时且耗资源。

近年来, 很大程度上XML已经被JSON (JavaScript对象表示法) 格式代替, 后者是一种更简单的数据格式, 而且特别容易被JavaScript处理 (顾名思义)。对于这个例子来说, 我已经生成了一个名为mydata.json的文件, 并把它和example.html文件保存在服务器上的同一目录。代码清单14-6展示了mydata.json文件的内容。

代码清单14-6 mydata.json的内容

```
[{"name": "Aster", "product": "aster", "stock": "10", "price": "2.99"},
{"name": "Daffodil", "product": "daffodil", "stock": "12", "price": "1.99"},
{"name": "Rose", "product": "rose", "stock": "2", "price": "4.99"},
{"name": "Peony", "product": "peony", "stock": "0", "price": "1.50"},
{"name": "Primula", "product": "primula", "stock": "1", "price": "3.12"},
{"name": "Snowdrop", "product": "snowdrop", "stock": "15", "price": "0.99"}]
```

这个文件包含着花店商品数据, 如你所见, JSON数据与我们使用JavaScript代码表示数据的方式一模一样。如代码清单14-7所示, 要使用Ajax载入并处理这些数据可以使用get方法。

代码清单14-7 使用get方法获取JSON数据

```
...
<script id="flowerTpl" type="text/x-handlebars-template">
  {{#flowers}}
  <div class="dcell">
    
    <label for="{{product}}">{{name}}:</label>
    <input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
      value="0" required />
  </div>
</script>
<script type="text/javascript">
  $(document).ready(function() {
    $("<button>Ajax</button>").appendTo("#buttonDiv").click(function (e) {
      $.get("mydata.json", function (data) {
        var template = $("#flowerTpl").template({flowers: data}).filter("*");
        template.slice(0, 3).appendTo("#row1");
        template.slice(3).appendTo("#row2");
      });
      e.preventDefault();
    });
  });
</script>
...
```

这个例子在有人点击按钮时向服务器请求JSON数据文件。和HTML片段的例子一样, 服务器返回的数据被传递给回调函数。在这儿我使用了数据模板插件 (参见第12章) 来处理数据并生成HTML元



素，然后使用slice和appendTo方法把HTML元素插入到页面。注意，我并没有必要为了把JSON字符串转换为JavaScript对象而做什么事情；这些事情jQuery都自动地替我们做好了。

**提示** 在不能识别文件的扩展名或者格式时，有些Web服务器（包括本书所用的Microsoft IIS版本）会拒绝返回文件内容给浏览器。为了让这个例子在IIS下面工作，我只好在文件扩展名（.json）和针对JSON数据的MIME类型（application/json）之间做了映射，否则请求mydata.json时IIS会返回404——找不到文件错误。

#### 4. 为GET请求提供数据

我们可以通过get方法（以及本章后面提到的load、getScript和getJSON方法）在GET请求中发送数据给服务器：只要给使用的快捷方法传递一个data对象就可以了。代码清单14-8提供了一个例子。

代码清单14-8 在执行GET请求的同时发送数据给服务器

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var requestData = {
            country: "US",
            state: "New York"
        };

        $("<button>Ajax</button>").appendTo("#buttonDiv").click(function (e) {
            $.get("mydata.json", requestData, function (data) {
                var tplData = $("#flowerTpl").template({flowers: data}).filter("*");
                tplData.slice(0, 3).appendTo("#row1");
                tplData.slice(3).appendTo("#row2");
            });
            e.preventDefault();
        });
    });
</script>
...
```

我们提供的数据会被作为请求字符串追加到目标URL中。对于上面的例子来说，这意味着我们的请求实际上是：

<http://www.jacquisflowershop.com/jquery/flowers.html?country=US&state=New+York>

服务器可根据我们提供的数据返回定制内容，比如我们可以为不同的州提供不同的花卉产品。不能直接从浏览器里看到Ajax所请求的URL，但我们可以利用开发工具（又称为F12工具，因为它是由F12键激活的）查看发出的请求。以Google Chrome为例，敲F12键，在打开的窗口中点击Network标签，然后点击XHR（XmlHttpRequest对象的缩写，它是jQuery发起Ajax请求要用到的DOM对象）过滤器。

图14-4展示了Chrome浏览器下如何查看本例中Ajax请求的细节。

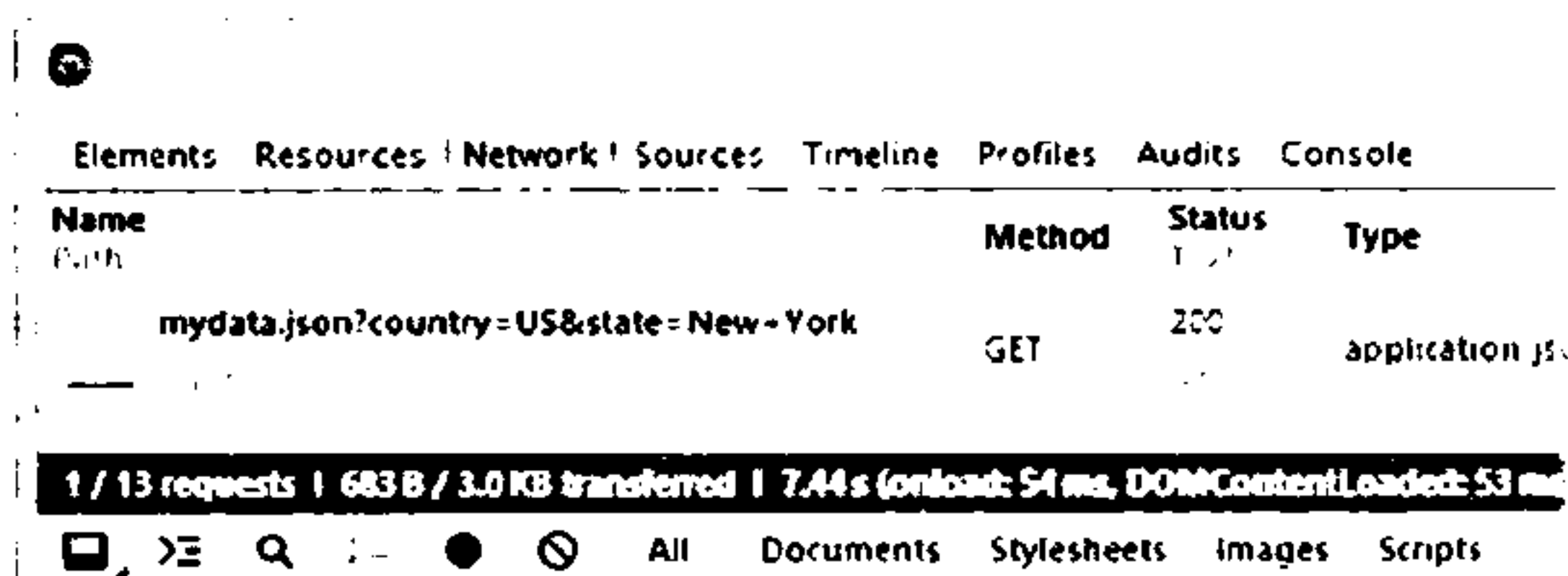


图14-4 使用Google Chrome F12工具检视Ajax请求

### GET还是POST？一定要选择正确

使用GET请求发送表单数据可能会有风险。所以，请小心一点儿。经验法则是GET请求应该仅用于获取只读的信息，而POST请求则用于任何会导致应用程序状态发生变化的操作。

用Web标准术语来讲，GET请求用于安全的交互（除了获取数据没有副作用），而POST请求用于非安全的交互（做一个什么决定或者改变什么东西）。这些惯例由W3C制定（[www.w3.org/Provider/Style/URI](http://www.w3.org/Provider/Style/URI)）。

因此，我们可以使用GET请求发送表单数据给服务器，只要这些表单数据不会改变服务器的状态即可。2005年，当Google Web Accelerator公开发布时，一些Web开发者辛酸地体会到了这一点。这个应用程序预先获取每个页面链接的所有内容，由于GET请求是安全的，这在HTTP范畴内是合法的。遗憾的是，许多Web开发者忽略了HTTP惯例，在站点上放置了许多类似这样的链接：“delete item”或者“add to shopping cart”。这简直是一片混乱。

曾有公司以为他们的内容管理系统（CMS）经常遭受恶意攻击，因为系统的内容不断被删除，后来他们找到原因，发现搜索引擎的爬虫访问到了管理页面的删除链接。

## 14.1.2 Ajax POST请求

现在我们已经知道如何从服务器获取数据，接下来我们研究如何发送数据，即如何把表单数据发送到服务器。jQuery也提供了一个快捷方法post，使用它发送表单数据异常简单。在演示post方法之前，需要在formserver.js中增加一些代码，以便Node.js能够接收并处理我们在示例中发起的POST请求。

### 1. 为接收表单数据而设置Node.js

我们需要一个服务器端脚本，由它接收浏览器POST过来的数据，并做简单处理，然后返回处理之后的数据。代码清单14-9列出的代码，是上一章的formserver.js文件改造之后的版本。

#### 代码清单14-9 新版本formserver.js

```
var http = require("http");
var querystring = require("querystring");
var url = require("url");

http.createServer(function (req, res) {
  console.log("[200 OK] " + req.method + " to " + req.url);
```

```

if (req.method == "OPTIONS") {
    res.writeHead(200, "OK", {
        "Access-Control-Allow-Headers": "Content-Type",
        "Access-Control-Allow-Methods": "*",
        "Access-Control-Allow-Origin": "http://www.jacquisflowershop.com"
    });
    res.end();
} else if (req.method == "POST") {
    var dataObj = new Object();
    var contentType = req.headers["content-type"];
    var fullBody = "";

    if (contentType) {
        if (contentType.indexOf("application/x-www-form-urlencoded") > -1) {
            req.on("data", function (chunk) { fullBody += chunk.toString(); });
            req.on("end", function () {
                var dBody = querystring.parse(fullBody);
                writeResponse(req, res, dBody,
                    url.parse(req.url, true).query["callback"]);
            });
        } else {
            req.on("data", function (chunk) { fullBody += chunk.toString(); });
            req.on("end", function () {
                dataObj = JSON.parse(fullBody);
                var dprops = new Object();
                for (var i = 0; i < dataObj.length; i++) {
                    dprops[dataObj[i].name] = dataObj[i].value;
                }
                writeResponse(req, res, dprops);
            });
        }
    }
} else if (req.method == "GET") {
    var data = url.parse(req.url, true).query;
    writeResponse(req, res, data, data["callback"])
}

function writeResponse(req, res, data, jsonp) {
    var total = 0;
    for (item in data) {
        if (item != "_" && data[item] > 0) {
            total += Number(data[item]);
        } else {
            delete data[item];
        }
    }
    data.total = total;
    jsonData = JSON.stringify(data);
    if (jsonp) {
        jsonData = jsonp + "(" + jsonData + ")";
    }

    res.writeHead(200, "OK", {

```

```
        "Content-Type": "application/json",
        "Access-Control-Allow-Origin": "*"
    });
    res.write(jsonData);
    res.end();
}

}).listen(port);
console.log("Ready on port " + port);
```

---

**提示** 得到这个脚本的最简单方法是从Apress.com下载本书的源代码。第1章对如何获取Node.js有详细的介绍。

---

和以前一样，我在命令行输入以下命令运行这个脚本：

---

```
node.exe formserver.js
```

---

这个修改后的Node.js脚本处理浏览器发送过来的数据，并响应JSON数据。我曾经让这个脚本返回HTML，然而JSON更简洁，而且通常更容易处理。我返回的JSON对象非常简单：一个包含用户选中产品总数以及每种产品数量的对象。因此，举例来说，如果我选中了1个aster、2个daffodil和3个rose，Node.js脚本返回的数据就是这样：

```
{"aster": "1", "daffodil": "2", "rose": "3", "total": 6}
```

本章前面我展示过的JSON数据是一个对象数组，而这次服务器端脚本返回的是一个简单对象，它的属性分别表示选中的花卉。total属性包含着选中各产品的总和。我并不认为这是服务端能实施的最有价值的行为，不过毕竟我们的主题是Ajax应用而非服务器端开发。

## 2. 理解跨源Ajax请求

仔细查看新的formserver.js脚本，你会发现我在响应浏览器时设置了以下HTTP头：

```
Access-Control-Allow-Origin: http://www.jacquisflowershop.com
```

默认情况下，浏览器只允许脚本对与页面同源的服务器发起请求。一个源就是由协议、主机名和端口组成的URL。如果两个URL有着同样的协议、主机名和端口，那么它们就是同源的。如果三者之一有所不同，那么它们就是不同源的。

---

**提示** 制定这条规则的目的是降低跨站脚本攻击（欺骗浏览器或者用户执行恶意脚本）的风险。跨站脚本攻击这个主题超出了本书讲述范围，不过维基百科有一篇很棒的文章可以参考：  
[http://en.wikipedia.org/wiki/Cross-site\\_scripting](http://en.wikipedia.org/wiki/Cross-site_scripting)。

---

表14-3列出了几个URL与主页面（<http://www.jacquisflowershop.com/jquery/example.html>）来源的比较结果。

表14-3 URL比较

URL	比较来源
http://www.jacquisflowershop.com/apps/mydoc.html	同源
https://www.jacquisflowershop.com/apps/mydoc.html	协议不同, 非同源
http://www.jacquisflowershop.com:81/apps/mydoc.html	端口不同, 非同源
http://node.jacquisflowershop.com/order	主机名不同, 非同源

我配置了两台服务器: `www.jacquisflowershop.com` 处理静态内容, 而 `node.jacquisflowershop.com` 运行着 Node.js。通过上表得知, 来自第一台服务器的页面与来自第二台服务器的页面不同源。当我们从一个来源向另一个来源发起请求时, 这就是跨源请求。

问题是这条规则其实是一条禁令; 跨源请求是不被允许的。开发者不得不使用一些非常丑陋的技巧欺骗浏览器, 以设法绕过这条规则发出跨源请求。幸运的是, 今天终于有了一条支持跨源请求的合法通道, 这就是 CORS (Cross-Origin Resource Sharing, 跨源资源共享规范)。在这儿我只是简要介绍一下 CORS, 完整内容请参阅 [www.w3.org/TR/cors](http://www.w3.org/TR/cors) 查阅标准全文。

**提示** CORS 规范问世不久, 可以说是崭新的标准。只有最新的浏览器才支持这一标准, 那些较老的浏览器一如故我地直接忽略跨站请求, 权当这些请求不存在。还有一个更可靠的解决方案 (支持旧浏览器), 也就是 JSONP 方案, 详见 14.3.3 节中的 “使用 JSONP”。

CORS 的工作方式是这样的: 浏览器请求第二个服务器 (在这里是 Node.js 服务器) 时, 发送 Origin (来源) 报头。这个报头的值就是请求的来源。

如果目标服务器认识这个来源并且允许浏览器发出跨源请求, 它会添加 Access-Control-Allow-Origin 报头, 并把这个报头的值匹配来源请求的 Origin 报头。如果服务器响应信息缺少这个报头, 浏览器就忽略响应信息。

**提示** 支持 CORS 意味着浏览器必须在连接目标服务器并得到响应报头之后使用跨源安全策略; 也就是说, 只有在请求成功发出并得到响应报头之后, 才可以判断响应信息是否缺少指定的报头以及报头中是否指定了不正确的域, 然后根据判断结果返回或者扔掉响应信息。这是一条崭新的路径, 那些尚不支持 CORS 的浏览器只会直接扔掉这个请求, 根本不会连接目标服务器。

在 `formserver.js` 脚本中, 我手工把 Access-Control-Allow-Origin 报头设置为可信源 `www.jacquisflowershop.com`<sup>①</sup>。不过, 我们很容易把这个报头的值设置为 Origin 报头的值以应对更复杂的决策程序。我们也可以把 Access-Control-Allow-Origin 报头设置为星号 (\*), 从而允许任意来源的跨站请求。这个设置用于测试环境时非常方便, 但在把代码部署到生产环境之前, 你一定要想清楚这个设置是否会造成安全隐患。

① 上面的脚本并没有把这个报头设置为 `www.jacquisflowershop.com`, 而是设置成了 \*, 即允许任意来源报头。

### 3. 使用post方法提交表单数据

现在我们已经把服务器设置好了, 并且也理解了CORS的工作原理。如代码清单14-10所示, 我们可以使用post方法发送表单数据给服务器了。

代码清单14-10 使用post方法提交表单数据

```
...
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js" type="text/javascript"></script>
  <script src="handlebars-jquery.js" type="text/javascript"></script>
  <script src="jquery.validate.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}</label>
      <input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
        value="0" required />
    </div>
    {{/flowers}}
  </script>
  <script id="totalTpl" type="text/x-handlebars-template">
    <div id="totalDiv" style="clear: both; padding: 5px">
      <div style="text-align: center">Total Items:
        <span id="total">{{total}}</span></div>
    </div>
  </script>
  <script type="text/javascript">
    $(document).ready(function () {

      $.get("flowers.html", function (data) {
        var elems = $(data).filter("div").addClass("dcell");
        elems.slice(0, 3).appendTo("#row1");
        elems.slice(3).appendTo("#row2");
      });

      $("button").click(function (e) {
        var formData = $("form").serialize();
        $.post("http://node.jacquisflowershop.com/order",
          formData, processServerResponse);
        e.preventDefault();
      });

      function processServerResponse(data) {
        var inputElems = $("div.dcell").hide();
        for (var prop in data) {
          var filtered = inputElems.has("input[name=" + prop + "]")
            .appendTo("#row1").show();
        }
      }
    });
  </script>

```

```

    }
    $("#buttonDiv").remove();
    $("#totalTpl").template(data).appendTo("body");
  }
});
</script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="oblock">
      <div class="dtable">
        <div id="row1" class="drow">
        </div>
        <div id="row2" class="drow">
        </div>
      </div>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
</html>
...

```

这个例子乍一看很复杂，其实不然。首先我使用getJSON方法得到mydata.json文件的内容（花卉产品数据），然后使用数据模板插件生成需要的元素，并把这些元素添加到页面。如图14-5所示，这就为发送数据给服务器准备好了必要的环境和数据。你可以看到，我输入了一些值：12枝紫菀（aster）、20枝水仙（daffodil），还有4枝报春花（primula）。

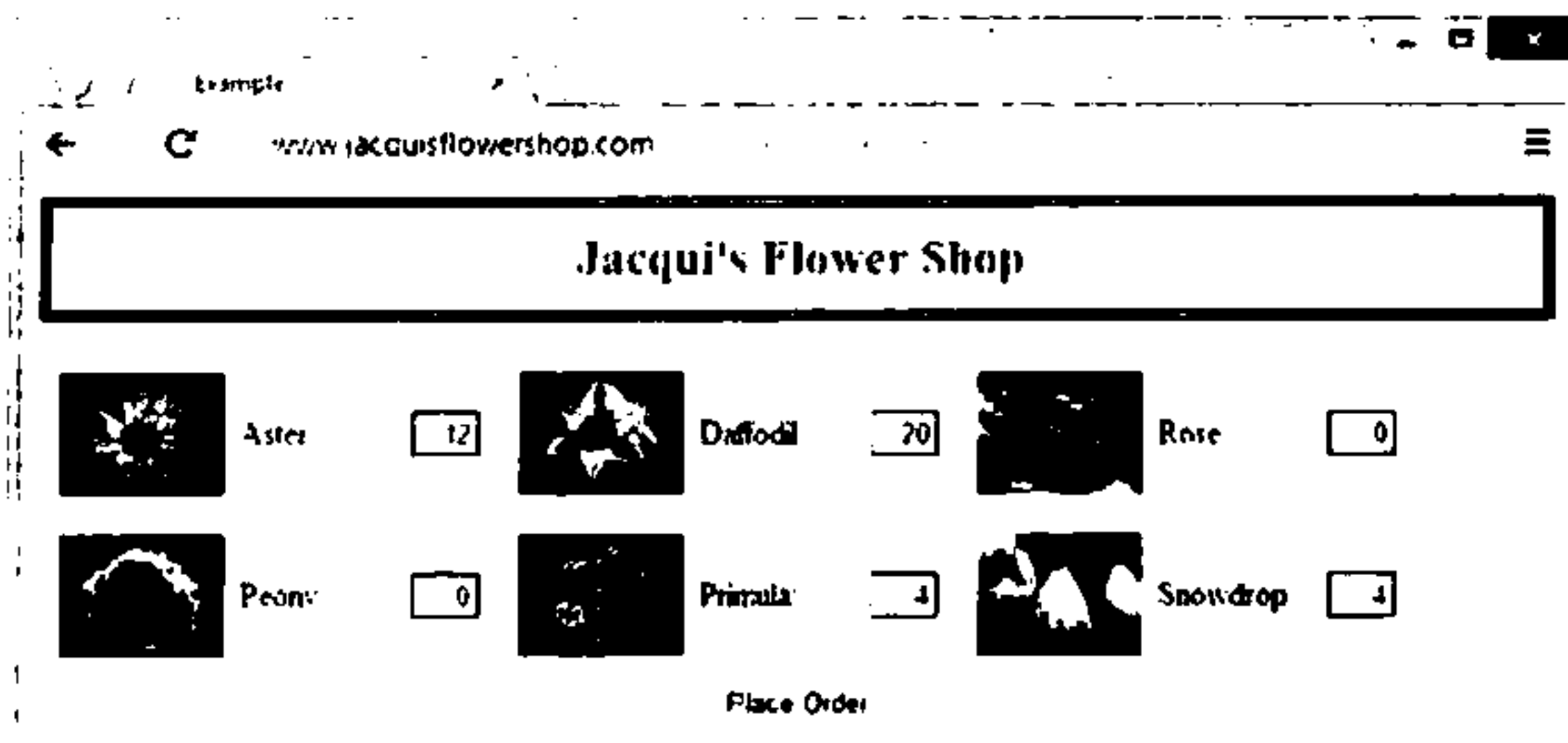


图14-5 一切就绪，准备发送数据给服务器

接着，使用click方法为button元素注册事件处理函数：

```

...
$("#button").click(function (e) {
  var formData = $("#form").serialize();
  $.post("http://node.jacquisflowershop.com/order", formData, processServerResponse);
  e.preventDefault();
});
...

```



在事件处理函数中，我做的第一件事就是在表单元素上调用serialize方法。这是一个非常有用的方法，它会收集所有表单元素的数据，并生成可直接发送到服务器的已经编码的查询字符串。

**提示** 注意，我在事件处理函数内调用了Event对象的preventDefault方法。我必须这么做以阻止浏览器以常规方式提交表单：送出数据，然后在一个单独的HTML页面中载入响应信息。

根据我输入的数据，serialize方法生成查询字符串如下：

```
aster=12&daffodil=20&rose=0&peony=0&primula=4&snowdrop=0
```

我之所以使用serialize方法，是因为post方法发送数据默认要求使用URL编码格式（不过可以通过第15章要讲到的ajaxSetup全局事件处理方法改变这一设置）。一旦准备好来自input元素的数据，我马上调用post方法来初始化Ajax请求。

post方法的第一个参数是我们发送数据的目的地（无需与form元素的action属性对应的URL相同），第二个参数是打算发送的数据，第三个参数是请求成功后需要执行的回调函数。在本例中，我拿到服务器响应数据之后就把响应传递给了processServerResponse函数，这个函数的定义如下：

```
...
function processServerResponse(data) {
    var inputElems = $("div.dcell").hide();
    for (var prop in data) {
        var filtered = inputElems.has("input[name=" + prop + "]")
            .appendTo("#row1").show();
    }
    $("#buttonDiv, #totalDiv").remove();
    $("#totalTpl").tmpl(data).appendTo("body");
}
...
```

在这个函数中，我隐藏了所有的div.dcell元素，然后根据服务器返回的JSON对象的属性显示相应的花卉。我同样使用了数据模板插件来生成选中花卉总数的显示代码。这两项活动都在客户端（浏览器端）完成，关键之处在于数据来自于Ajax POST请求。这个函数的执行结果见图14-6。

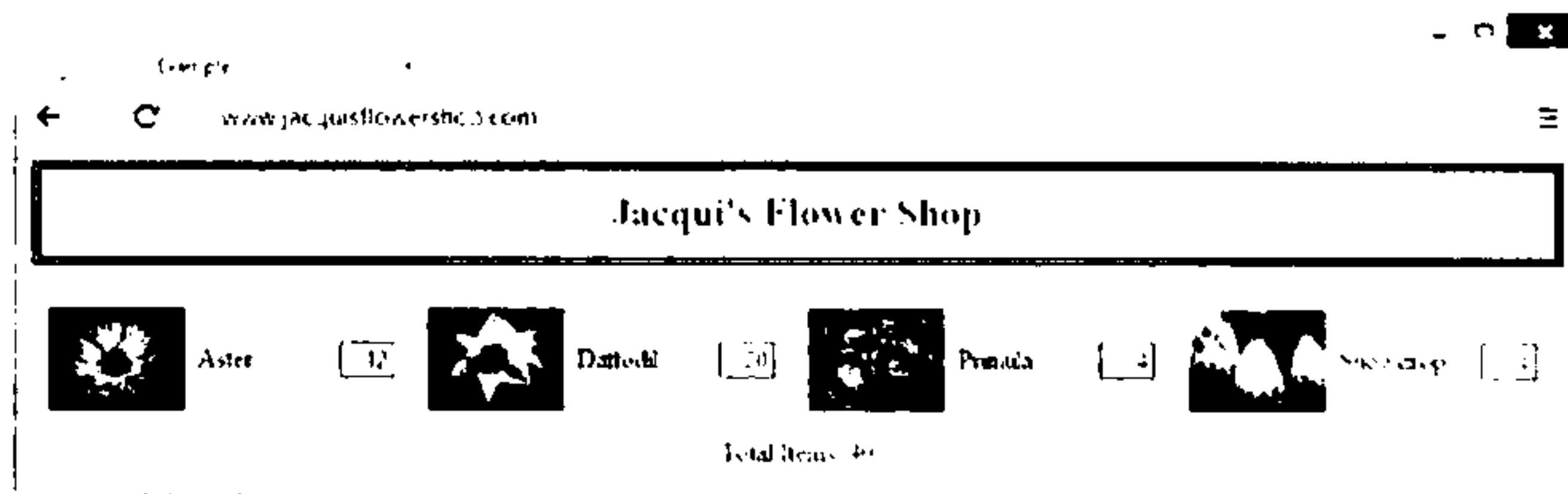


图14-6 处理来自Ajax POST请求的数据之后的显示结果

看到了没，提交数据到服务器是多么容易！（没错，处理服务器响应也这么容易，特别是当服务器响应的是JSON数据时。）



**提示** 如果你的响应结果与图中不符，很可能是因为Node.js脚本的CORS报头中没有设置正确的域（domain）。在报头中把domain设置为\*，然后再运行一次，看看会发生什么。

#### 4. 使用post方法发送其他数据

尽管我们通常使用post方法提交表单数据，但实际上可以用它发送各种类型的数据，只需要生成包含要发送数据的一个对象，然后调用serialize方法对数据格式做适当处理，传递给post方法就行了。

手工收集用户输入数据时（不使用表单）或者希望有选择地提交部分数据时（不是提交所有数据），这一技术非常有用。代码清单14-11展示了post方法的这种用法。

**代码清单14-11 使用post方法发送非表单数据到服务器**

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $('button').click(function(e) {
            var requestData = {
                apples: 2,
                oranges: 10
            };

            $.post("http://node.jacquisflowershop.com/order", requestData,
                function(responseData) {
                    console.log(JSON.stringify(responseData));
                });
            e.preventDefault();
        })
    });
</script>
...
```

在这段脚本中，我创建了一个对象并明确定义了几个属性。然后，我把这个对象传递给post方法，并使用console.log方法显示来自服务器的响应（formserver.js脚本才不关心浏览器提交过来的是什么数据，它只是简单地做加法以生成一个总数）。该脚本在控制台的输出结果如下：

```
{"apples": "2", "oranges": "10", "total": 12}
```

14

**提示** 服务器响应的JSON字符串会被jQuery自动转换为JSON对象。为方便在控制台中显示，我使用JSON.stringify方法（这一方法绝大多数现代浏览器都支持）把JSON对象重新转换为JSON字符串。

### 14.1.3 指定数据类型

在使用get和post方法时，jQuery必须设法搞清服务器端响应数据的类型；有可能是HTML、JavaScript代码，或者其他类型。jQuery根据服务器响应的信息，特别是Content-type报头来判断响应

数据的类型。绝大部分时候，这种机制工作良好，然而有时（通常都是因为Web服务器为响应数据使用了错误的MIME类型）jQuery需要一些提示。

我们可以为get或post方法多提供一个参数，直接告诉jQuery我们需要的数据类型。下面这些值中的任意一个都可以作为这个参数的值：

- ☐ xml
- ☐ json
- ☐ jsonp
- ☐ script
- ☐ html
- ☐ text

代码清单14-12展示了直接在get方法中指定数据类型的方法。

#### 代码清单14-12 指定期待的数据类型

```
...
<script type="text/javascript">
    $(document).ready (function() {
        $.get("mydata.json", function (responseData) {
            console.log(JSON.stringify(responseData));
        }, "json");

    });
</script>
...
```

如你所见，我们用快捷方法的最后一个参数指定服务器响应的数据类型。在这个例子中，我告诉jQuery我们期待JSON数据。jQuery不再关心服务器报告的数据类型，而是直接把响应数据当做JSON数据来处理。这个例子在控制台的输出结果如下：

---

```
[{"name": "Aster", "product": "aster", "stock": "10", "price": "2.99"},
{"name": "Daffodil", "product": "daffodil", "stock": "12", "price": "1.99"},
{"name": "Rose", "product": "rose", "stock": "2", "price": "4.99"},
{"name": "Peony", "product": "peony", "stock": "0", "price": "1.50"},
{"name": "Primula", "product": "primula", "stock": "1", "price": "3.12"},
{"name": "Snowdrop", "product": "snowdrop", "stock": "15", "price": "0.99"}]
```

---

浏览器输出的内容与mydata.json文件一样，这当然是我们期望的结果。然而，如果要指定数据类型，就一定要正确指定。如代码清单14-13所示，如果服务器实际返回的数据并非指定的类型，就会产生问题。

#### 代码清单14-13 指定了错误的数据类型

```
...
<script type="text/javascript">
    $(document).ready(function () {

        $.get("flowers.html", function (responseData) {
```

```
        console.log(JSON.stringify(responseData));
    }, "json");
});
</script>
...
```

在这个例子中，我请求了一个HTML文件，但是告诉jQuery要把HTML内容当成JSON数据处理。问题在于，如果是JSON数据，jQuery自然会根据数据生成JavaScript对象，可是这次服务器返回的数据并非JSON，而是HTML。

---

**提示** 我将在第15章讲解如何检测Ajax错误。

---

## 14.2 避免最常见的 Ajax 陷阱

在进一步学习之前，我要展示一个Web程序员最容易犯的Ajax错误：把异步请求当成同步请求处理。具体示例见代码清单14-14。

代码清单14-14 一个常见的Ajax错误

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var elems;

        $.get("flowers.html", function (data) {
            elems = $(data).filter("div").addClass("dcell");
        });

        elems.slice(0, 3).appendTo("#row1");
        elems.slice(3).appendTo("#row2");
    });
</script>
...
```

在这个例子里，我定义了一个变量elems，接着在Ajax回调函数中把服务器返回数据的处理结果赋值给elems。然后，我使用slice和appendTo方法把elems变量中保存的这些元素添加到页面中。如果运行这个例子，你会在控制台看到一条错误信息。Google Chrome浏览器报告的错误信息是：

---

Uncaught TypeError: Cannot call method 'slice' of undefined

---

问题在于script元素中的语句并不总是按照书写顺序执行，而我假定代码按下列顺序执行：

- (1) 定义elems变量；
- (2) 从服务器获取数据并赋值给elems变量；
- (3) 对elems变量中保存的元素分段并把它们添加到页面。

然而，实际发生的事情是：

- (1) 定义elems变量；
- (2) 开始向服务器发起异步请求；
- (3) 对elems变量中保存的元素分段并把它们添加到页面。

并且，在不确定的一小段时间之后会发生以下事情：

- (1) 从服务器接收响应数据；
- (2) 处理这些数据并赋值给elems变量。

长话短说，之所以发生错误，这是因为我们调用slice方法时变量elems还没有任何元素（因为这一刻elems尚未被赋值）。更坏的事情是，上面的错误代码有时候能够“正常”工作。这是因为偶尔Ajax响应得足够快，因此开始处理elems变量时elems中已经包含了我们期望的元素。（典型的情况是数据已经被浏览器缓存，或者我们在开始处理服务器响应数据之前执行了一些复杂的耗时操作。）现在你应该明白为什么代码会有此种“异常”行为了。

## 14.3 处理特定任务的快捷方法

jQuery提供了3个处理专门任务的快捷方法，简化了这些任务的处理，如表14-4所示。

表14-4 jQuery Ajax特定任务快捷方法

名 称	描 述
load()	载入HTML元素并把这些元素添加到调用load方法的jQuery对象所包含的元素中
\$.getScript()	获取JavaScript代码，并执行这些代码
\$.getJSON()	获取JSON数据

### 14.3.1 获取HTML片段

load方法仅用来获取HTML数据，允许我们从服务器端拿到HTML片段，并把它处理成一系列元素，只需一步操作便能把它们添加到页面中。代码清单14-15演示了load方法的用法。

代码清单14-15 使用load方法

```
...
<script type="text/javascript">
    $(document).ready(function () {
        $("#row1").load("flowers.html");
    });
</script>
...
```

我们选中作为容器的元素，然后调用load方法从服务器端载入HTML数据。如果请求成功完成，并且服务器响应的确实是有有效的HTML数据，如图14-7所示，这些数据就会被插入到容器元素内。

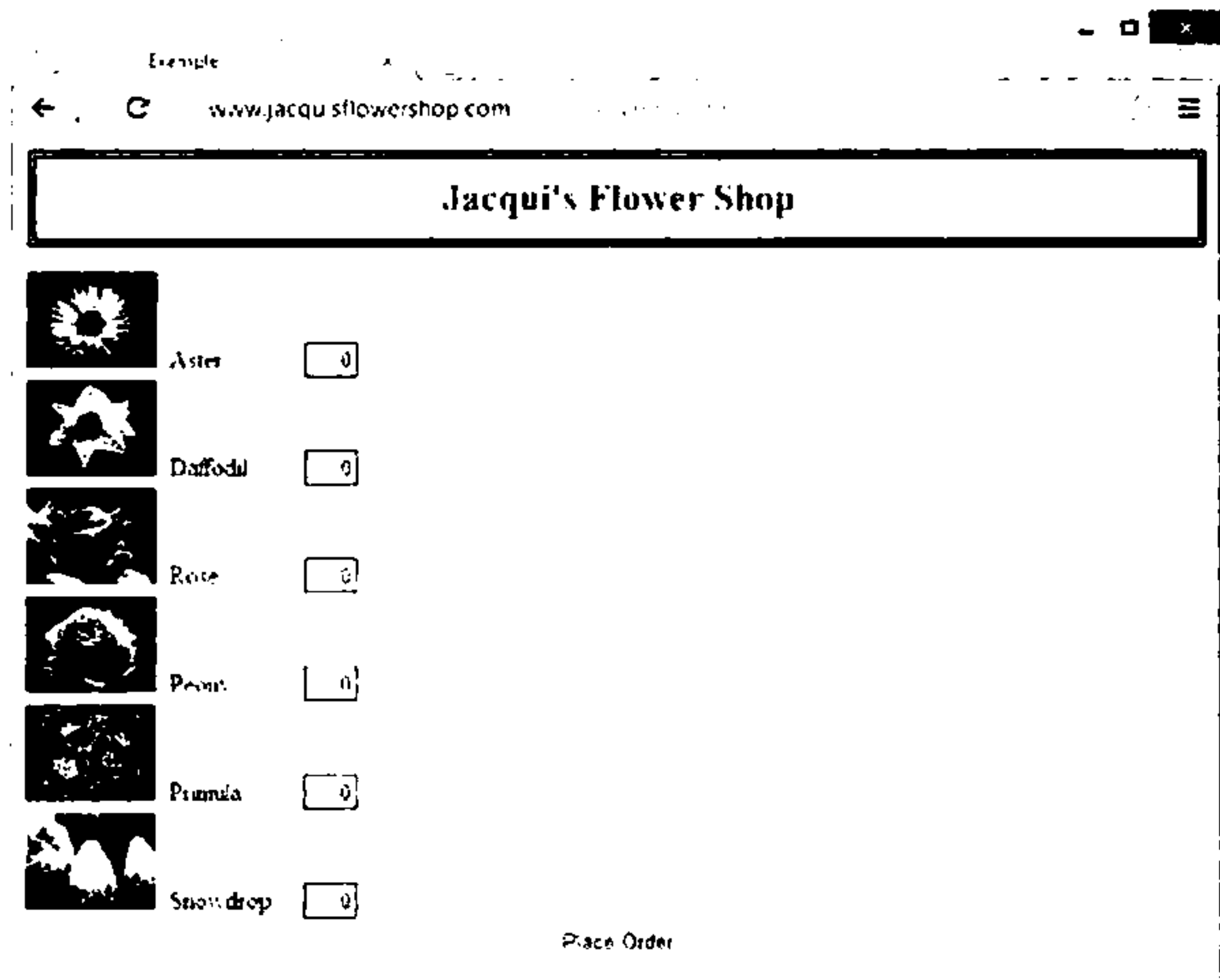


图14-7 使用load方法在页面中添加新元素

flower.html的内容已经添加到页面中。然而，由于它们缺少class属性，其在页面中的布局有些问题。因此，在所有元素都需要插入到单一位置，并且在添加到页面之前无需对内容做任何修改的场合，特别适合使用load方法。

#### 处理load方法载入的元素

load方法会返回包含着表示载入HTML元素的jQuery对象，这里有一个关键点，即load方法采用异步请求获取这些HTML元素。也就是说，如果你打算对load方法载入并插入到DOM中的这些HTML元素做些处理，就得加倍小心，因为常规的jQuery技术在这里不可行。代码清单14-16展示了在实际项目中使用load方法时最常见的一个问题。

#### 代码清单14-16 load方法常见陷阱

```
...
<script type="text/javascript">
    $(document).ready(function () {
        $("#row1").load("flowers.html").children().addClass("dcell");
    });
</script>
...
```

这段代码的目标很明确：使用load方法载入flowers.html的内容，然后选中新添加的元素，为它们添加dcell类（作为表格布局的一部分，让它们水平排列）。

然而，如果你运行这个例子，就会发现这些代码并没有发挥作用（见图14-7）。这是因为load方法采用异步方式获取数据，无需等待获取数据的请求完成，我们的jQuery代码就已执行。因此，Ajax请

求尚未完成（新数据尚未插入DOM），选择所有子元素并添加dcell类的任务已经完成。之后，新元素才插入到DOM之中（这样它们就没有机会添加dcell类了）。

为解决这个问题，load方法提供了一个可选的回调函数参数。当且仅当Ajax请求完成，新元素插入到DOM之后，这个回调函数参数对应的函数才会执行。这就确保了我们的修改操作能够顺利完成。修正之后的代码见清单14-17。

代码清单14-17 load方法的回调函数参数

```
...
<script type="text/javascript">
    $(document).ready(function () {
        var targetElems = $("#row1");
        targetElems.load("flowers.html", function () {
            targetElems.children().addClass("dcell");
        });
    });
</script>
...
```

结果如何？选择子元素并添加类的操作直到flowers.html中的内容载入并插入到DOM之后才执行。这个例子的实际效果见图14-8。

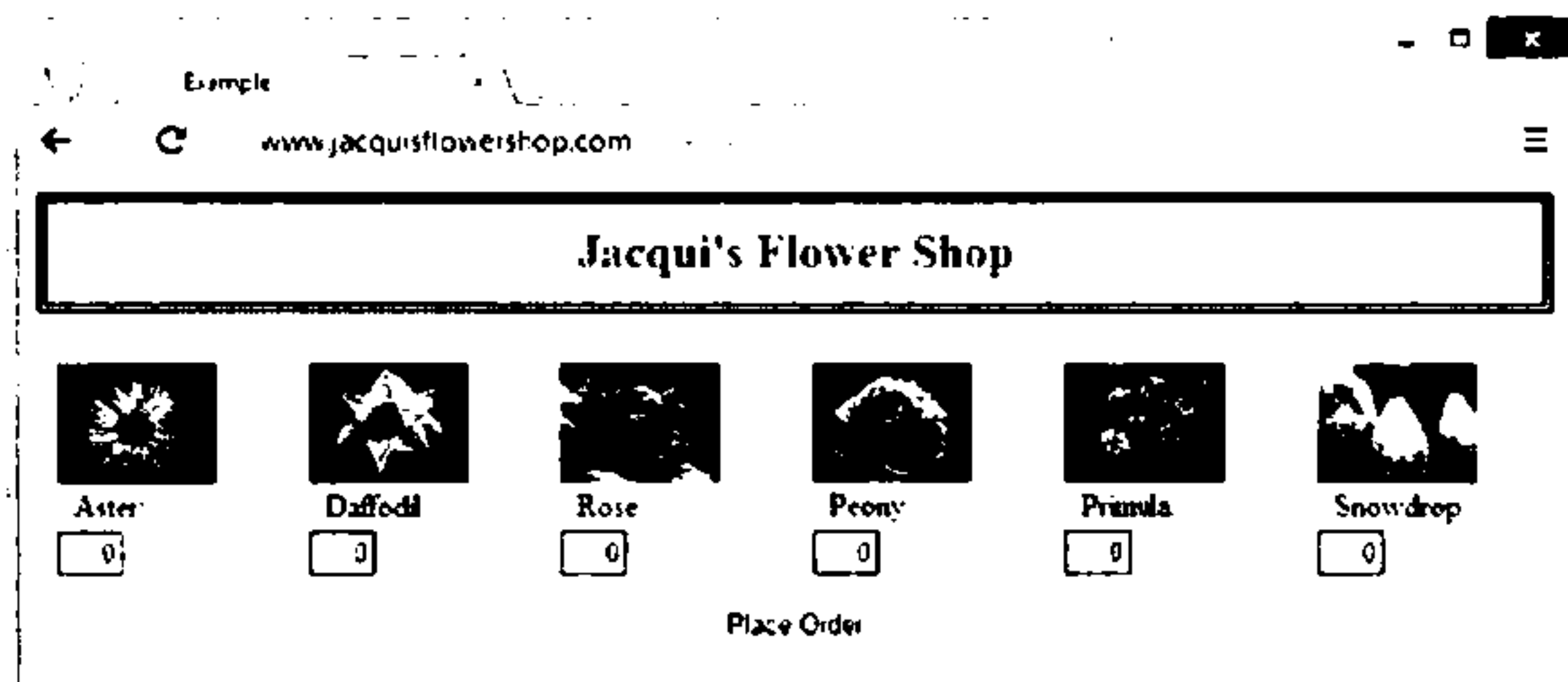


图14-8 使用回调函数处理load方法添加到DOM中的元素

### 14.3.2 获取并执行脚本

getScript方法用来载入JavaScript文件，并执行其中的语句。为了演示这个方法，我创建了一个myscript.js文件，并把它保存在服务器上example.html文件所在目录之下。代码清单14-18展示了这个文件的内容。

代码清单14-18 myscript.js文件的内容

```
var flowers = [
    ["Aster", "Daffodil", "Rose"],
    ["Peony", "Primula", "Snowdrop"],
    ["Carnation", "Lily", "Orchid"]
];
```

```

$("<div id=row3 class=drow/>").appendTo("div.dtable");

var fTemplate = $("<div class=dcell><img/><label/><input/></div>");

for (var row = 0; row < flowers.length; row++) {
    var fNames = flowers[row];

    for (var i = 0; i < fNames.length; i++) {
        fTemplate.clone().appendTo("#row" + (row + 1)).children()
            .filter("img").attr("src", fNames[i] + ".png").end()
            .filter("label").attr("for", fNames[i]).text(fNames[i]).end()
            .filter("input").attr({name: fNames[i], value: 0})
    }
}

```

这些语句用来生成3行展示花卉的元素。我采用循环的方式生成这些元素，这样就不必调用预定义模板了。（一般情况下，我更喜欢使用第12章中讲到的数据模板。）可以使用getScript方法载入并执行myscript.js文件的内容，如代码清单14-19所示。

#### 代码清单14-19 使用getScript方法

```

...
<script type="text/javascript">
    $(document).ready(function () {
        $.getScript("myscript.js");
    });
</script>
...

```

只有当DOM准备好之后，我们才调用getScript方法。如图14-9所示，载入并执行myscript.js后，页面中增加了三行花卉元素。

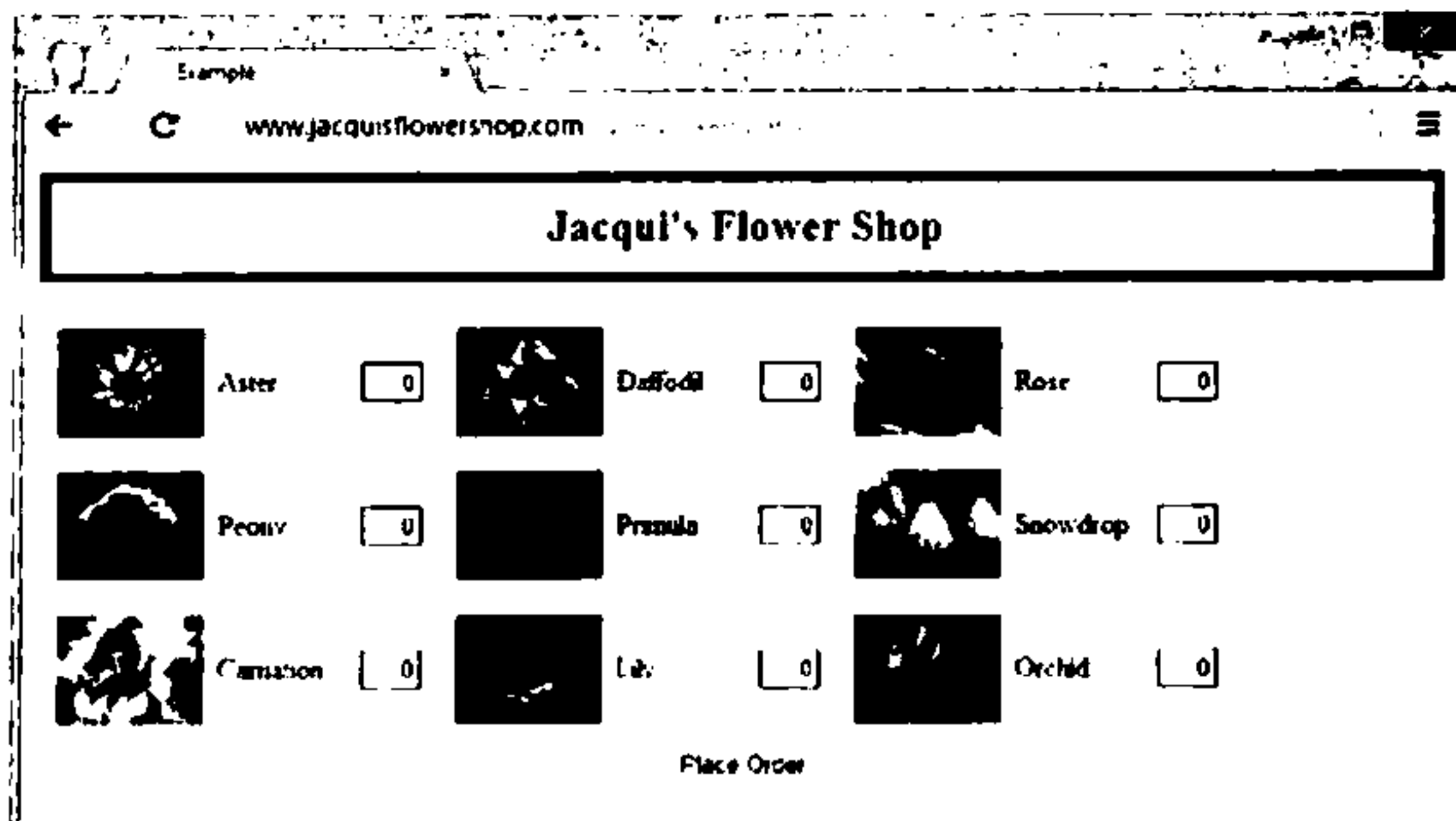


图14-9 使用getScript方法载入并执行JavaScript文件

像这样处理脚本时，我们需要了解一个最为重要的事实：从初始化Ajax请求到脚本语句执行期间，页面状态可能会发生变化。代码清单14-20提供了一个这样的例子：主页面脚本使用getScript方法载入



了脚本，在Ajax请求完成之前还修改了DOM。

#### 代码清单14-20 使用getScript方法请求并执行脚本

```
...
<script type="text/javascript">
    $(document).ready(function () {
        $.getScript("myscript.js");
        $("#row2").remove();
    });
</script>
...
```

**提示** getScript方法可用于载入任意JavaScript脚本文件，不过我发现它特别适合用来载入并执行那些在Web应用中提供边缘功能的脚本文件，比如说统计脚本或者与获取用户地理位置有关的脚本。用户并不关心这些脚本是否放到了正确的位置（站点统计系统才关心这个），而只关心那些必须执行的功能脚本的加载时间。利用getScript方法，我们不必阻塞用户，一样能够得到需要的信息。有一点你必须清楚，我并不是建议你做什么事情都背着用户，而是对于那些适合延迟载入和执行的功能（时间价值超过功能价值）来说，getScript再合适不过了。

在这个例子里，前一行代码使用getScript方法发起Ajax请求载入并执行myscript.js，后一行代码用remove方法从页面中删除了#row2元素，然而myscript.js文件需要利用#row2元素插入一些新元素。

结果，由于#row2选择器不再匹配页面中的任何元素，这些新元素就被浏览器悄无声息地忽略掉了。这个例子的效果参见图14-10。根据不同的应用场景，我们可以把这种面对各种页面变化仍尽力工作视为一种健壮的设计，也可能会为这种一声不吭的失败而感到气恼。不管怎么说，在外部JavaScript文件中不应对页面状态做过多的假设。

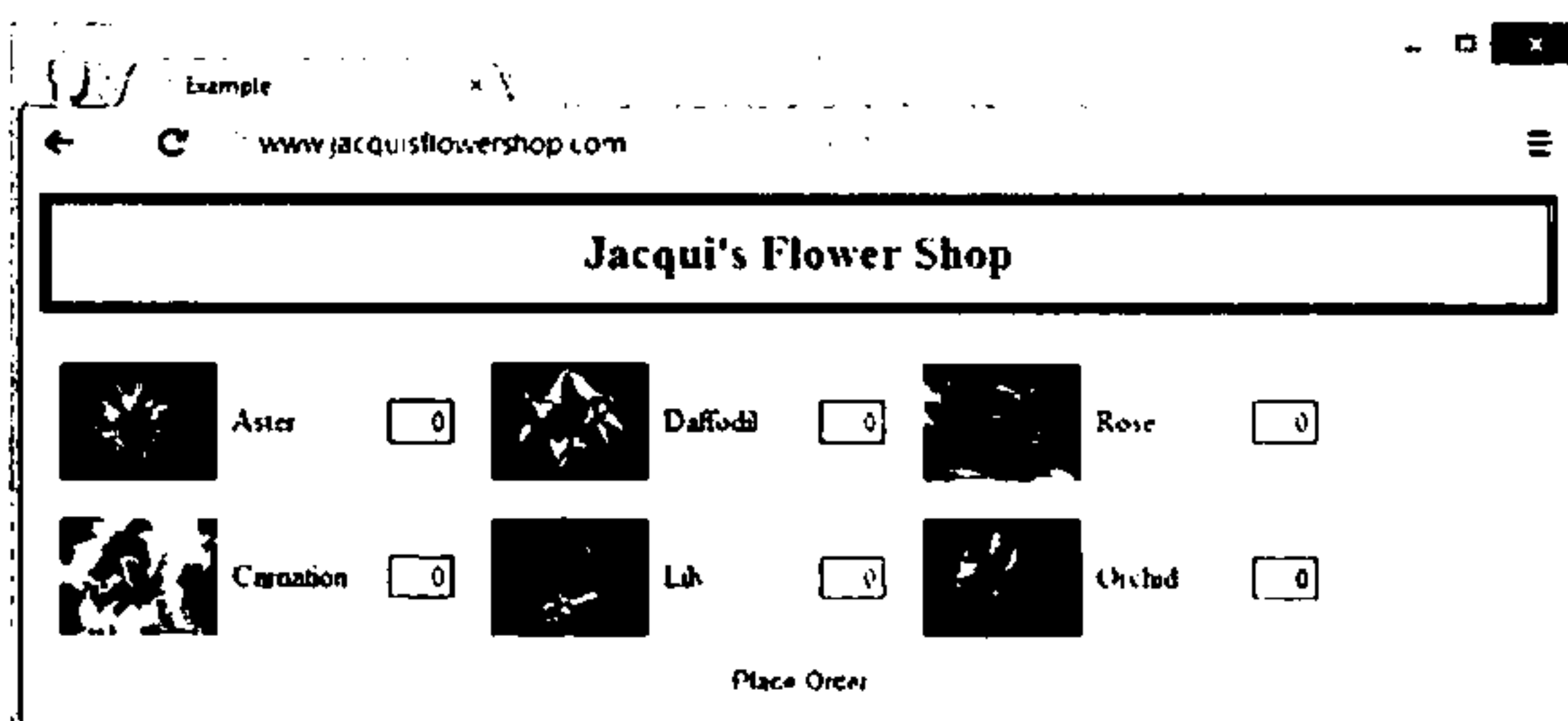


图14-10 Ajax请求期间页面发生变化产生的影响

### 14.3.3 获取JSON数据

getJSON方法用来从服务器获取JSON数据。它可能是3个快捷方法中使用频率最低的一个，因为它并没有比基本的get方法多做任何事情。代码清单14-21演示了getJSON方法的用法。



## 代码清单14-21 使用getJSON方法

```

...
<script type="text/javascript">
    $(document).ready(function () {
        $.getJSON("mydata.json", function (data) {
            var template = $("#flowerTpl").template({ flowers: data }).filter("");
            tmplElems.slice(0, 3).appendTo("#row1");
            tmplElems.slice(3).appendTo("#row2");
        });
    });
</script>
...

```

服务器返回的JSON数据被传递给回调函数，几乎与本章前面使用的get方法一模一样。我使用了数据模板插件（详见第12章）来处理得到的数据并生成HTML元素，然后调用slice和appendTo方法把它们添加到页面中。

---

**提示** 回调函数的参数已经是一个JavaScript对象。你不需要做任何转换工作，jQuery已经自动把JSON字符串转换为JSON对象。

---

## 使用JSONP

JSONP是CORS的替代方案，不同之处在于它支持那些仅允许同源Ajax请求的环境。它依赖于这样一个事实，即浏览器允许我们从任意服务器上载入JavaScript代码，这正是script元素src属性的工作方式。要使用JSONP，首先要在页面中定义一个处理数据的函数：

```

...
function processJSONP(data) {
    //处理数据
}
...

```

然后我们向服务器发起请求，并在查询字符串中加上表单数据及callback属性，把callback的值设置为刚才定义的函数名，就像下面这样：

```

http://node.jacquisflowershop.com/order?callback=processJSONP&aster=1
&daffodil=2&rose=2&peony=0&primula=0&snowdrop=0

```

服务器（需要知道这是一个JSONP请求）像往常一样生成JSON数据，然后创建一条JavaScript语句，调用刚才定义的函数，并把数据作为这个函数的参数：

```
processJSONP({"aster": "1", "daffodil": "2", "rose": "2", "total": 5})
```

服务器同时还把响应内容的content-type设置为text/javascript，这样浏览器就知道收到的是JavaScript代码，应该执行它们。这样实际上调用了我们之前定义的函数，而且以服务器返回的数据作为参数。就这样，我们无需使用CORS，一样完美地绕过了同源问题。

---

**警告** 限制跨源请求的理由非常充分。不要随意使用JSONP，它有可能造成严重的安全问题。

---

jQuery对JSONP的支持相当完美。我们只需要在使用getJSON方法时在URL的查询字符串中增加callback=?。jQuery会自动生成一个随机名字的回调函数，并使用它和服务器进行通信，也就是说你根本不需要修改代码。代码清单14-22演示了如何发起JSONP请求。

代码清单14-22 使用getJSON方法发起JSONP请求

```
...
<script type="text/javascript">
  $(document).ready(function () {
    $.getJSON("mydata.json", function (data) {
      var template = $("#flowerTpl").tempalte({ flowers: data }).filter("*");
      tmplElems.slice(0, 3).appendTo("#row1");
      tmplElems.slice(3).appendTo("#row2");
    });

    $("#button").click(function (e) {
      var formData = $("#form").serialize();
      $.getJSON("http://node.jacquisflowershop.com/order?callback=?",
        formData, processServerResponse)
      e.preventDefault();
    })

    function processServerResponse(data) {
      var inputElems = $("#div.dcell").hide();
      for (var prop in data) {
        var filtered = inputElems.has("input[name=' + prop + ']")
          .appendTo("#row1").show();
      }
      $("#buttonDiv, #totalDiv").remove();
      $("#totalTpl").tmpl(data).appendTo("body");
    }
  });
</script>
...
```

## 14.4 Ajax Forms 插件

目前为止，我一直使用jQuery内建的Ajax支持。前面说过，易于扩展和添加新功能是jQuery最伟大的功能之一，这直接催生出jQuery无比活跃的插件世界。在结束本章之前，我简要地向大家介绍一个有用的表单插件。

如果你对使用Ajax提交表单数据感兴趣，应该会喜欢上jQuery Forms插件，下载地址是[www.malsup.com/jquery/form](http://www.malsup.com/jquery/form)，我把它保存为jquery.form.js。如代码清单14-23所示，这个插件简化了在表单上应用Ajax的工作。

代码清单14-23 使用Ajax Forms插件

```
<!DOCTYPE html>
<html>
```

```

<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js" type="text/javascript"></script>
  <script src="handlebars-jquery.js" type="text/javascript"></script>
  <script src="jquery.validate.js" type="text/javascript"></script>
  <script src="jquery.form.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}</label>
      <input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
        value="0" required />
    </div>
    {{/flowers}}
  </script>
  <script id="totalTpl" type="text/x-handlebars-template">
    <div id="totalDiv" style="clear: both; padding: 5px">
      <div style="text-align: center">Total Items:
        <span id="total">{{total}}
      </div>
    </div>
  </script>
  <script type="text/javascript">
    $(document).ready(function () {

      $.getScript("myscript.js");

      $("form").ajaxForm(function (data) {
        var inputElems = $("div.dcell").hide();
        for (var prop in data) {
          var filtered = inputElems.has("input[name=' + prop + ']")
            .appendTo("#row1").show();
        }
        $("#buttonDiv, #totalDiv").remove();
        $("#totalTpl").template(data).appendTo("body");
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="oblock">
      <div class="dtable">
        <div id="row1" class="drow">
        </div>
        <div id="row2" class="drow">
        </div>
      </div>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>

```

```
</body>  
</html>
```

在这个例子中，我在页面中添加了脚本文件jquery.form.js（在该插件下载包里有这个文件），而且在script元素里选中表单元素并调用了ajaxForm方法。ajaxForm方法的参数是一个回调函数，它用来处理来自服务器的数据。这是实现简单Ajax表单最简洁的方案，实际上它直接将表单元素的action属性作为POST URL。

这个插件做了很多事情，甚至包括一些基本的表单验证功能。不过，如果希望精确控制自己的Ajax请求，我建议你使用第15章中讲到的Ajax底层函数。如果是一些需要快速完成的工作，使用这个精心设计的插件是非常方便的。

## 14.5 小结

本章介绍了jQuery提供的一些Ajax快捷方法。我不但演示了使用get和post方法发起异步请求的方法，还讲解了JSON数据的处理办法，以及如何使用快捷方法处理特定的数据类型。同时，我还演示了Ajax应用中最常见的陷阱，解释了什么是跨源请求以及如何处理跨源请求，并于最后简要介绍了一个jQuery插件，它使得使用Ajax处理表单比使用快捷方法还要容易。下一章，我将带大家学习Ajax底层API。你很快就会看到，实际上它并没有那么底层，其使用体验也是相当得好。

本章的主题是jQuery Ajax底层API。术语“底层”暗示着我们将深入请求的内部细节，其实并非如此。本章讲到的API没有第14章中的那些方便，不过是在快捷方法和便捷方法不太能胜任时，为满足需要而必须多做一点配置工作而已。表15-1列出了本章概要。

表15-1 本章概要

问 题	解决方法	代码清单
如何使用底层API发起Ajax请求	使用ajax方法	1
如何像原生XMLHttpRequest对象那样得到请求的细节	使用jqXHR对象	2
如何指定Ajax请求的目标URL	使用url选项	3
如何指定Ajax请求使用的HTTP方法	使用type选项	4
如何响应成功的请求	使用success选项	5
如何响应出错的请求	使用error选项	6
如何响应已经完成的请求（不论是成功还是失败）	使用complete选项	7、8
如何在请求发出之前配置请求	使用beforeSend选项	9
如何为成功完成、出错或者已经完成的请求指定多个回调函数	分别为success、error或complete选项指定一个函数数组	10
如何指定一个元素作为success、error、complete回调函数中的this变量的值	使用context选项	11
如何响应所有Ajax请求中发生的事件	使用Ajax全局事件方法	12
如何明确指定一个请求是否触发全局事件	使用global选项	13
如何指定请求时间的超时选项	使用timeout选项	14
如何为一个请求添加指定报头	使用headers选项	14
如何指定发送给服务器的内容类型	使用contentType报头	15
如何指定一个请求以异步还是同步方式发送	使用async选项	16
如何在必要时忽略重复数据	使用ifModified选项	17
如何响应服务器返回的HTTP状态码	使用statusCode选项	18
如何净化服务器响应数据	使用dataFilter选项	19
如何控制响应数据的转换方式	使用converters选项	20

(续)

问 题	解决方法	代码清单
如何为所有Ajax请求设置全局配置	使用ajaxSetup方法	21
如何为某个请求动态改变某些选项	使用ajaxPrefilter方法	22

### 自本书上一版本至今jQuery发生的变化

自jQuery1.9/2.0版本开始, 设定Ajax全局事件仅可以在document对象上调用(之前的版本可在任意元素上调用)。如需详细了解这些方法, 请参阅15.6节。

## 15.1 使用底层 API 发起简单的 Ajax 请求

与第14章讲到的快捷方法相比, 使用底层API发起Ajax请求并没有复杂多少。区别在于我们能够针对请求的方方面面进行配置, 而且可以得到当前请求更详细的信息。底层API的核心方法是ajax。代码清单15-1演示了ajax方法的用法。

代码清单15-1 使用ajax方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js" type="text/javascript"></script>
  <script src="handlebars-jquery.js" type="text/javascript"></script>
  <script src="jquery.validate.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}</label>
      <input name="{{product}}" data-price="{{price}}" data-stock="{{stock}}"
        value="0" required />
    </div>
    {{/flowers}}
  </script>
  <script type="text/javascript">
    $(document).ready(function () {
      $.ajax("mydata.json", {
        success: function (data) {
          var tplElems = $("#flowerTpl");
            .template({flowers: data}).filter("*");
          tplElems.slice(0, 3).appendTo("#row1");
          tplElems.slice(3).appendTo("#row2");
        }
      });
    });
  </script>
</head>
<body>
  <div id="row1">
  </div>
  <div id="row2">
  </div>
</body>
</html>
```

```

</script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="oblock">
      <div class="dtable">
        <div id="row1" class="drow">
        </div>
        <div id="row2" class="drow">
        </div>
      </div>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
</html>

```

在这个例子中，ajax方法的第一个参数是要请求的URL，第二个参数是定义了许多属性的（键值对）映射对象，它的每一个属性都对应着一个请求选项。

**注意** 本章代码依赖于第14章中使用的Node.js脚本。

在这个例子里，参数对象只有一个选项：success。success选项指定请求成功完成时执行的回调函数。在本例中，和上一章中使用快捷方法的做法一样，我向服务器请求mydata.json文件，并利用请求得来的数据data与数据模板生成需要的元素，然后把这些元素添加到页面中。默认情况下，ajax方法向服务器发出HTTP get请求，也就是说这个例子大致等同于第14章中的get或者getJSON方法。（在15.4节我会介绍如何发起POST请求。）

ajax方法的选项非常多，我会利用本章后面的篇幅讲解这些选项。除此之外，jQuery还提供了许多有用的方法，这些方法简化了Ajax的使用。

## 15.2 理解 jqXHR 对象

ajax方法会返回一个jqXHR对象，利用这个对象我们可以了解请求的细节信息并与它交互。jqXHR是XMLHttpRequest对象（由W3C标准定义的浏览器端实现Ajax功能的底层对象）的一个超集，它能够与jQuery延迟对象协同工作（详见第35章）。

对于绝大多数Ajax请求，我们完全可以忽略jqXHR对象的存在。只有在需要的信息（服务器响应信息）无法使用其他方式获得的情况下，我们才会用到jqXHR对象。表15-2列出了jqXHR对象的成员。

表15-2 jqXHR对象成员

成 员	描 述
readyState	返回请求进度（生命周期），从未发送（值为0）到请求完成（值为4）
status	返回服务器响应的HTTP状态码
statusText	返回HTTP状态码的文本描述

(续)

成 员	描 述
responseXML	若服务器返回数据是XML的话，在该属性中保存着相应的XML对象
responseText	返回文本形式的服务器响应数据
setRequestHeader(name, value)	为请求设置一个报头
getAllResponseHeaders()	返回字符串形式的服务器响应报头
getResponseHeader(name)	返回指定服务器响应信息中指定报头的值
abort()	终止当前请求

**提示** jqXHR对象可以用来配置Ajax请求，但是使用ajax方法的配置选项会更容易实现，后面我会作进一步的解释。

使用jQuery时，我们会在一些场合遇到jqXHR对象。首先，如代码清单15-2所示，ajax方法的返回值就是一个jqXHR对象。

**代码清单15-2 使用jqXHR对象**

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var jqxhr = $.ajax("mydata.json", {
            success: function (data) {
                var tmpLElems = $("#flowerTpl").template({flowers: data}).filter("*");
                tmpLElems.slice(0, 3).appendTo("#row1");
                tmpLElems.slice(3).appendTo("#row2");
            }
        });

        var timerID = setInterval(function () {
            console.log("Status: " + jqxhr.status + " " + jqxhr.statusText);
            if (jqxhr.readyState == 4) {
                console.log("Request completed: " + jqxhr.responseText);
                clearInterval(timerID);
            }
        }, 100);

    });
</script>
...
```

在这个例子中，我把ajax方法的返回值保存在jqxhr变量中，然后使用setInterval方法每隔100 ms输出一次请求信息。在处理jqXHR对象时，我们一定要保持警惕，记住：利用ajax方法的返回值这种做法并不会改变请求是异步的这一事实。我使用readyState属性检查请求的状态（值4表示请求顺利完成），并在请求完成后把服务器响应信息输出到控制台。这个脚本会输出以下信息（如果你使用的浏览器与我的不同，看到的信息可能会有细微差别）：



---

Status: 200 OK

```
Request completed: [{"name": "Aster", "product": "aster", "stocklevel": "10", "price": "2.99"},
{"name": "Daffodil", "product": "daffodil", "stocklevel": "12", "price": "1.99"},
{"name": "Rose", "product": "rose", "stocklevel": "2", "price": "4.99"},
{"name": "Peony", "product": "peony", "stocklevel": "0", "price": "1.50"},
{"name": "Primula", "product": "primula", "stocklevel": "1", "price": "3.12"},
{"name": "Snowdrop", "product": "snowdrop", "stocklevel": "15", "price": "0.99"}]
```

---

**提示** 我很少使用jqXHR对象，而作为ajax方法返回值的jqXHR对象我一次也没有使用过（这次除外）。如果需要使用jqXHR对象（典型应用场景以此得到来自服务器的额外信息），我通常会使用有关的事件处理函数（15.5节有详细讲解）。这些事件处理函数会根据请求状态提供相应的上下文信息，而且我也不必费气力查询请求状态。

---

## 15.3 设置请求 URL

要设置请求URL，除了把请求URL作为ajax方法的第一个参数之外，还可以有另一种方法，那就是在映射对象中定义一个url属性，见代码清单15-3。

代码清单15-3 使用url选项

```
...
<script type="text/javascript">
  $(document).ready (function () {
    $.ajax({
      url: "mydata.json",
      success: function (data) {
        var tmplElems = $("#flowerTpl").template({flowers: data}).filter("*");
        tmplElems.slice(0, 3).appendTo("#row1");
        tmplElems.slice(3).appendTo("#row2");
      }
    });
  });
</script>
...
```

## 15.4 发起 POST 请求

type选项用来设置Ajax请求使用的HTTP方法。前面的例子已经表明，Ajax请求默认使用GET请求。代码清单15-4展示了如何使用ajax方法发起POST请求并提交表单数据到服务器。

代码清单15-4 使用ajax方法发起POST请求

```
...
<script id="totalTpl" type="text/x-handlebars-template">
```

```

    <div id="totalDiv" style="clear: both; padding: 5px">
        <div style="text-align: center">Total Items:
            <span id="total">{{total}}
        </div>
    </script>
    <script type="text/javascript">
        $(document).ready(function () {
            $.ajax({
                url: "mydata.json",
                success: function (data) {
                    var tplElems = $("#flowerTpl").template({flowers: data}).filter("");
                    tplElems.slice(0, 3).appendTo("#row1");
                    tplElems.slice(3).appendTo("#row2");
                }
            });

            $("button").click(function (e) {
                $.ajax({
                    url: $("form").attr("action"),
                    data: $("form").serialize(),
                    type: "post",
                    success: processServerResponse
                });
                e.preventDefault();
            })

            function processServerResponse(data) {
                var inputElems = $("div.dcell").hide();
                for (var prop in data) {
                    var filtered = inputElems.has("input[name=' + prop + ']")
                        .appendTo("#row1").show();
                }
                $("#buttonDiv, #totalDiv").remove();
                $("#totalTpl").template(data).appendTo("body");
            }
        });
    </script>
    ...

```

除了type选项, 在这个例子中我还使用了其他几个选项。为指定POST请求的目标URL, 我使用了url属性, 该属性可以从页面中form元素的action属性中得到。我在form元素上调用serialize方法得到需要发送给服务器的数据, 并把这些数据用作data属性的值, serialize方法的详细解释见第33章。

### GET/POST之外的其他方法

我们可以使用type属性指定任意合法的HTTP请求方法, 不过使用GET/POST之外的其他方法可能会遇到问题, 这主要是因为许多防火墙和应用程序服务器都被配置成忽略其他类型的请求。如果确实希望使用其他HTTP方法, 你可发起POST请求, 同时添加一个X-HTTP-Method-Override报头, 把这个报头的值设置为真正希望使用的方法, 比如:

```
X-HTTP-Method-Override: PUT
```

几乎所有Web应用程序框架都支持这种转换，这也是创建REST式Web应用程序的一种通用做法。如果你希望详细了解这一技术，我推荐维基百科上的一篇文章：[http://en.wikipedia.org/wiki/Representational\\_state\\_transfer](http://en.wikipedia.org/wiki/Representational_state_transfer)。欲详细了解如何为一个jQuery Ajax请求设置报头，请参阅15.7.1节。

## 15.5 Ajax 事件

ajax方法支持一些属性，利用这些属性，我们可以在Ajax请求生存期内的一些关键时间点指定回调函数。在前面的例子里，我们已经用过一个属性：`success`。所有可用来设置回调函数的属性见表15-3。

表15-3 Ajax事件有关选项

选 项	描 述
<code>beforeSend</code>	指定在Ajax请求发起之前执行的函数
<code>complete</code>	指定Ajax请求成功结束或者失败后执行的函数
<code>error</code>	指定Ajax请求失败之后执行的函数
<code>success</code>	指定Ajax请求成功完成之后执行的函数

**提示** 表15-3中的列出的这些选项针对的都是“局部”（针对具体Ajax请求的）回调，我们还可以设置一系列“全局”（针对所有Ajax请求的）事件。我会在15.6节详细讲解全局事件。

### 15.5.1 处理成功完成的请求

在前面演示`success`选项的用法时，我省略了回调函数的两个参数，一个是描述请求结果状态文本信息的`status`参数，一个是jqXHR对象。代码清单15-5演示了在回调函数中同时处理3个参数的方法。

代码清单15-5 在`success`回调函数中处理所有参数

```
...
<script type="text/javascript">
  $(document).ready(function () {
    $.ajax({
      url: "mydata.json",
      success: function(data, status, jqxhr) {

        console.log("Status: " + status);
        console.log("jqXHR Status: " + jqxhr.status + " " + jqxhr.statusText);
        console.log(jqxhr.getAllResponseHeaders());

        var tplElems = $("#flowerTpl").template({flowers: data}).filter("*");
        tplElems.slice(0, 3).appendTo("#row1");
        tplElems.slice(3).appendTo("#row2");
      }
    });
  });
});
```

```
</script>
...
```

`status`参数是一个字符串，它描述本次请求的结果。由于只有Ajax请求成功完成才会执行`success`选项指定的回调函数，因此这个参数的值一般都是`success`。不过，如果你使用了`ifModified`选项（详见15.8.2节），那么这个参数的值就有可能不是`success`。Ajax事件的回调函数都遵循同样的模式，对于某些其他事件，这个参数会更有用。

最后一个参数是`jqXHR`对象。由于我们知道函数仅当Ajax请求成功完成之后才会执行，因此无需轮询Ajax请求的状态。在这个例子中，我使用`jqXHR`对象得到请求的状态信息以及服务器响应的报头信息，并把这些信息输出到控制台。上面的例子会在控制台产生以下输出（由于你使用的Web服务器可能与我的不同，因此看到的报头可能与此有所不同）：

```
Status: success
jqXHR Status: 200 OK
Date: Thu, 20 Jun 2013 12:06:30 GMT
Last-Modified: Wed, 19 Jun 2013 16:29:49 GMT
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET
ETag: "b680cf37a6dce1:0"
Content-Type: application/json
Cache-Control: no-cache
Accept-Ranges: bytes
Content-Length: 405
```

## 15.5.2 处理失败的请求

`error`选项用来指定请求失败后执行的回调函数，具体示例见代码清单15-6。

代码清单15-6 使用`error`选项

```
...
<style type="text/css">
    .error {color: red; border: medium solid red; padding: 4px;
        margin: auto; width: 200px; text-align: center}
</style>
<script type="text/javascript">
    $(document).ready(function () {
        $.ajax({
            url: "NoSuchFile.json",
            success: function (data, status, jqxhr) {
                var tplElems = $("#flowerTpl").template({flowers: data}).filter("*");
                tplElems.slice(0, 3).appendTo("#row1");
                tplElems.slice(3).appendTo("#row2");
            },
            error: function (jqxhr, status, errorMsg) {
                $("<div>").addClass("error")
                    .text("Status: " + status + " Error: " + errorMsg)
                    .insertAfter("h1");
            }
        })
    })
}
```

```

    });
  });
</script>
...

```

在这个例子里，我请求了一个名为NoSuchFile.json的文件，没错，我的Web服务器上根本没有这个文件。因此这个请求注定会失败，error选项中指定的函数会执行。

这个函数接受3个参数，分别是jqXHR对象、描述请求状态的信息，以及服务器返回的出错信息。如图15-1所示，在error回调函数中，我向页面添加了一个div元素，以此显示status参数和errorMsg参数的值。

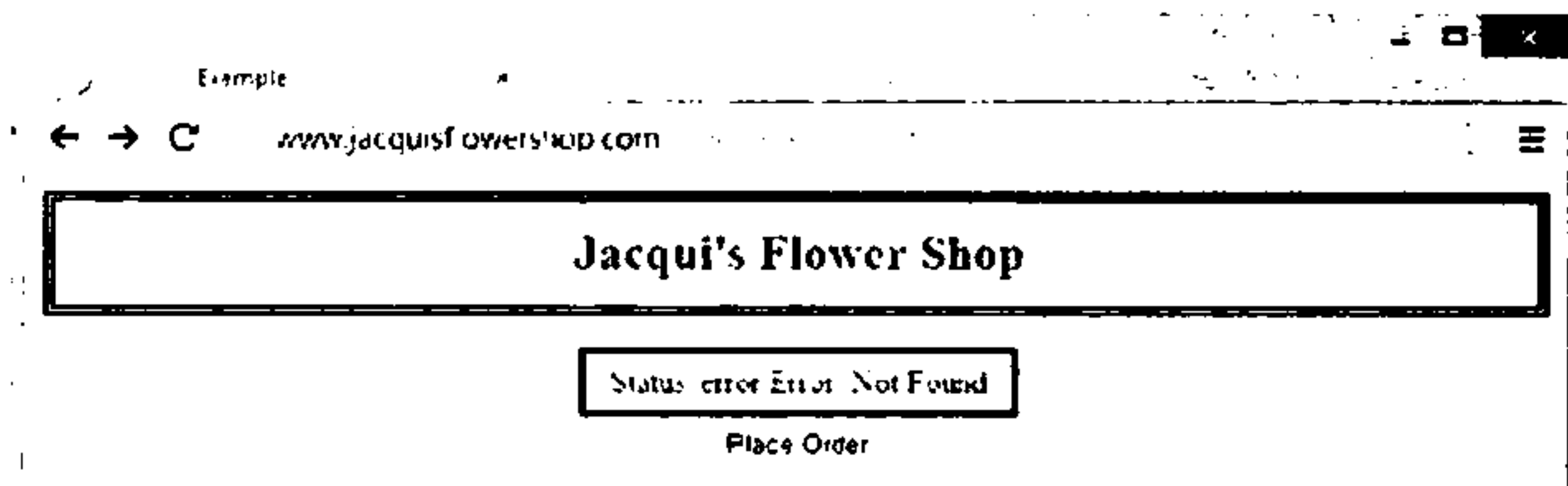


图15-1 显示出错信息

status参数可能的值见表15-4。

表15-4 出错状态参数status可能出现的值

值	描 述
abort	表示请求是jqXHR对象的abort方法终止的
error	表示一般的错误，这类错误通常是由服务器报告的
parsererror	表示服务器返回的数据无法正常解析
timeout	表示在服务器返回信息之前请求已经超时

errorMsg参数的值随status参数的变化而变化。若status的值为error，那么errorMsg的值就是服务器响应信息的文本部分。就上面的例子来说，既然服务器响应的信息是404 Not Found，那么errorMsg的值就是Not Found。

如果status的值是timeout，errorMsg的值就同样是timeout。我们可以使用timeout选项指定请求的最长时间限制，关于这一选项，我会在15.7.1节详细介绍。

如果status的值是parsererror，那么errorMsg参数中保存的就是详细出错信息。这种错误一般是因为服务器返回的数据格式不对，或者服务器返回的数据格式与数据相应的MIME类型不符造成的。我们可以使用ajax方法的dataType选项明确指定jQuery服务器返回的数据格式。最后，如果请求被人为终止，status和errorMsg参数的值就都是abort。

**提示** 尽管我们已经把错误状态和出错信息显示到页面中，但是由于这些信息提示过于专业，而且没有报告产生问题的可能原因，这些信息通常帮不了用户多少忙。

### 15.5.3 处理已经完成的请求

`complete`选项用来指定Ajax请求完成之后需要执行的函数，不管请求是成功完成还是失败，具体示例见代码清单15-7。

代码清单15-7 使用complete选项

```
...
<script type="text/javascript">
    $(document).ready(function () {
        $.ajax({
            url: "mydata.json",
            success: function (data, status, jqxhr) {
                var tplElems = $("#flowerTpl").template({flowers: data}).filter("*");
                tplElems.slice(0, 3).appendTo("#row1");
                tplElems.slice(3).appendTo("#row2");
            },
            error: function (jqxhr, status, errorMsg) {
                $("<div>").addClass("error")
                    .text("Status: " + status + " Error: " + errorMsg)
                    .insertAfter('h1');
            },
            complete: function (jXHR, status) {
                console.log("Completed: " + status);
            }
        });
    });
</script>
...
```

使用complete选项指定的函数会在success和error选项指定的回调函数执行之后执行。jQuery会传递jqXHR对象和status字符串给complete回调函数，其中status可取值见表15-5。

表15-5 status参数可能取值一览表

值	描 述
abort	表示请求是jqXHR对象的abort方法终止的
error	表示一般的错误，这类错误通常是由服务器报告的
notmodified	表示这次请求得到的内容与上一次请求的没有差别（详见15.8.2节）
parsererror	表示服务器返回的数据无法正常解析
success	表示请求成功完成
timeout	表示在服务器返回信息之前请求已经超时

你也许会想要使用complete选项指定一个回调函数，以处理Ajax请求可能遭遇的各种情况。然而如果那样做，因为complete指定的回调函数只支持status和jqxhr两个参数，所以你就没有机会处理服务器返回的数据或者得到具体的出错信息。一个较好的方案（参见代码清单15-8）是定义一个通用函数，然后在success和error选项的回调函数中调用这个通用函数并正确组织调用参数。

代码清单15-8 使用一个函数处理所有情况

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $.ajax({
            url: "mydata.json",
            success: function (data, status, jqxhr) {
                handleResponse(status, data, null, jqxhr);
            },
            error: function (jqxhr, status, errorMsg) {
                handleResponse(status, null, errorMsg, jqxhr);
            }
        });

        function handleResponse(status, data, errorMsg, jqxhr) {
            if (status == "success") {
                var tplElems = $("#flowerTpl").template({ flowers: data }).filter("*");
                tplElems.slice(0, 3).appendTo("#row1");
                tplElems.slice(3).appendTo("#row2");
            } else {
                $("<div>").addClass("error")
                    .text("Status: " + status + " Error: " + errorMsg)
                    .insertAfter('h1');
            }
        }
    });
</script>
...

```

#### 15.5.4 在请求发出之前配置请求

`beforeSend`选项用来指定一个在请求启动之前调用的回调函数。它提供了最后的配置请求的机会，可以用于补充或者覆盖传递给ajax方法的配置选项（尤其适合许多请求使用同样配置的情况）。代码清单15-9演示了这个选项的用法。

代码清单15-9 使用beforeSend选项

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $.ajax({
            url: "NoSuchFile.json",
            success: function (data, status, jqxhr) {
                handleResponse(status, data, null, jqxhr);
            },
            error: function (jqxhr, status, errorMsg) {
                handleResponse(status, null, errorMsg, jqxhr);
            },
            beforeSend: function(jqxhr, settings) {
                settings.url = "mydata.json";
            }
        });
    });

```

```

    });

    function handleResponse(status, data, errorMsg, jqxhr) {
        if (status == "success") {
            var tplElems = $("#flowerTpl").template({ flowers: data }).filter("*");
            tplElems.slice(0, 3).appendTo("#row1");
            tplElems.slice(3).appendTo("#row2");
        } else {
            $("<div>").addClass("error")
                .text("Status: " + status + " Error: " + errorMsg)
                .insertAfter("h1");
        }
    }
});
</script>
...

```

这个函数会收到两个参数：第一个是jqXHR对象；第二个参数是我们传递给ajax方法的配置对象。在这个例子里，我使用了url选项指定本次请求的目标URL，从而覆盖了url属性定义的值。

### 15.5.5 为同一事件指定多个处理函数

前面的示例代码都只为Ajax事件指定一个回调函数。如果我们为success、error、complete和beforeStart选项指定一个函数数组，那么当事件发生时数组中的每个函数都会执行，具体示例见代码清单15-10。

代码清单15-10 为同一事件指定多个处理函数

```

...
<script type="text/javascript">
    $(document).ready(function () {
        $.ajax({
            url: "NoSuchFile.json",
            success: [processData, reportStatus]
        });

        function processData(data, status, jqxhr) {
            var tplElems = $("#flowerTpl").template({ flowers: data }).filter("*");
            tplElems.slice(0, 3).appendTo("#row1");
            tplElems.slice(3).appendTo("#row2");
        }

        function reportStatus(data, status, jqxhr) {
            console.log("Status: " + status + " Result code: " + jqxhr.status);
        }
    });
</script>
...

```

在这个例子里，我为success选项指定了一个拥有两个函数成员的数组，一个使用服务器返回的数据data在页面中添加一些元素，一个则在控制台打印输出一些信息。



### 15.5.6 设置事件上下文对象

context选项用来指定一个对象，这个对象将成为事件处理函数中this变量的值。这样我们就能够直接在事件处理函数中使用这个元素，节省了选择元素的过程，具体示例见代码清单15-11。

代码清单15-11 使用context选项

```
...
<script type="text/javascript">
  $(document).ready(function () {
    $.ajax({
      url: "mydata.json",
      context: $("#h1"),
      success: function (data, status, jqxhr) {
        var tmpLElems = $("#flowerTpl").template({ flowers: data }).filter("*");
        tmpLElems.slice(0, 3).appendTo("#row1");
        tmpLElems.slice(3).appendTo("#row2");
      },
      complete: function(jqxhr, status) {
        var color = status == "success" ? "green" : "red";
        this.css("border", "thick solid " + color);
      }
    });
  });
</script>
...
```

在这个例子中，我把包含着h1元素的jQuery对象设置为context选项的值。在complete指定的回调函数中，我在这个jQuery对象上调用css方法，根据不同的请求结果（成功/失败）为对象中的元素设置了不同的边框样式。在图15-2中分别可以看到成功和失败的请求结果。

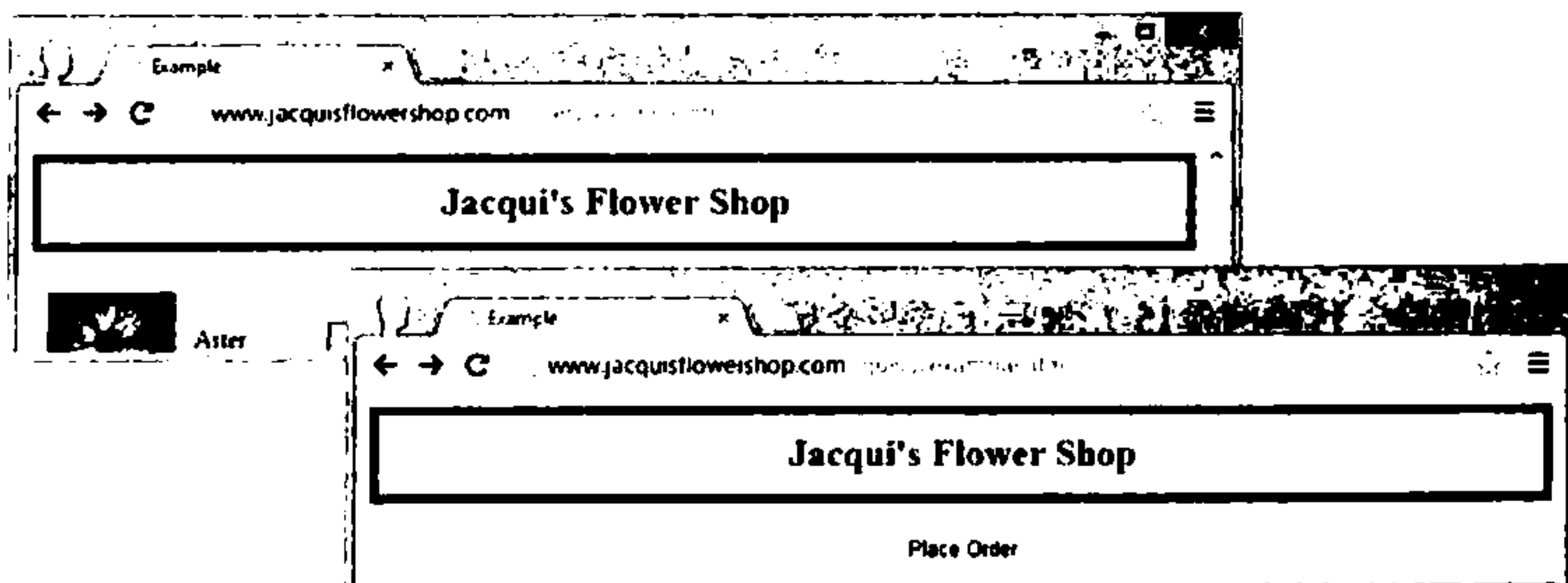


图15-2 使用context选项显示Ajax请求的结果

**提示** context选项可设置为任意的值，而为此选项设置合适的值是我们的责任。举例来说，如果把context选项的值设置为某个HTMLElement对象，那么在调用任何jQuery方法之前，一定要想着先调用\$函数把它包装成一个jQuery对象。

## 15.6 全局 Ajax 事件方法

除了前面讲过的那些针对具体Ajax请求的Ajax事件选项，jQuery还定义了一套全局事件方法。我们可以利用这些方法监控Web应用程序中的所有Ajax请求。表15-6列出了所有Ajax全局事件方法。

表15-6 Query Ajax事件方法

方 法	描 述
<code>ajaxComplete(function)</code>	注册在Ajax请求完成之后执行的函数（无论请求成功还是失败）
<code>ajaxError(function)</code>	注册在Ajax请求失败后执行的函数
<code>ajaxSend(function)</code>	注册在Ajax请求开始之前执行的函数
<code>ajaxStart(function)</code>	注册在Ajax请求开始时执行的函数
<code>ajaxStop(function)</code>	注册在所有Ajax请求完成之后执行的函数
<code>ajaxSuccess(function)</code>	注册在Ajax请求成功完成之后执行的函数

**提示** 如果是jQuery 1.9之前的版本，我们可以在任意的元素上调用上表中定义的这些方法。然而通过本节的例子可以看到，从jQuery 1.9/2.0版本开始，仅允许在document对象上调用这些方法

这些方法用来注册处理函数，而且必须在document对象上调用（前面简单提过）。`ajaxStart`和`ajaxStop`回调函数不会收到任何参数，而其他回调函数则接收到以下参数：

- 描述当前事件的Event对象；
- 描述当前请求的jqXHR对象；
- 包含当前请求配置信息的配置对象。

`ajaxError`方法还会额外得到一个描述出错信息的参数。

关于这些方法有两点需要特别注意。第一，这些函数将在任一Ajax请求发生时执行，也就是我们必须特别小心，确保没有假定这些函数只适用于某个具体的请求。

第二，这些方法必须在发起任何Ajax请求之前调用，以确保每一个Ajax请求都正确触发注册的函数。如果你调用`ajax`方法之后才调用这些方法，就要冒Ajax请求已经完成，而jQuery尚未成功注册事件处理函数的风险。代码清单15-12演示了这些全局Ajax事件方法的使用法。

代码清单15-12 全局Ajax事件方法示例

```
...
<script type="text/javascript">
    $(document).ready(function () {

        $("<div>").append("Events:")
        .append("<input type='checkbox' id='globalevents' name='globalevents' checked>")
        .insertAfter("h1");
        $("<ol id='info' class='ajaxinfo'>").insertAfter("h1").append("<li>Ready</li>");

        function displayMessage(msg) {
            $("#info").append("<li>").text(msg));
        }
    });
</script>
```

```

}

$(document)
  .ajaxStart(function() {
    displayMessage("Ajax Start")
  })
  .ajaxSend(function (event, jqxhr, settings) {
    displayMessage("Ajax Send: " + settings.url)
  })
  .ajaxSuccess(function (event, jqxhr, settings) {
    displayMessage("Ajax Success: " + settings.url)
  })
  .ajaxError(function (event, jqxhr, settings, errorMsg) {
    displayMessage("Ajax Error: " + settings.url)
  })
  .ajaxComplete(function (event, jqxhr, settings) {
    displayMessage("Ajax Complete: " + settings.url)
  })
  .ajaxStop(function () {
    displayMessage("Ajax Stop")
  });

$("#button").click(function (e) {
  $("#row1, #row2, #info").empty();
  $.ajax({
    url: "mydata.json",
    global: $("#globolevents:checked").length > 0,
    success: function (data, status, jqxhr) {
      var tplElems = $("#flowerTpl")
        .template({ flowers: data }).filter("*");
      tplElems.slice(0, 3).appendTo("#row1");
      tplElems.slice(3).appendTo("#row2");
    }
  });
  e.preventDefault();
});
});
</script>
...

```

在代码清单15-12中，我注册了所有的全局Ajax事件处理函数。每个函数都调用了`displayMessage`以显示当前触发的事件名称。由于Ajax请求往往在极短的时间内完成，当收到消息时，我创建了一个事件列表，并使用一个`ol`元素显示这些信息。

我还为按钮元素的`click`事件注册了`click`事件处理函数，点击按钮时，它会触发Ajax请求。这个例子的实际效果见图15-3，它展示了当按下按钮后，由Ajax请求生成的各种信息。

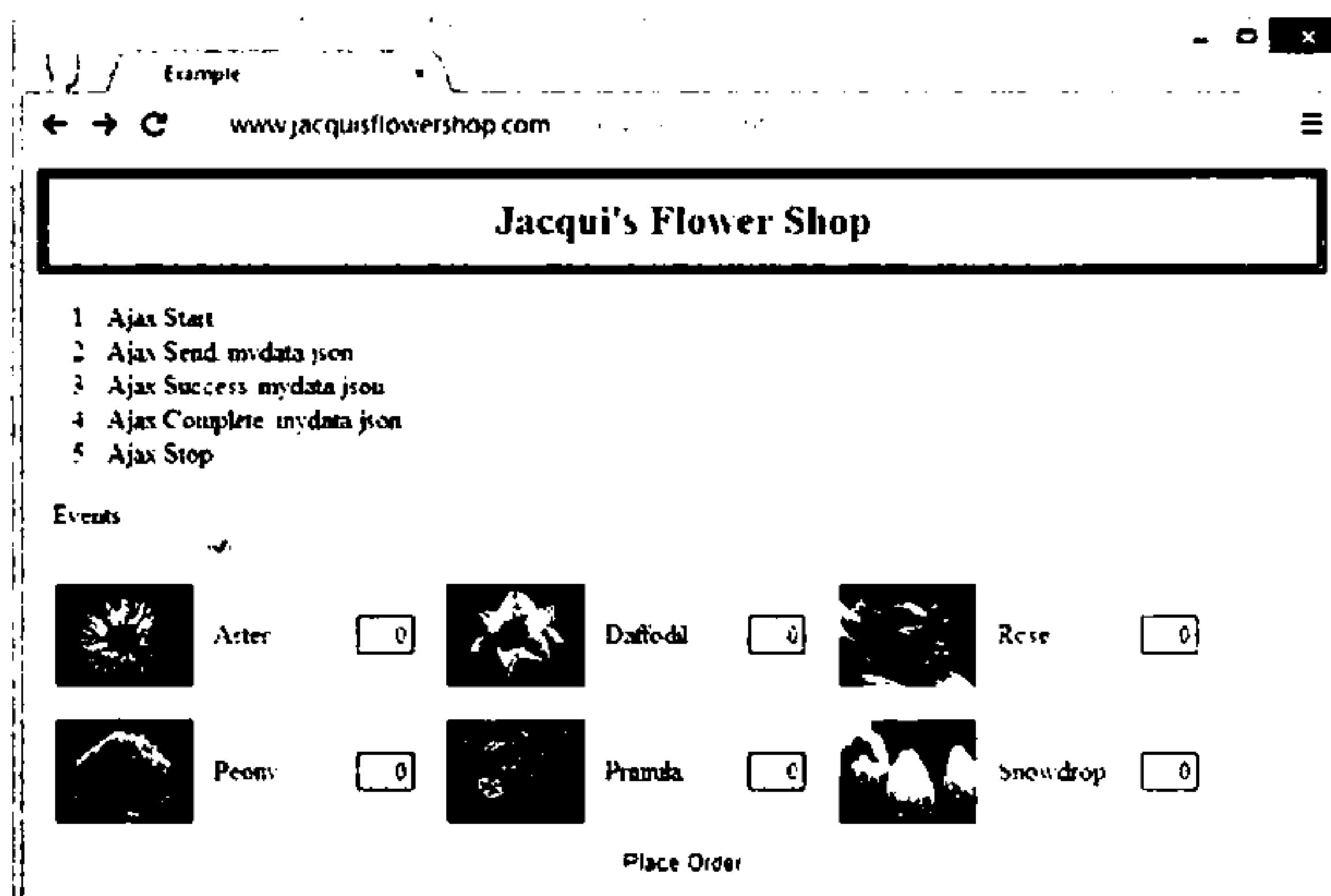


图15-3 全局Ajax事件效果

## 控制全局事件

也许你已经注意到，我在页面中添加了一个复选框。如代码清单15-13所示，在调用ajax方法时，我利用这个复选框设置global选项的值。

代码清单15-13 使用global选项

```
...
$.ajax({
    url: "mydata.json",
    global: $("#globolevents:checked").length > 0,
    success: function (data, status, jqxhr) {
        var tmplElems = $("#flowerTpl").template({ flowers: data }).filter("*");
        tmplElems.slice(0, 3).appendTo("#row1");
        tmplElems.slice(3).appendTo("#row2");
    }
})
...
```

若global选项的值为false，该Ajax请求就不会发生全局Ajax事件。你可以自己动手试试，反选这个复选框，然后点击按钮，你会看到Ajax请求成功完成，但没有显示任何状态信息。

## 15.7 为 Ajax 请求配置基础选项

ajax方法提供了一组选项，用于最基础的Ajax请求配置。我们只有极个别的情况需要使用这些选项，而这些选项的名字都清晰明了，可以望名知义。表15-7列出了这些选项，随后的几节会着重讲解其中一些选项。

表15-7 Ajax请求基础配置选项

选 项	描 述
accepts	设置Accept报头，浏览器根据这个值决定接受哪些MIME类型的数据。默认情况下，这个选项的值由dataType选项的值决定
cache	若设置为false，则不缓存服务器响应的数据。默认情况下，只有在请求script或jsonp数据时默认值是false，其他所有情况默认值都是true
contentType	设置Content-Type报头
dataType	指定服务器返回数据的类型。若使用此设置，jQuery就会忽略服务器响应数据声称的数据类型。欲了解该选项的工作原理详情，请参阅第14章
headers	为当前请求指定额外的其他报头，详见以下各节的讨论和例子
jsonp	进行JSONP请求时，不使用回调函数而是使用该选项指定的回调函数的名字，这需要服务器端程序的配合。关于JSONP，详见第14章
jsonpCallback	指定回调函数的名字，这里设置的名字会替掉jQuery默认随机生成的函数名。如果需要详细了解JSONP，参阅第14章
password	若服务器要求HTTP认证，使用这个选项指定密码
scriptCharset	若请求JavaScript脚本内容，使用该选项指定脚本文本使用的字符编码
timeout	指定请求超时选项（单位ms）。若请求超时，则jQuery自动调用error回调函数，并把status参数的值设置为timeout
username	若服务器要求HTTP认证，使用这个选项指定用户名

### 15.7.1 设置请求超时选项和报头

用户往往并不知道Ajax请求已经发生，因此设置一个最长等待时间是避免让用户傻等（甚至不知道系统正在做什么事）的好方法。代码清单15-14演示了为请求设置超时选项的方法。

代码清单15-14 设置请求超时选项

```

...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax("mydata.json", {
            timeout: 5000,
            headers: {
                "X-HTTP-Method-Override": "PUT" },
            success: function(data, status, jqxhr) {
                var template = $("#flowerTpl");
                template.tmpl(data.slice(0, 3)).appendTo("#row1");
                template.tmpl(data.slice(3)).appendTo("#row2");
            },
            error: function(jqxhr, status, errorMsg) {
                console.log("Error: " + status);
            }
        });
    });
});

```

```
</script>
```

```
...
```

在这个例子里，我使用`timeout`选项把请求的最长等待时间设置为5秒。如果5秒后请求仍未完成，`error`选项对应的回调函数就会自动执行，并把`status`参数值设置为`timeout`。

---

**警告** 定时器在请求被传递给浏览器那一刻就开始计时，而绝大多数浏览器对并发请求都有限制。这意味着有可能出现请求尚未发出就已经超时的极端情况。要避免这种情况，你必须对浏览器对并发请求个数的限制有所了解，并且对请求发生时可能发生的其他Ajax请求做到心中有数。

---

在这个例子中，我还使用`headers`选项为请求添加了一个报头：

```
...
```

```
headers: { "X-HTTP-Method-Override": "PUT" },
```

```
...
```

利用映射对象我们可以一次设置多个附加报头。在这个例子中设置的报头就是我在15.4节提到的那个。这个报头经常用于创建REST式Web应用程序，如果服务器支持的话。

## 15.7.2 发送JSON数据到服务器

如果需要发送数据到服务器，可以使用JSON格式——它是一种紧凑、表达力强的数据格式，而且很容易由JavaScript对象生成。发送JSON数据很简单，只要使用`contentType`选项为请求指定`Content-Type`报头，它会通知服务器我们正在发送何种数据，如代码清单15-15所示。

代码清单15-15 发送JSON数据到服务器

```
...
```

```
<script type="text/javascript">
```

```
$(document).ready(function () {
```

```
    $.ajax("mydata.json", {
```

```
        success: function (data, status, jqxhr) {
```

```
            var tmplElems = $("#flowerTpl").template({ flowers: data }).filter("*");
```

```
            tmplElems.slice(0, 3).appendTo("#row1");
```

```
            tmplElems.slice(3).appendTo("#row2");
```

```
        }
```

```
    });
```

```
    $("button").click(function (e) {
```

```
        $.ajax({
```

```
            url: $("form").attr("action"),
```

```
            contentType: "application/json",
```

```
            data: JSON.stringify($("form").serializeArray()),
```

```
            type: "post",
```

```
            success: processServerResponse
```

```
        });
```

```
        e.preventDefault();
```

```
    });
```

```

function processServerResponse(data) {
    var inputElems = $("#div.dcell").hide();
    for (var prop in data) {
        var filtered = inputElems.has("input[name=" + prop + "]")
            .appendTo("#row1").show();
    }
    $("#buttonDiv, #totalDiv").remove();
    $("#totalTpl").template(data).appendTo("body");
}
});
</script>
...

```

我把contentType选项的值设置为application/json, 即JSON文件的MIME类型。可以发送任意对象到服务器, 不过在这儿我希望演示一下如何把表单数据表达成JSON格式, 也就是这一句:

```

...
data: JSON.stringify($("#form").serializeArray()),
...

```

我先选中form元素, 然后调用serializeArray方法。它会生成一个数组对象, 其中每一个元素都是一个表示表单input元素的有着name属性和value属性的对象。最后, 我使用JSON.stringify方法把这个对象转换成下面这样的JSON字符串:

```

[{"name":"aster","value":"1"}, {"name":"daffodil","value":"1"},
{"name":"rose","value":"1"}, {"name":"peony","value":"1"},
{"name":"primula","value":"1"}, {"name":"snowdrop","value":"1"}]

```

现在, 我们有了一个适合发送到服务器的JSON字符串对象。本章用到的Node.js脚本能够解析和处理这个字符串对象。

## 15.8 高级配置选项

我将在接下来的几节中讲解最有趣也最有用的Ajax请求高级选项。这些选项并不常用, 但在需要时能帮上大忙。我们可以利用这些选项微调jQuery发起的Ajax请求。

### 15.8.1 发起同步请求

async选项用来指定Ajax请求是否以异步方式发出。它的默认值是true, 表示以异步方式发起Ajax请求; 如果把这个选项的值设置为false, 就会以同步方式发起Ajax请求。

如果请求以同步方式发起, ajax方法的行为就像一个普通函数, 浏览器会一直等待请求完成, 然后才会执行脚本的后续语句。具体示例见代码清单15-16。

代码清单15-16 发起同步请求

```

...
<script type="text/javascript">
    $(document).ready(function() {
        var elems;

        $.ajax("flowers.html", {

```



```

        async: false,
        success: function(data, status, jqxhr) {
            elems = $(data).filter("div").addClass("dcell");
        }
    });

    elems.slice(0, 3).appendTo("#row1");
    elems.slice(3).appendTo("#row2");
});
</script>
...

```

这个例子发起的请求与第14章演示的那个Ajax最容易犯错的例子相同，只是改用了底层API。不同之处在于这个例子把`async`选项的值设置成了`false`，因此浏览器在请求完成（把返回结果赋值给`elems`变量，如果请求成功完成的话）之前，不会执行后面语句中的`slice`和`appendTo`方法。使用Ajax方法发起同步请求是一件非常别扭的事情，我建议你自己在自己开发的应用程序中做这种事之前一定认真想清楚这么做的理由。

---

**提示** 不要因为异步请求处理困难就改用同步请求，回调处理方式其实非常优雅（我非常欣赏这种机制）。异步处理请求的结果其实没有那么可怕。时间将证明在理解回调机制上花费时间非常值得，它可使我们彻底弄清Web编程的这种方法。

---

## 15.8.2 忽略重复（未修改过的）数据

我们可以利用`ifModified`选项把Ajax请求设置为仅接收自上次接收数据以来更新过的数据。这个选项利用服务器响应中的`Last-Modified`报头工作。如果在响应用户行为时需要不断向服务器发出同样的请求，我们经常要处理服务器响应数据，并根据这些数据修改页面呈现（这些数据有可能已经呈现过）。这个选项的默认值是`false`，也就是说jQuery会忽略这个报头，总是提供数据给我们。代码清单15-17是使用该选项的一个具体示例。

代码清单15-17 使用`ifModified`选项

```

...
<script type="text/javascript">
    $(document).ready(function () {

        $("button").click(function (e) {
            $.ajax("mydata.json", {
                ifModified: true,
                success: function (data, status) {
                    if (status == "success") {
                        $("#row1, #row2").children().remove();
                        var tmplElems = $("#flowerTpl")
                            .template({ flowers: data }).filter("*");
                        tmplElems.slice(0, 3).appendTo("#row1");
                        tmplElems.slice(3).appendTo("#row2");
                    } else if (status == "notmodified") {

```



```

        $("img").css("border", "thick solid green");
    }
}
});
e.preventDefault();
});
});
</script>
...

```

这个例子把ifModified选项设置成了true。success回调函数总是执行，然而如果数据在上次请求之后并未改变，data参数的值就是undefined，而status参数则被设置为notmodified。

在这个例子里，status的值决定回调函数的行为。如果status的值是success，我就使用data参数在页面中添加元素（花卉产品）。如果status的值是notmodified，我就使用css方法为页面中已有的元素添加一个边框。

当用户点击button元素时，我调用ajax方法发起Ajax请求。这样我们就能不断地重复发起请求，以演示ifModified选项的效果（参见图15-4）。

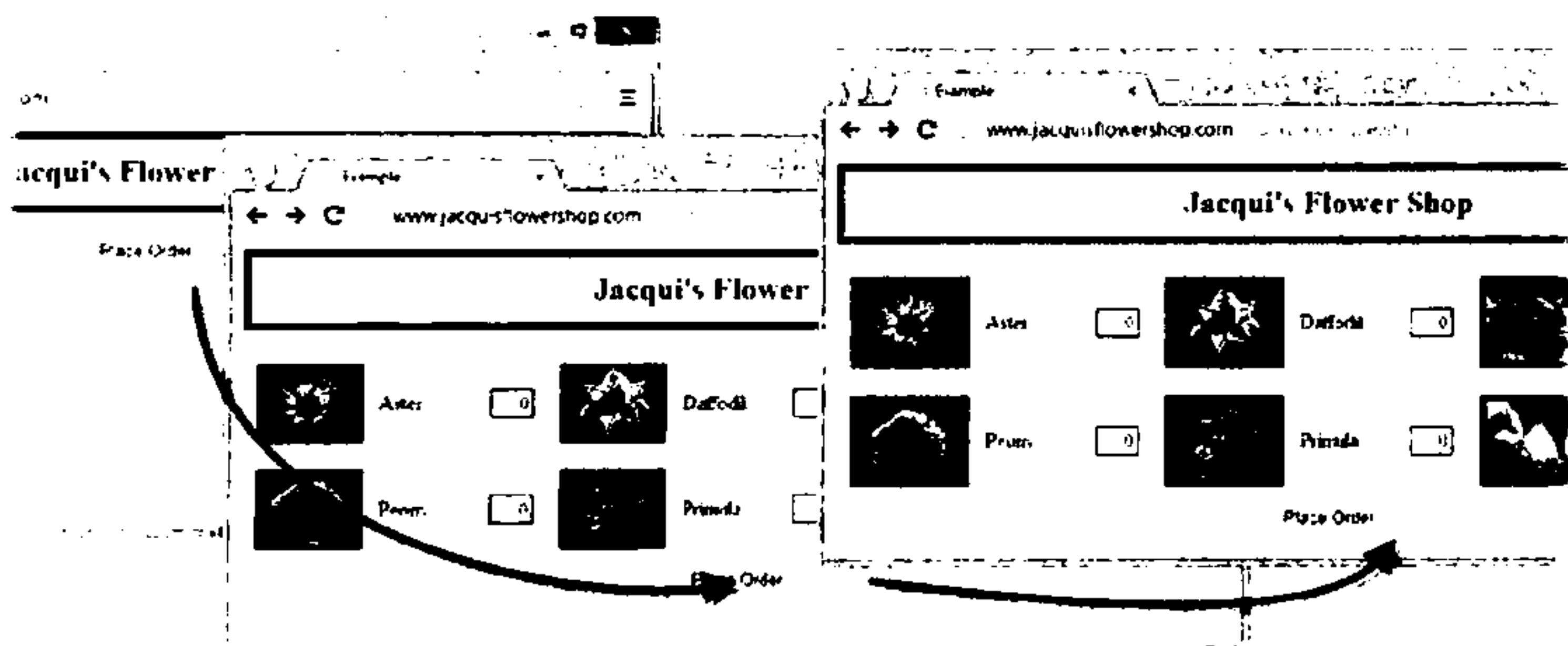


图15-4 使用ifModified选项

**警告** 有时候这个选项很有用，不过我还是建议慎用这个选项。如果是由用户行为（比如点击一个按钮）发起的Ajax请求，用户之所以重复按下同一个按钮很可能是因为第一次按下按钮没有反应。假设我们请求了数据，但是success回调函数里有一个bug导致没有正确更新页面内容，用户就会再一次按下这个按钮，以便试着让页面显示正常。然而不明智地使用ifModified选项可能导致忽略用户行为，让他们只得采取其他更不方便的步骤解决问题。

### 15.8.3 处理statusCode

利用statusCode选项，我们能够针对服务器返回的状态码执行专门的回调函数。我们可以用这一选项代替success和error选项，或者用作它们的补充。代码清单15-18演示了statusCode选项的用法。

代码清单15-18 使用statusCode选项

```

...
<style type="text/css">
    .error {color: red; border: medium solid red; padding: 4px;
        margin: auto; width: 200px; text-align: center}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax({
            url: "mydata.json",
            statusCode: {
                200: handleSuccessfulRequest,
                404: handleFailedRequest,
                302: handleRedirect
            }
        });

        function handleSuccessfulRequest(data, status, jqxhr) {
            $("#row1, #row2").children().remove();
            var template = $("#flowerTpl");
            template.tmpl(data.slice(0, 3)).appendTo("#row1");
            template.tmpl(data.slice(3)).appendTo("#row2");
        }

        function handleRedirect() {
            // 此函数永远不会被调用
        }

        function handleFailedRequest(jqxhr, status, errorMsg) {
            $("<div class=error>Code: " + jqxhr.status + " Message: "
                + errorMsg + "</div>").insertAfter("h1");
        }
    });
</script>
...

```

statusCode选项的值是一个映射对象，它的键是HTTP状态码，值则是对应该状态码需要执行的回调函数。在这个例子里我定义了3个回调函数，分别对应着状态码200、404和302。

状态码不同，传递给回调函数的参数也不同。如果状态码表示成功完成（比如200），传递给回调函数的参数就与success选项对应的回调函数一样。否则（比如表示文件未找到的404），传递的参数就与error选项对应的回调函数一样。

注意，我还在映射对象里定义了状态码302。当服务器希望重定向到另一个URL的时候，状态码302就会被返回给浏览器。jQuery将自动追踪重定向，直到收到一些内容或者遇到错误。这意味着我为302状态码定义的回调函数永远不会被调用。

---

**提示** 状态码304表示内容自从上次请求之后没有发生变化。只有把ifModified选项设置为true才有可能出现这个状态码，否则jQuery总是提供状态码200。关于ifModified选项，请阅读15.8.2节。

---

如果需要在浏览器与服务器之间调试用户交互，特别是在查找为什么jQuery实际行为与我们的预期不同时，这个功能非常有用。我一般将statusCode选项作为success及error选项的补充，利用回调函数在控制台输出信息。

---

**提示** success和error回调函数要先于statusCode选项指定的回调函数执行。

---

## 15.8.4 整理响应数据

dataFilter选项用来指定处理服务器返回数据的回调函数。如果服务器返回的数据格式与我们的需要不匹配（通常是因为格式不合适或者包含一些不需要的数据），这个功能就非常有用。代码清单15-19演示了这个选项的用法。

代码清单15-19 使用dataFilter选项

```
...
<script type="text/javascript">
    $(document).ready(function () {

        $.ajax({
            url: "mydata.json",
            success: function (data, status, jqxhr) {
                $("#row1, #row2").children().remove();
                var tplElems = $("#flowerTpl").template({ flowers: data }).filter("*");
                tplElems.slice(0, 3).appendTo("#row1");
                tplElems.slice(3).appendTo("#row2");
            },
            dataType: "json",
            dataFilter: function (data, dataType) {
                if (dataType == "json") {
                    var filteredData = $.parseJSON(data);
                    filteredData.shift();
                    return JSON.stringify(filteredData.reverse());
                } else {
                    return data;
                }
            }
        });
    });
</script>
...
```

服务器返回数据以及dataType选项的值分别被作为data和dataType参数传递给处理函数。如果未定义dataType选项，那么第二个参数的值为undefined。处理函数的任务是返回过滤后的数据。在这个例子中，如以下代码所示，我使用的是json数据：

```
...
var filteredData = $.parseJSON(data);
filteredData.shift();
return JSON.stringify(filteredData.reverse());
...
```

首先，我使用jQuery的parseJSON方法把JSON数据转换为JavaScript数组对象。然后，我调用数组的shift方法删除数组中的第一个元素，接着调用reverse方法反转剩余元素的顺序。

dataFilter回调函数必须返回一个字符串，因此我调用了JSON.stringify方法，虽然我知道在调用success回调函数之前，jQuery还是会把这个字符串转换成JavaScript对象。不管怎么说，在这个例子中我们确实删除了数组中的一个元素并反转了剩余数据的顺序。这不是最有用的变换形式，但它确实演示了过滤效果，如图15-5所示。

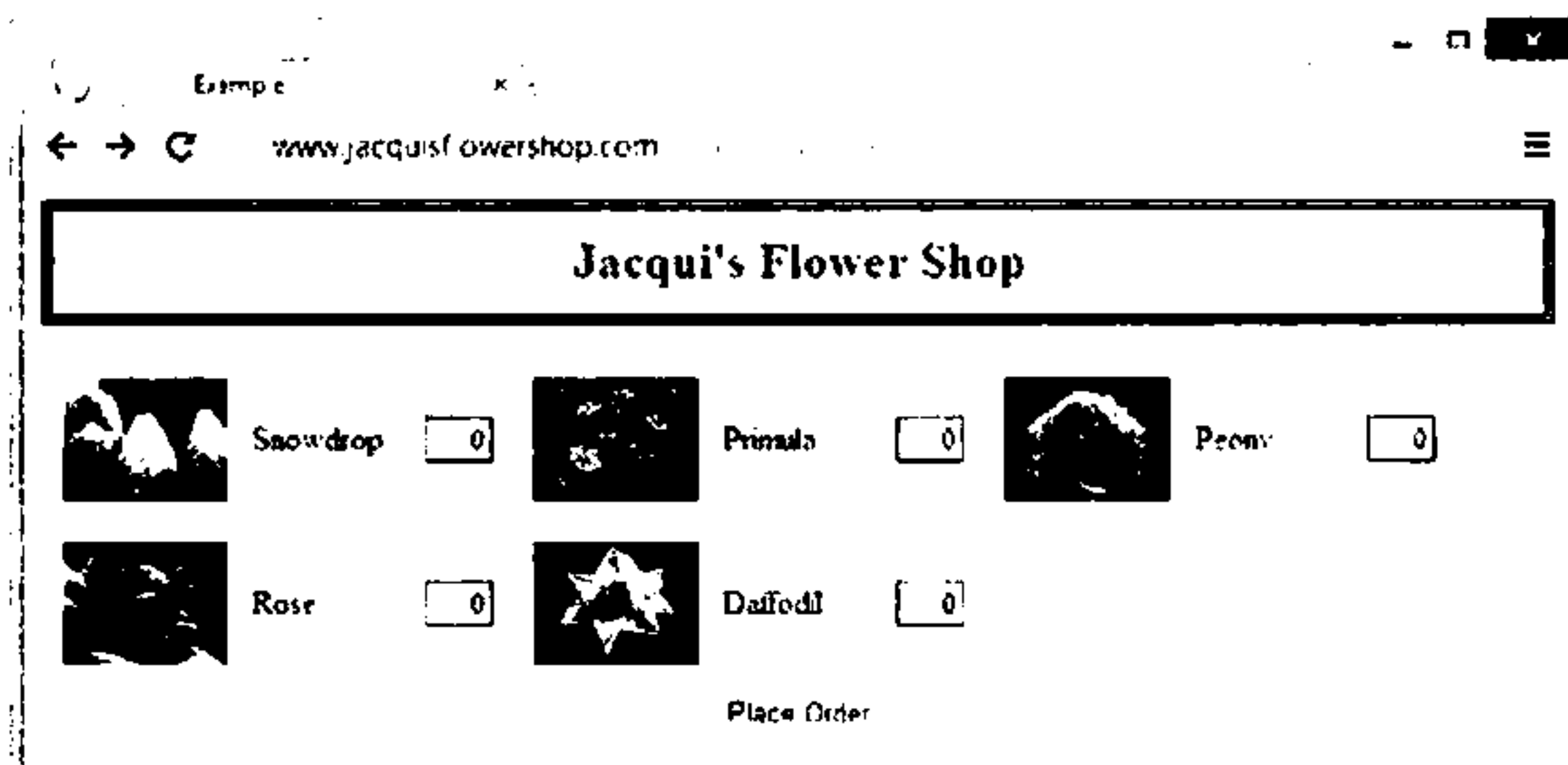


图15-5 使用dataFilter选项删除一条数据并反转剩余数据的顺序

### 15.8.5 控制数据格式转换

现在介绍最后一个选项，它也是我最喜欢的一个选项。你可能已经注意到，jQuery在接收到特定类型的数据之后会主动做一些格式转换。比如，jQuery收到的是一些JSON数据，但它转交给success回调函数的却是一个JavaScript对象，而不是未经处理的JSON字符串。

我们可以使用converters选项控制格式转换。这个选项的值是一个映射对象（键是数据类型，值是相应的处理函数）。代码清单15-20演示了如何利用这个选项自动把HTML数据转换为jQuery对象。

代码清单15-20 使用converters选项

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $.ajax({
            url: "flowers.html",
            success: function(data, status, jqxhr) {
                var elems = data.filter("div").addClass("dcell");
                elems.slice(0, 3).appendTo("#row1");
                elems.slice(3).appendTo("#row2");
            },
            converters: {
                "text html": function(data) {
                    return $(data);
                }
            }
        });
    });
</script>
```

```

    }
  });
});
</script>
...

```

在这个例子中，我为text/html类型的数据注册了一个转换函数。注意，这里使用空格而不是斜线（如text/html）表示MIME数据类型。这个函数接受服务器返回的数据，返回处理之后的数据。在本例中，我把服务器返回的HTML片段（flowers.html文件的内容）传递给jQuery的\$函数并返回。这意味着传递给success回调函数的data参数已经是一个完整的jQuery对象，我可以直接在这个对象上使用jQuery的各种方法。

---

**提示** 这个映射对象使用的数据类型并不总是和服务器声称的MIME类型相符。比如在converters选项中，application/json类型通常使用"text json"字符串表示。

---

使用这些转换器时很容易就会做过头。我们应该极力避免在这些函数中做“份外”之事。举例来说，我有时候忍不住想把服务器返回的JSON数据套上模板，然后把模板引擎的渲染结果HTML传递给success回调函数。尽管这（听起来）是一个很棒的技巧，然而一旦有人希望扩展你的代码，或者在你不需要如此“重度”转换，渴望使用服务器返回的原始数据时，你就会发现自己愚蠢得作茧自缚了。

## 15.9 设置及过滤 Ajax 请求

接下来，我们来了解jQuery额外提供的两个方法，它们简化了Ajax请求的配置。

### 15.9.1 定义默认选项

我们可以使用ajaxSetup方法指定一些对所有Ajax请求都有效的全局选项，这样就不必为每一个请求重复定义所有的选项了。代码清单15-21演示了这个方法的用法。

代码清单15-21 使用ajaxSetup方法

```

...
<script type="text/javascript">
  $(document).ready(function() {

    $.ajaxSetup({
      timeout: 15000,
      global: false,
      error: function(jqxhr, status, errorMsg) {
        $("

15


```

```

    }
  });

  $.ajax({
    url: "flowers.html",
    success: function(data, status, jqxhr) {
      var elems = data.filter("div").addClass("dcell");
      elems.slice(0, 3).appendTo("#row1");
      elems.slice(3).appendTo("#row2");
    }
  });
});
</script>
...

```

我们直接在\$函数上调用ajaxSetup方法, 就像调用ajax方法一样。ajaxSetup的参数是一个对象, 它的成员就是那些全局选项。在这个例子中, 我定义了timeout、global、error和converters选项的默认值。执行过ajaxSetup方法之后, 我只需要定义那些未提供默认值的选项或者希望采用不同值的那些选项。在需要发起大量类似配置的Ajax请求的场合, 这个方法能有效地减少代码重复。

---

**提示** ajaxSetup方法指定的选项也会影响我们在第14章讲到的快捷方法。这实在是一个既能利用底层API控制请求细节, 又可享受快捷方法提供的便利性的好方法。

---

## 15.9.2 过滤请求

ajaxPrefilter方法用来在必要时临时调整某些请求的选项, 如代码清单15-22所示。

代码清单15-22 使用ajaxPrefilter方法

```

...
<script type="text/javascript">
  $(document).ready(function () {

    $.ajaxSetup({
      timeout: 15000,
      global: false,
      error: function (jqxhr, status, errorMsg) {
        $("<div class=error/>")
          .text("Status: " + status + " Error: " + errorMsg)
          .insertAfter("h1");
      },
      converters: {
        "text html": function(data) {
          return $(data);
        }
      }
    });

    $.ajaxPrefilter("json html", function (settings, originalSettings, jqxhr) {
      if (originalSettings.dataType == "html") {

```

```

        settings.timeout = 2000;
    } else {
        jqxhr.abort();
    }
});

$.ajax({
    url: "flowers.html",
    dataType: "html",
    success: function (data, status, jqxhr) {
        var elems = data.filter("div").addClass("dcell");
        elems.slice(0, 3).appendTo("#row1");
        elems.slice(3).appendTo("#row2");
    }
});
});
</script>
...

```

ajaxPrefilter方法的参数由需要处理的数据类型和回调函数组成，若请求返回数据的类型匹配参数类型，相应的回调函数就会执行。如果我们省去数据类型的定义，只指定回调函数，那么不论请求返回什么类型的数据，回调函数都会执行。

这个函数的第一个参数是请求选项（包括我们使用ajaxSetup定义的那些默认值），第二个参数是传递给ajax方法的originalSettings选项（排除了所有默认设置），最后一个参数是对应本次请求的jqXHR对象。如本例所示，我们修改了作为第一个参数的请求选项对象。

在这段脚本中，我过滤了html请求和json请求。如果ajax方法的originalSettings中定义的数据类型为html，就把timeout选项设置为2秒。对数据类型不是html的请求，我们调用jqXHR对象的abort方法终止这个请求。

## 15.10 小结

本章讲解了jQuery底层Ajax接口的用法。我想你会认同，与第14章提到的快捷方法相比，它并没有难多少。只需要再多努力一点，你就能控制Ajax请求的方方面面，按照自己的需要与偏好进行调整。在下一章，我将重构示例页面，把本书这一部分讲到的功能和技术串联起来。

本书这一部分已经介绍了许多实用有趣的功能。类似于第一次重构（见第11章），我要把这些功能汇集到一起，让大家从更宽广的视角了解jQuery。

**提示** 在这一章，由于添加到示例中的这些功能严重依赖于JavaScript，我将不再维持一个没有JavaScript也能工作的页面结构。

## 16.1 重温重构示例

在第11章，我们使用jQuery核心功能重构了包括DOM操控、特效和事件等示例。代码清单16-1展示了我们最后得到的页面，这个页面是本章的起点，我们会把书中这一部分内容的示例整合到这个页面。

代码清单16-1 本章的起点

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    a.arrowButton {
      background-image: url(leftarrows.png); float: left;
      margin-top: 15px; display: block; width: 50px; height: 50px;
    }
    #right {background-image: url(rightarrows.png)}
    h1 { min-width: 0px; width: 95%; }
    #oblock { float: left; display: inline; border: thin black solid; }
    form { margin-left: auto; margin-right: auto; width: 885px; }
    #bbox {clear: left}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      var fNames = ["Carnation", "Lily", "Orchid"];
```



```

var fRow = $("<div id=row3 class=drow/>").appendTo("div.dtable");
var fTemplate = $("<div class=dcell><img/><label/><input/></div>");
for (var i = 0; i < fNames.length; i++) {
    fTemplate.clone().appendTo(fRow).children()
        .filter("img").attr("src", fNames[i] + ".png").end()
        .filter("label").attr("for", fNames[i]).text(fNames[i]).end()
        .filter("input").attr({
            name: fNames[i],
            value: 0,
            required: "required"
        });
}

$("<a id=left></a><a id=right></a>").prependTo("form")
    .addClass("arrowButton").click(handleArrowPress).hover(handleArrowMouse);
$("#right").appendTo("form");

$("#row2, #row3").hide();

var total = $("#buttonDiv")
    .prepend("<div>Total Items: <span id=total>0</span></div>")
    .css({clear: "both", padding: "5px"});
$("<div id=bbox />").appendTo("body").append(total);

$("input").change(function(e) {
    var total = 0;
    $("input").each(function(index, elem) {
        total += Number($(elem).val());
    });
    $("#total").text(total);
});

function handleArrowMouse(e) {
    var propValue = e.type == "mouseenter" ? "-50px 0px" : "0px 0px";
    $(this).css("background-position", propValue);
}

function handleArrowPress(e) {
    var elemSequence = ["row1", "row2", "row3"];

    var visibleRow = $("div.drow:visible");
    var visibleRowIndex = jQuery.inArray(visibleRow.attr("id"), elemSequence);

    var targetRowIndex;

    if (e.target.id == "left") {
        targetRowIndex = visibleRowIndex - 1;
        if (targetRowIndex < 0) {targetRowIndex = elemSequence.length - 1;}
    } else {
        targetRowIndex = (visibleRowIndex + 1) % elemSequence.length;
    }
    visibleRow.fadeOut("fast", function() {
        $("#" + elemSequence[targetRowIndex]).fadeIn("fast");
    });
}

```

```

    }
  });
</script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="oblock">
      <div class="dtable">
        <div id="row1" class="drow">
          <div class="dcell">
            <label for="aster">Aster:</label>
            <input name="aster" value="0" />
          </div>
          <div class="dcell">
            <label for="daffodil">Daffodil:</label>
            <input name="daffodil" value="0" />
          </div>
          <div class="dcell">
            <label for="rose">Rose:</label>
            <input name="rose" value="0" />
          </div>
        </div>
        <div id="row2" class="drow">
          <div class="dcell">
            <label for="peony">Peony:</label>
            <input name="peony" value="0" />
          </div>
          <div class="dcell">
            <label for="primula">Primula:</label>
            <input name="primula" value="0" />
          </div>
          <div class="dcell">
            <label for="snowdrop">Snowdrop:</label>
            <input name="snowdrop" value="0" />
          </div>
        </div>
      </div>
      <div id="buttonDiv"><button type="submit">Place Order</button></div>
    </form>
  </body>
</html>

```

---

**提示** 脚本中有好几个点会往页面中动态插入元素。不必把这些动态内容静态化，让它们保持原样，这样我们就能集中精力为页面添加新功能。

---

这个代码清单与我们在第11章得到的最终页面略有不同。我整理了示例中用到的样式并把它们添加到style元素中，从而不必在各个结果集上使用css方法。图16-1展示了这个文档在浏览器中的样子，不过为了达到这一效果，在第11章我们认真剖析了对页面的种种修改。

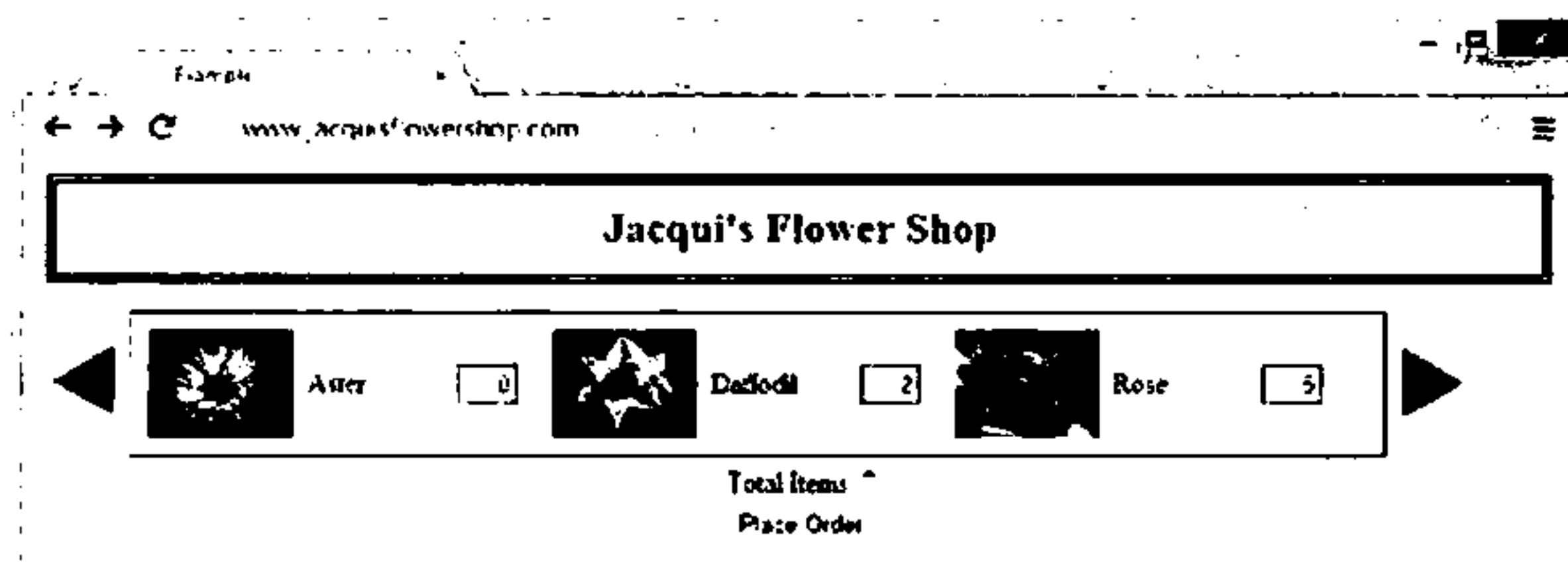


图16-1 本章示例页面的起始样子

## 16.2 更新 Node.js 脚本

首先，我们需要升级服务器端脚本。如代码清单16-2所示，这次升级使表单提交数据后服务器返回的数据更有意义，而且提供了数据验证的新功能。和本书中的所有示例一样，你可以从Apress网站（[www.apress.com](http://www.apress.com)）的源代码/下载区免费下载formserver.js升级文件。

### 代码清单16-2 修订之后的Node.js脚本

```
var http = require("http");
var querystring = require("querystring");
var url = require("url");

var port = 80;

http.createServer(function (req, res) {
  console.log("[200 OK] " + req.method + " to " + req.url);

  if (req.method == "OPTIONS") {
    res.writeHead(200, "OK", {
      "Access-Control-Allow-Headers": "Content-Type",
      "Access-Control-Allow-Methods": "**",
      "Access-Control-Allow-Origin": "http://www.jacquisflowershop.com"
    });
    res.end();
  } else if (req.method == "POST") {
    var dataObj = new Object();
    var contentType = req.headers["content-type"];
    var fullBody = "";

    if (contentType) {
      if (contentType.indexOf("application/x-www-form-urlencoded") > -1) {
        req.on("data", function (chunk) { fullBody += chunk.toString(); });
        req.on("end", function () {
          var dBody = querystring.parse(fullBody);
          writeResponse(req, res, dBody,
            url.parse(req.url, true).query["callback"])
        });
      }
    }
  }
});
```

```

    } else {
        req.on("data", function (chunk) { fullBody += chunk.toString(); });
        req.on("end", function () {
            dataObj = JSON.parse(fullBody);
            var dprops = new Object();
            for (var i = 0; i < dataObj.length; i++) {
                dprops[dataObj[i].name] = dataObj[i].value;
            }
            writeResponse(req, res, dprops);
        });
    }
}
} else if (req.method == "GET") {
    var data = url.parse(req.url, true).query;
    writeResponse(req, res, data, data["callback"])
}

var flowerData = {
    aster: { price: 2.99, stock: 10, plural: "Asters"},
    daffodil: {price: 1.99, stock: 10, plural: "Daffodils"},
    rose: {price: 4.99, stock: 2, plural: "Roses"},
    peony: {price: 1.50, stock: 3, plural: "Peonies"},
    primula: {price: 3.12, stock: 20, plural: "Primulas"},
    snowdrop: {price: 0.99, stock: 5, plural: "Snowdrops"},
    carnation: {price: 0.50, stock: 1, plural: "Carnations"},
    lily: {price: 1.20, stock: 2, plural: "Lillies"},
    orchid: {price: 10.99, stock: 5, plural: "Orchids"}
}

function writeResponse(req, res, data, jsonp) {
    var jsonData;
    if (req.url == "/stockcheck") {
        for (flower in data) {
            if (flowerData[flower].stock >= data[flower]) {
                jsonData = true;
            } else {
                jsonData = "We only have " + flowerData[flower].stock + " "
                    + flowerData[flower].plural + " in stock";
            }
            break;
        }
        jsonData = JSON.stringify(jsonData);
    } else {
        var totalCount = 0;
        var totalPrice = 0;
        for (item in data) {
            if(item != "_" && data[item] > 0) {
                var itemNum = Number(data[item])
                totalCount += itemNum;
                totalPrice += (itemNum * flowerData[item].price);
            } else {
                delete data[item];
            }
        }
    }
}

```

```

    data.totalItems = totalCount;
    data.totalPrice = totalPrice.toFixed(2);

    jsonData = JSON.stringify(data);
    if (jsonp) {
        jsonData = jsonp + "(" + jsonData + ")";
    }
}
res.writeHead(200, "OK", {
    "Content-Type": jsonp ? "text/javascript" : "application/json",
    "Access-Control-Allow-Origin": "*"
});
res.write(jsonData);
res.end();
}

```

```

}).listen(port);
console.log("Ready on port " + port);

```

提交表单到服务器之后, 现在服务器响应的JSON数据中增加了表单提交产品的总价 (totalPrice), 类似于这样:

```

{"aster": "1", "daffodil": "2", "rose": "4", "totalItems": 7, "totalPrice": "26.93"}

```

在命令行键入以下命令执行这个脚本:

```

node.exe formserver.js

```

## 16.3 为 Ajax 做准备

首先, 我要往页面中添加一些简单的元素和样式显示Ajax请求的错误信息, 并为所有Ajax请求设置 (共用的) 基本配置信息, 如代码清单16-3所示。

代码清单16-3 Ajax请求设置及错误处理

```

...
<style type="text/css">
    a.arrowButton {
        background-image: url(leftarrows.png); float: left;
        margin-top: 15px; display: block; width: 50px; height: 50px;
    }
    #right {background-image: url(rightarrows.png)}
    h1 { min-width: 0px; width: 95%; }
    #oblock { float: left; display: inline; border: thin black solid; }
    form { margin-left: auto; margin-right: auto; width: 885px; }
    #bbox {clear: left}
    #error {color: red; border: medium solid red; padding: 4px; margin: auto;
        width: 300px; text-align: center; margin-bottom: 5px}
</style>
<script type="text/javascript">

```

```

$(document).ready(function () {
    $.ajaxSetup({
        timeout: 5000,
        converters: {"text html": function (data) { return $(data); }}
    });

    $(document).ajaxError(function (e, jqxhr, settings, errorMsg) {
        $("#error").remove();
        var msg = "An error occurred. Please try again"
        if (errorMsg == "timeout") {
            msg = "The request timed out. Please try again"
        } else if (jqxhr.status == 404) {
            msg = "The file could not be found";
        }
        $("<div id=error/>").text(msg).insertAfter("h1");
    }).ajaxSuccess(function () {
        $("#error").remove();
    });

    var fNames = ["Carnation", "Lily", "Orchid"];
    var fRow = $("<div id=row3 class=drow/>").appendTo("div.dtable");
    var fTemplate = $("<div class=dcell><img/><label/><input/></div>");
    for (var i = 0; i < fNames.length; i++) {
        fTemplate.clone().appendTo(fRow).children()
            .filter("img").attr("src", fNames[i] + ".png").end()
            .filter("label").attr("for", fNames[i]).text(fNames[i]).end()
            .filter("input").attr({
                name: fNames[i], value: 0, required: "required"
            })
    }
};

$("<a id=left></a><a id=right></a>").prependTo("form")
    .addClass("arrowButton").click(handleArrowPress).hover(handleArrowMouse);
$("#right").appendTo("form");

$("#row2, #row3").hide();

var total = $("#buttonDiv")
    .prepend("<div>Total Items: <span id=total>0</span></div>")
    .css({clear: "both", padding: "5px"});
$("<div id=bbox />").appendTo("body").append(total).css("clear: left");

$("input").change(function (e) {
    var total = 0;
    $("input").each(function (index, elem) {
        total += Number($(elem).val());
    });
    $("#total").text(total);
});

function handleArrowMouse(e) {
    var propValue = e.type == "mouseenter" ? "-50px 0px" : "0px 0px";
    $(this).css("background-position", propValue);
}

```

```

function handleArrowPress(e) {
    var elemSequence = ["row1", "row2", "row3"];
    var visibleRow = $("#div.drow:visible");
    var visibleRowIndex =
        jQuery.inArray(visibleRow.attr("id"), elemSequence);

    var targetRowIndex;
    if (e.target.id == "left") {
        targetRowIndex = visibleRowIndex - 1;
        if (targetRowIndex < 0) { targetRowIndex = elemSequence.length - 1 };
    } else {
        targetRowIndex = (visibleRowIndex + 1) % elemSequence.length;
    }

    visibleRow.fadeOut("fast", function () {
        $("#" + elemSequence[targetRowIndex]).fadeIn("fast")
    });
}
});
</script>
...

```

在上面的代码中，我利用了全局Ajax事件来显示Ajax出错信息。当Ajax请求出错时，描述出错信息的新元素就立即显示在屏幕上。这个简单的错误信息来自jQuery。但对于一个真实的Web应用程序来说，出错信息应该尽量把发生的问题说清楚，并尽可能地提供解决办法。因为我在这里使用的是全局事件，所以仍然能在单个请求上使用success和error选项（不必操心函数数组的连接问题）。图15-2展示了一个请求出错示例。

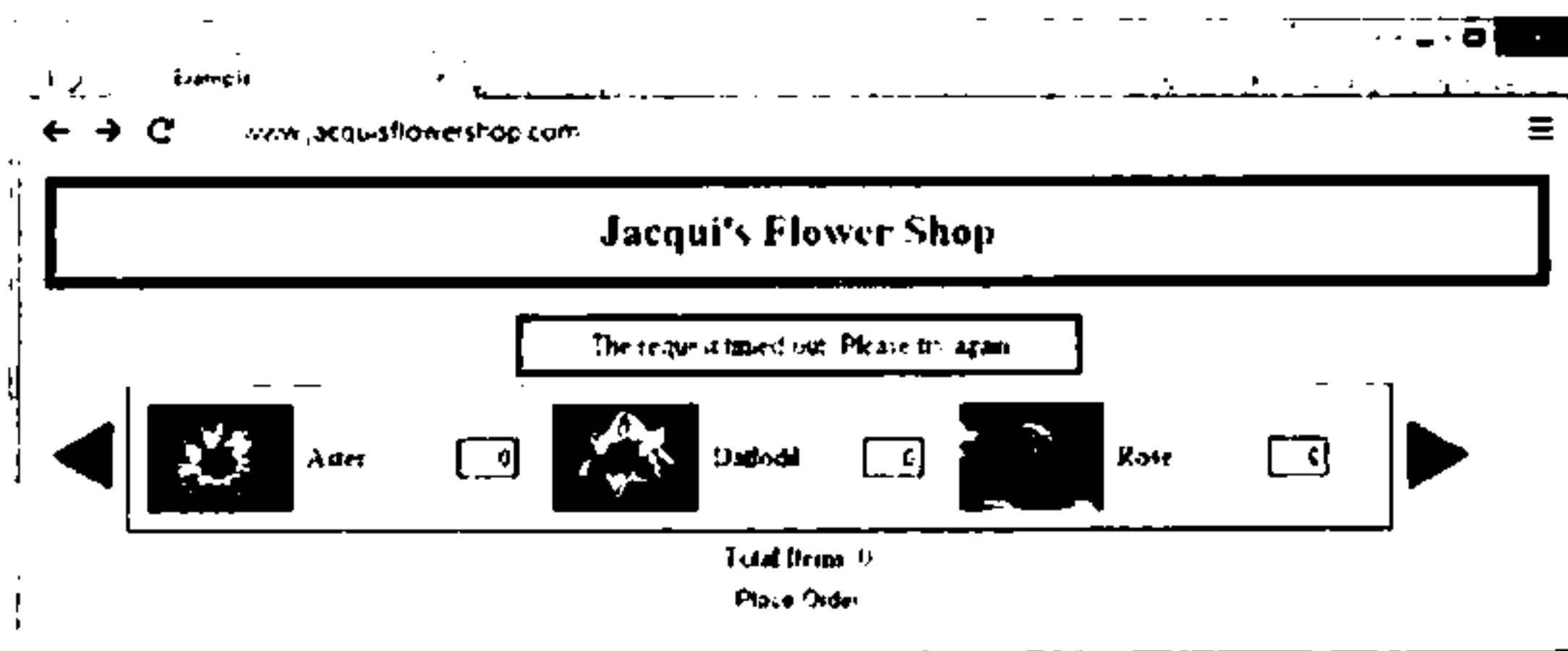


图16-2 显示Ajax出错信息

出错信息将一直显示，直到发出的请求成功完成或者发生另一个错误，无论哪种情况，现有的#error元素都会被从页面中移除。除了事件，我还使用ajaxSetup方法定义了timeout选项，并且针对HTML片段提供了一个转换器（converter），这样服务器响应的数据就能自动转换为jQuery对象。

## 16.4 处理产品数据来源

接下来，我要删除页面中的静态产品元素并在列表中额外添加另外3个花卉产品，使用两次Ajax调用和数据模板替换原来的代码。首先，我要创建一个新文件additionalflowers.json，把它作为其他示

例文件放在同一个目录下，其内容见代码清单16-4。

**代码清单16-4** additionalflowers.json文件的内容

```
[{"name": "Carnation", "product": "carnation"},
 {"name": "Lily", "product": "lily"},
 {"name": "Orchid", "product": "orchid"}]
```

这个文件的内容是一些JSON数据，是计划增加显示的3种产品的最基本描述。我会处理JSON数据，得到这些产品的HTML片段，然后把它们添加到页面中。代码清单16-5展示了这些变更。

**代码清单16-5** 利用通过Ajax得来的HTML和JSON数据生成产品信息

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js" type="text/javascript"></script>
  <script src="handlebars-jquery.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    a.arrowButton {
      background-image: url(leftarrows.png); float: left;
      margin-top: 15px; display: block; width: 50px; height: 50px;
    }
    #right {background-image: url(rightarrows.png)}
    h1 { min-width: 0px; width: 95%; }
    #oblock { float: left; display: inline; border: thin black solid; }
    form { margin-left: auto; margin-right: auto; width: 885px; }
    #bbox {clear: left}
    #error {color: red; border: medium solid red; padding: 4px; margin: auto;
      width: 300px; text-align: center; margin-bottom: 5px}
  </style>
  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}:
      <input name="{{product}}" value="0" />
    </div>
    {{/flowers}}
  </script>
  <script type="text/javascript">
    $(document).ready(function () {

      $.ajaxSetup({
        timeout: 5000,
        converters: {"text html": function(data) { return $(data); }}
      });

      $(document).ajaxError(function (e, jqxhr, settings, errorMsg) {
        $("#error").remove();
        var msg = "An error occurred. Please try again"
```



```

    if (errorMsg == "timeout") {
        msg = "The request timed out. Please try again"
    } else if (jqxhr.status == 404) {
        msg = "The file could not be found";
    }
    $("

>").text(msg).insertAfter("h1");
}).ajaxSuccess(function() {
    $("#error").remove();
});

$("<a id=left></a><a id=right></a>").prependTo("form")
    .addClass("arrowButton").click(handleArrowPress).hover(handleArrowMouse);
$("#right").appendTo("form");

$("#row2, #row3").hide();

$.get("flowers.html", function (data) {
    var elems = data.filter("div").addClass("dcell");
    elems.slice(0, 3).appendTo("#row1");
    elems.slice(3).appendTo("#row2");
});

$.getJSON("additionalflowers.json", function (data) {
    $("#flowerTpl").template({ flowers: data }).appendTo("#row3");
});

var total = $("#buttonDiv")
    .prepend("<div>Total Items: <span id=total>0</span></div>")
    .css({ clear: "both", padding: "5px" });
$("<div id=bbox />").appendTo("body").append(total).css("clear: left");

$("input").change(function (e) {
    var total = 0;
    $("input").each(function (index, elem) {
        total += Number($(elem).val());
    });
    $("#total").text(total);
});

function handleArrowMouse(e) {
    var propValue = e.type == "mouseenter" ? "-50px 0px" : "0px 0px";
    $(this).css("background-position", propValue);
}

function handleArrowPress(e) {
    var elemSequence = ["row1", "row2", "row3"];
    var visibleRow = $("div.drow:visible");
    var visibleRowIndex =
        jQuery.inArray(visibleRow.attr("id"), elemSequence);

    var targetRowIndex;
    if (e.target.id == "left") {
        targetRowIndex = visibleRowIndex - 1;
        if (targetRowIndex < 0) {targetRowIndex = elemSequence.length - 1};
    }


```

```

        } else {
            targetRowIndex = (visibleRowIndex + 1) % elemSequence.length;
        }

        visibleRow.fadeOut("fast", function () {
            $("#" + elemSequence[targetRowIndex]).fadeIn("fast")
        });
    }
});
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2" class="drow"></div>
                <div id="row3" class="drow"></div>
            </div>
            <div id="buttonDiv"><button type="submit">Place Order</button></div>
        </form>
    </body>
</html>

```

我在这里使用了Ajax快捷方法获取生成页面内容所需的HTML片段和JSON数据。尽管无法从脚本中一眼看出来，快捷方法本质上仅仅是底层API的封装。这意味着我们通过ajaxSetup方法设置的各种选项一样会在这些快捷方法上生效，就像是直接调用的ajax方法一样。除了get和getJSON方法的调用，我还添加了一个简单的数据模板，这样处理起JSON数据来既简单又轻松。页面的外观没有发生什么变化，但页面的内容来源却变了。

## 16.5 添加表单验证

接下来我们要对表单中的input元素做一些验证工作。代码清单16-6展示了需要添加的代码。

代码清单16-6 添加表单验证

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="handlebars.js" type="text/javascript"></script>
    <script src="handlebars-jquery.js" type="text/javascript"></script>
    <script src="jquery.validate.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <style type="text/css">
        a.arrowButton {
            background-image: url(leftarrows.png); float: left;
            margin-top: 15px; display: block; width: 50px; height: 50px;
        }
        #right {background-image: url(rightarrows.png)}
    </style>

```

```

h1 { min-width: 0px; width: 95%; }
#oblock { float: left; display: inline; border: thin black solid; }
form { margin-left: auto; margin-right: auto; width: 885px; }
#bbox {clear: left}
#error {color: red; border: medium solid red; padding: 4px; margin: auto;
        width: 300px; text-align: center; margin-bottom: 5px}
.invalidElem {border: medium solid red}
#errorSummary {border: thick solid red; color: red; width: 350px; margin: auto;
               padding: 4px; margin-bottom: 5px}
</style>
<script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
    <div class="dcell">
        
        <label for="{{product}}">{{name}}:
        <input name="{{product}}" value="0" />
    </div>
    {{/flowers}}
</script>
<script type="text/javascript">
    $(document).ready(function () {

        $.ajaxSetup({
            timeout: 5000,
            converters: {"text html": function (data) { return $(data); }}
        });

        $(document).ajaxError(function (e, jqxhr, settings, errorMsg) {
            $("#error").remove();
            var msg = "An error occurred. Please try again"
            if (errorMsg == "timeout") {
                msg = "The request timed out. Please try again"
            } else if (jqxhr.status == 404) {
                msg = "The file could not be found";
            }
            $("<div id=error/>").text(msg).insertAfter("h1");
        }).ajaxSuccess(function () {
            $("#error").remove();
        });

        $("<a id=left></a><a id=right></a>").prependTo("form")
            .addClass("arrowButton").click(handleArrowPress).hover(handleArrowMouse);
        $("#right").appendTo("form");

        $("#row2, #row3").hide();

        var flowerReq = $.get("flowers.html", function (data) {
            var elems = data.filter("div").addClass("dcell");
            elems.slice(0, 3).appendTo("#row1");
            elems.slice(3).appendTo("#row2");
        });

        var jsonReq = $.getJSON("additionalflowers.json", function (data) {
            $("#flowerTpl").template({ flowers: data }).appendTo("#row3");
        });
    });

```

```

    });

    $("<div id=errorSummary>").text("Please correct the following errors:")
        .append("<ul id='errorsList'></ul>").hide().insertAfter("h1");

    $("form").validate({
        highlight: function(element, errorClass) {
            $(element).addClass("invalidElem");
        },
        unhighlight: function(element, errorClass) {
            $(element).removeClass("invalidElem");
        },
        errorContainer: "#errorSummary",
        errorlabelContainer: "#errorsList",
        wrapper: "li",
        errorElement: "div"
    });

    var plurals = {
        aster: "Asters", daffodil: "Daffodils", rose: "Roses",
        peony: "Peonies", primula: "Primulas", snowdrop: "Snowdrops",
        carnation: "Carnations", lily: "Lillies", orchid: "Orchids"
    }

    $.when(flowerReq, jsonReq).then(function() {
        $("input").each(function(index, elem) {
            $(elem).rules("add", {
                required: true,
                min: 0,
                digits: true,
                messages: {
                    required: "Please enter a number for " + plurals[elem.name],
                    digits: "Please enter a number for " + plurals[elem.name],
                    min: "Please enter a positive number for "
                        + plurals[elem.name]
                }
            })
        })
        .change(function(e) {
            if ($("form").validate().element($(e.target))) {
                var total = 0;
                $("input").each(function(index, elem) {
                    total += Number($(elem).val());
                });
                $("#total").text(total);
            }
        });
    });

    var total = $("#buttonDiv")
        .prepend("<div>Total Items: <span id=total>0</span></div>")
        .css({ clear: "both", padding: "5px" });
    $("<div id=bbox />").appendTo("body").append(total).css("clear: left");

    $("input").change(function (e) {

```

```

        var total = 0;
        $("input").each(function (index, elem) {
            total += Number($(elem).val());
        });
        $("#total").text(total);
    });

    function handleArrowMouse(e) {
        var propValue = e.type == "mouseenter" ? "-50px 0px" : "0px 0px";
        $(this).css("background-position", propValue);
    }

    function handleArrowPress(e) {
        var elemSequence = ["row1", "row2", "row3"];
        var visibleRow = $("div.drow:visible");
        var visibleRowIndex =
            jQuery.inArray(visibleRow.attr("id"), elemSequence);

        var targetRowIndex;
        if (e.target.id == "left") {
            targetRowIndex = visibleRowIndex - 1;
            if (targetRowIndex < 0) {targetRowIndex = elemSequence.length - 1};
        } else {
            targetRowIndex = (visibleRowIndex + 1) % elemSequence.length;
        }

        visibleRow.fadeOut("fast", function () {
            $("#" + elemSequence[targetRowIndex]).fadeIn("fast")
        });
    }
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2" class="drow"></div>
                <div id="row3" class="drow"></div>
            </div>
        </div>
        <div id="buttonDiv"><button type="submit">Place Order</button></div>
    </form>
</body>
</html>

```

在这个代码清单中，首先我加载了jQuery的验证插件，接着定义了一些显示出错信息所需的样式。然后，我在表单元素上调用validate方法设置表单验证选项，并指定了单一验证出错摘要显示容器，这些方法与第13章的处理方式相同。

现在，使用Ajax生成花卉产品内容会造成一个问题。没错，异步调用的副作用：我们不能假定在

Ajax调用之后，页面中的input元素就立即就绪。我曾在第14章中指出，这是一个相当容易犯错的地方。如果浏览器在两个Ajax请求完成之前就开始选取input元素，就不会匹配到任何元素（尽管一小会儿之后它们就会被创建并添加到页面中），我们的验证表单设置就会失败。

为了解决这个问题，我使用了when和then方法，它们都是jQuery延迟对象的属性。关于jQuery延迟对象的细节，请阅读第35章。下面列出了有关语句：

```
...
$.when(flowerReq, jsonReq).then(function() {
    $("input").each(function(index, elem) {
        $(elem).rules("add", {
            required: true,
            min: 0,
            digits: true,
            messages: {
                required: "Please enter a number for " + plurals[elem.name],
                digits: "Please enter a number for " + plurals[elem.name],
                min: "Please enter a positive number for "
                    + plurals[elem.name]
            }
        });
    }).change(function(e) {
        if ($("form").validate().element($(e.target))) {
            var total = 0;
            $("input").each(function(index, elem) {
                total += Number($(elem).val());
            });
            $("#total").text(total);
        }
    });
});
...

```

我不得不打破计划，提前在这儿讲讲when和then方法的使用：每个Ajax方法返回一个jqXHR对象，把这两个对象作为参数传递给when方法，只有当两个Ajax请求都成功完成之后，then方法指定的回调函数才会执行。

我在then回调函数中处理表单验证，选中所有input元素，为每个元素添加必要的验证规则。我要求每个input元素都必须填写，并且值必须为不小于0的数字。我为每个检查定制了出错提示：利用plurals数组得到当前花卉的名字，从而让出错提示更有用。

既然已经得到这些input元素，我顺便为它们绑定了change事件处理函数。如果输入框中的数字发生了变化，就会触发change事件。注意，我在上面的代码中使用了element方法：

```
...
if ($("form").validate().element($(e.target))) {
...

```

这行代码会立即对值发生变化的元素进行验证，如果验证通过element方法就返回true，否则返回false。这里的if语句帮助我们在汇总订货数目时，扔掉在选中元素中填写的那些无效数据。

## 添加服务器端验证

在上一个例子中，我们实施的表单验证以及在第13章讲到的那些示例都是本地验证，也就是说这些规则和数据要求只能保证它们在当前页面范围内有效。

验证插件也支持服务器端验证。也就是说，当用户输入一个值时，这个值被发送给服务器，并由服务器端规则检查是否有效。有些时候我们无法让浏览器来检查数据，因为这样需要传递给浏览器过多的数据（比如检查一个用户名是否已被注册，要让浏览器来做验证，就得把所有用户名都传递给浏览器，这是不现实的）。

---

**警告** 当我们使用服务器端验证时，有一些事项需要注意。如果使用不当，服务器端验证会显著加重服务器的负担。在这个例子中，input元素的值一旦发生变化，我就实施服务器端验证。如果这是一个真实的项目，这种做法会产生大量的请求。一种更实用的做法是在用户提交表单之前实施服务器端验证。

---

我之所以没有在第13章讲解服务器端验证，是因为服务器端验证依赖于JSON和Ajax，而那时候我们还没有讲到这些主题。代码清单16-7展示了在示例页面新加的服务器端验证代码。我利用这些代码确保用户不会订购超出库存数量（服务器知道最新的库存数量）的产品。

### 代码清单16-7 实施服务器端验证

```
...
$.when(flowerReq, jsonReq).then(function() {
    $("input").each(function(index, elem) {
        $(elem).rules("add", {
            required: true,
            min: 0,
            digits: true,
            remote: {
                url: "http://node.jacquisflowershop.com/stockcheck",
                type: "post",
                global: false
            },
            messages: {
                required: "Please enter a number for " + plurals[elem.name],
                digits: "Please enter a number for " + plurals[elem.name],
                min: "Please enter a positive number for "
                    + plurals[elem.name]
            }
        });
    });
}).change(function(e) {
    if ($("#form").validate().element($(e.target))) {
        var total = 0;
        $("input").each(function(index, elem) {
            total += Number($(elem).val());
        });
        $("#total").text(total);
    }
});
```

```

    });
  });
  ...

```

设置由服务器端进行验证很简单：只需把remote属性设置为配置Ajax请求的选项对象，验证插件就会根据这个对象向服务器发起请求。在这个例子中，我把url选项设置为实施服务器端验证的URL，把type选项设置为POST请求，把global选项设置为false以禁止全局事件。

之所以禁用全局事件，这是因为我不希望验证请求发生的错误被当成用户其他请求的普通错误。相反，鉴于在提交表单之后服务器还会执行更多的验证（示例里的formserver.js脚本并未实施任何验证，不过在真实的Web项目中，我在第13章中就说过，服务器端验证非常重要），我希望验证请求在发生错误时一声不吭。

验证插件使用标准的jQuery Ajax设置对服务器端验证URL发起请求，把input元素的name和用户输入的值发送给服务器。若用户在Aster输入框输入数字22，然后离开输入框触发change事件，验证插件就会带着以下信息向服务器发起POST请求。

---

```

aster=22

```

---

服务器返回的信息相当简单。如果返回信息是字符串true，则这个值有效。任何其他响应都将视为出错信息，并显示给用户。formserver.js脚本会返回类似下面这样的出错信息：

---

```

We only have 10 Asters in stock

```

---

这个信息被当作本地验证信息来处理，如图16-3所示。

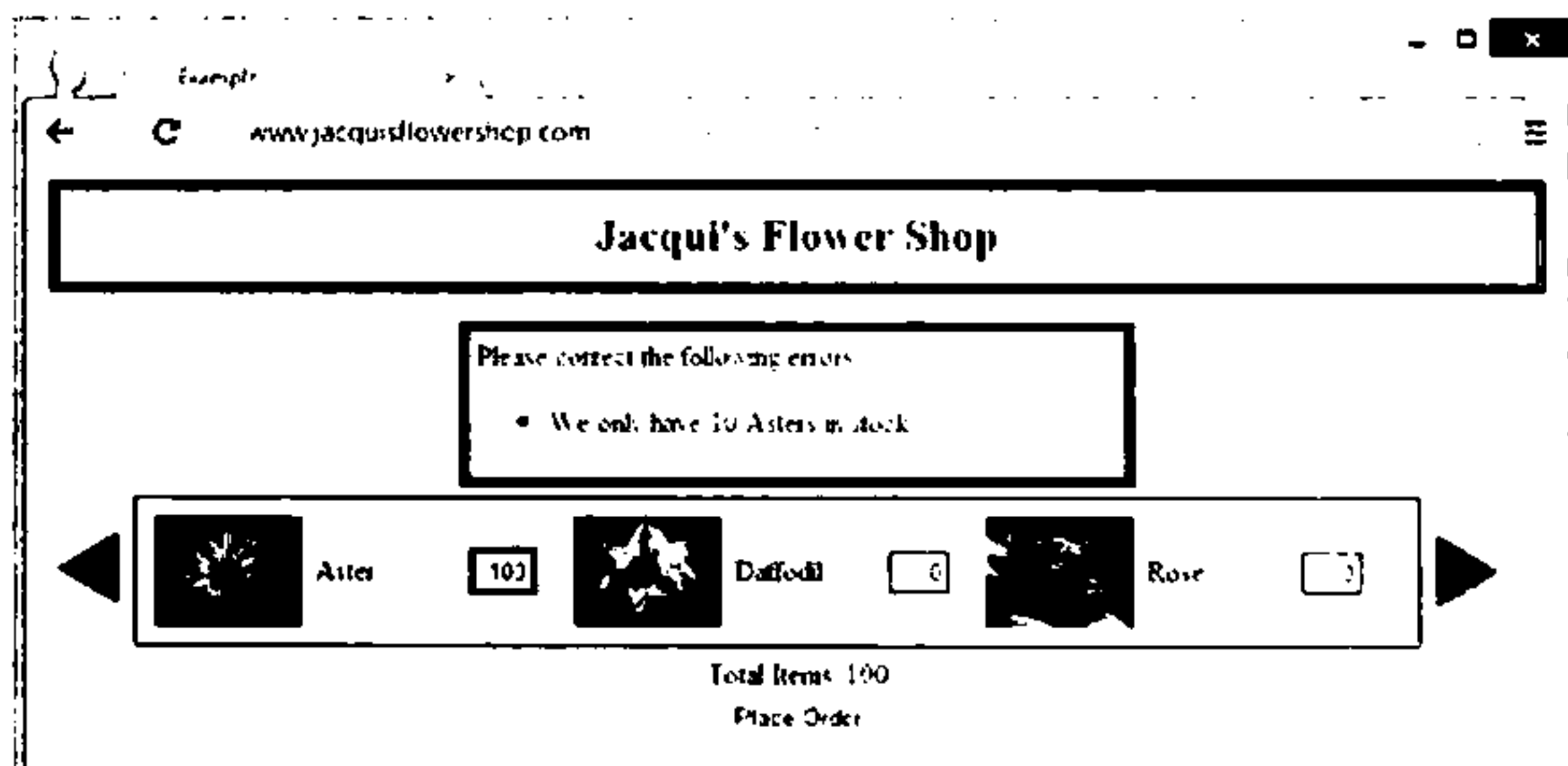


图16-3 显示服务器端验证信息

## 16.6 使用 Ajax 提交表单数据

提交表单数据特别简单，在代码清单16-8中我们使用了第15章中用过的技术。



## 代码清单16-8 使用Ajax提交表单数据

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js" type="text/javascript"></script>
  <script src="handlebars-jquery.js" type="text/javascript"></script>
  <script src="jquery.validate.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    a.arrowButton {
      background-image: url(leftarrows.png); float: left;
      margin-top: 15px; display: block; width: 50px; height: 50px;
    }
    #right {background-image: url(rightarrows.png)}
    h1 { min-width: 0px; width: 95%; }
    #oblock { float: left; display: inline; border: thin black solid; }
    form { margin-left: auto; margin-right: auto; width: 885px; }
    #bbox {clear: left}
    #error {color: red; border: medium solid red; padding: 4px; margin: auto;
      width: 300px; text-align: center; margin-bottom: 5px}
    .invalidElem {border: medium solid red}
    #errorSummary {border: thick solid red; color: red; width: 350px; margin: auto;
      padding: 4px; margin-bottom: 5px}
    #popup {
      text-align: center; position: absolute; top: 100px;
      left: 0px; width: 100%; height: 1px; overflow: visible; visibility: visible;
      display: block }
    #popupContent {
      color: white; background-color: black; font-size: 14px ;
      font-weight: bold; margin-left: -75px; position: absolute; top: -55px;
      left: 50%; width: 150px; height: 60px; padding-top: 10px; z-index: 2;}
  </style>
  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}:
      <input name="{{product}}" value="0" />
    </div>
    {{/flowers}}
  </script>
  <script type="text/javascript">
    $(document).ready(function() {

      $("<div id='popup'><div id='popupContent'><img src='progress.gif'"
        + "alt='progress' /><div>Placing Order</div></div></div>")
        .appendTo("body");

      $.ajaxSetup({
        timeout: 5000,
        converters: {"text html": function(data) { return $(data); }}
      })
    })
  </script>

```

```

    });

    $(document).ajaxError(function (e, jqxhr, settings, errorMsg) {
        $("#error").remove();
        var msg = "An error occurred. Please try again";
        if (errorMsg == "timeout") {
            msg = "The request timed out. Please try again";
        } else if (jqxhr.status == 404) {
            msg = "The file could not be found";
        }
        $("#<div id=error/>").text(msg).insertAfter("h1");
    }).ajaxSuccess(function() {
        $("#error").remove();
    });

    $("#<a id=left></a><a id=right></a>").prependTo("form")
        .addClass("arrowButton").click(handleArrowPress).hover(handleArrowMouse);
    $("#right").appendTo("form");

    $("#row2, #row3, #popup").hide();

    var flowerReq = $.get("flowers.html", function (data) {
        var elems = data.filter("div").addClass("dcell");
        elems.slice(0, 3).appendTo("#row1");
        elems.slice(3).appendTo("#row2");
    });

    var jsonReq = $.getJSON("additionalflowers.json", function (data) {
        $("#flowerTpl").template({ flowers: data }).appendTo("#row3");
    });

    $("#<div id=errorSummary>").text("Please correct the following errors:")
        .append("<ul id='errorsList'></ul>").hide().insertAfter("h1");

    $("#form").validate({
        highlight: function(element, errorClass) {
            $(element).addClass("invalidElem");
        },
        unhighlight: function(element, errorClass) {
            $(element).removeClass("invalidElem");
        },
        errorContainer: "#errorSummary",
        errorLabelContainer: "#errorsList",
        wrapper: "li",
        errorElement: "div"
    });

    var plurals = {
        aster: "Asters", daffodil: "Daffodils", rose: "Roses",
        peony: "Peonies", primula: "Primulas", snowdrop: "Snowdrops",
        carnation: "Carnations", lily: "Lillies", orchid: "Orchids"
    }

    $.when(flowerReq, jsonReq).then(function() {

```

```

$("input").each(function(index, elem) {
    $(elem).rules("add", {
        required: true,
        min: 0,
        digits: true,
        remote: {
            url: "http://node.jacquisflowershop.com/stockcheck",
            type: "post",
            global: false
        },
        messages: {
            required: "Please enter a number for " + plurals[elem.name],
            digits: "Please enter a number for " + plurals[elem.name],
            min: "Please enter a positive number for "
                + plurals[elem.name]
        }
    });
}).change(function(e) {
    if ($("#form").validate().element($(e.target))) {
        var total = 0;
        $("input").each(function(index, elem) {
            total += Number($(elem).val());
        });
        $("#total").text(total);
    }
});
});

$("button").click(function(e) {
    e.preventDefault();
    var formData = $("#form").serialize();
    $("body *").not("#popup, #popup *").css("opacity", 0.5);
    $("input").attr("disabled", "disabled");
    $("#popup").show();
    $.ajax({
        url: "http://node.jacquisflowershop.com/order",
        type: "post",
        data: formData,
        complete: function() {
            setTimeout(function() {
                $("body *").not("#popup, #popup *").css("opacity", 1);
                $("input").removeAttr("disabled");
                $("#popup").hide();
            }, 1500);
        }
    });
});

var total = $("#buttonDiv")
    .prepend("<div>Total Items: <span id=total>0</span></div>")
    .css({clear: "both", padding: "5px"});
$("#<div id=bbox />").appendTo("body").append(total).css("clear: left");

$("input").change(function (e) {

```

```

        var total = 0;
        $("input").each(function (index, elem) {
            total += Number($(elem).val());
        });
        $("#total").text(total);
    });

    function handleArrowMouse(e) {
        var propValue = e.type == "mouseenter" ? "-50px 0px" : "0px 0px";
        $(this).css("background-position", propValue);
    }

    function handleArrowPress(e) {
        var elemSequence = ["row1", "row2", "row3"];
        var visibleRow = $("div.drow:visible");
        var visibleRowIndex =
            jQuery.inArray(visibleRow.attr("id"), elemSequence);

        var targetRowIndex;
        if (e.target.id == "left") {
            targetRowIndex = visibleRowIndex - 1;
            if (targetRowIndex < 0) { targetRowIndex = elemSequence.length - 1 };
        } else {
            targetRowIndex = (visibleRowIndex + 1) % elemSequence.length;
        }

        visibleRow.fadeOut("fast", function() {
            $("#" + elemSequence[targetRowIndex]).fadeIn("fast");
        });
    }
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                <div id="row2" class="drow">
                <div id="row3" class="drow">
            </div>
        </div>
        <div id="buttonDiv">Place Order
    </form>
</body>
</html>

```

我没有满足于发出Ajax POST请求，为了演示在真实项目中如何实施服务器端验证，我将额外讲解一些背景知识。首先，我在页面中添加了一个元素，它会显示在所有其他元素之上，告诉用户订单已经发出。下面这些代码负责创建这些元素：

```

...
$("<div id='popup'><div id='popupContent'><img src='progress.gif'"

```

```
+ "alt='progress'"/><div>Placing Order</div></div></div>").appendTo("body");
...
```

在style标签中，我为这些新元素设置了如下样式：

```
...
#popup {
  text-align: center; position: absolute; top: 100px;
  left: 0px; width: 100%; height: 1px; overflow: visible; visibility: visible;
  display: block }
#popupContent { color: white; background-color: black; font-size: 14px ;
  font-weight: bold; margin-left: -75px; position: absolute; top: -55px;
  left: 50%; width: 150px; height: 60px; padding-top: 10px; z-index: 2;
}
```

添加这个元素，把它放到屏幕上适当的位置，并让它看起来像一个弹出层非常麻烦。可以看到，我们需要编写相当多的CSS样式才能达成这一目标。相比较而言，HTML代码则简单许多，最终由jQuery语句生成的HTML如下所示：

```
...
<div id="popup" style="display: none;">
  <div id="popupContent">
    
    <div>Placing Order</div>
  </div>
</div>
...
```

指定给img元素的图片（progress.gif）是一个GIF动画。有许多网站能够帮我们生成表示进度的图片，我随便从中选了一个。如何你不想自己生成这种图片，那就直接用示例中的这一个好了（可免费从Apress.com下载本书配套的代码包）。这些元素最终的样子见图16-4，为了让大家看得更清楚，我删掉了其他元素。

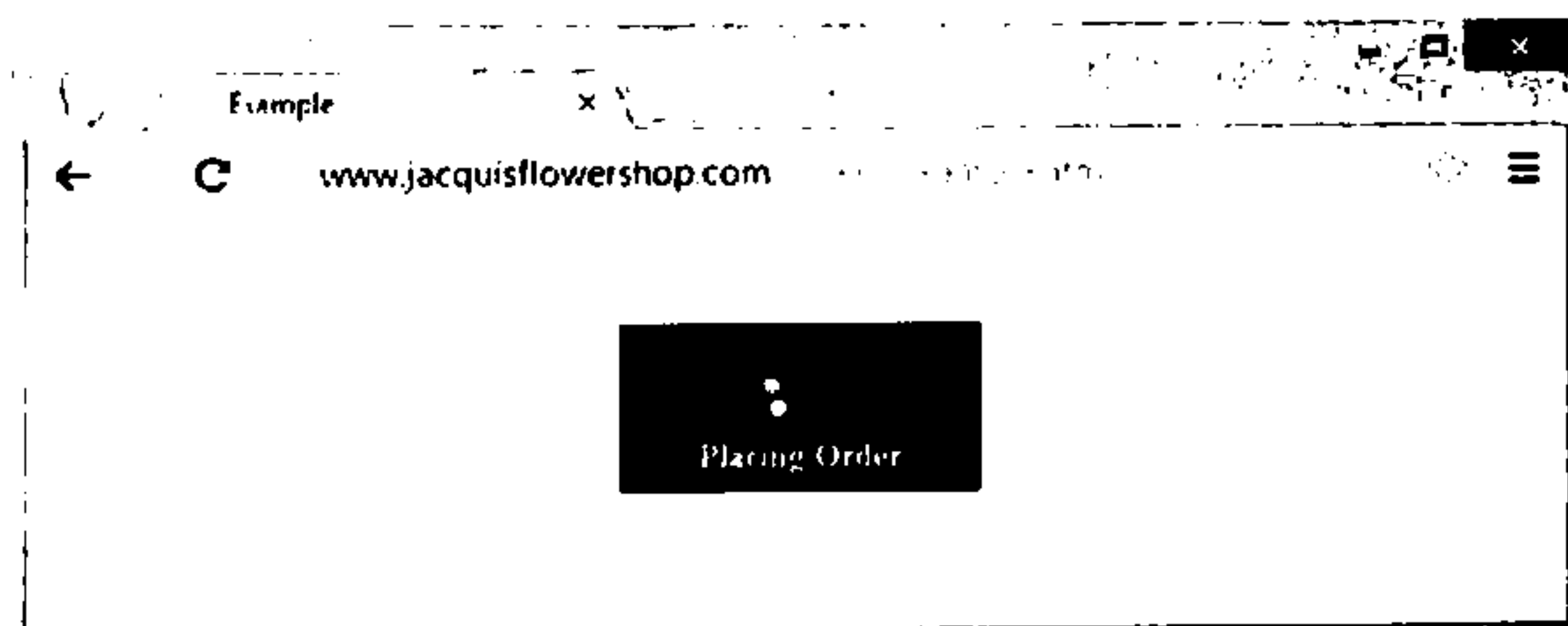


图16-4 显示进度

首先让这些元素隐藏，因为在真正提交订单之前显示进度没有什么用：

```
...
$("#row2, #row3, #popup").hide();
...
```

在这些元素就绪并隐藏起来之后，我们开始处理表单提交部分。如以下代码所示，我为button元

素注册了一个click事件处理函数：

```
...
$("button").click(function (e) {
    e.preventDefault();
    var formData = $("form").serialize();
    $("body *").not("#popup, #popup *").css("opacity", 0.5);
    $("input").attr("disabled", "disabled");
    $("#popup").show();
    $.ajax({
        url: "http://node.jacquisflowershop.com/order",
        type: "post",
        data: formData,
        complete: function () {
            setTimeout(function () {
                $("body *").not("#popup, #popup *").css("opacity", 1);
                $("input").removeAttr("disabled");
                $("#popup").hide();
            }, 1500);
        }
    });
});
...

```

在发起Ajax请求之前，我显示弹出层，并让所有其他元素变得半透明。发送数据给服务器时，我们不希望用户修改任意input元素的值，为此禁用了input元素（给它们设置disabled属性）：

```
...
$("body *").not("#popup, #popup *").css("opacity", 0.5);
$("input").attr("disabled", "disabled");
$("#popup").show();
...

```

这时我们遇到一个问题，在发送给服务器的数据里并不包含被禁用的input元素的值。serialize方法根据HTML标准只会包含那些可用input元素的值，这就排除了那些被禁用的input元素以及没有name属性的input元素。我们当然可以自己遍历所有的input元素以得到它们的值，然而像下面这样在禁用input元素之前先收集数据更简单更容易：

```
...
var formData = $("form").serialize();
...

```

我使用complete选项来恢复所有元素到正常状态（由半透明到不透明），删除input元素的disabled属性，并隐藏弹出层元素。在Ajax请求完成之后，恢复界面之前，我使用下面的代码人为地制造了1.5s的延时：

```
...
complete: function() {
    setTimeout(function() {
        $("body *").not("#popup, #popup *").css("opacity", 1);
        $("input").removeAttr("disabled");
        $("#popup").hide();
    }, 1500);
}

```

```

}
...

```

在真实的项目中我不会这么做。由于我们的开发机器与服务器在同一个局域网上,出于演示目的,加上这个延时有助于强调状态的切换。图16-5展示了Ajax请求期间页面在浏览器中的呈现效果。

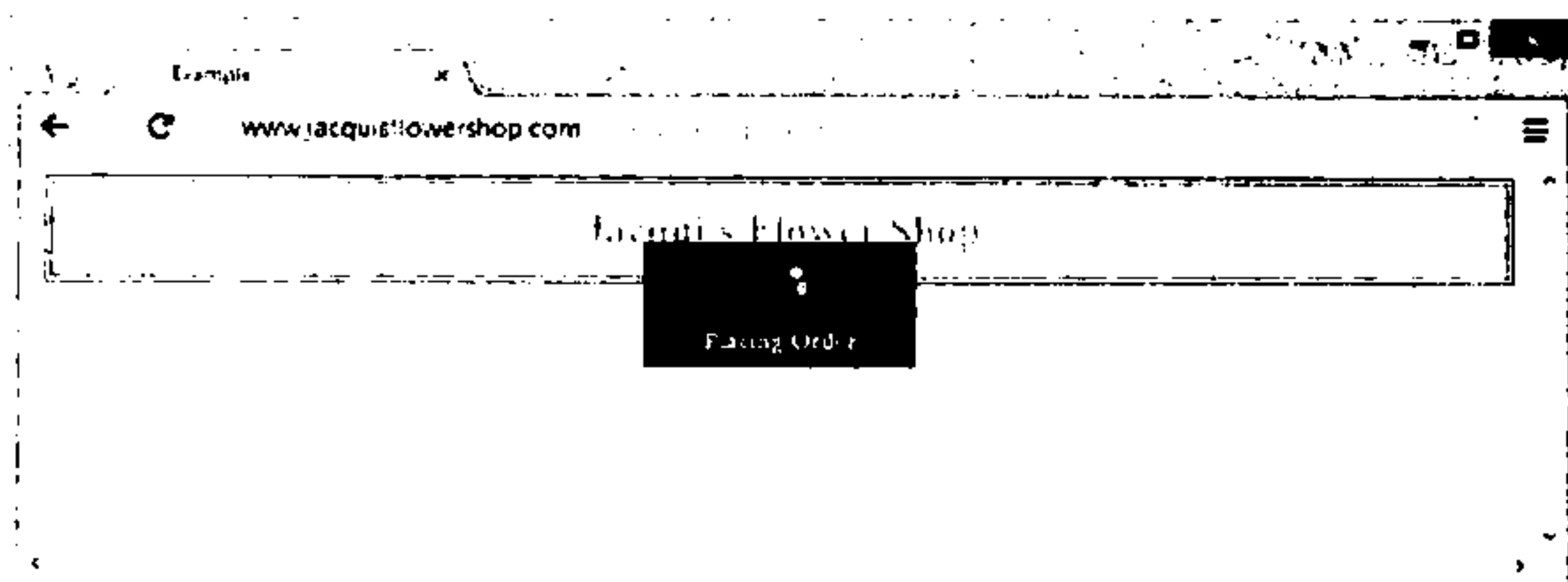


图16-5 提交表单数据期间浏览器的呈现效果

## 16.7 处理服务器响应

剩下的工作是处理服务器返回的数据。本章会使用一个简单的表格。在本书的下一部分中,我们将会了解如何使用jQuery UI创建更华丽的用户界面。因此,我不会在这里花大力气手工实现那些效果,因为可以用jQuery UI组件更优雅地实现它们。最终效果见图16-6。

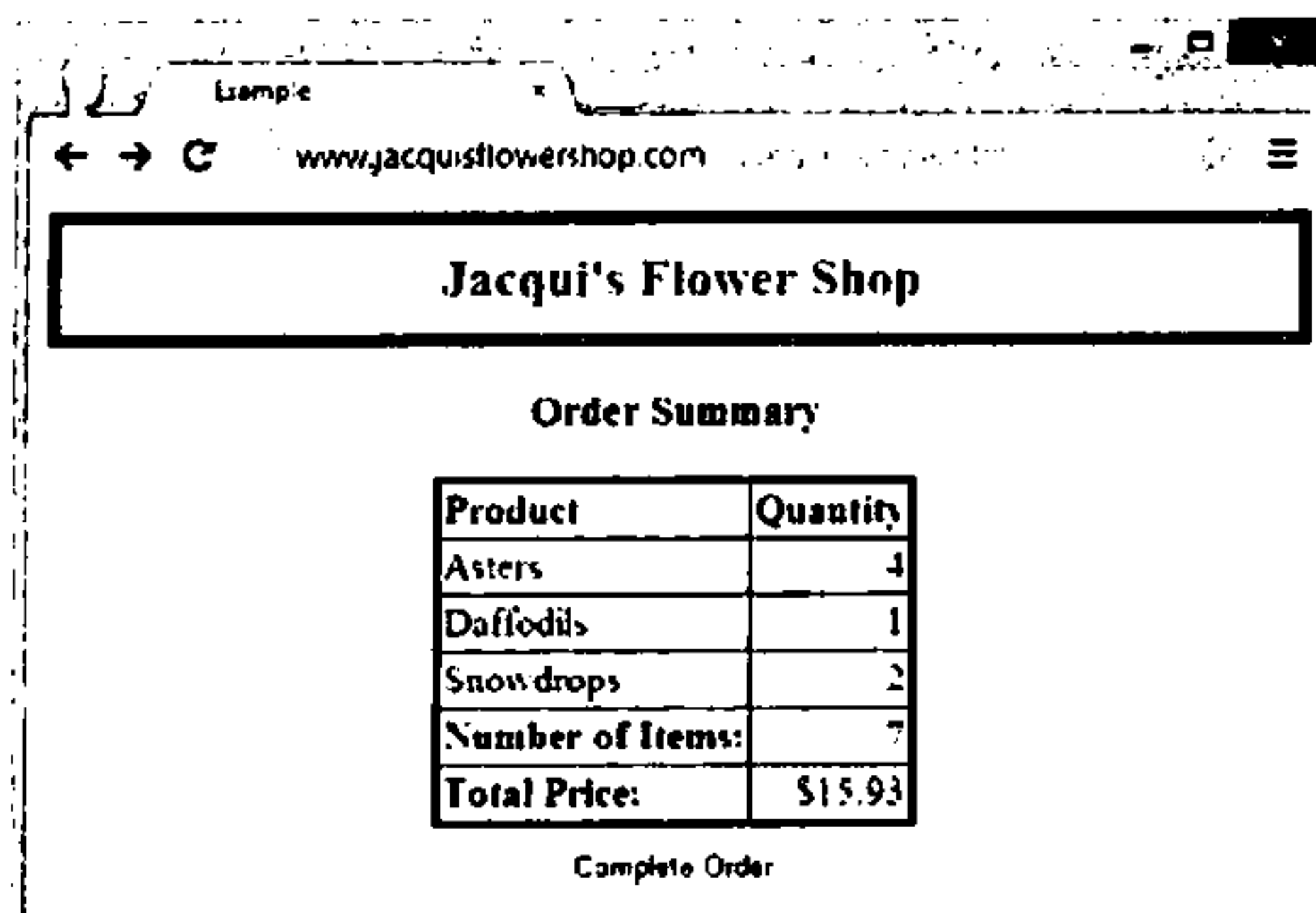


图16-6 显示订单摘要

代码清单16-9展示了完成这一增强之后的完整代码。在后面几个小节中,我会针对这些修改分步进行讲解。

### 代码清单16-9 处理服务器响应

```

<!DOCTYPE html>
<html>
<head>

```

```

<title>Example</title>
<script src="jquery-2.0.2.js" type="text/javascript"></script>
<script src="handlebars.js" type="text/javascript"></script>
<script src="handlebars-jquery.js" type="text/javascript"></script>
<script src="jquery.validate.js" type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<style type="text/css">
  a.arrowButton {
    background-image: url(leftarrows.png); float: left;
    margin-top: 15px; display: block; width: 50px; height: 50px;
  }
  #right {background-image: url(rightarrows.png)}
  h1 { min-width: 0px; width: 95%; }
  #oblock { float: left; display: inline; border: thin black solid; }
  #orderForm { margin-left: auto; margin-right: auto; width: 885px; }
  #bbox {clear: left}
  #error {color: red; border: medium solid red; padding: 4px; margin: auto;
    width: 300px; text-align: center; margin-bottom: 5px}
  .invalidElem {border: medium solid red}
  #errorSummary {border: thick solid red; color: red; width: 350px; margin: auto;
    padding: 4px; margin-bottom: 5px}
  #popup {
    text-align: center; position: absolute; top: 100px;
    left: 0px; width: 100%; height: 1px; overflow: visible; visibility: visible;
    display: block }
  #popupContent { color: white; background-color: black; font-size: 14px ;
    font-weight: bold; margin-left: -75px; position: absolute; top: -55px;
    left: 50%; width: 150px; height: 60px; padding-top: 10px; z-index: 2; }
  #summary {text-align: center}
  table {border-collapse: collapse; border: medium solid black; font-size: 18px;
    margin: auto; margin-bottom: 5px;}
  th {text-align: left}
  th, td {padding: 2px}
  tr > td:nth-child(1) {text-align: left}
  tr > td:nth-child(2) {text-align: right}
</style>
<script id="flowerImpl" type="text/x-handlebars-template">
  {{#flowers}}
  <div class="dcell">
    
    <label for="{{product}}">{{name}}:
    <input name="{{product}}" value="0" />
  </div>
  {{/flowers}}
</script>
<script id="productRowImpl" type="text/x-handlebars-template">
  {{#rows}}
    <tr><td>{{name}}</td><td>{{quantity}}</td></tr>
  {{/rows}}
</script>
<script type="text/javascript">
  $(document).ready(function () {

    $("<div id='popup'><div id='popupContent'><img src='progress.gif'"

```



```

    + "alt='progress'/><div>Placing Order</div></div></div>")
    .appendTo("body");

$.ajaxSetup({
    timeout: 5000,
    converters: {"text html": function(data) { return $(data); }}
});

$(document).ajaxError(function (e, jqxhr, settings, errorMsg) {
    $("#error").remove();
    var msg = "An error occurred. Please try again"
    if (errorMsg == "timeout") {
        msg = "The request timed out. Please try again"
    } else if (jqxhr.status == 404) {
        msg = "The file could not be found";
    }
    $("

>").text(msg).insertAfter("h1");
}).ajaxSuccess(function() {
    $("#error").remove();
});

$("<a id=left></a><a id=right></a>").prependTo("#orderForm")
    .addClass("arrowButton").click(handleArrowPress).hover(handleArrowMouse);
$("#right").appendTo("#orderForm");

$("#row2, #row3, #popup, #summaryForm").hide();

var flowerReq = $.get("flowers.html", function (data) {
    var elems = data.filter("div").addClass("dcell");
    elems.slice(0, 3).appendTo("#row1");
    elems.slice(3).appendTo("#row2");
});

var jsonReq = $.getJSON("additionalflowers.json", function (data) {
    $("#flowerTpl").template({ flowers: data }).appendTo("#row3");
});

$("<div id=errorSummary>").text("Please correct the following errors:")
    .append("<ul id='errorsList'></ul>").hide().insertAfter("h1");

$("#orderForm").validate({
    highlight: function(element, errorClass) {
        $(element).addClass("invalidElem");
    },
    unhighlight: function(element, errorClass) {
        $(element).removeClass("invalidElem");
    },
    errorContainer: "#errorSummary",
    errorLabelContainer: "#errorsList",
    wrapper: "li",
    errorElement: "div"
});

var plurals = {


```

```

        aster: "Asters", daffodil: "Daffodils", rose: "Roses",
        peony: "Peonies", primula: "Primulas", snowdrop: "Snowdrops",
        carnation: "Carnations", lily: "Lillies", orchid: "Orchids"
    }

    $.when(flowerReq, jsonReq).then(function() {
        $("input").each(function(index, elem) {
            $(elem).rules("add", {
                required: true,
                min: 0,
                digits: true,
                remote: {
                    url: "http://node.jacquisflowershop.com/stockcheck",
                    type: "post",
                    global: false
                },
                messages: {
                    required: "Please enter a number for " + plurals[elem.name],
                    digits: "Please enter a number for " + plurals[elem.name],
                    min: "Please enter a positive number for " + plurals[elem.name]
                }
            });
        });
        $.change(function(e) {
            if ($("#orderForm").validate().element($(e.target))) {
                var total = 0;
                $("input").each(function(index, elem) {
                    total += Number($(elem).val());
                });
                $("#total").text(total);
            }
        });
    });

    $("#orderForm button").click(function(e) {
        e.preventDefault();
        var formData = $("#orderForm").serialize();
        $("body *").not("#popup, #popup *").css("opacity", 0.5);
        $("input").attr("disabled", "disabled");
        $("#popup").show();
        $.ajax({
            url: "http://node.jacquisflowershop.com/order",
            type: "post",
            data: formData,
            dataType: "json",
            dataFilter: function (data, dataType) {
                data = $.parseJSON(data);
                var cleanData = {
                    totalItems: data.totalItems,
                    totalPrice: data.totalPrice
                };
                delete data.totalPrice; delete data.totalItems;
                cleanData.products = [];
                for (prop in data) {
                    cleanData.products.push({

```

```

        name: plurals[prop],
        quantity: data[prop]
    })
    }
    return cleanData;
},
converters: {"text json": function(data) { return data; } },
success: function (data) {
    processServerResponse(data);
},
complete: function () {
    $("body *").not("#popup, #popup *").css("opacity", 1);
    $("input").removeAttr("disabled");
    $("#popup").hide();
}
});
});

function processServerResponse(data) {
    if (data.products.length > 0) {
        $("body > *:not(h1)").hide();
        $("#summaryForm").show();
        $("#productRowTpl")
            .template({ rows: data.products }).appendTo("tbody");
        $("#totalitems").text(data.totalItems);
        $("#totalprice").text(data.totalPrice);
    } else {
        var elem = $("input").get(0);
        var err = new Object();
        err[elem.name] = "No products selected";
        $("#orderForm").validate().showErrors(err);
        $(elem).removeClass("invalidElem");
    }
}

var total = $("#buttonDiv")
    .prepend("<div>Total Items: <span id=total>0</span></div>")
    .css({ clear: "both", padding: "5px" });
$("<div id=bbbox />").appendTo("body").append(total).css("clear: left");

$("input").change(function (e) {
    var total = 0;
    $("input").each(function (index, elem) {
        total += Number($(elem).val());
    });
    $("#total").text(total);
});

function handleArrowMouse(e) {
    var propValue = e.type == "mouseenter" ? "-50px 0px" : "0px 0px";
    $(this).css("background-position", propValue);
}

function handleArrowPress(e) {

```

```

    var elemSequence = ["row1", "row2", "row3"];
    var visibleRow = $("div.drow:visible");
    var visibleRowIndex =
        jQuery.inArray(visibleRow.attr("id"), elemSequence);

    var targetRowIndex;
    if (e.target.id == "left") {
        targetRowIndex = visibleRowIndex - 1;
        if (targetRowIndex < 0) {targetRowIndex = elemSequence.length - 1};
    } else {
        targetRowIndex = (visibleRowIndex + 1) % elemSequence.length;
    }

    visibleRow.fadeOut("fast", function () {
        $("#"+ elemSequence[targetRowIndex]).fadeIn("fast");
    });
}
});
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form id="orderForm" method="post" action="http://node.jacquisflowershop.com/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow"></div>
                <div id="row2" class="drow"></div>
                <div id="row3" class="drow"></div>
            </div>
            <div id="buttonDiv"><button type="submit">Place Order</button></div>
        </form>
        <form id="summaryForm" method="post" action="">
            <div id="summary">
                <h3>Order Summary</h3>
                <table border="1">
                    <thead>
                        <tr><th>Product</th><th>Quantity</th>
                    </thead>
                    <tbody>
                    </tbody>
                    <tfoot>
                        <tr><th>Number of Items:</th><td id="totalitems"></td></tr>
                        <tr><th>Total Price:</th><td id="totalprice"></td></tr>
                    </tfoot>
                </table>
                <div id="buttonDiv2"><button type="submit">Complete Order</button></div>
            </div>
        </form>
    </body>
</html>

```

## 16.7.1 添加新表单

我做的第一件事是在页面的静态HTML中加上一个新表单：

```
...
<form id="summaryForm" method="post" action="">
  <div id="summary">
    <h3>Order Summary</h3>
    <table border="1">
      <thead>
        <tr><th>Product</th><th>Quantity</th>
      </thead>
      <tbody>
      </tbody>
      <tfoot>
        <tr><th>Number of Items:</th><td id="totalitems"></td></tr>
        <tr><th>Total Price:</th><td id="totalprice"></td></tr>
      </tfoot>
    </table>
    <div id="buttonDiv2"><button type="submit">Complete Order</button></div>
  </div>
</form>
...
```

这是新功能的核心。当用户把他们选中的产品提交到服务器时，我们的脚本就会利用表单中的表格显示服务器返回的数据。

---

**提示** 在前面的例子里我使用了\$('form')选择器，然而现在页面中有两个表单（再使用这个选择器就不合适了），因此我改用form元素的id属性来引用它们。

---

我并不想立即显示新的表单，因此把它添加到脚本中需要隐藏的元素列表当中：

```
...
$('#row2, #row3, #popup, #summaryForm').hide();
...
```

而且，你一定想到了，既然添加了新元素，就一定要为它们编写样式：

```
...
#summary {text-align: center}
table {border-collapse: collapse; border: medium solid black; font-size: 18px;
  margin: auto; margin-bottom: 5px;}
th {text-align: left}
th, td {padding: 2px}
tr > td:nth-child(1) {text-align: left}
tr > td:nth-child(2) {text-align: right}
...
```

这些样式确保表格居中显示在浏览器窗口之中，并且各列的文本也都对齐到正确的位置。

## 16.7.2 完成Ajax请求

接下来完成Ajax请求：

```
...
$("#orderForm button").click(function (e) {
    e.preventDefault();
    var formData = $("#orderForm").serialize();
    $("body *").not("#popup, #popup *").css("opacity", 0.5);
    $("input").attr("disabled", "disabled");
    $("#popup").show();
    $.ajax({
        url: "http://node.jacquisflowershop.com/order",
        type: "post",
        data: formData,
        dataType: "json",
        dataFilter: function(data, dataType) {
            data = $.parseJSON(data);
            var cleanData = {
                totalItems: data.totalItems,
                totalPrice: data.totalPrice
            };
            delete data.totalPrice; delete data.totalItems;
            cleanData.products = [];
            for (prop in data) {
                cleanData.products.push({
                    name: plurals[prop],
                    quantity: data[prop]
                });
            }
            return cleanData;
        },
        converters: {"text json": function(data) { return data; } },
        success: function (data) {
            processServerResponse(data);
        },
        complete: function () {
            $("body *").not("#popup, #popup *").css("opacity", 1);
            $("input").removeAttr("disabled");
            $("#popup").hide();
        }
    });
});
...
```

我在complete选项中删除了那个特意的延迟，并为请求添加了dataFilter、converters和success选项。我使用dataFilters选项指定了一个函数，利用它把服务器返回的JSON数据格式化成为更有用的形式。服务器返回给我的JSON字符串如下：

---

```
{"aster": "4", "daffodil": "1", "snowdrop": "2", "totalItems": 7, "totalPrice": "15.93"}
```

---

我分析这些JSON数据并重新组织数据的结构，最终得到：

```
{
  "totalItems": 7, "totalPrice": "15.93",
  "products": [
    { "name": "Asters", "quantity": "4" },
    { "name": "Daffodils", "quantity": "1" },
    { "name": "Snowdrops", "quantity": "2" }
  ]
}
```

这种格式有两个优点。首先，这样我就能把products属性直接传递给template方法，因此它更便于结合数据模板使用。其次，我可以根据products.length的值知道用户选择了几种产品。这是两个非常细微的改进，然而我希望尽可能多地从前面的章节中整合功能进来。注意，我把产品的名字（比如orchid）替换成了复数形式（orchids）。

在成功把JSON数据解析为JavaScript对象之后（使用parseJSON方法，这个方法我会在第34章详细讲解），我希望禁用内建的转换器（它做同样的事）。最后，我为JSON数据定制了一个转换器，它不做任何处理，直接返回服务器响应的数据：

```
...
converters: { "text json": function(data) { return data; } }
...
```

### 16.7.3 处理数据

我为ajax方法调用的success选项指定了一个回调函数processServerResponse，它的定义如下：

```
...
function processServerResponse(data) {
  if (data.products.length > 0) {
    $("body > *:not(h1)").hide();
    $("#summaryForm").show();
    $("#productRowTpl").template({ rows: data.products }).appendTo("tbody");
    $("#totalItems").text(data.totalItems);
    $("#totalPrice").text("$" + data.totalPrice);
  } else {
    var elem = $("input").get(0);
    var err = new Object();
    err[elem.name] = "No products selected";
    $("#orderForm").validate().showErrors(err);
    $(elem).removeClass("invalidElem");
  }
}
...
```

如果来自服务器的数据包含产品信息，我就隐藏页面中所有不需要的数据（包括最初的表单元素和使用脚本追加的元素），然后显示新的表单。使用下面的数据模板生成表格内容：

```
...
<script id="productRowTpl" type="text/x-handlebars-template">
  {{#rows}}
    <tr><td>{{name}}</td><td>{{quantity}}</td></tr>
  {{/rows}}
```

```
</script>
```

```
...
```

这个模板为用户选中的每个产品生成一个产品行。最后，我使用text方法为总数和总价单元格设置正确的内容。

```
...
```

```
$("#totalitems").text(data.totalItems);
```

```
$("#totalprice").text(data.totalPrice);
```

```
...
```

不过，如果服务器返回的数据里不包含任何产品信息（用户根本一个产品也没有选择），我就做点别的什么事情。首先，选中第一个input元素：

```
...
```

```
var elem = $("input").get(0);
```

```
...
```

然后生成一个对象，它有一个属性。属性的名字是这个input元素的name属性值，属性值是显示给用户看的提示信息。然后，我在form元素上调用validate方法，并接着调用返回结果的showErrors方法：

```
...
```

```
var err = new Object();
```

```
err[elem.name] = "No products selected";
```

```
$("#orderForm").validate().showErrors(err);
```

```
...
```

这样我们就能手动为验证系统注入一个错误，从而享受前面准备好结构与格式的便利。我不得不提供一个元素的名字，以便验证系统高亮问题发生的位置。如图16-7所示，效果并不是十分理想。

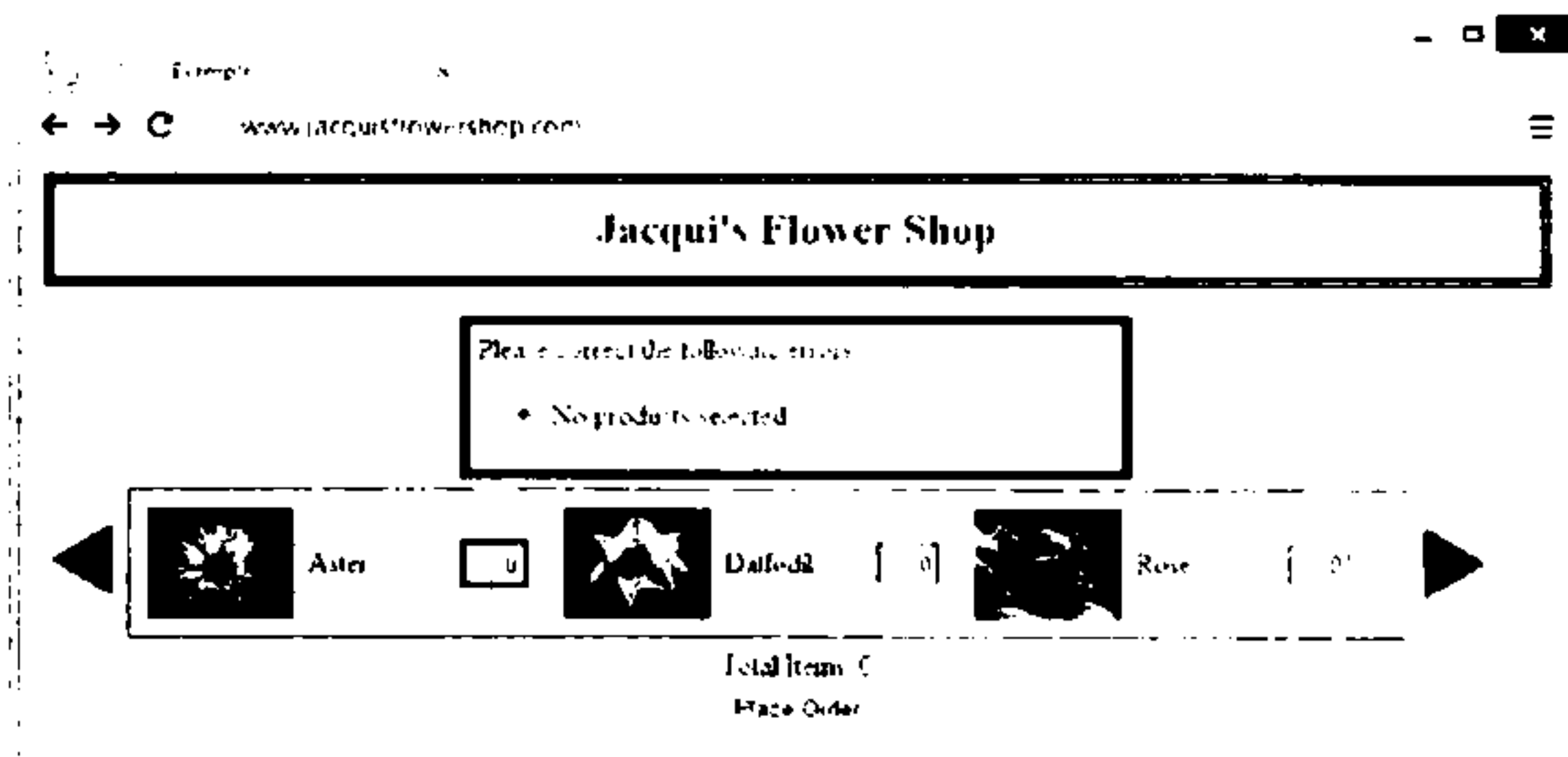


图16-7 显示“未选中任何产品”提示信息

这里显示了一条普通提示信息，然而却只高亮了一个input元素，这可能会让用户感到迷惑。为解决这个问题，我删除了验证插件添加的用来进行高亮显示的invalidElem类：

```
...
```

```
$(elem).removeClass("invalidElem");
```

```
...
```

新的效果见图16-8。



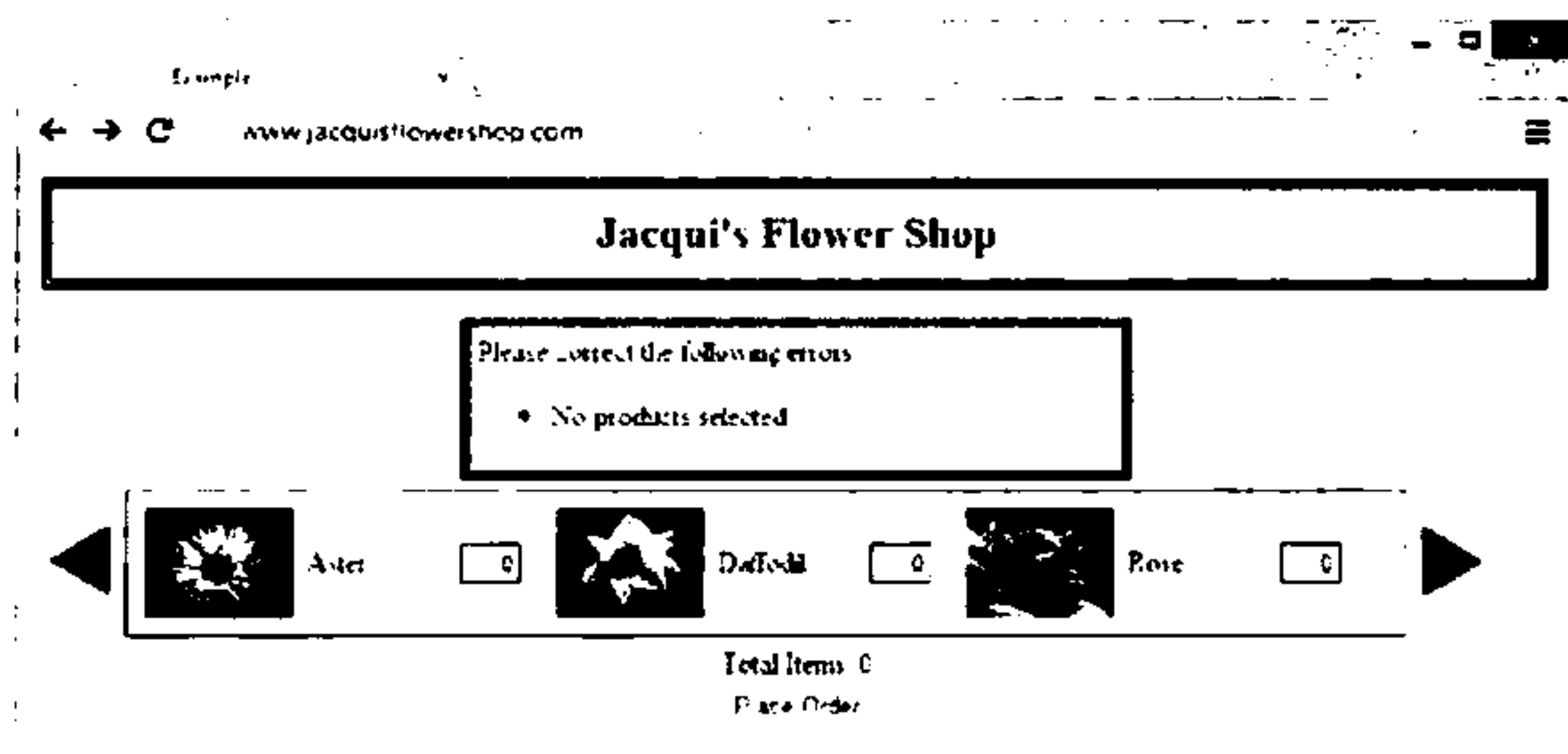


图16-8 删除关联在出错提示上的input元素的高亮效果

## 16.8 小结

本章把本书这一部分中讲到的主题和其他功能结合到一起，重构了示例页面。我们广泛地使用了Ajax技术（快捷方法和底层方法都有应用），还应用了两个数据模板，并且使用验证插件执行了本地检查和远程检查，且手动控制错误信息的显示。在本书第四部分，我们将把目光转向jQuery UI，等到再一次重构示例页面时，它将变成一个完全不同的样子。

与其他JavaScript库相比，下载并安装jQuery UI有一点点复杂。安装过程虽说并不痛苦，但还是需要做一些解释。没错，这就是本章的任务。对学习本书来说，我们只需要设置好开发环境。尽管如此，我仍然详细讲解了如何在生产环境下部署压缩版jQuery UI，以及如何使用CDN版jQuery UI。

**注意** 正如我在第1章所提到的，随着版本1.10的发布，jQuery UI API也有些许变化，在后面的章节中我会指出这些变化。

## 17.1 获取 jQuery UI

jQuery UI的功能划分为四大块，而我们能够自由选择下载和配置需要的部分。本部分涵盖jQuery UI的所有特性，不过对于一个真实的Web项目而言，我们需要删除那些不必要的部分以得到一个方便浏览器下载的体积更小的库。

**提示** 虽然jQuery UI并不是基于jQuery的唯一UI库，但却是最流行的UI库。另一个选择是jQuery Tools，它也是开源的，不需要任何许可证，也没有任何下载限制，地址是<http://flowplayer.org/tools>。还有一些商业性质的UI库，比如jQWidgets ([www.jqwidgets.com](http://www.jqwidgets.com)) 和Wijmo (<http://wijmo.com>)。此外，当然还有jQuery Mobile，详见本书第五部分。

### 17.1.1 选定一个主题

在定制jQuery UI库之前，我们需要先选定一个主题。jQuery UI是高度可配置的，我们能够改变用到的每一个功能的每一个小部件的外观。事实上，它的选项是如此之多，甚至多得有点儿“过分”。jQuery UI站点提供了一个方便用户创建定制主题的工具，而且还提供了许多事先定义好的主题供我们选用。

直接打开<http://jqueryui.com>，然后点击Themes按钮，你将看到ThemeRoller页面，如图17-1所示。这个页面由jQuery UI组件及左边的控制面板组成，后者方便我们配置主题设置。

如果你的站点或者应用程序有一个明确的视觉主题，并且希望jQuery UI也符合这个主题，那么Roll Your Own标签（默认选中）就是为你准备的。在这个标签中，你可以修改jQuery UI使用CSS样式的任何一个部分。

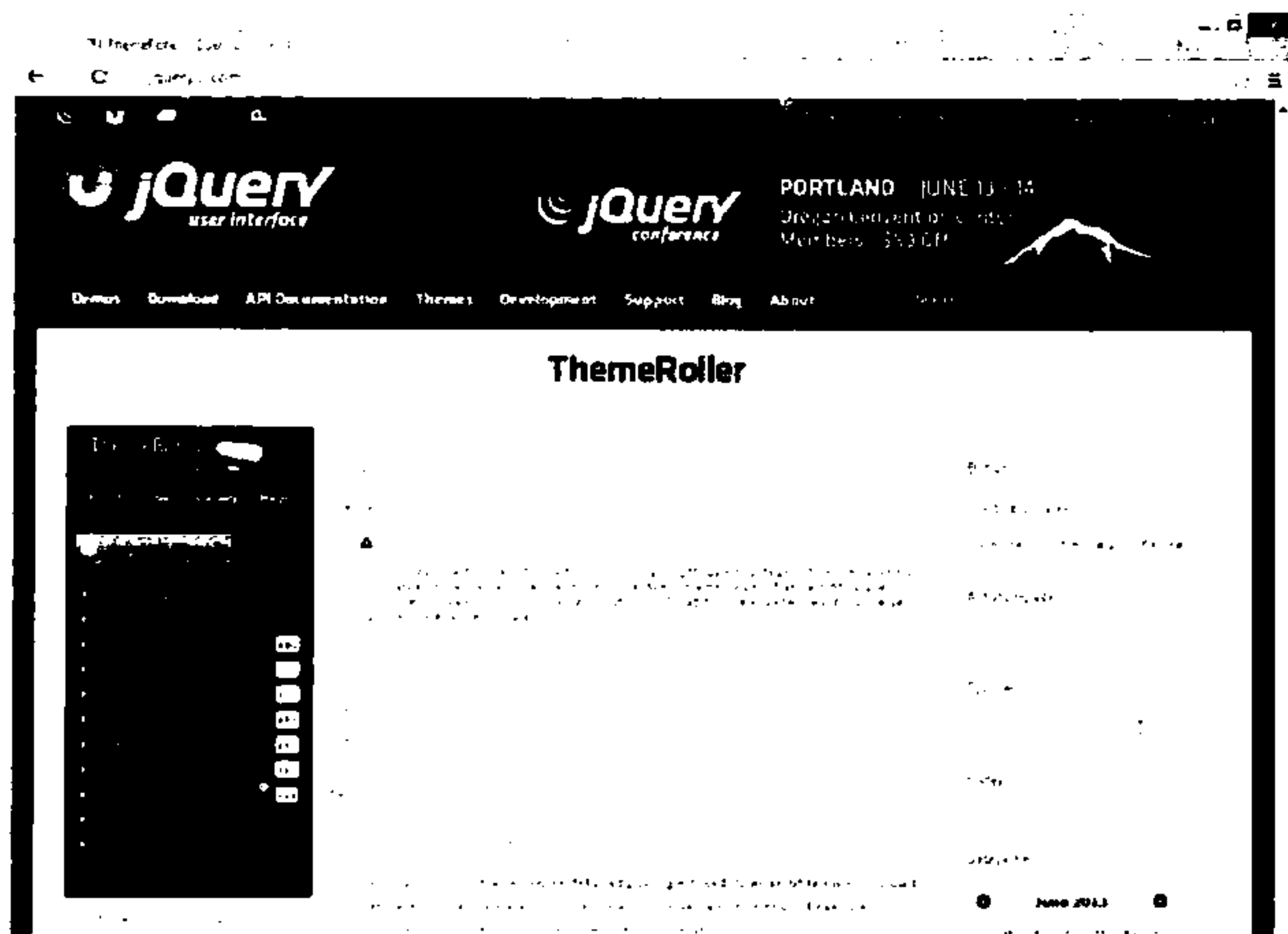


图17-1 jQuery UI网站的主题页

Gallery标签存放的是预定义主题。编写本书之时，这个标签中共有24个主题，其排列顺序是从朴素淡雅到光彩艳丽。如图17-2所示，点击一个主题时，页面的组件部分会随即更新，从而让你看到使用这个主题后Web站点（或应用程序）的预览效果。

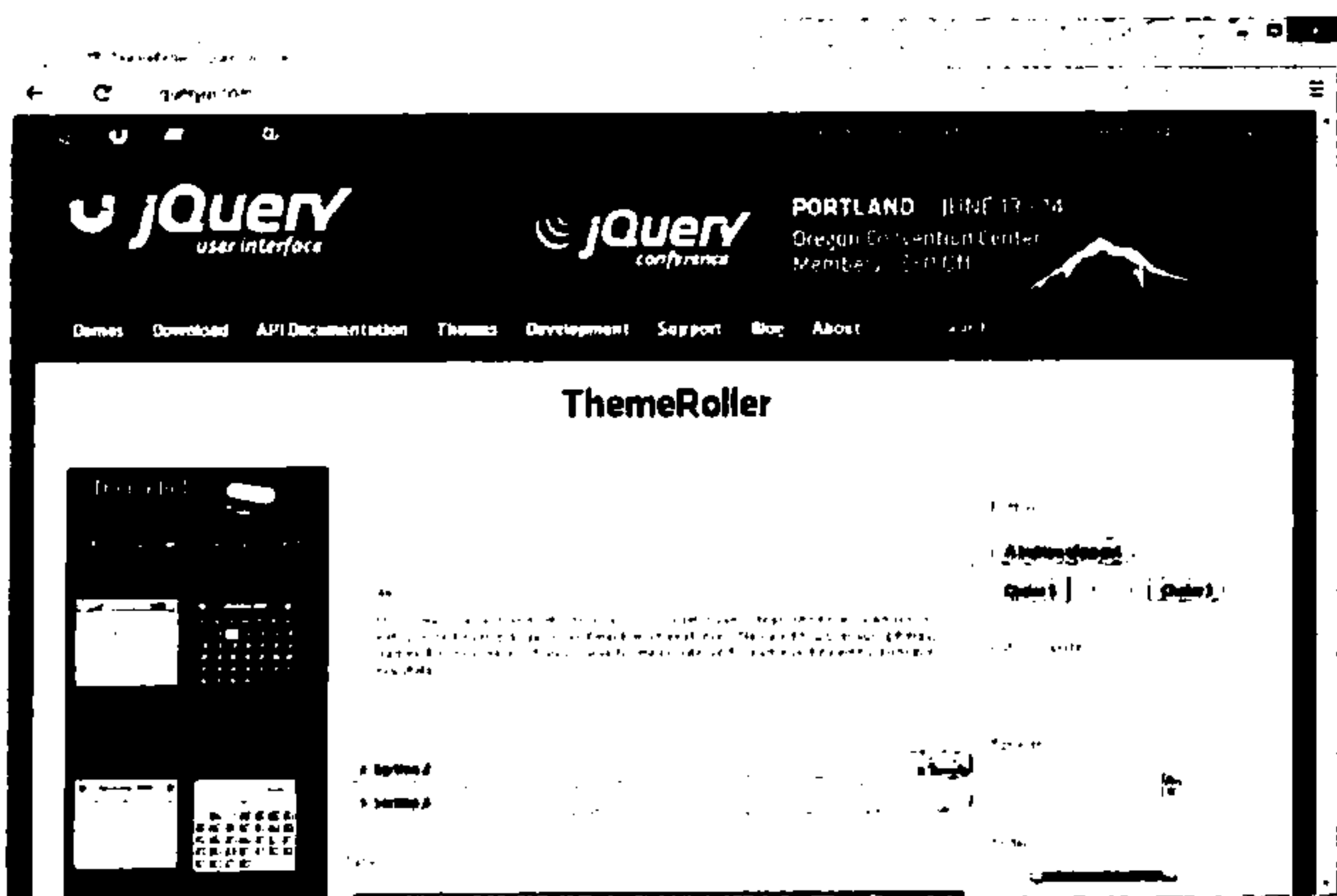


图17-2 主题库中Sunny（阳光）主题的预览效果

jQuery UI的默认主题名为UI lightness（淡雅UI），这个主题由于对比度不够，并不适合印在书页

上，所以我选择了Sunny主题，它在书上的显示效果好一些。现在，我们只需要记住要使用这个主题就够了，并不需要对这个主题作任何修改。这些主题印在纸上的效果远不如显示在屏幕上的效果好，因此我建议你挨个儿查看这些主题，直到选中自己喜欢的一个。

---

**提示** 你完全可以有自己的选择，不必和我一样。不过，如果你选择了另一种主题，看到的效果就会与书上的显著不同。

---

### 17.1.2 生成jQuery UI定制下载版本

现在你已经有了心仪的主题，可以开始生成jQuery UI定制下载包了。点击页面顶部的Download按钮，转到Build Your Download页面。

第一步是选择需要下载的版本。在写作本书时，最新版是1.10.3，这也是我在本书中使用的版本。

除了指定jQuery UI的版本外，这个页面列有一个jQuery UI组件清单，分为四大块：UI Core（核心）、Interactions（交互组件）、Widgets（小部件）和Effects（特效）。

通过选择自己项目所需的功能，我们可以生成一个对于浏览器来说体积较小的下载包。我认为这是一个好主意，但并不想使用这一功能。我认为节省传输jQuery UI所耗带宽的最好方式是使用CDN网络，本章后面会介绍这一方法。

就本章来说，我们需要所有的组件，因此请务必选中所有的组件。

---

**提示** 列表中的一些组件依赖于其他组件。不过不必担心，定制下载包页面会自动处理依赖问题，在你选中某个组件时自动选中依赖的组件。

---

下一步是选择需要的主题。如图17-3所示，选择器位于页面底部，Download按钮正上方。



图17-3 选择一个主题

选定所有组件之后，我们选好自己喜欢的主题，然后选择稳定版本，点击Download按钮即可下载定制好的jQuery UI库。

## 17.2 安装 jQuery UI 开发版

jQuery UI 下载包中包含开发版和生产环境版所需要的全部文件。对于本书来说，我们需要使用源代码未压缩的开发版文件。遇到问题时，使用这个版本更容易通过查看 jQuery UI 源代码查找深层次的原因。我们需要复制以下文件到示例页面所在的目录：

- js/jquery-ui-1.10.3.custom.js 文件
- themes/sunny/jquery-ui-1.10.3.custom.css 文件
- themes/sunny/images 文件夹

你会发现，ui 和 themes 目录下还有一些用于独立组件和功能的 JavaScript 及 CSS 文件。我们不需要使用它们，只有当需要使用一个有限功能的 jQuery UI 库时，它们才会派上用场。

---

**提示** 下载的 JavaScript 文件和 CSS 文件名中都包含着版本号。写作本书时，最新版本是 1.10.3。jQuery UI 的开发相当活跃，你所下载的很可能是一个更新版本。若是如此，你需要修改 HTML 例子文件中对 jQuery UI 文件的引用代码。

---

### 添加 jQuery UI 到 HTML 文档

接下来我们把 jQuery UI 添加到 HTML 文档。如代码清单 17-1 所示，我们在页面中添加一个 script 和一个 link 元素，引用前面列出的 JavaScript 文件和 CSS 文件。

代码清单 17-1 把 jQuery UI 添加到 HTML 页面

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  $(document).ready(function() {
    $('a').button();
  });
</script>
</head>
<body>
  <a href="http://apress.com">Visit Apress</a>
</body>
</html>
```

---

**提示** 这里不需要直接引用 images 目录，只要把 images 目录和 css 文件放在同一个地方，jQuery UI 就会自动找到它需要的资源文件。

---

**提示** jQuery UI依赖于jQuery。我们必须在页面中包含jQuery才能使用jQuery UI。

代码清单17-1所示页面里包含了一个简单的测试，用于检查jQuery UI是否能够正常工作。如果在浏览器中查看这个页面，应该会看到如图17-4所示的孤零零的一个按钮。不必理会代码清单中script元素内调用的button方法是什么意思，我会在第18章详细介绍它的功能和原理。

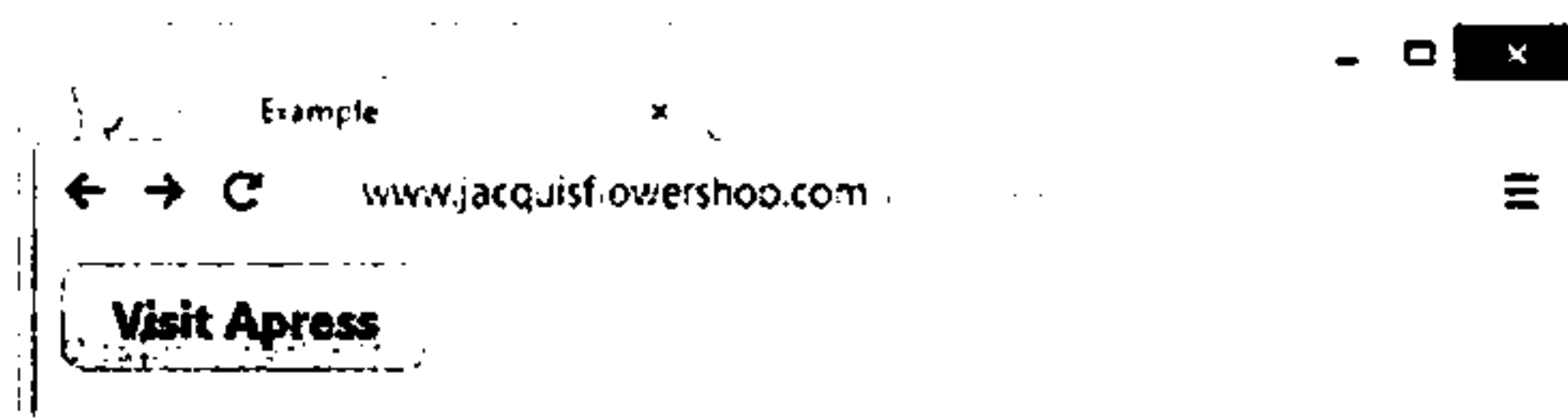


图17-4 检查jQuery UI是否安装并正确地添加到页面

如果未正确指定至两个文件中任意一个的路径，你就只能看到一个如图17-5所示的简单元素。



图17-5 标识导入jQuery UI到页面失败的问题

## 17.3 把 jQuery UI 部署到生产环境

结束手头项目的开发，准备好把它部署到生产环境时，我们应该使用下载包中的jQuery UI压缩版本。这些文件的体积要小很多，不过可读性很差（不方便调试）。使用压缩版本，只需要把以下文件复制到Web服务器上的适当目录：

- ❑ js/jquery-ui-1.10.3.custom.min.js文件
- ❑ themes/sunny/jquery-ui-1.10.3.custom.css文件
- ❑ themes/sunny/images文件夹

和开发版本一样，images目录要和CSS文件放在同一目录；只有JavaScript文件有所变化。

### 使用CDN网络上的jQuery UI

在第5章我提到过使用CDN网络上的jQuery库。如果喜欢这个方案，那你一定很高兴知道这个方案也支持jQuery UI。谷歌和微软都在它们的CDN网络上部署了jQuery UI文件。由于微软不仅提供jQuery UI的JavaScript文件，还提供了标准主题服务，因此我在下面的例子里使用了微软提供的服务。

要使用CDN服务，我们需要文件所需的URL。对于微软服务来说，首先请访问[www.asp.net/ajaxlibrary/cdn.ashx](http://www.asp.net/ajaxlibrary/cdn.ashx)。从上往下看，你会看到一个jQuery UI Releases on the CDN链接，点击这个链接就

会看到jQuery UI的许多版本。点击与你的项目使用版本相符的链接，你会看到jQuery UI库正常版与压缩版文件所在的URL。写作本书时，压缩版文件的URL就是：

---

```
http://ajax.aspnetcdn.com/ajax/jquery.ui/1.10.3/jquery-ui.min.js
```

---

页面后面还展示了每种jQuery UI预定义主题，主题的下面是相应CSS文件的URL。Sunny主题的URL是：

---

```
http://ajax.aspnetcdn.com/ajax/jquery.ui/1.10.3/themes/sunny/jquery-ui.css
```

---

如代码清单17-2所示，使用CDN上的这些文件，只需要将script和link元素中的URL由引用本地文件改为使用上面的URL。

#### 代码清单17-2 使用CDN网络上的jQuery UI

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="http://ajax.aspnetcdn.com/ajax/jquery.ui/1.10.3/jquery-ui.min.js"
    type="text/javascript"></script>
  <link
    href="http://ajax.aspnetcdn.com/ajax/jquery.ui/1.10.3/themes/sunny/jquery-ui.css"
    rel="stylesheet" />
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script type="text/javascript">
    $(document).ready(function() {
      $("a").button();
    });
  </script>
</head>
<body>
  <a href="http://apress.com">Visit Apress</a>
</body>
</html>
```

同样，我们可以在浏览器中载入示例页面，查看页面中按钮的样式是否正确，以此确认使用的URL没有问题。

## 17.4 小结

本章演示了定制jQuery UI下载包的过程。jQuery UI为我们的Web应用程序提供了大量灵活的功能和默认的外观。我特别喜欢ThemeRoller应用程序，它为适应一个现有视觉主题提供了一种创建完全定制主题的优雅方式，特别适合为现有的企业级站点添加jQuery UI支持。下一章，我们从最常用的功能区——小部件——开始介绍jQuery UI提供的具体功能。

# 按钮、进度条与滑动条组件

现在，我们已经配置好并下载安装了jQuery UI，可以看看它包含的组件了。这些组件是jQuery UI的主要功能块，虽然还有一些其他功能（比如第35章介绍的特效），但正是这些组件真正让jQuery UI名声在外。

本章要介绍3个最简单的组件：按钮、进度条和滑动条组件。所有组件都有一些共同的特征：选项、方法和事件。精通一个组件就为使用所有其他组件打下了坚实基础，因此我会在本章开头花些时间介绍一些背景知识。

我们很难把所有的组件都“置入”花店示例，因此本书这一部分的许多示例都是些规模很小、自包含的HTML文档，每个文档演示一个组件。我们会在第26章回顾花店示例，重构这个示例以便包含jQuery UI功能。本章概要见表18-1。

表18-1 本章概要

问 题	解决方法	代码清单
如何得到jQuery UI按钮	选中目标元素，然后调用button方法	1
如何配置jQuery UI按钮	调用button方法时传递给button方法一个选项对象，或者使用option方法	2、3
如何在jQuery UI按钮上使用图标	使用icons选项	4
如何在jQuery UI按钮上使用自定义图标	将img元素作为按钮的内容使用	5
如何删除jQuery UI按钮组件	使用destroy方法	6
如何启用或禁用jQuery UI按钮	使用enable或者disable方法	7
如何更新jQuery UI按钮的状态以便告诉底层元素已经发生了某些变化	使用refresh方法	8
如何响应正在生成jQuery UI按钮这一事件	为create事件绑定一个处理函数	9
如何使不同类型的底层元素呈现统一样式的按钮	在input、button或者a元素上调用button方法	10
如何得到切换按钮	在一个checkbox上调用button方法	11
如何得到一组按钮	使用buttonset方法	12、13
如何得到jQuery UI进度条	使用progressbar方法	14、15
如何获取或者设置进度条的进度	使用value方法	16
如何响应进度条的变化	为create、change或者complete事件指定回调函数	17



(续)

问 题	解决方法	代码清单
如何得到jQuery UI滑动条组件	使用slide方法	18
如何改变jQuery UI滑动条组件的布局方向	使用orientation选项	19、20
如何在用户点击滑动条组件时以动画的方式移动滑动把手	使用animate选项	21
如何得到可以自定义数值范围的滑动条组件	使用range和values选项	22
如何使用脚本控制jQuery UI滑动条组件	使用value或者values方法	23
如何响应滑动把手的位置变化	处理start、stop、change或者slide事件	24

### 新版jQuery UI与本章有关的变化

自jQuery UI 1.10起, 进度条组件可以用于显示模糊任务的进度。要了解细节或者查看示例, 请参阅18.3节。

## 18.1 jQuery UI 按钮

我们即将看到第一个jQuery UI组件, 它很好地展示了jQuery UI世界。按钮(button)组件相对简单, 但能使HTML页面发生变化。按钮组件会在按钮元素和a元素上应用jQuery UI主题样式, 这意味着按钮元素的大小、形状、字体和颜色都会发生变化以适应我们选中的主题(在17章生成自定义jQuery UI主题包时)。如代码清单18-1所示, 得到一个jQuery UI组件相当容易。

代码清单18-1 一个简单的HTML文档

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <link rel="stylesheet" type="text/css" href="styles.css"/>

  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}:</label>
      <input name="{{product}}" data-price="{{price}}" data-stock="{{stocklevel}}"
        value="0" required />
    </div>
    {{/flowers}}
  </script>
```

```

<script type="text/javascript">
    $(document).ready(function () {
        $.ajax("mydata.json", {
            success: function (data) {
                var tmplData = $("#flowerTpl")
                    .template({ flowers: data }).filter("*");
                tmplData.slice(0, 3).appendTo("#row1");
                tmplData.slice(3).appendTo("#row2");
            }
        });

        $("#button").button();
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="oblock">
            <div class="dtable">
                <div id="row1" class="drow">
                </div>
                <div id="row2" class="drow">
                </div>
            </div>
            <div id="buttonDiv"><button type="submit">Place Order</button></div>
        </form>
    </body>
</html>

```

要得到一个jQuery UI按钮，我们只需要使用jQuery选中目标元素，然后调用button方法，剩下的事情由jQuery UI打理。这个示例的效果见图18-1。

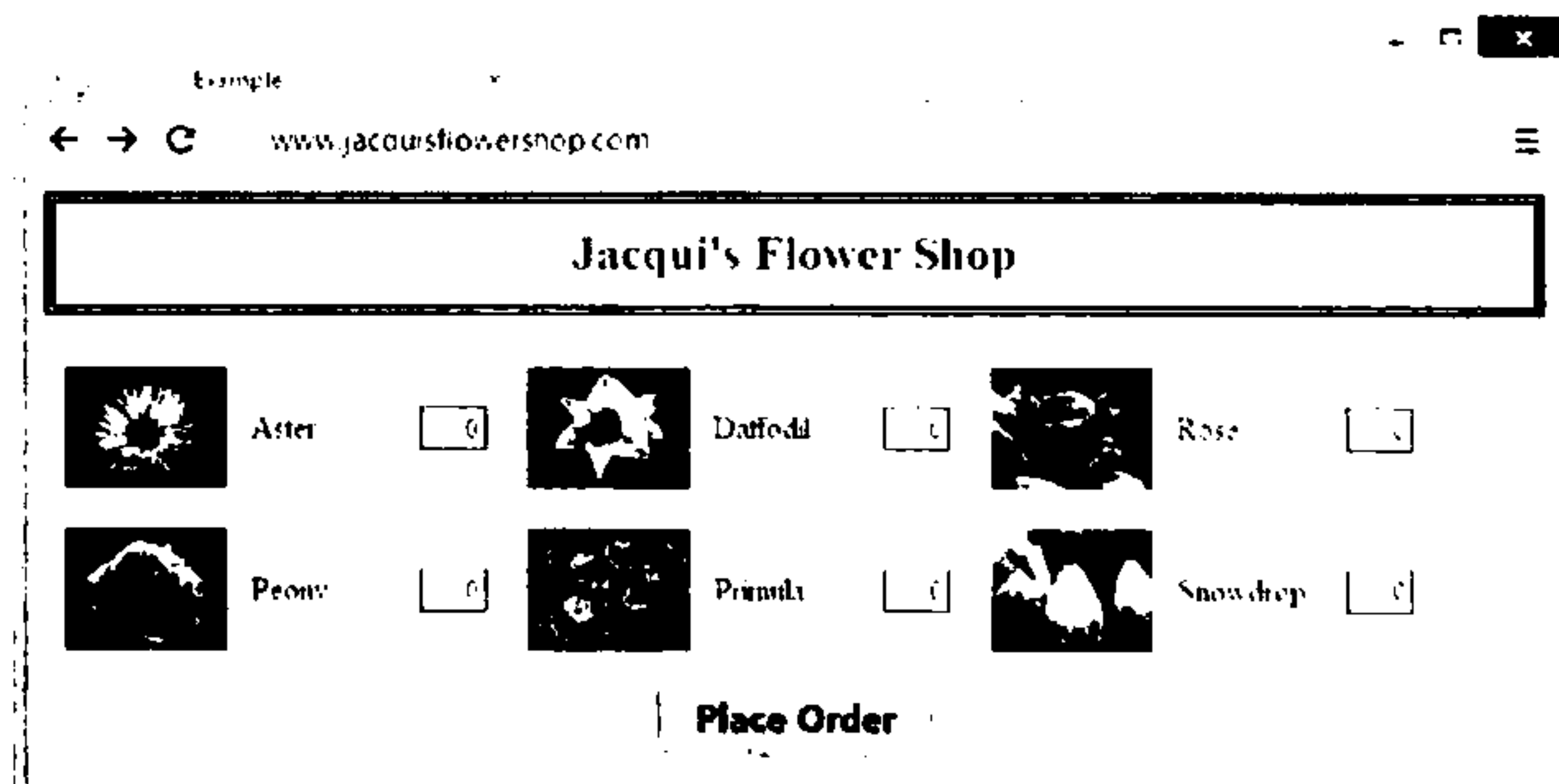


图18-1 jQuery UI按钮组件

**提示** 注意，我们是在一个jQuery结果集对象上调用的button方法。jQuery与jQuery UI结合得非常密切，也就是说使用jQuery UI只是使用jQuery核心库技术（我在本书前面已经多次展示过）的一个自然延伸。

与所有的jQuery UI组件一样，图中所示的按钮组件是由一系列CSS样式施加到已有HTML元素上的结果。button方法把按钮元素由：

```
<button type="submit">Place Order</button>
```

变为：

```
<button type="submit" class="ui-button ui-widget ui-state-default ui-corner-all
    ui-button-text-only" role="button" aria-disabled="false">
    <span class="ui-button-text">Place Your Order</span>
</button>
```

这个方法很巧妙，因为它只用到标准的HTML元素。这意味着我们在创建HTML时完全可以像往常一样，无需为了jQuery UI而做出什么改变。

### 18.1.1 配置按钮组件

jQuery UI的按钮组件有许多属性，我们可以利用这些属性配置按钮，从而控制按钮组件的创建方式。表18-2列出了这些属性。

表18-2 按钮组件的选项属性

属 性	描 述
disabled	得到或设置按钮的可用状态，值为true表示按钮不可用。jQuery UI并不考虑底层HTML元素的实际状态
text	得到按钮文本或者设定按钮文本。若icons选项的值为false，则jQuery UI会忽略这一设置（总是显示按钮文本）
icons	得到按钮图标设置或者设定按钮上要显示的图标
label	得到或者设置按钮上显示的文本

我们可以通过两种方法使用这些选项。第一种见代码清单18-2中列出的加粗代码：将一个映射对象用作button方法的参数。

代码清单18-2 使用映射对象配置按钮组件

```
...
<script type="text/javascript">
    $(document).ready(function () {
        $.ajax("mydata.json", {
            success: function (data) {
                var tmplData = $("#flowerTpl")
```

```

        .template({ flowers: data }).filter("");
        tmplData.slice(0, 3).appendTo("#row1");
        tmplData.slice(3).appendTo("#row2");
    }
    });
    $("button").button({
        label: "Place Your Order",
        disabled: true
    });

    $("button").button("option", "disabled", false);
    });
</script>
...

```

我使用label选项设定了按钮上显示的文本，还使用disabled选项禁用了这个按钮。使用映射对象可以在初始化一个按钮时对它做一些配置，它延用了我们最近经常使用的Ajax请求配置风格（参见第15章）。

代码清单18-2展示了适用于改变已有按钮组件的第二种方法，即给某个属性赋一个新值，见代码：

```

...
$("button").button("option", "disabled", false);
...

```

我们又调用了一次button方法，不过这次提供了三个参数。第一个参数是option，告知jQuery UI我们要修改设置。第二个参数是我们打算改变的选项，第三个参数则是这个选项的新值。该行语句把disabled选项设置为false，改变了该选项的初始设置（创建组件时的设置），使按钮可用。

这两种技术可以结合使用。如代码清单18-3所示，我们这样调用button方法：第一个参数是option，第二个参数是一个映射对象，这样就可以一步应用多个设置。

### 代码清单18-3 使用option参数和映射对象参数

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $.ajax("mydata.json", {
            success: function (data) {
                var tmplData = $("#flowerTpl")
                    .template({ flowers: data }).filter("");
                tmplData.slice(0, 3).appendTo("#row1");
                tmplData.slice(3).appendTo("#row2");
            }
        });

        $("button").button();

        $("button").button("option", {
            label: "Place Your Order",
            disabled: false
        });

        console.log("Enabled? " + $("button").button("option", "disabled"));
    });

```

```
    });
</script>
...
```

我们使用同样的方法读取选项的值，在本例中使用两个参数调用button方法。第一个参数仍是字符串option，第二个参数则是我们希望获取值的选项名称：

```
...
console.log("Disabled? " + $("button").button("option", "disabled"));
...
```

以这种方式调用button方法，button方法会返回参数选项的值。这行语句读取disable选项的值并把它输出到控制台，产生以下输出：

---

```
Disabled? false
```

---

### 18.1.2 在按钮上使用jQuery UI图标

jQuery UI主题内建了许多图标。我们可以把这些图标用于任何目的，当然也包括把它显示在按钮上。代码清单18-4展示了在jQuery UI按钮上使用图标的具体做法。

代码清单18-4 在按钮上显示图标

```
...
<script type="text/javascript">
    $(document).ready(function () {
        $.ajax("mydata.json", {
            success: function (data) {
                var tplData = $("#flowerTpl")
                    .template({ flowers: data }).filter("");
                tplData.slice(0, 3).appendTo("#row1");
                tplData.slice(3).appendTo("#row2");
            }
        });

        $("button").button({
            icons: {
                primary: "ui-icon-star",
                secondary: "ui-icon-circle-arrow-e"
            }
        });
    });
</script>
...
```

我们使用icons选项指定要显示的图标。按钮组件有两个位置可以显示图标。主图标显示在文本的左边，副图标显示在文本的右边，两个图标是在同一个配置对象（icons选项的值）里指定的。我们可以省略其中的任意一个，从而只显示一个图标。如图18-2所示，图标本身都很小。

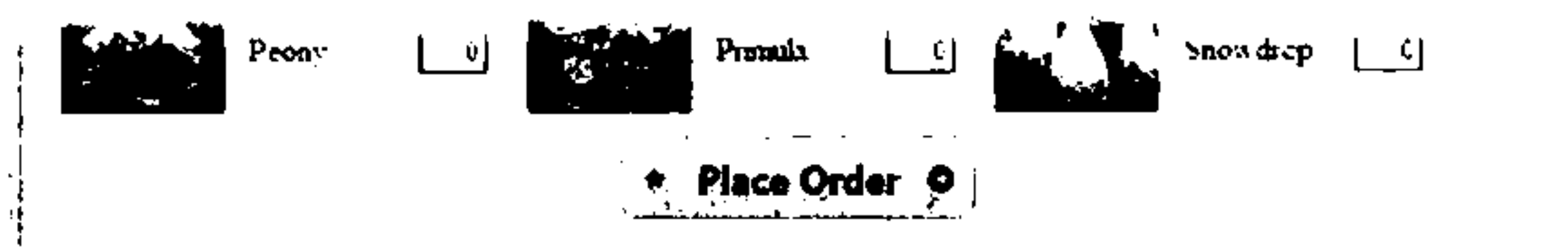


图18-2 在按钮上显示图标

我们使用jQuery UI样式文件中定义的class来指定图标。jQuery UI一共定义了173个不同样子的图标，要在这一章里都列出来未免太多了。查看所有图标名字最简单的办法是访问<http://jqueryui.com>，进入Themes页，然后滚动到页尾。如图18-3所示，你会在一个网格中看到所有的图标，把鼠标移到某个图标上就会看到这个图标使用的class名。

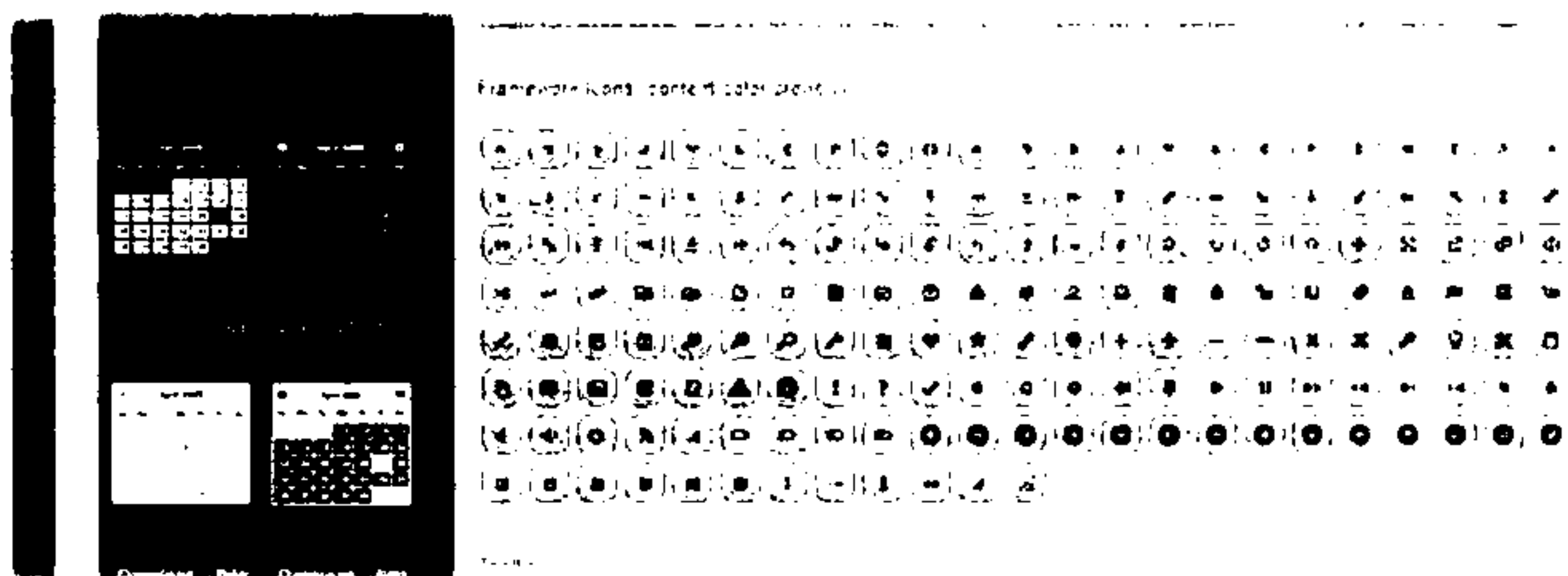


图18-3 jQuery UI图标网格

**提示** 弹出来的class名都有一个小数点前缀，在设置图标时这个小数点一定要省略。比如我们把鼠标放到风格的第一个图标上时会浮出`.ui-icon-caret-1-n`，要在按钮上使用这个图标，就要把`primary`或者`secondary`属性设置为`ui-icon-caret-1-n`。

### 18.1.3 在按钮上使用自定义图片

jQuery UI图标并不总是适用，它们常常太小，不能满足需要。好在我们有两种方法可以在一个jQuery UI按钮上显示自定义图片。<sup>①</sup>

第一种是在`button`元素（即用于生成jQuery UI按钮组件的底层元素）内放一个`img`元素。jQuery UI按钮组件与底层`button`元素的配合相当默契，只要使用的是透明背景的图片，就不用担心图片与当前主题的匹配问题。代码清单18-5是一个简单的例子。

①虽然作者说有两种方法可以在按钮上显示自定义图片，但只在书中提供了一种方法。译者猜测另一种方法可能是使用自定义样式为按钮指定背景图片。——译者注

代码清单18-5 在jQuery UI按钮上使用自定义图片

```

...
<script type="text/javascript">
    $(document).ready(function () {
        $.ajax("mydata.json", {
            success: function (data) {
                var tmplData = $("#flowerTpl")
                    .template({ flowers: data }).filter("*");
                tmplData.slice(0, 3).appendTo("#row1");
                tmplData.slice(3).appendTo("#row2");
            }
        });

        $("button")
            .text("")
            .append("<img src=rightarrows.png width=100 height=30 />")
            .button();
    });
</script>
...

```

既然我们既不要按钮文本，也不想使用jQuery UI提供的图标，那么可以使用jQuery的text方法把按钮元素的内容设置为空字符串。接着，我使用append方法插入一个img元素，最后调用button方法生成一个jQuery UI按钮。这个例子最终的效果见图18-4。

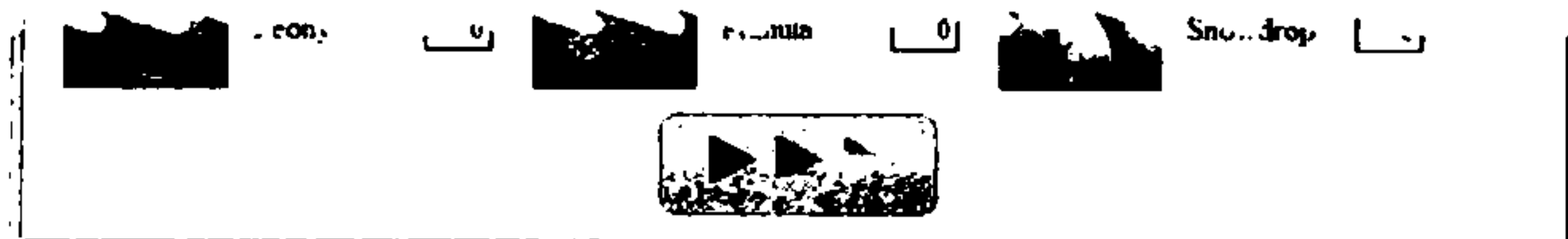


图18-4 在按钮上显示自定义图片

### 18.1.4 button方法

jQuery UI组件也定义了一些“方法”用以控制已创建的组件。在大家看来，称这些“方法”为方法多少有点怪，因为我们只是调用同一个JavaScript方法，传递不同的参数值来改变jQuery UI的行为。不过，既然jQuery UI团队已经把它们命名为方法，我也就顺水行舟，使用方法之名引用它们。表18-3列出了这些jQuery UI方法以及它们的作用。

表18-3 button方法

方 法	描 述
button("destroy")	使按钮恢复原始模样
button("disable")	禁用这个按钮
button("enable")	使按钮可用
button("option")	设置一个或多个选项，参阅18.1.1节
button("refresh")	刷新按钮状态，参阅本节第3部分

### 1. 使按钮恢复原始模样

destroy方法用来从HTML button元素上剥离jQuery UI组件，恢复其本来的面目，具体示例见代码清单18-6。

代码清单18-6 使用destroy方法

```
...
<script type="text/javascript">
    $(document).ready(function () {
        $.ajax("mydata.json", {
            success: function (data) {
                var tmplData = $("#flowerTpl")
                    .template({ flowers: data }).filter("*");
                tmplData.slice(0, 3).appendTo("#row1");
                tmplData.slice(3).appendTo("#row2");
            }
        });

        $("button").button().click(function (e) {
            $("button").button("destroy");
            e.preventDefault();
        });
    });
</script>
...
```

在代码清单18-6中，我使用jQuery的click方法为button元素注册了一个事件处理函数。注意，这一技术与第9章中介绍的技术完全一样，无需为jQuery UI做任何改变。如图18-5所示，当我们点击这个按钮，事件处理函数就会删除该jQuery UI组件。

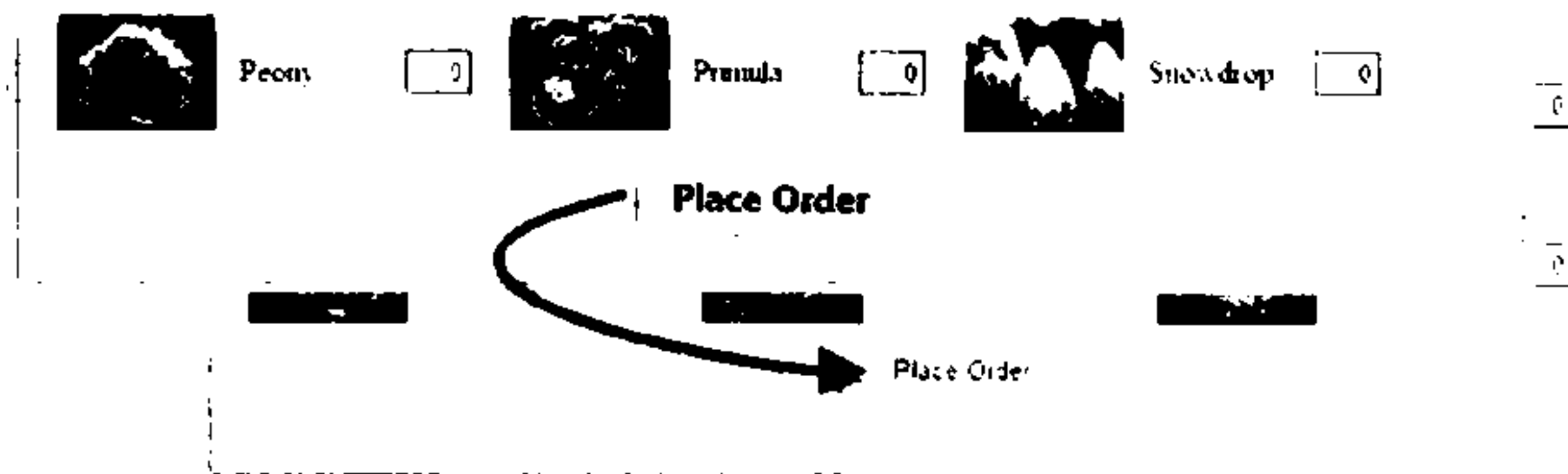


图18-5 恢复按钮组件底层元素的本来面目

### 2. 改变按钮的可用状态

如代码清单18-7所示，enable和disable方法用来改变jQuery UI按钮的可用状态。

代码清单18-7 改变按钮的可用状态

```
...
<script type="text/javascript">
    $(document).ready(function () {
        $.ajax("mydata.json", {
            success: function (data) {
```



```

        var tmplData = $("#flowerTpl")
            .template({ flowers: data }).filter("");
        tmplData.slice(0, 3).appendTo("#row1");
        tmplData.slice(3).appendTo("#row2");
    }
});

$("<span>Enabled:<span><input type=checkbox checked />").prependTo("#buttonDiv");
$(":checkbox").change(function (e) {
    $("button").button(
        $(":checked").length == 1 ? "enable" : "disable"
    );
});

$("button").button();
});
</script>
...

```

这段脚本在页面中加入了一个复选框，并使用change方法为它注册了一个事件处理函数。这样，当复选框被选中或者反选时就会执行这个函数。在这个函数中我调用了enable和disable方法来改变按钮的状态，以匹配复选框的状态。这个示例产生的效果见图18-6。

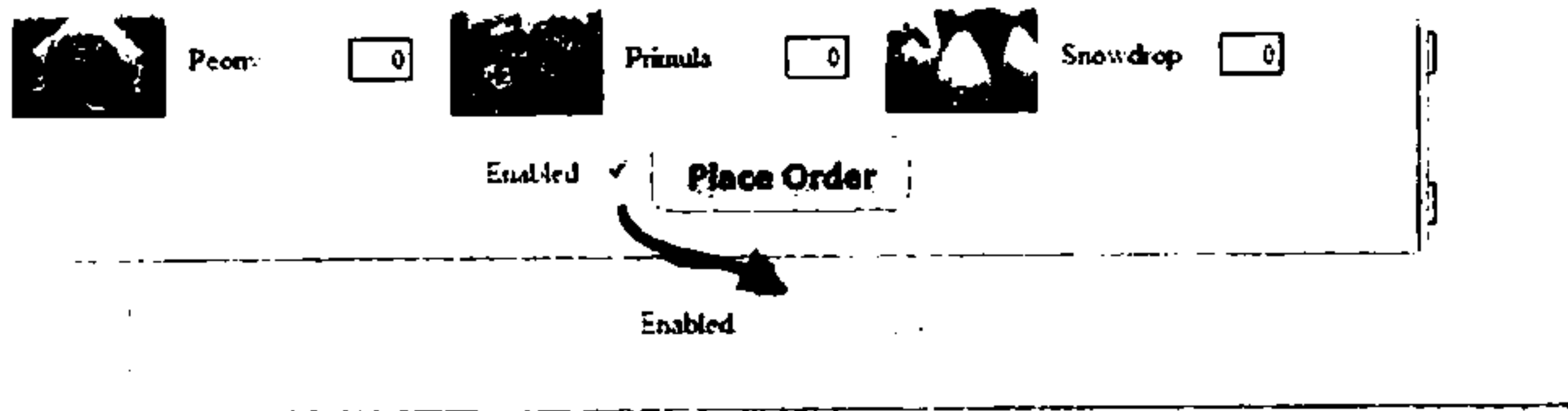


图18-6 改变jQuery UI按钮的可用状态

### 3. 更新jQuery UI按钮的状态

refresh方法刷新按钮组件的状态，以反映底层HTML元素的变化。如代码清单18-8所示，如果使用脚本修改底层元素，就会用到这个方法。

#### 代码清单18-8 更新jQuery UI按钮

```

...
<script type="text/javascript">
    $(document).ready(function () {
        $.ajax("mydata.json", {
            success: function (data) {
                var tmplData = $("#flowerTpl")
                    .template({ flowers: data }).filter("");
                tmplData.slice(0, 3).appendTo("#row1");
                tmplData.slice(3).appendTo("#row2");
            }
        });

        $("<span>Enabled:<span><input type=checkbox checked />").prependTo("#buttonDiv");
    }
});

```

```

    $(":checkbox").change(function (e) {
        var buttons = $("button");
        if ($(":checked").length == 1) {
            buttons.removeAttr("disabled");
        } else {
            buttons.attr("disabled", "disabled");
        }
        buttons.button("refresh");
    });

    $("button").button();
});
</script>
...

```

在这个例子中，我使用复选框触发对HTML button元素的disabled属性的添加或删除。jQuery UI无法自动检测到这种变化，因此我调用refresh方法来让按钮的表现与内在性质同步。

---

**提示** 你可能会奇怪为什么不直接使用jQuery UI的enable和disable方法，其实代码清单18-8中的这种场景非常常见，因为经常是到了开发阶段的尾声才会引入jQuery UI，或者是对已有项目做界面美化。在这种情况下，HTML元素由脚本生成和维护，它们根本不知道有jQuery按钮组件这回事。现在你知道提供update方法来反映底层元素的变化有多么重要了吧！

---

### 18.1.5 按钮事件

jQuery UI组件也定义了一些事件，作为对底层HTML元素事件的补充。button组件只定义了一个事件，即create事件，它在我们创建jQuery UI按钮完成时触发。如代码清单18-9所示，定义jQuery UI事件，需要调用组件方法（对按钮组件来说，就是button方法），并传递一个事件配置对象为参数。

**代码清单18-9** jQuery UI按钮的create事件

```

...
<script type="text/javascript">
    $(document).ready(function () {
        $.ajax("mydata.json", {
            success: function (data) {
                var tplData = $("#flowerTpl")
                    .template({ flowers: data }).filter("");
                tplData.slice(0, 3).appendTo("#row1");
                tplData.slice(3).appendTo("#row2");
            }
        });

        $("button").button({
            create: function (e) {
                $(e.target).click(function (e) {
                    alert("Button was pressed");
                    e.preventDefault();
                });
            }
        });
    });

```

```

    }
  });
});
</script>
...

```

在这个例子中，我利用create事件为按钮的click事件绑定了处理函数。我认为按钮组件的create事件并不是很有用。因为在这个事件里能做的事都可以用更加一致并且更直白的jQuery方式去做。

## 18.2 创建不同类型的按钮

jQuery UI按钮组件能根据调用它的元素类型做出不同的行为。在button元素、a元素，或者type属性被设置为submit、reset或button的input元素上调用button方法，这会触发button方法最基本的行为，即生成一个标准的按钮。代码清单18-10展示了如何把所有这些元素转换成jQuery UI按钮。

代码清单18-10 生成标准按钮

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function() {
      $(".jqButton").click(function(e) {
        e.preventDefault();
        $(this).button();
      });
    });
  </script>
</head>
<body>
  <form>
    <input class="jqButton" type="submit" id="inputSubmit" value="Submit">
    <input class="jqButton" type="reset" id="inputReset" value="Reset">
    <input class="jqButton" type="button" id="inputButton" value="Input Button">
    <button class="jqButton">Button Element</button>
    <a class="jqButton" href="http://apress.com">A Element</a>
  </form>
</body>
</html>

```

对于前面提到的每个类型的HTML元素，我都进行了定义。这里使用了click方法，这样每个元素被点击时就会转换为jQuery UI按钮。转换前后的按钮见图18-7。

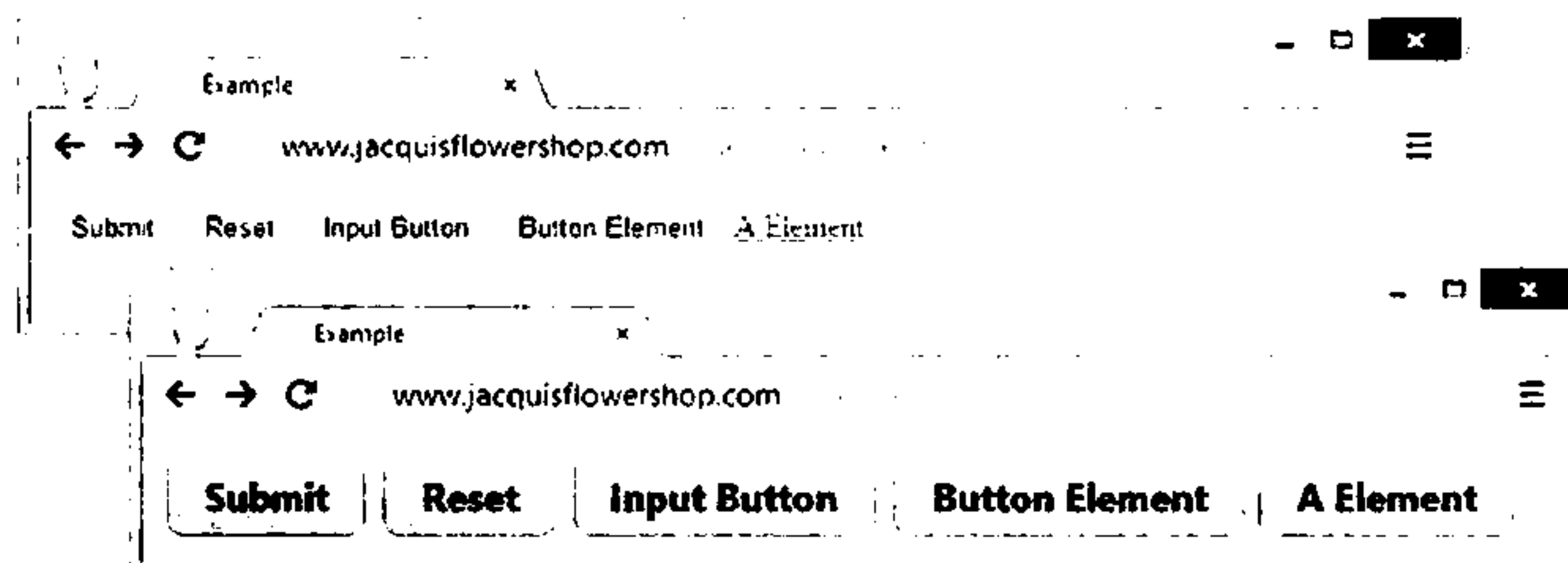


图18-7 jQuery UI标准按钮

### 18.2.1 切换按钮

如果在一个复选框上调用button方法，我们就会得到一个切换按钮。点击切换按钮组件，这个按钮就会根据底层元素的状态开或者关。具体示例见代码清单18-11。

代码清单18-11 在复选框上调用button方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <script type="text/javascript">
    $(document).ready(function () {
      $(".jqButton").button();
    });
  </script>
</head>
<body>
  <form>
    <input class="jqButton" type="checkbox" id="toggle">
    <label for="toggle">Toggle Me</label>
  </form>
</body>
</html>
```

如代码清单18-11所示，要把一个复选框变成一个jQuery UI切换按钮，我们必须要有☐元素和一个配套的label元素。jQuery UI会生成一个有着标准按钮一样外观的切换按钮，点击时会切换状态。这个示例的具体效果见图18-8。

记住，jQuery UI并未改变底层的HTML元素，因此当复选框位于表单中时，浏览器仍会像对待普通复选框一样对待这个外观已经变为切换按钮的复选框。切换按钮的外观变化对应着复选框checked属性的变化，就如同根本没有使用jQuery UI库一样。

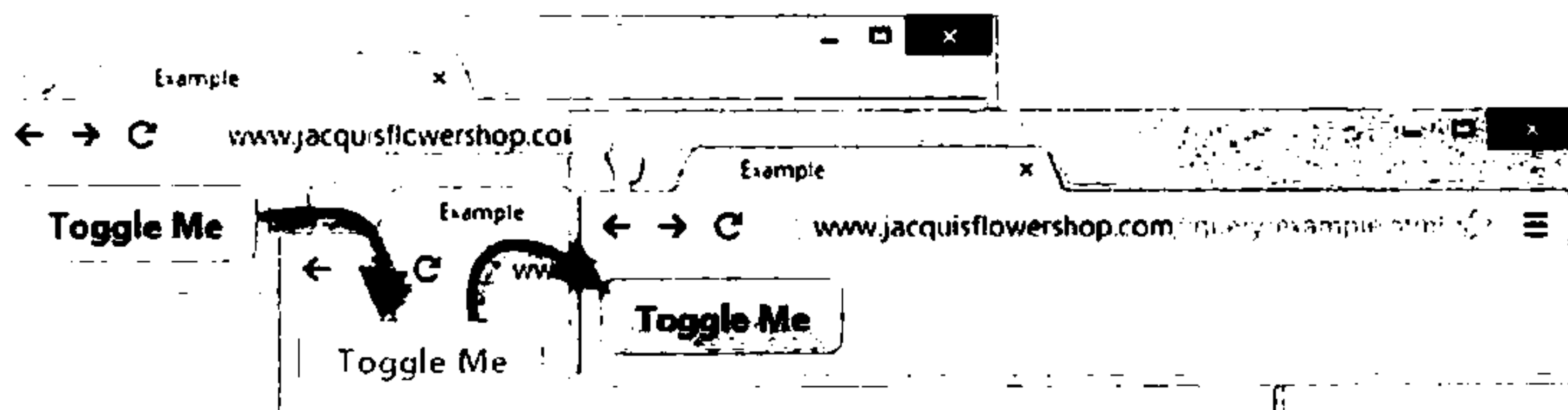


图18-8 由复选框生成切换按钮

## 18.2.2 按钮组

如代码清单18-12所示，buttonset方法利用一些单选钮生成按钮组。

### 代码清单18-12 生成按钮组

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <script type="text/javascript">
    $(document).ready(function () {
      $("#radioDiv").buttonset();
    });
  </script>
</head>
<body>
  <form>
    <div id="radioDiv">
      <input type="radio" name="flower" id="rose" checked />
      <label for="rose">Rose</label>
      <input type="radio" name="flower" id="lily"/><label for="lily">Lily</label>
      <input type="radio" name="flower" id="iris"/><label for="iris">Iris</label>
    </div>
  </form>
</body>
</html>
```

注意，我选中的是包含单选按钮的div元素，在这个元素上调用buttonset方法。我们不用在每个input元素上挨个调用button方法。调用buttonset方法后单选按钮的外观变化见图18-9。

这一组按钮是互斥的，即同一时刻至多只能有一个按钮处于选中状态。这样一来，我们可以提供给用户一些固定的选项，同时保证其和其他jQuery UI按钮拥有一致的外观。注意，jQuery UI对这组按钮之间的边框施以直角（而不是边缘上的圆角）样式，强调了这些按钮之间的关系。这一点在图18-10上表现得更加清楚。

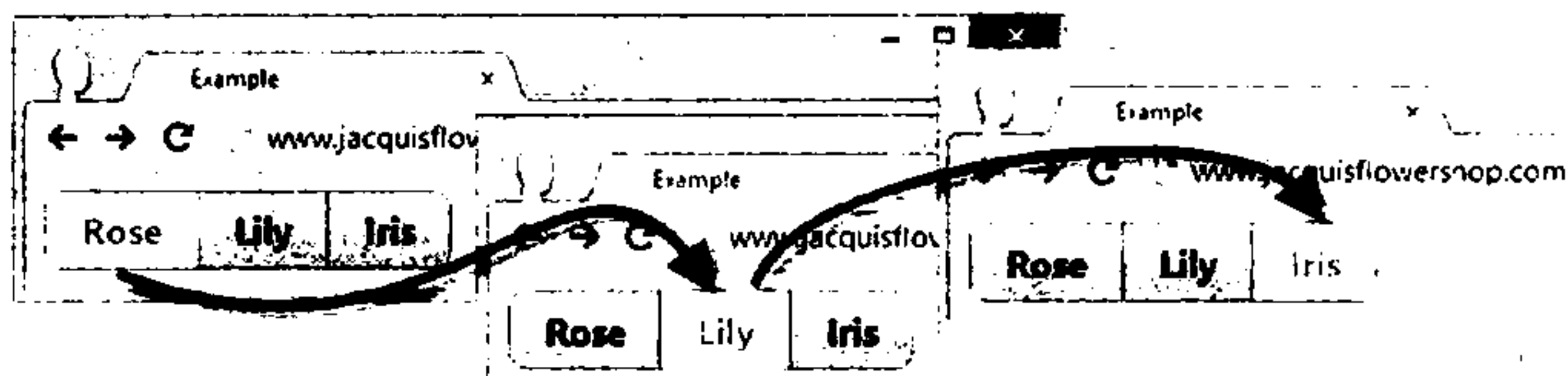


图18-9 生成按钮组



图18-10 jQuery UI为按钮组定义的样式

### 使用普通按钮生成按钮组

就像可以在任意元素上调用button方法一样，我们也可以在任意元素上调用buttonset方法。这会产生类似单选按钮组的外观，却没有互斥的行为。代码清单18-13演示了buttonset的这种用法。

#### 代码清单18-13 使用普通按钮生成按钮组

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <script type="text/javascript">
    $(document).ready(function() {
      $("#radioDiv").buttonset();
    });
  </script>
</head>
<body>
  <form>
    <div id="radioDiv">
      <input type="submit" value="Submit"/>
      <input type="reset" value="Reset"/>
      <input type="button" value="Press Me"/>
      <a href="http://apress.com">Visit Apress</a>
    </div>
  </form>
</body>
</html>
```

如图18-11所示，div容器中任何合适的元素都会转换为一个按钮，而且按钮之间的边框都会像单

选按钮组那样进行处理。

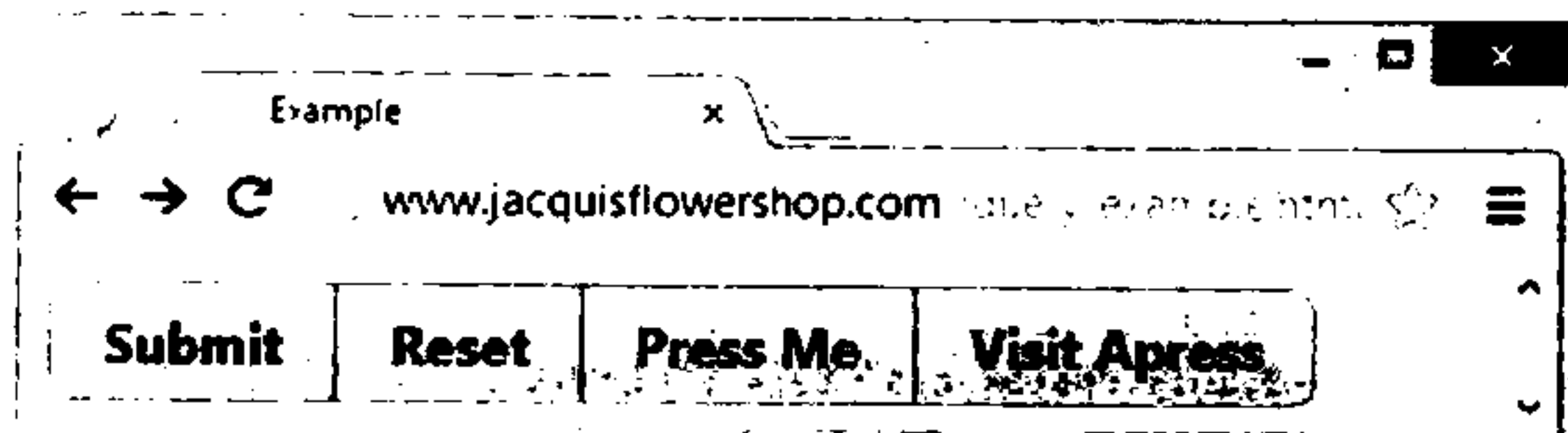


图18-11 利用普通按钮生成按钮组

**提示** 尽量不要这么做，它会让用户犯迷糊，特别是当我们在同一页面或者同一Web应用程序的其他地方使用了互斥单选按钮的情况下。

## 18.3 jQuery UI 进度条

通过按钮组件我们已经了解了jQuery UI组件的基本结构，现在是时候了解更多jQuery UI组件了。首先来看看进度条组件。

进度条组件用来显示任务进度，它适用于明确的任务。对于这类任务，我们能够以百分比的形式精确地告诉用户已经完成了多少。与此相对的是模糊任务，对模糊任务我们无法得知具体的进度。

**提示** 从jQuery UI 1.10起，进度条组件开始支持模糊任务，欲了解详情请参阅18.3.2节。

### 显示有意义的进度信息

在Web应用程序中，一个任务应该使用什么组件并没有一定之规，然而用户期待类似Windows和Mac OS这样的由操作系统提供的进度条标准控件。为了让用户能看懂进度条，我们应遵守两条规则。

**第一条**，让进度条一直前进。当任务遇到比计划更多的步骤时，不要有让进度后退的想法。进度条表示的是任务已经完成的百分比，并不表示任务剩余时间。如果任务有多条执行路径，那就按照最悲观的情况显示进度。让进度突然前进一大块总比让用户困惑要好。

**第二条**，不要让进度条走到头又从头开始。如果你有足够的信息让用户知道相对准确的任务完成情况，那就使用模糊的进度指示器。要知道当进度接近100%时，用户期待着任务结束。如果进度条到了100%又回到开头重新再来，这不但让用户困惑，也让进度条失去了意义。

### 18.3.1 创建进度条

如代码清单18-14所示，选中一个div元素，然后调用progressbar方法，我们就得到了一个进度条。

## 代码清单18-14 创建进度条

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <script type="text/javascript">
    $(document).ready(function () {
      $("#progressDiv").progressbar({
        value: 21
      });
    });
  </script>
</head>
<body>
  <div id="progressDiv"></div>
</body>
</html>

```

在这个例子中，页面有一个

#progressDiv元素。为了创建进度条，我们必须使用空的div元素。如果div元素非空，那么它的内容会影响进度条组件的布局。我使用jQuery选中了这个progressDiv元素，接着调用progressbar方法，并传递给它一个映射对象作为初始配置。进度条组件支持三个选项，其用途见表18-4。

表18-4 进度条组件选项

选 项	描 述
disabled	如果为true，进度条组件被禁用。默认值为false
value	设置用户看到的已完成进度（百分比），默认值为0。把该属性的值设置为false则显示模糊进度（参考18.3.2节）
max	设置进度条的最大值，默认值为100

在这个例子里，我把初始进度的值设置为21（由于我并未改变max属性，因此值21即进度21%）。浏览器中的实际效果见图18-12。



图18-12 创建一个进度条

## 18.3.2 创建模糊进度

如代码清单18-15所示，从jQuery UI 1.10开始，只要把value属性设置为false，进度条组件即支持



模糊进度。

#### 代码清单18-15 创建模糊进度条

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <script type="text/javascript">
    $(document).ready(function () {
      $("#progressDiv").progressbar({
        value: false
      });
    });
  </script>
</head>
<body>
  <div id="progressDiv"></div>
</body>
</html>
```

jQuery UI在整个进度条上显示一个简单动画，表示任务正在进行。实际运行这个例子才能看到动画效果，图18-13展示的是动画的一帧。

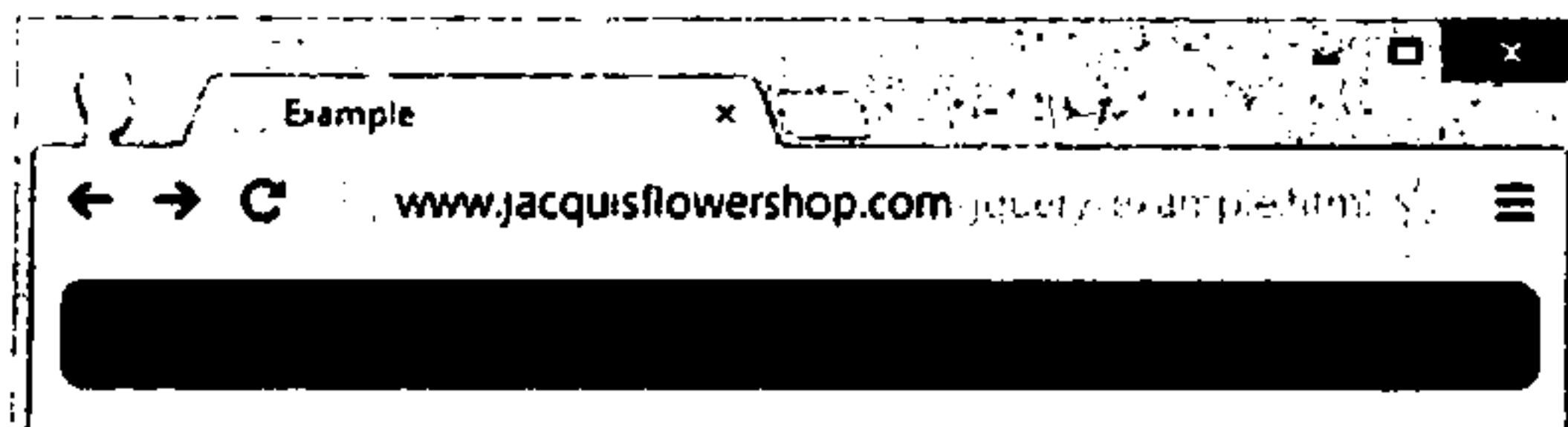


图18-13 创建模糊进度

### 18.3.3 进度条方法

与button组件，进度条组件定义了一些相同风格的方法。也就是说，我们要通过以方法名为progressbar方法第一个参数的方式使用jQuery UI组件方法。表18-5列出了这些方法及其作用。

表18-5 进度条方法

方 法	描 述
progressbar("destroy")	恢复div元素的初始状态
progressbar("disable")	禁用进度条组件
progressbar("enable")	启用进度条组件
progressbar("option")	设置一个或多个选项。欲了解配置jQuery UI组件的详细信息，请参阅18.1.1节
progressbar("value", value)	获取或设置进度条的当前进度，或者切换精确进度模式与模糊进度模式

绝大多数方法的工作方式都与按钮组件一模一样，因此我就不一一演示它们的用法了。唯一的例外是value方法，它用于获取或者设置进度条当前进度，还可以用来切换精确进度与模糊进度。代码清单18-16演示了value方法的用法。

代码清单18-16 进度条组件的value方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <script type="text/javascript">
    $(document).ready(function () {

      $("#progressDiv").progressbar({
        value: 21
      });

      $("#button").click(function (e) {
        var divElem = $("#progressDiv");
        if (this.id == "mode") {
          divElem.progressbar("value", false);
        } else {
          var currentProgress = divElem.progressbar("value");
          if (!currentProgress) {
            divElem.progressbar("value", 21);
          } else {
            divElem.progressbar("value",
              this.id == "decr" ? currentProgress - 10 :
              currentProgress + 10)
          }
        }
      });
    });
  </script>
</head>
<body>
  <div id="progressDiv"></div>
  <button id="decr">Decrease</button>
  <button id="incr">Increase</button>
  <button id="mode">Indeterminate</button>
</body>
</html>
```

我在这个例子里添加了三个按钮，用来增加进度、减小进度以及切换进度模式。如图18-14所示，每点击一次Increase按钮或者Decrease按钮，进度就变化10%。

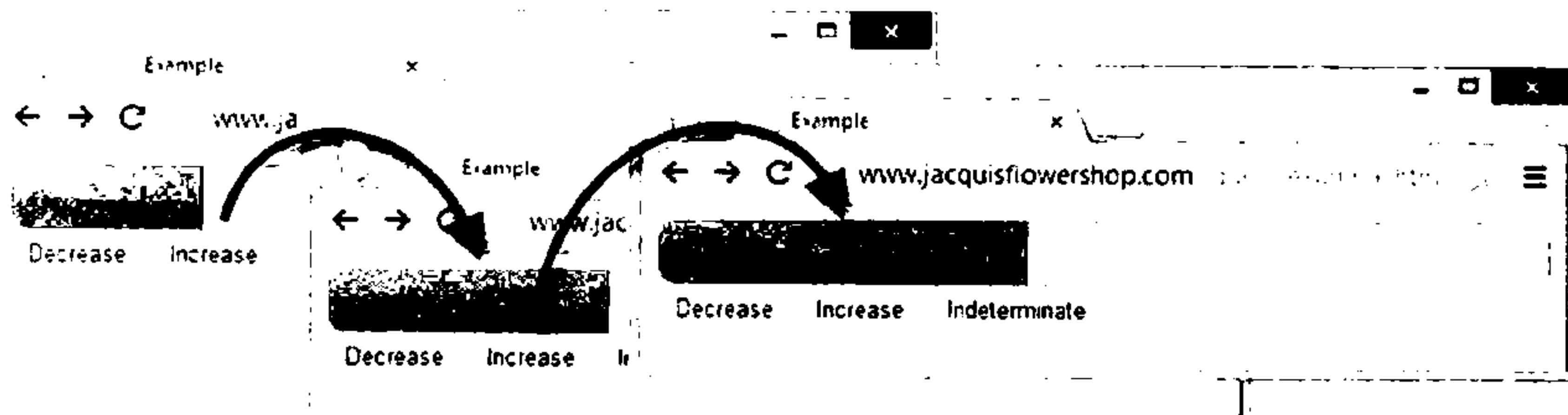


图18-14 使用value方法改变显示进度

Indeterminate按钮调用value方法，把进度条切换到模糊进度模式。如图18-15所示，点击此按钮之外的任意一个按钮，都会把进度重新设置为数字值，从而切换回精确进度模式。

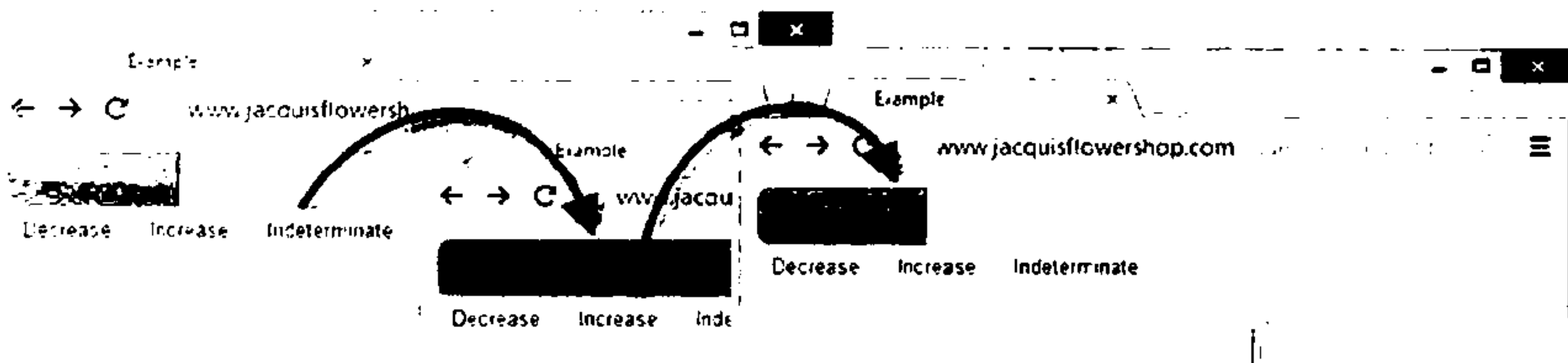


图18-15 进度条模式切换

### 18.3.4 进度条事件

如表18-16所示，jQuery UI进度条组件定义了3个事件。

表18-6 进度条事件

事 件	描 述
create	在进度条组件创建完成时触发
change	在进度条进度发生变化时触发
complete	当进度条的值达到100（显示为进度完成100%）时触发

代码清单18-17演示了进度条事件的使用。

#### 代码清单18-17 进度条事件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
```

```

<script type="text/javascript">
    $(document).ready(function () {

        $("button").button();

        $("#progressDiv").progressbar({
            value: 21,
            create: function (e) {
                $("#progVal").text($("#progressDiv").progressbar("value"));
            },
            complete: function (e) {
                $("#incr").button("disable")
            },
            change: function (e) {
                var currentValue = $("#progressDiv").progressbar("value");
                if (!currentValue) {
                    $("#progWrapper").hide();
                } else {
                    if ($(this).progressbar("value") < 100) {
                        $("#incr").button("enable")
                    }
                    $("#progVal").text(currentValue);
                    $("#progWrapper").show();
                }
            }
        });

        $("button").click(function (e) {
            var divElem = $("#progressDiv");
            if (this.id == "mode") {
                divElem.progressbar("value", false);
            } else {
                var currentProgress = divElem.progressbar("value");
                if (!currentProgress) {
                    divElem.progressbar("value", 21);
                } else {
                    divElem.progressbar("value",
                        this.id == "decr" ? currentProgress - 10 :
                        currentProgress + 10)
                }
            }
        });
    });
</script>
</head>
<body>
    <div id="progressDiv"></div>
    <button id="decr">Decrease</button>
    <button id="incr">Increase</button>
    <button id="mode">Indeterminate</button>
    <span id="progWrapper">Progress: <span id="progVal"></span>X</span>
</body>
</html>

```

在这个例子中，我添加了一个span元素，用它显示当前进度。还在create事件中使用组件的value属性为进度条设置了初始进度。

**提示** 注意，我在组件选项和组件事件上使用了同样的映射对象。这不是必需的，不过确实可以利用它通过一次方法调用完成组件创建和配置。

当进度条达到100%时，我利用complete事件禁用了Increase按钮，而change事件则保证了进度小于100%时，Increase按钮总是可用的，并显示当前进度以及切换精确模式和模糊模式。这个例子在浏览器中的实际显示效果见图18-16。

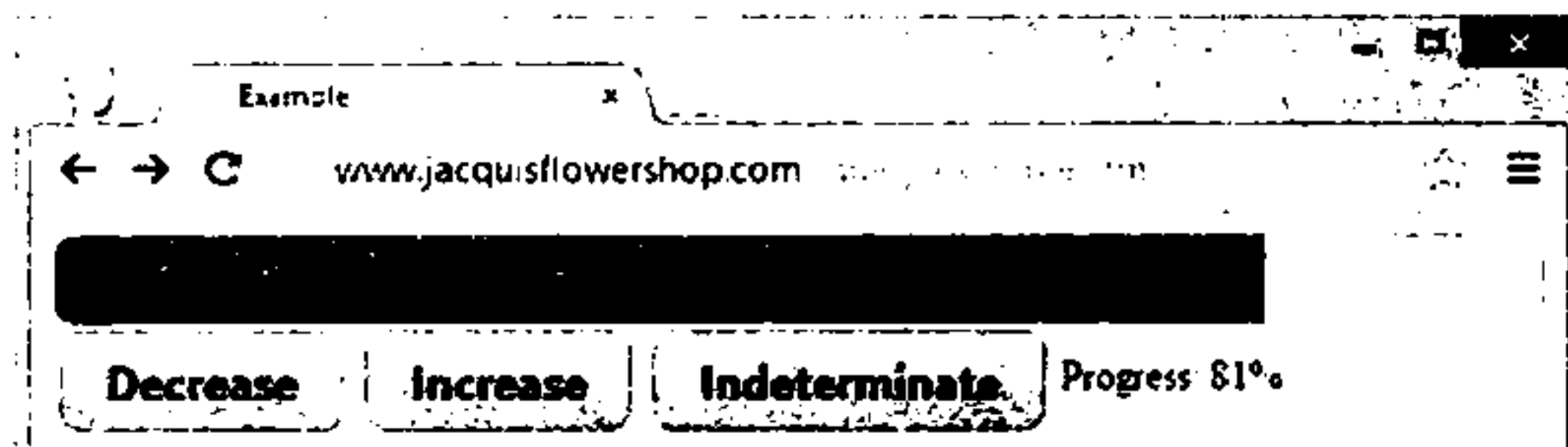


图18-16 响应进度条组件事件

**提示** 使用这些事件时要注意两件事。首先，每次把进度的value设置为100或者更大值都会触发complete事件。这意味着complete事件处理函数有可能执行多次。其次，当进度的value变为100或更大值时，change事件和complete事件都会发生，因此当任务完成时我们必须能够同时处理这两个事件。

## 18.4 jQuery UI 滑动条组件

顾名思义，滑动条组件在网页中生成一个滑动条，让用户不必在input文本框里输入值，而是能够利用滑动条选择一个值。如代码清单18-18所示，在底层元素上调用slider方法即可生成滑动条组件。

代码清单18-18 生成一个滑动条

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <style>
    #slider { margin: 10px; }
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
```

```

        $("#slider").slider();
    });
</script>
</head>
<body>
    <div id="slider"></div>
</body>
</html>

```

这个滑动条的风格与其他jQuery UI组件完全一致，可以用鼠标在滑动范围内移动滑块，也支持光标键。这个简单的滑动条在浏览器中的显示效果见图18-17。注意，我为本例中对应滑动条的底层元素定义了一条样式规则，如果缺少这条规则，滑动条就会紧贴父元素（缺少留白）。



图18-17 一个简单的jQuery UI滑动条

### 18.4.1 配置滑动条

和其他jQuery UI组件一样，滑动条组件也定义了一些选项，我们可以利用这些选项配置滑动条的外观和行为。表18-7列出了这些选项。在接下来的几节中，我将演示如何利用这些选项配置滑动条组件。

表18-7 滑动条组件的选项

选 项	描 述
animate	若设置为true，当用户点击滑块之外的区域时滑动条会动一下。默认值为false
disabled	若设置为true，则滑动条被禁用。默认值为false
max	滑动条的最大值，默认值为100
min	滑动条的最小值，默认值为0
orientation	滑动条的滑动方向（参见代码清单18-19）
range	和values选项一起生成具有多个滑块的滑动条
step	定义滑块从min到max滑动的最小间隔
value	滑动条表示的值
values	与range选项一起创建具有多个滑块的滑动条

**提示** 用户可以选择的值不包括最小值和最大值。也就是说，如果我们把最小值设置为0，最大值设置为100，那么用户的选择范围是1到99。

### 1. 改变滑动方向

滑动条组件的默认滑动方向是水平方向,不过我们可以使用orientation选项生成一个垂直滑动的滑动条。代码清单18-19就是一个这样的例子。

代码清单18-19 使用orientation选项

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <style>
    #hslider, #vslider { margin: 10px}
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#hslider").slider({
        value: 35
      });

      $("#vslider").slider({
        orientation: "vertical",
        value: 35
      });
    });
  </script>
</head>
<body>
  <div id="hslider"></div>
  <div id="vslider"></div>
</body>
</html>
```

在这个例子里我创建了两个滑动条,并将其中一个设置为垂直滑动。我还改变了一个style元素,以便控制两个滑动条之间的距离,避免它们靠得过近。通过控制底层元素的样式,我们可以控制滑动条(以及其他任意jQuery UI组件)的大小和位置。这也是最好使用div元素的原因,它们是那么容易样式化。图18-18展示了这两个滑动条的样子。注意,例子中还使用value选项设置了滑块的初始位置。

虽然在上一个例子中,我有意把两个滑动条的初始化和配置代码分开书写,但确实可以重写上一个例子以更好地利用底层的jQuery功能,如代码清单18-20所示。



图18-18 垂直滑动条与水平滑动条

## 代码清单18-20 更好地使用jQuery

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <style>
    #hslider, #vslider { margin: 10px}
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#hslider, #vslider").slider({
        value: 35,
        orientation: "vertical"
      }).filter("#hslider").slider("option", "orientation", "horizontal");
    });
  </script>
</head>
<body>
  <div id="hslider"></div>
  <div id="vslider"></div>
</body>
</html>

```

这虽然只是一个小问题，我希望借此让你牢牢记住：jQuery UI构建在jQuery之上，并与jQuery紧密结合。你尽可以使用本书前面讲到的jQuery提供的各种选项与维护功能。

**提示** 注意，我在例子中先把初始滑动方向设置为vertical（垂直方向），然后再修改为horizontal（水平方向）。滑动条中有个bug：如果先设置为水平方向，再修改为vertical，jQuery UI会出错（滑块位置错误）。



## 2. 为滑动条增加动画效果

利用animate选项，我们可以在用户点击滑动条目标位置（相对于滑动条本身）时让滑块的移动更平滑。把animate选项设置为true即可打开默认的动画效果。我们还可以把animate的值设置为fast或者slow，或者指定一个数值（动画持续时间，单位ms）。代码清单18-21演示了animate选项的用法。

代码清单18-21 使用animate选项

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <style>
    #slider {margin: 10px}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#slider").slider({
        animate: "fast"
      });
    });
  </script>
</head>
<body>
  <div id="slider"></div>
</body>
</html>
```

在这个例子中，我把animate选项的值设置为fast。截图很难展示出动画效果，不过图18-19还是展示了animate选项的行为。

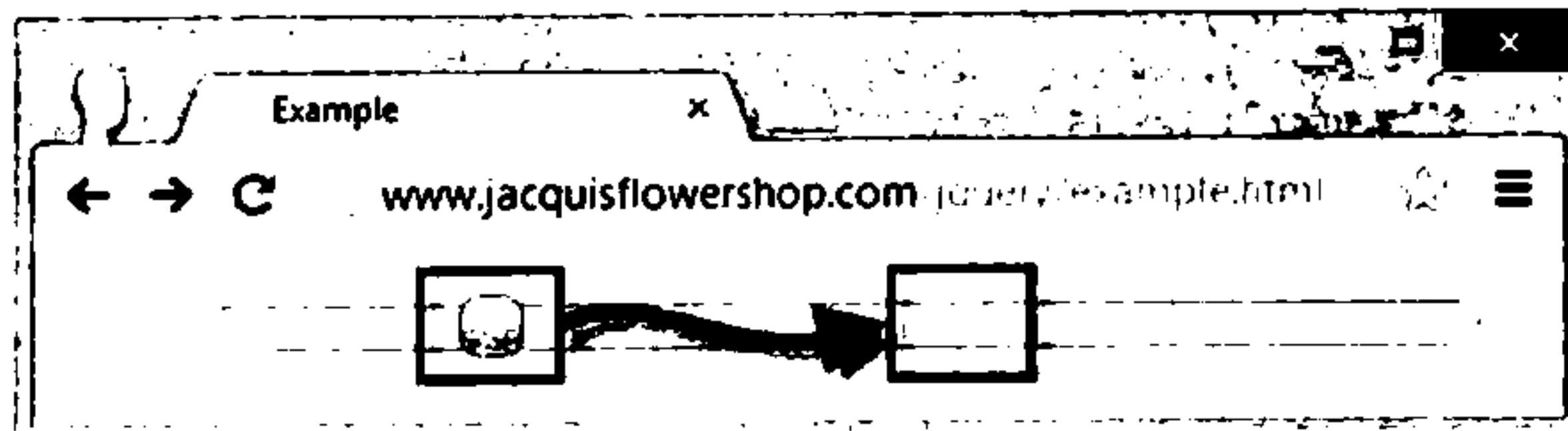


图18-19 动画演示滑块的移动

这张截图展示了我点击鼠标按钮之前滑动条的样子。如果禁用动画效果，代码就会立即为滑动条设置新值，滑动会一下子跳到我点击鼠标的位置。然而在我设置了animate选项之后，滑动就以一种平滑且优雅的方式移到新的位置。不过，和所有动画效果一样，物极必反。这也是我选择fast选项值的原因。

**提示** 这是一个需要在浏览器中体验的例子。如果你不想手工输入这些代码和HTML，可以从Apress.com的源代码/下载区免费下载。（在Apress.com网站上可以免费下载包括书中所有示例在内的源代码。）

### 3. 创建范围选择器

范围选择器就是带有两个滑块的滑动条，用户可以利用它设定一个范围。比方说，我们希望用户表达愿意为某个产品支付的价格范围，滑动条可以帮助限定用户的选择范围。代码清单18-22展示了创建范围选择器的示例。

代码清单18-22 创建范围选择器

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <style>
    #slider { margin: 20px}
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#slider").slider({
        values: [35, 65],
        range: true,
        create: displaySliderValues,
        slide: displaySliderValues
      });

      function displaySliderValues() {
        $("#lower").text($("#slider").slider("values", 0));
        $("#upper").text($("#slider").slider("values", 1));
      }
    });
  </script>
</head>
<body>
  <div id="slider"></div>
  <div>Lower Value: <span id="lower">
    </span> Upper Value: <span id="upper"></span></div>
</body>
</html>
```

为了得到范围选择器，我们需要把range选项设置为true，并把values选项设置为一个包含最低及最高边界值的数组。（创建普通滑动条使用value选项，而创建范围选择器则要使用values选项。）在这个例子里，我把边界设置为35和65。这个例子的效果见图18-20。

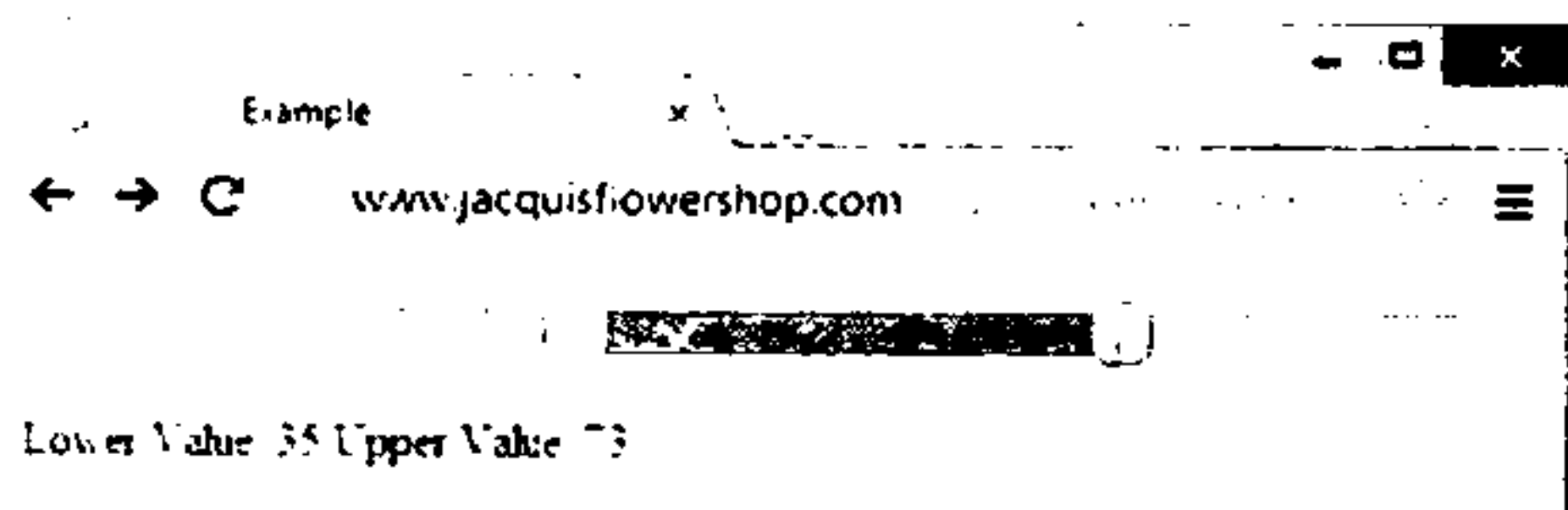


图18-20 创建一个范围选择器

我在这个例子里为create和slide事件绑定了处理函数。18.4.3节中会深入讲解滑动条有关的事件，现在我们先来演示怎样在范围选择器中获得某个滑块的位置。我们使用values方法，并指定需要获取位置的滑块的顺序号，就像下面这样：

```
...
$("#slider").slider("values", 0);
...
```

滑块的顺序号从0开始计算，换言之，第一个滑块的顺序号是0。这样，上面这句代码就得到第一个滑块的值（表示范围的下限）。create和slide事件处理函数会在事件发生时设置两个span元素的内容。

## 18.4.2 滑动条组件方法

和其他jQuery UI组件一样，滑动条组件定义了同样一组基本方法，而且还多定义了两个方法。利用这两个方法，我们可以设置其中一个滑块的值，也可以一次设置整个范围的值。表18-8列出了这些方法及其作用。

表18-8 滑动条组件方法

方 法	描 述
slider("destroy")	让组件底层元素恢复原始状态
slider("disable")	禁用滑动条组件
slider("enable")	启用滑动条组件
slider("option")	改变滑动条组件的选项。欲了解配置jQuery UI组件的详细信息，请参阅18.1.1节
slider("value", value)	得到或者设置常规滑动条的值（表示滑块位置的值）
slider("values", [values])	得到或者设置范围选择器的值

代码清单18-23展示了如何使用脚本调用value和values方法来控制滑动条组件。

### 代码清单18-23 使用脚本控制滑动条组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <style>
```

```

    #slider, #rangeslider, *.inputDiv { margin: 10px}
    label {width: 80px; display: inline-block; margin: 4px}
</style>
<script type="text/javascript">
    $(document).ready(function () {

        $("#slider").slider({
            value: 50,
            create: function () {
                $("#slideVal").val($("#slider").slider("value"));
            }
        });

        $("#rangeslider").slider({
            values: [35, 65],
            range: true,
            create: function () {
                $("#rangeMin").val($("#rangeslider").slider("values", 0));
                $("#rangeMax").val($("#rangeslider").slider("values", 1));
            }
        });

        $("input").change(function (e) {
            switch (this.id) {
                case "rangeMin":
                case "rangeMax":
                    var index = (this.id == "rangeMax") ? 1 : 0;
                    $("#rangeslider").slider("values", index, $(this).val())
                    break;
                case "slideVal":
                    $("#slider").slider("value", $(this).val())
                    break;
            }
        });
    });
</script>
</head>
<body>
    <div id="rangeslider"></div>
    <div class="inputDiv">
        <label for="rangeMin">Range Min: </label><input id="rangeMin" />
        <label for="rangeMax">Range Max: </label><input id="rangeMax" />
    </div>
    <div id="slider"></div>
    <div class="inputDiv">
        <label for="slideVal">Slide Val: </label><input id="slideVal" />
    </div>
</body>
</html>

```

这个页面中有2个滑动条和3个文本输入框（input元素）。利用这些input元素我们无需移动滑块就可改变滑动条的值。图18-21展示了这个页面在浏览器中的显示效果。

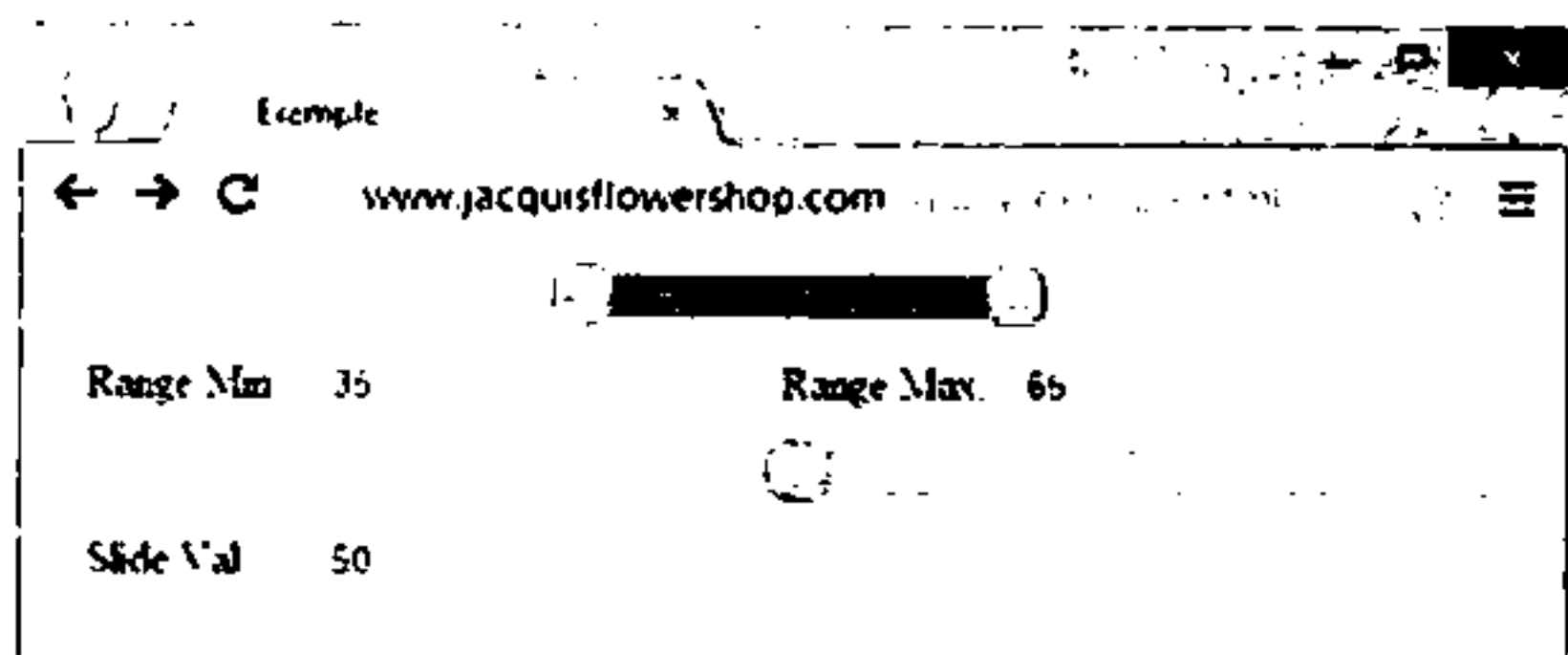


图18-21 使用脚本控制滑动条上滑块的位置

我先使用jQuery选中input元素，然后调用change方法，为change事件绑定处理函数。这样当input元素的值发生变化时，处理函数就会自动执行。

在处理函数中，我根据触发事件元素的id属性判断应该调整哪个滑动条，接着调用value或values方法设定滑块的位置。在这个例子中，input元素与滑动条之间的关系是单向的，移动滑块并不会更新文本框中的值。在下一节，我将介绍如何在二者之间建立双向关系。

### 18.4.3 滑动条组件事件

表18-9列出了滑动条组件支持的事件。这些事件中最有用的是change和stop。利用这两个事件，我们能够辨别出是用户滑动滑块产生的变化，还是我们使用脚本设置新值造成的变化。

表18-9 滑动条组件事件

事 件	描 述
create	当滑动条创建完成时触发
start	用户开始滑动滑块时触发
slide	滑动滑块过程中，鼠标的每一次移动都会触发
change	当用户停止滑动或者我们使用脚本改变滑动条组件的值时触发
stop	用户停止滑动滑块时触发

与上一节的例子类似，代码清单18-24展示了如何利用滑动条事件在文本框和滑动条之间建立双向关系，为简便起见，我从例子中删除了一个滑动条。这样我们就能够同时支持用户和滑动条交互，以及脚本和滑动条之间的交互了。

代码清单18-24 利用滑动条事件在文本框与滑动条之间创建双向关系

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js"></script>
  <script src="jquery-ui-1.10.3.custom.js"></script>
  <link href="jquery-ui-1.10.3.custom.css" rel="stylesheet" />
  <style>
    #rangeslider, *.inputDiv { margin: 10px}
    label {width: 80px; display: inline-block; margin: 4px}
  </style>
  <script type="text/javascript">
```

```

$(document).ready(function () {
    $("#rangeslider").slider({
        values: [35, 65],
        range: true,
        create: setInputsFromSlider,
        slide: setInputsFromSlider,
        stop: setInputsFromSlider
    });

    function setInputsFromSlider() {
        $("#rangeMin").val($("#rangeslider").slider("values", 0));
        $("#rangeMax").val($("#rangeslider").slider("values", 1));
    }

    $("input").change(function(e) {
        var index = (this.id == "rangeMax") ? 1 : 0;
        $("#rangeslider").slider("values", index, $(this).val());
    });
});
</script>
</head>
<body>
    <div id="rangeslider"></div>
    <div class="inputDiv">
        <label for="rangeMin">Range Min: </label><input id="rangeMin" />
        <label for="rangeMax">Range Max: </label><input id="rangeMax" />
    </div>
</body>
</html>

```

在这个例子中，我们使用了create、slide和stop事件。现在，当我们在文本框中输入新值的时候，滑块位置会变化，而且当用鼠标滑动滑块时，文本框中的数字也会同步变化。图18-22展示了这个例子在浏览器中的效果，不过你最好在真正的浏览器中运行一下这个例子，亲自执行交互以得到充分的体验。

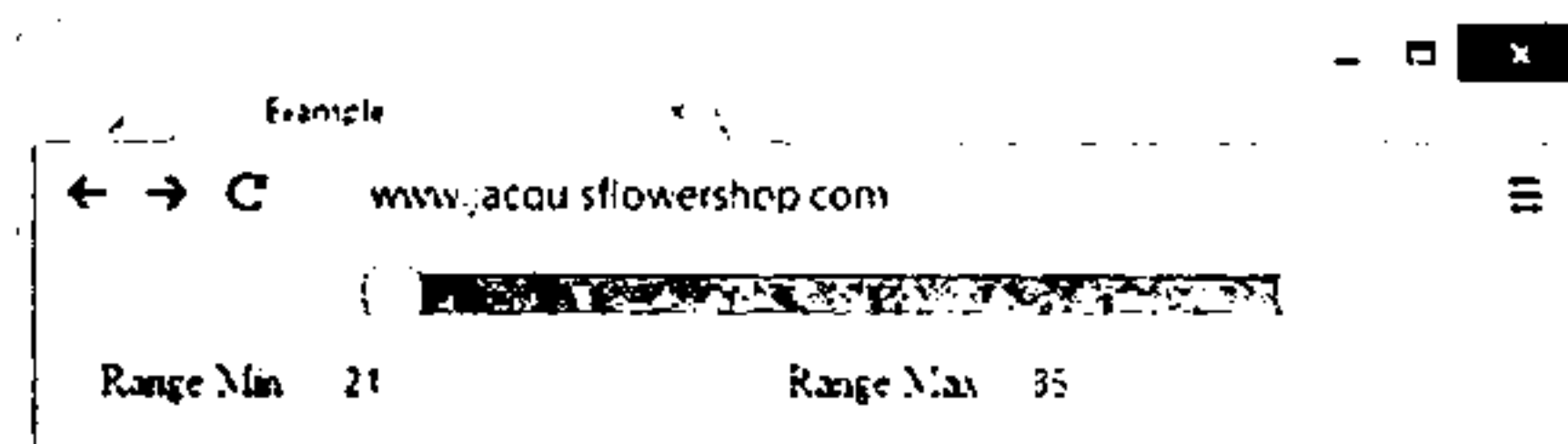


图18-22 响应滑动条组件事件

## 18.5 小结

本章介绍了前3个jQuery UI组件：按钮组件、进度条组件和滑动条组件。这些组件都有着同样的基础架构。它们都是使用一个方法创建和配置组件，也是使用同一个方法配置组件事件。有些组件方法和事件是通用的，每个组件都有。也有一些是组件独一无二的，对应着这个组件的某个特殊功能。现在，我们已经了解了jQuery UI的基础知识，接下来的几章来看一些更灵活、更复杂的组件。在下一章中，我将介绍自动完成与折叠菜单组件。

# 自动完成和折叠菜单组件

本章的主题是自动完成（autocomplete）和折叠菜单。虽然这两个组件比第18章中讲到的那3个组件要复杂许多，但它们的配置方式、内建方法和事件与其他jQuery UI组件一样，遵循同样的模式。这些用户界面组件支持高度定制，既灵活又巧妙，得到了广泛的应用。善用jQuery UI组件能够有效地增强页面或应用程序的外观及可用性。表19-1列出了本章概要。

表19-1 本章概要

问 题	解决方法	代码清单
如何让一个input元素支持自动完成	调用autocomplete方法	1、2
如何从远程服务器得到自动完成推荐数据	把source选项设置为数据源的URL	3~5
如何动态生成推荐数据	把source选项设置为处理函数	6
如何使用异步任务返回自动完成结果	调用response方法	7
如何控制自动完成组件弹出层的显示位置	使用position属性	8
如何用脚本控制自动完成功能	使用search和close方法	9
如何接收选中自动完成项的通知信息	使用focus、select及change事件	10
如何修改自动完成组件的显示结果	处理response事件	11
如何覆盖默认的自动完成行为	覆盖select事件的默认行为	12
如何生成jQuery UI折叠菜单	调用accordion方法	13
如何设置折叠菜单的高度及内容面板高度	使用heightStyle选项	14~16
如何改变激活内容块所使用的事件（用户行为）	使用event选项	17
如何设置折叠菜单组件中默认展开的内容块	使用active和collapsible选项	18、19
如何改变折叠菜单使用的图标	使用icons选项	20
如何响应折叠菜单组件中的内容块变化	响应activate和beforeactivate事件	21

## 新版jQuery UI与本章有关的变化

在jQuery UI 1.10 中，折叠菜单组件的许多API都发生了变化：配置选项、方法和事件都变了。本章的相关章节中列出了这些变化。

至于我们将要介绍的自动完成组件，它增加了一项有用的新特性：控制自动完成弹出层的显示位置（参阅19.1.2节中“设置弹出层的位置”部分）。

## 19.1 jQuery UI 自动完成组件

当用户在文本框输入数据时，我们可使用自动完成组件向其提供建议数据。利用得当的话，这个组件能有效地替用户节省时间，加快数据输入速度并且减少错误。在下面的内容中，我将演示如何创建、配置以及使用自动完成组件。

### 19.1.1 让输入框支持自动完成

在一个文本输入框（input元素）上调用autocomplete方法就会得到一个自动完成组件。代码清单19-1展示了如何得到一个基础的自动完成组件。

代码清单19-1 让文本输入框支持自动完成

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function() {

      var flowers = ["Aster", "Daffodil", "Rose", "Peony", "Primula", "Snowdrop",
                    "Poppy", "Primrose", "Petuna", "Pansy"];
      $("#acInput").autocomplete({
        source: flowers
      });
    });
  </script>
</head>
<body>
  <form>
    <div class="ui-widget">
      <label for="acInput">Flower Name: </label><input id="acInput"/>
    </div>
  </form>
</body>
</html>
```

与其他jQuery UI组件类似，我们先选中底层HTML元素，然后调用autocomplete方法。autocomplete方法的参数对象必须定义有source属性，它负责指定自动完成列表的数据来源。

在本章后面，我将演示如何为自动完成结果定义多个数据源。在代码清单19-1中，使用一个简单的字符串数组作为数据源。图19-1展示了自动完成组件的工作界面。

---

**警告** 自动完成组件并不会对输入数据做任何形式的验证，用户可以在文本框里输入任意值，并不受source选项中提供值的限制。

---



图中有两个屏幕截图，第一个展示的是输入字母P时的情况。在图中我们看到了包含字母P的一个数据列表，里面有以字母P开头的所有花卉名字，也包括了Snowdrop（因为其中含有字母p）。在第二个屏幕截图中，我输入了Pe两个字母，这时jQuery UI仅仅显示出了包含Pe的花卉名字。用户当然可以继续键入，也可以从自动完成列表中选择一项。

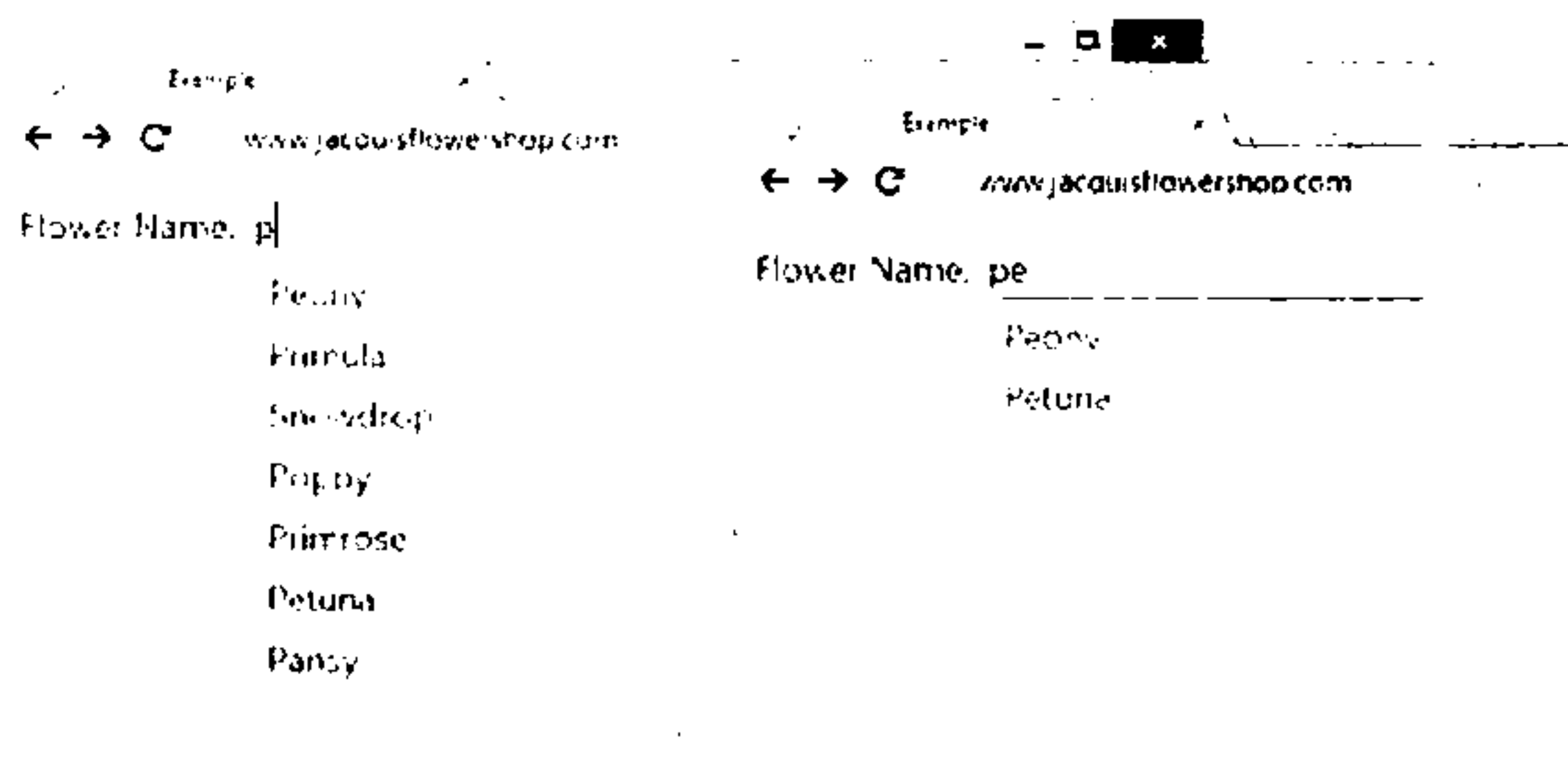


图19-1 一个简单的jQuery UI自动完成文本框

**提示** 在这个页面中，我把input元素和它的标签（label）放到了一个具有ui-widget class的div元素内。这样标签和input元素的字体样式就会与自动完成弹出菜单中的字体相匹配。我会在第35章详细介绍jQuery UI样式class的用法。

### 将对象数组用作数据源

也可以不使用字符串数组，而是使用一个对象数组作为数据源。这样我们就能实现这样的效果：用户看到的是描述更详细的文本（对象的label属性），而实际输入的值却是对象的value属性，如清单19-2所示。

#### 代码清单19-2 将对象数组作为自动完成功能的数据源

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function() {

      var flowers = [{label: "Aster (Purple)", value: "Aster"},
        {label: "Daffodil (White)", value: "Daffodil"},
        {label: "Rose (Pink)", value: "Rose"},
        {label: "Peony (Pink)", value: "Peony"}];

      $("#acInput").autocomplete({
```

```

        source: flowers
    });

    });
</script>
</head>
<body>
    <form>
        <div class="ui-widget">
            <label for="acInput">Flower Name: </label><input id="acInput"/>
        </div>
    </form>
</body>
</html>

```

当我们将对象数组用作数据源时，自动完成组件就去寻找label属性和value属性。label属性用来生成弹出菜单，value属性对应值用来在用户选中该项时插入到文本框。在这个例子中，我在label属性中额外加入了一些value中没有的颜色信息。这个例子的具体效果见图19-2。

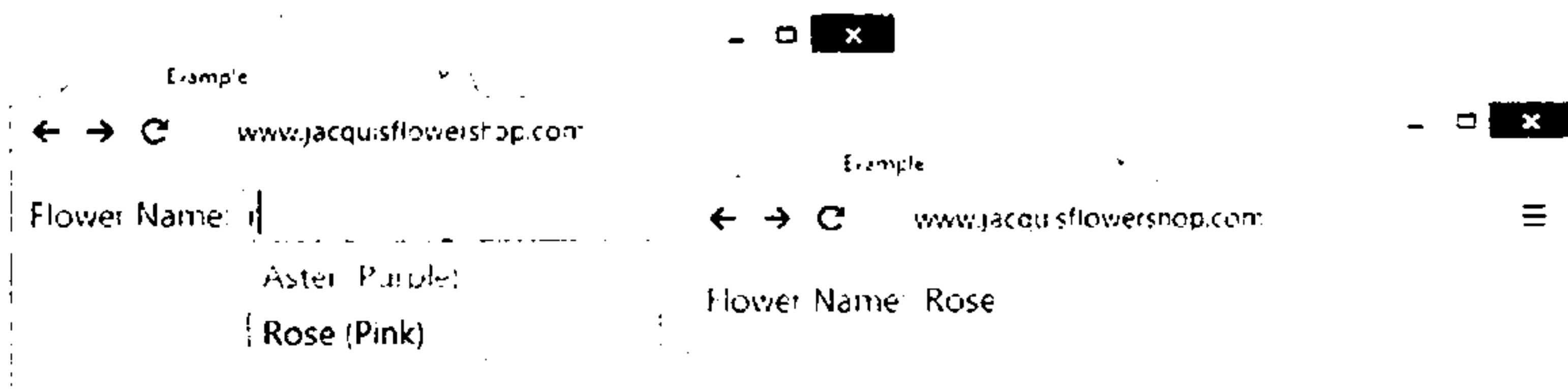


图19-2 使用对象数组定义显示文本和实际输入的值

### 19.1.2 配置自动完成组件

自动完成组件提供了许多选项，这些选项可用于控制该组件的方方面面。表19-2列出了这些选项及其作用。我会在接下来几节里演示如何使用这些选项配置自动完成组件。

表19-2 自动完成组件选项

选 项	描 述
appendTo	指定弹出层元素的DOM插入位置，默认值是body元素
autoFocus	默认值是false，若设置为true，则弹出层的第一项会自动得到焦点，也就是说用户可以直接按下回车键选中这一项
delay	指定用户按下一个键之后多长时间更新弹出层中的数据，默认值是300（单位ms）
disabled	默认值是false，若设置为true，则禁用文本框的自动完成功能。这个选项不会影响底层input元素的功能
minLength	指定用户输入最少多少个字符之后才开始弹出自动完成菜单，默认值是1

(续)

19

选 项	描 述
position	设置弹出层相对input元素的位置
source	指定自动完成弹出层项的数据来源。该选项没有默认值，在调用autocomplete方法时，必须指定一个数据源

### 1. 远程数据源

source选项是自动完成组件最激动人心的选项，因为我们可以使用各种各样的数据源来生成弹出层。我在清单19-2中使用了JavaScript数组，这对于静态的数据列表来说很合适。对于更复杂的应用场合，我们可以从远程服务器得到匹配数据。如代码清单19-3所示，我们可以指定一个URL来生成弹出层所需要的数据。

代码清单19-3 远程数据源

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function() {

      $("#acInput").autocomplete({
        source: "http://node.jacquisflowershop.com/auto"
      });
    });
  </script>
</head>
<body>
  <form>
    <div class="ui-widget">
      <label for="acInput">Flower Name: </label><input id="acInput"/>
    </div>
  </form>
</body>
</html>
```

“jQuery UI需要数据以生成弹出层时，它就向我们指定的URL发起GET请求。用户键入的内容会包含在查询字符串（名字term）中，比方说用户键入的字母是s，则jQuery UI就会请求下面的URL：

---

```
http://node.jacquisflowershop.com/auto?term=s
```

---

如果用户接着键入了字母n，那么jQuery UI请求的URL就变为：

---

```
http://node.jacquisflowershop.com/auto?term=sn
```

---

当数据量非常大，我们无法把全部数据发送给客户端时，这项技术非常有用。此外，该技术也适用于数据会不断更新，并且我们希望用户总是能看到最新数据的情况。

服务器负责从查询字符串中拿到term参数的值，然后返回对应用户需要查看的数组数据的JSON字符串。代码清单19-4展示了修改之后的formserver.js文件，里面是一段Node.js脚本，由它完成这件事情。（阅读第1章详细了解如何获取和安装Node.js。）

代码清单19-4 支持远程自动完成的Node.js脚本

```
var http = require("http");
var querystring = require("querystring");
var url = require("url");

var port = 80;

http.createServer(function (req, res) {
  console.log("[200 OK] " + req.method + " to " + req.url);

  var flowers = ["Aster", "Daffodil", "Rose", "Peony", "Primula", "Snowdrop",
    "Poppy", "Primrose", "Petuna", "Pansy"];

  var matches = [];
  var term = url.parse(req.url, true).query["term"];

  if (term) {
    var pattern = new RegExp("^" + term, "i");
    for (var i = 0; i < flowers.length; i++) {
      if (pattern.test(flowers[i])) {
        matches.push(flowers[i]);
      }
    }
  } else {
    matches = flowers;
  }

  res.writeHead(200, "OK", {
    "Content-Type": "application/json",
    "Access-Control-Allow-Origin": "*"
  });
  res.write(JSON.stringify(matches));
  res.end();

}).listen(port);
console.log("Ready on port " + port);
```

这个脚本使用了代码清单19-2中用到的花卉名字数据，并返回那些匹配term参数的花卉名字。我改进了搜索算法，只返回那些从名字开头匹配term参数的花卉。举例来说，假设jQuery UI发送的请求是：

```
http://node.jacquisflowershop.com/auto?term=p
```

那么Node.js脚本返回的JSON字符串就是：

```
["Peony", "Primula", "Poppy", "Primrose", "Petuna", "Pansy"]
```

如图19-3所示, 由于我改从花卉名字的开头匹配用户输入数据, 因此Snowdrop不会出现在列表中。

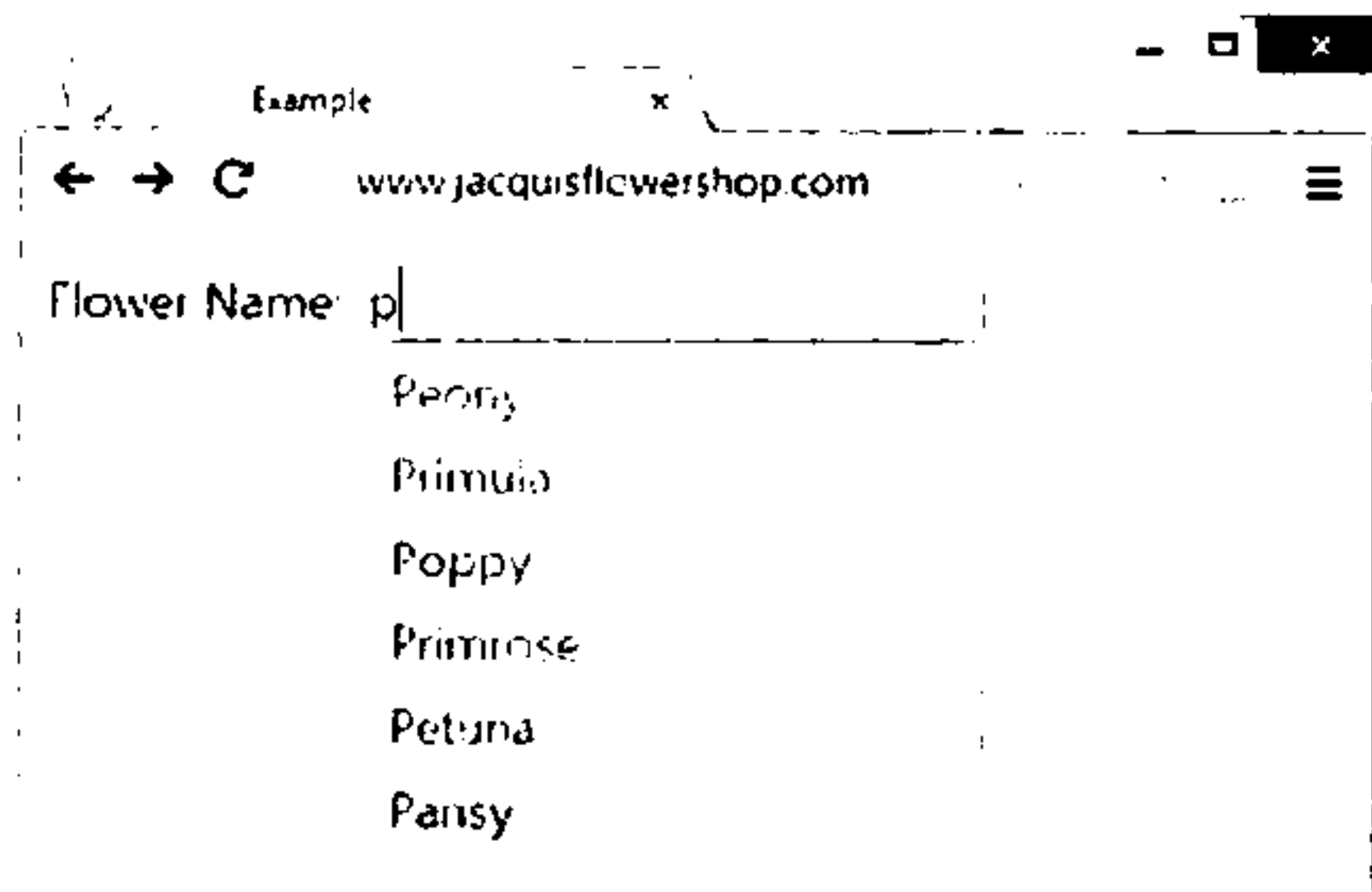


图19-3 从远程服务器获取自动完成数据

这是一项非常棒的技术, 但它会向服务器发起非常多的请求。对于这个例子来说, 这并不重要, 因为我实施的是非常简单的搜索, 而且服务器和浏览器都在同一个局域网上。然而, 对于互联网上复杂的搜索来说, 这很容易造成服务器过载, 从而产生让人痛苦的延迟。

管理自动完成请求频率的最佳方式是使用`minLength`和`delay`选项。`minLength`选项规定用户必须输入足够的字符才请求数据。我们可以利用这个选项, 只在用户输入几个字符之后向服务器发起请求, 这时我们已经有了足够多的信息来缩小搜索范围。

`delay`选项用来指定用户敲键之后等待多长时间才向服务器发起请求。利用这个选项, 我们能避免在用户正快速输入数据时向服务器发起不必要的请求。比方说, 用户输入了s, 接着输入了n, 我们就不必先向服务器请求关于s的数据列表并接着向服务器请求关于sn的数据列表。综合使用这两个选项, 我们就能大幅降低发往服务器的请求次数, 同时仍然能在用户需要时提供指导。

#### 代码清单19-5 利用`delay`和`minLength`选项降低对服务器的请求次数

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function() {

      $("#acInput").autocomplete({
        source: "http://node.jacquisflowershop.com/auto",
        minLength: 3,
        delay: 1000
      });
    });
  </script>
</head>
</html>
```

```

    });
  </script>
</head>
<body>
  <form>
    <div class="ui-widget">
      <label for="acInput">Flower Name: </label><input id="acInput"/>
    </div>
  </form>
</body>
</html>

```

在这个例子中，用户输入3个字符之前，程序不会向服务器发起任何请求，并且在至少1s内没有输入任何字符时才向服务器发起请求。

## 2. 函数数据源

我们也可以将函数用作数据源，以生成自动完成所需的数据。我们把source选项的值设置为一个函数，在需要生成弹出层时调用这个函数。代码清单19-6演示了这种用法。

代码清单19-6 使用函数生成自动完成数据

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function() {

      var flowers = ["Aster", "Daffodil", "Rose", "Peony", "Primula", "Snowdrop",
                    "Poppy", "Primrose", "Petuna", "Pansy"];

      $("#acInput").autocomplete({
        source: function(request, response) {
          var term = request.term;
          var pattern = new RegExp("^" + term, "i");

          var results = $.map(flowers, function(elem) {
            if (pattern.test(elem)) {
              return elem;
            }
          });
          response(results);
        }
      });
    });
  </script>
</head>
<body>
  <form>
    <div class="ui-widget">
      <label for="acInput">Flower Name: </label><input id="acInput"/>
    </div>
  </form>
</body>
</html>

```

```

    </div>

    </form>
</body>
</html>

```

jQuery UI会传递两个参数给这个函数。第一个参数是一个对象，它只有一个属性，即term属性。这个属性的值就是用户在文本框内键入的内容。第二个参数是一个函数，当我们准备好要显示的数据之后就调用这个函数生成弹出层。这个函数的参数可以是一个字符串数组，也可能是对象数组。

在这个例子中，我重新实现了上一个例子中服务器端实现的功能，生成一个起始字符符合term参数的字符串数组。

---

**提示** 在这里我使用了jQuery的map实用方法（详见第34章）处理数组内容。

---

接下来，我把处理结果作为response函数的参数又传递回jQuery UI:

```

...
response(results);
...

```

这种处理数据的方式有点怪，但它支持我们在异步任务完成之后调用该函数。在代码清单19-7中，你将看到如何使用函数发起Ajax请求，得到花卉的详细数据，并对返回数据实施本地搜索，最后调用response函数得到jQuery UI需要的最终结果。

#### 代码清单19-7 发起异步任务的自定义数据源

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    button {margin-bottom: 5px}
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#acInput").autocomplete({
        source: function (request, response) {
          $.getJSON("http://node.jacquisflowershop.com/auto",
            function(flowers) {
              var term = request.term;
              var pattern = new RegExp("^" + term, "i");

              var results = $.map(flowers, function (elem) {
                if (pattern.test(elem)) {
                  return elem;
                }
              });
              response(results);
            });
    });
  </script>

```

```

        }
    });
});
</script>
</head>
<body>
    <form>
        <div class="ui-widget">
            <label for="acInput">Flower Name: </label><input id="acInput"/>
        </div>

    </form>
</body>
</html>

```

在这个例子中，我调用getJSON方法从Node.js服务器获取完整的花卉数据，然后执行本地搜索匹配用户键入数据，得到供jQuery UI显示的建议列表。

### 3. 设置弹出层的位置

默认情况下，弹出的选择列表显示在input元素的下方。不过我们可以利用position属性改变这一位置，尽管设置位置语法比较晦涩。代码清单19-8演示了如何改变弹出层的位置。

代码清单19-8 改变弹出层的位置

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
    <style>
        #target { margin-top: 40px; display:inline-block }
    </style>
    <script type="text/javascript">
        $(document).ready(function () {

            var flowers = [{ label: "Aster (Purple)", value: "Aster" },
                { label: "Daffodil (White)", value: "Daffodil" },
                { label: "Rose (Pink)", value: "Rose" },
                { label: "Peony (Pink)", value: "Peony" }];

            $("#acInput").autocomplete({
                source: flowers,
                position: {
                    my: "left top",
                    at: "right bottom+20",
                    of: "#target",
                    collision: "fit"
                }
            });

        });
    </script>

```



```

</head>
<body>
  <form>
    <div class="ui-widget">
      <label for="acInput">Flower Name: </label><input id="acInput"/>
    </div>
  </form>
  <span id="target">Target</span>
</body>
</html>

```

position属性的值是一个对象，它指定了摆放弹出层的一系列规则。例子中用到的四个属性见表19-3。

表19-3 自动完成组件的位置属性

属 性	描 述
my	指定是弹出层自身的什么位置相对于目标元素（可取值参见后文）
at	指定`my`属性定义的弹出层位置位于目标元素的什么位置（可取值参见后文）
of	指定弹出层的目标元素，若省略则为对应的input元素，可取值是HTMLElement对象、选择器或jQuery对象
collision	在弹出层遮住窗口时，调整弹出层位置的方式（可取值详见表19-4）

**提示** 自动完成组件的弹出层摆放使用了jQuery UI的position应用程序功能，该程序拥有许多选项，本章只使用了其中的几个选项。访问<http://api.jqueryui.com/position>可了解更多细节。

my和at属性均使用中间以空格相隔的水平与垂直位置值指定。水平方向可取值为left、right和center；重直方向可取值为top、bottom和center。也可以用百分比或者像素值指定位置偏移量。在上面的例子里，我使用的my、at和of属性值如下：

```

...
my: "left top",
at: "right bottom+20",
of: "#target",
...

```

这个组合的含义如下：自动完成弹出层的左上角位于低于目标元素#target右下角20像素的位置。不要把of属性设置为不相干的普通元素，那样会打破弹出层与输入框之间的视觉关联。在这里之所以这么做，是为了演示可以灵活设置弹出层的显示位置。这个配置组合的实际显示效果见图19-4。

collision属性用来指定当弹出层显示空间不够时的调整策略。表19-4列出了这个属性支持的可取值。

表19-4 collision属性可取值

属 性	描 述
flip	jQuery UI会检查of属性指定元素对面（上面或下面）哪一边空间最大，选择空间较大的一边显示弹出层
fit	jQuery UI会移动弹出层的位置，以避免弹出层超出窗口范围
flipfit	综合两种属性的行为：先检查哪边空间大，确定显示在哪一边，再移动弹出层位置避免超出窗口范围
none	对弹出层的位置不做调整

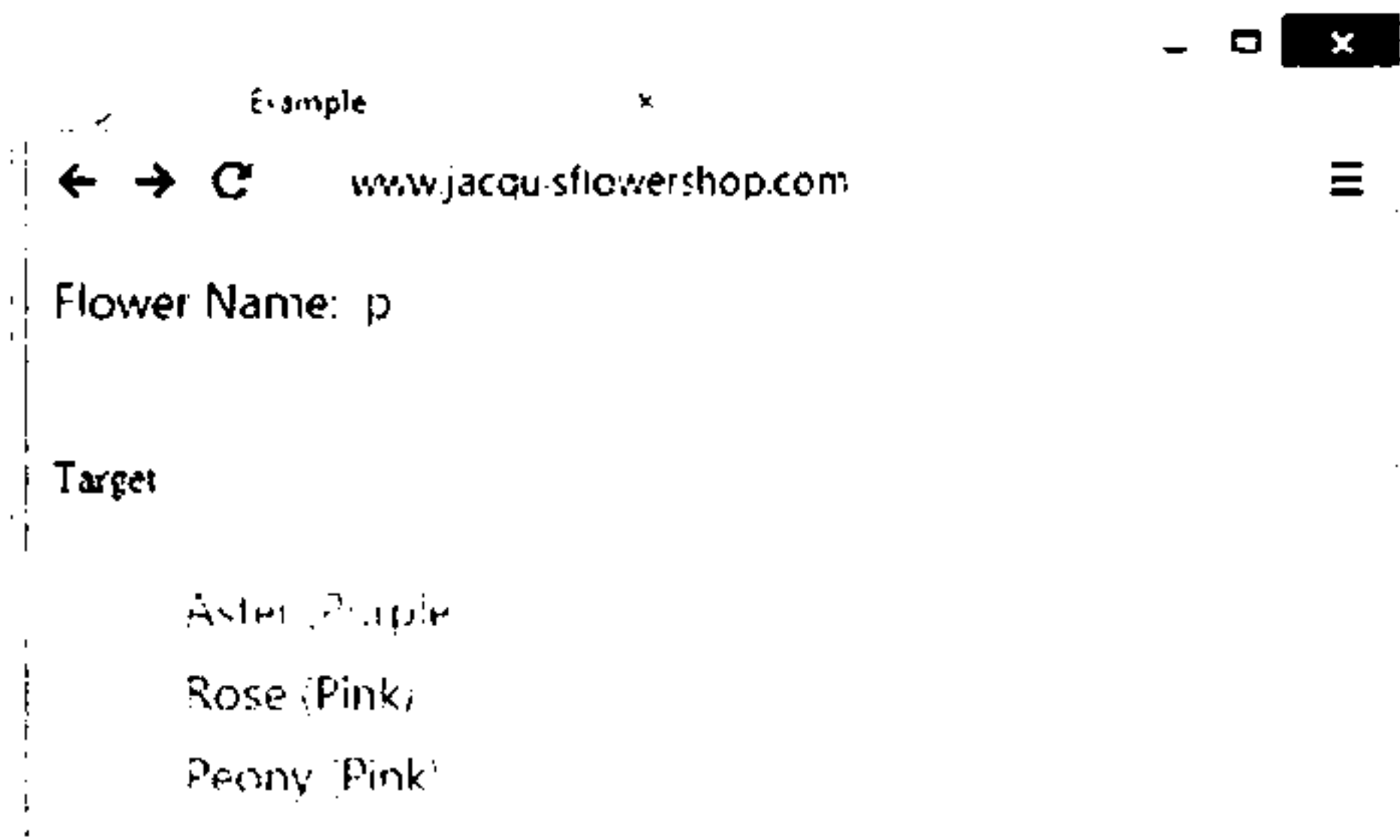


图19-4 配置自动完成组件弹出层的显示位置

如图19-5所示，我把collision属性的值设置为fit，这意味着jQuery UI在必要时会移动弹出层的位置以适应浏览器窗口。

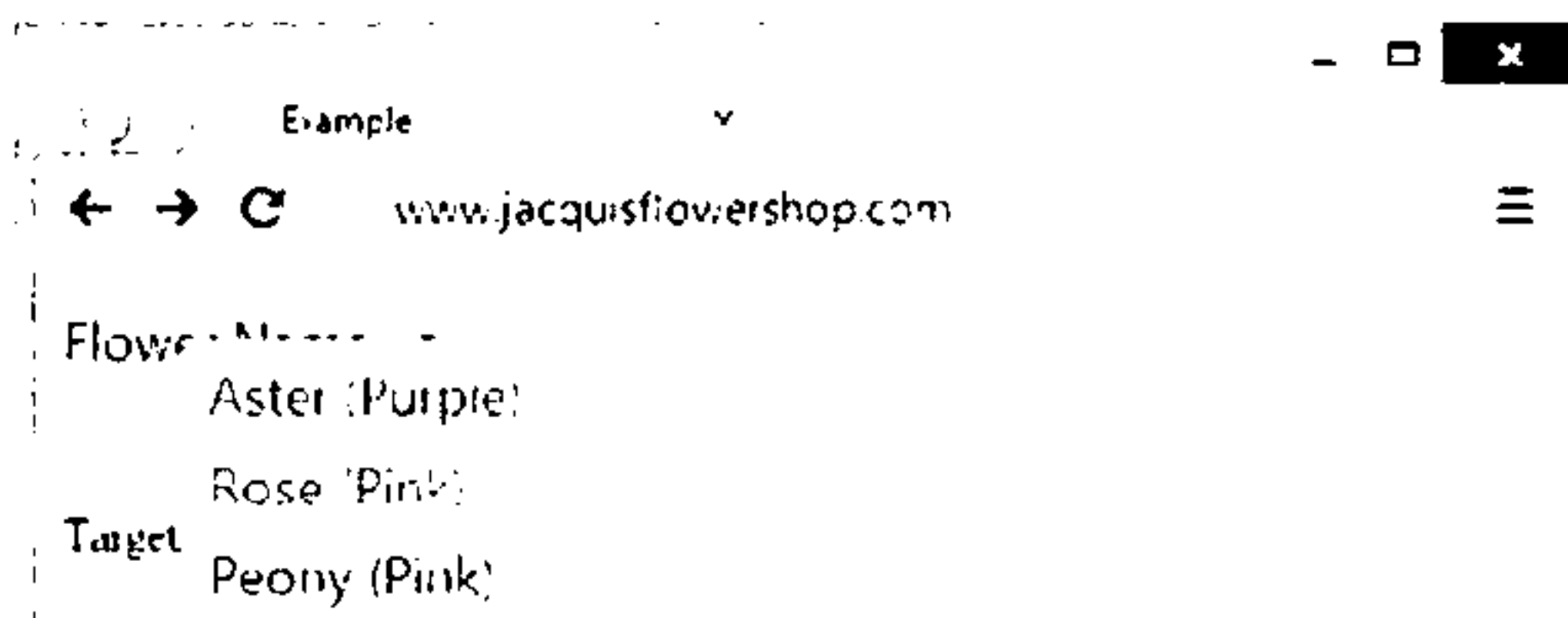


图19-5 移动弹出层以适应浏览器窗口

### 19.1.3 自动完成组件方法

jQuery UI自动完成组件还定义了一些方法，我们可以利用它们对组件进行控制。表19-5列出了这些方法。

表19-5 autocomplete方法

方 法	描 述
<code>autocomplete("close")</code>	关闭自动完成弹出菜单
<code>autocomplete("destroy")</code>	删除自动完成组件功能
<code>autocomplete("disable")</code>	禁用自动完成功能
<code>autocomplete("enable")</code>	启用文本框自动完成功能
<code>autocomplete("option")</code>	设置一个或多个选项
<code>autocomplete("search", value)</code>	用某个值主动触发自动完成菜单，若没有提供value参数，则会将用户输入的值用作参数

如代码清单19-9所示，search和close方法是自动完成组件特有的，我们可以使用它们主动弹出或者关闭自动完成列表。

代码清单19-9 使用search和close方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    button {margin-bottom: 5px}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      var flowers = ["Aster", "Daffodil", "Rose", "Peony", "Primula", "Snowdrop",
                    "Poppy", "Primrose", "Petuna", "Pansy"];

      $("#acInput").autocomplete({
        source: flowers
      });

      $("button").click(function(e) {
        e.preventDefault();
        switch (this.id) {
          case "close":
            $("#acInput").autocomplete("close");
            break;
          case "input":
            $("#acInput").autocomplete("search");
            break;
          default:
            $("#acInput").autocomplete("search", this.id);
            break;
        }
      });
    });
  </script>
</head>
<body>
  <form>
    <button id="s">S</button>
    <button id="p">P</button>
    <button id="input">Input Content</button>
    <button id="close">Close</button>
    <div class="ui-widget">
      <label for="acInput">Flower Name: </label><input id="acInput"/>
    </div>
  </form>
</body>
</html>
```

在这个例子里，我添加了几个按钮，并使用click方法为它们绑定了不同的组件方法调用。当点击S或者P按钮时，我调用search方法，并把选中的字符作为搜索使用的值。如图19-6所示，这会触发基于选中字母的自动完成行为，完全不考虑文本输入框中的内容。

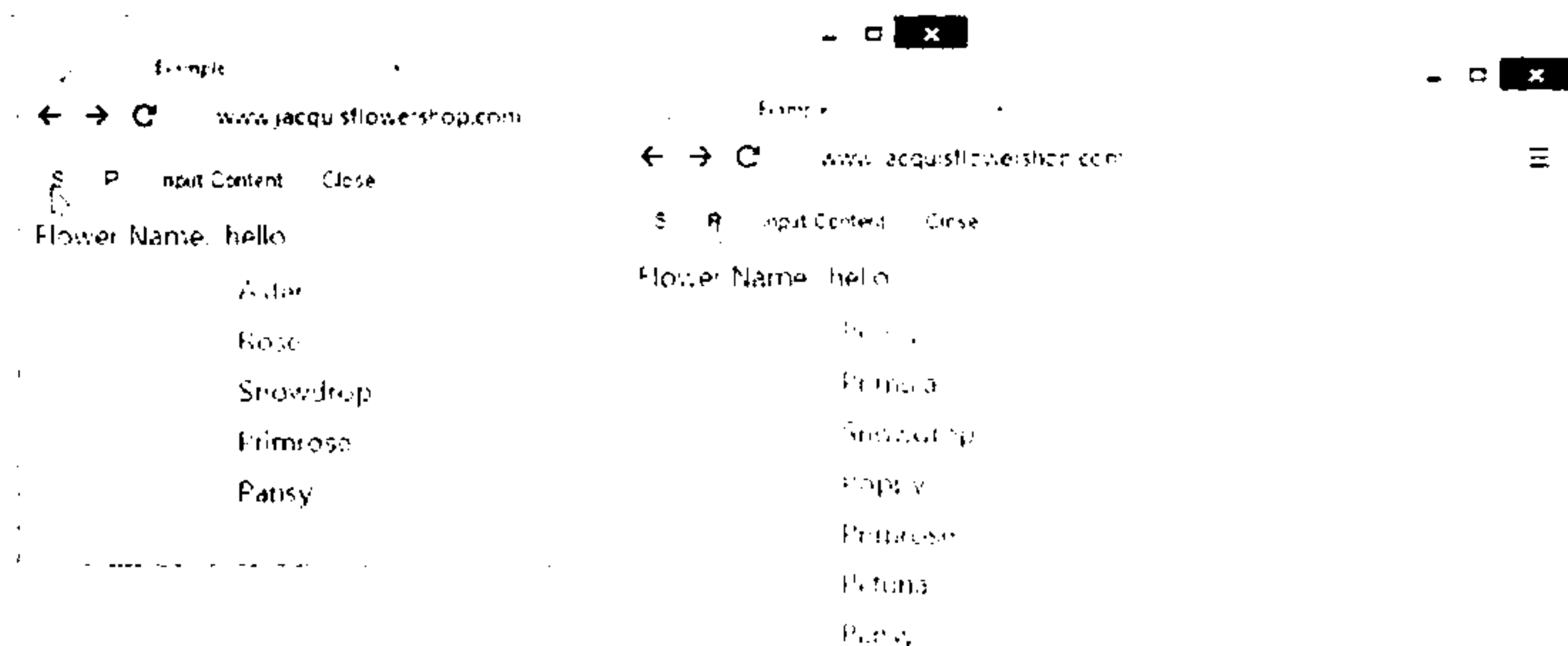


图19-6 使用search方法搜索特定参数

如图19-6所示，弹出菜单里显示的列表项匹配按钮上的文本，即便文本框中已经输入了内容hello。点击Input Content（输入内容）按钮则会基于文本框中输入的内容弹出自动完成菜单，如图19-7所示。

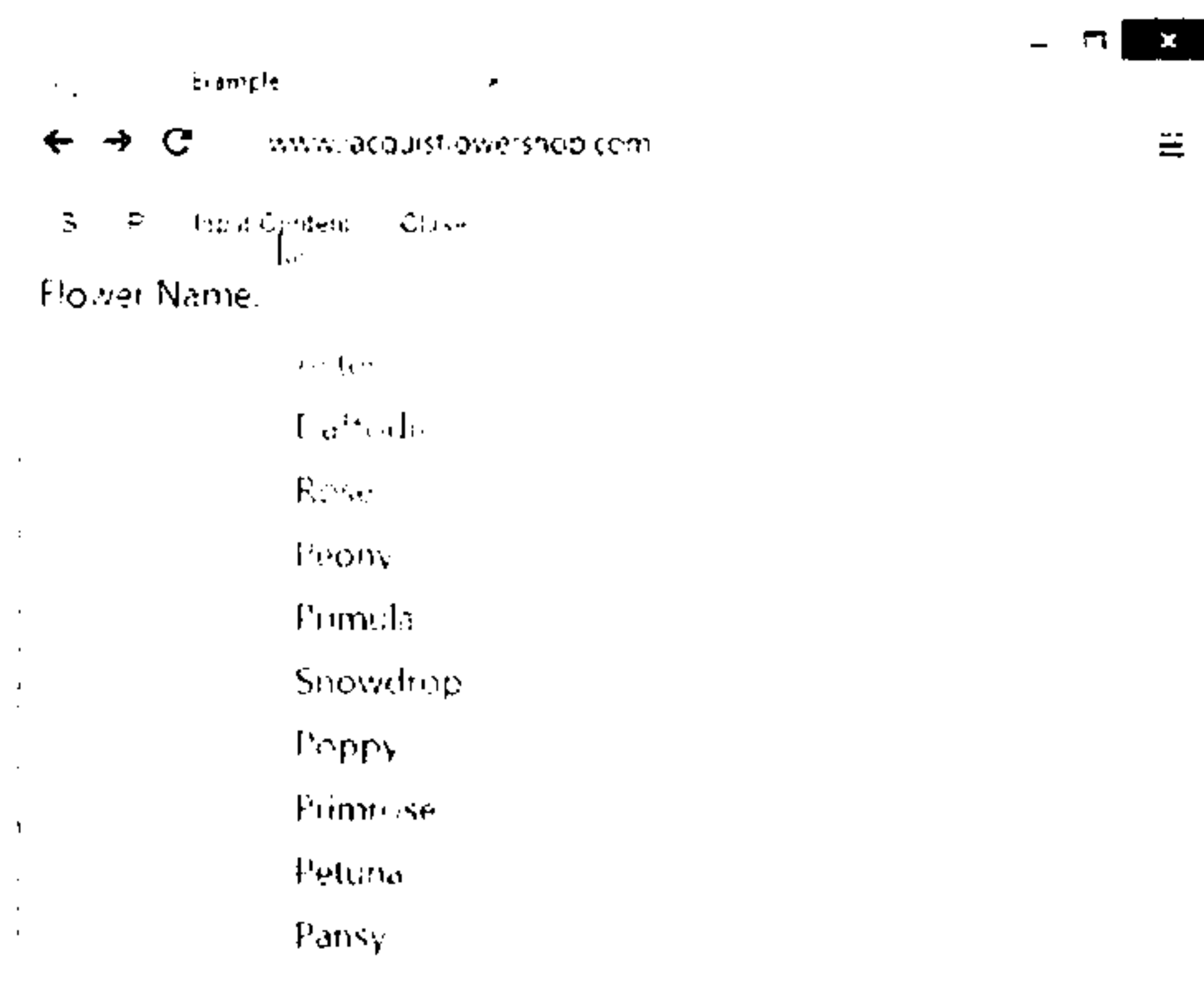


图19-7 使用文本框中的内容搜索推荐数据

最后一个按钮是Close，点击时会调用组件的close方法关闭弹出菜单。

### 19.1.4 自动完成组件事件

自动完成组件定义了若干事件，参见表19-6。

表19-6 自动完成事件

事 件	描 述
change	当文本框的值发生了变化，且文本框失去焦点时触发
close	当弹出菜单关闭时触发
create	当自动完成组件创建完成时触发
focus	当弹出菜单上的一项得到焦点时触发
open	当弹出菜单显示出来时触发
response	在搜索已经完成但尚未显示给用户看时触发
search	在自动完成数据生成完毕之前或请求完成之前触发
select	当弹出菜单中的一项被选中时触发

#### 1. 获取选中项详细信息

jQuery UI在调用事件处理函数时利用第二个参数提供了一些附加信息，习惯上我们把这个参数叫做ui对象。对于change、focus和select事件来说，jQuery UI提供的ui对象有一个item属性，这个属性保存了描述弹出菜单中被选中项或者得到焦点的项。代码清单19-10所示的这个例子充分利用了这一特性来获取该数据项的信息。

代码清单19-10 在事件处理函数中使用ui对象

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function() {

      var flowers = ["Aster", "Daffodil", "Rose", "Peony", "Primula", "Snowdrop",
                    "Poppy", "Primrose", "Petuna", "Pansy"];

      $("#acInput").autocomplete({
        source: flowers,
        focus: displayItem,
        select: displayItem,
        change: displayItem
      })

      function displayItem(event, ui) {
        $("#itemLabel").text(ui.item.label)
      }
    });
  </script>
</head>
</html>
```

```

</script>
</head>
<body>
  <form>
    <div class="ui-widget">
      <label for="acInput">Flower Name: </label><input id="acInput"/>
      Item Label: <span id="itemLabel"></span>
    </div>
  </form>
</body>
</html>

```

在这个例子中我增加了一个span元素，用它显示选中数据项的label属性。jQuery UI会自动为数据项对象创建label和value属性，即使我们使用简单的字符串数组作为source选项的数据源。因此我们总是需要读取ui.item对象的某个属性。在这个例子中，我使用同一个函数显示focus事件、select事件及change事件中数据项的信息。图19-8是该例子在浏览器中的呈现效果。

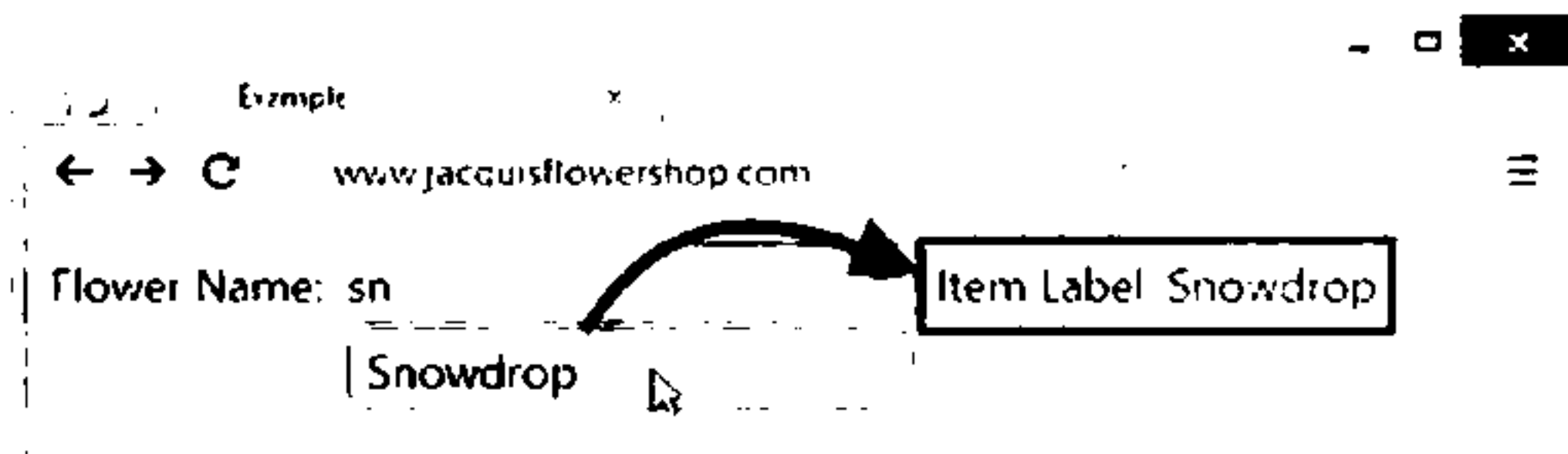


图19-8 得到选中项的详细信息

## 2. 修改搜索结果

response事件在数据显示给用户看之前，提供了最后的修改机会。在代码清单19-11中，我演示了如何使用response事件避免用户看到Peony项。

### 代码清单19-11 处理response事件

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function () {

      var flowers = ["Aster", "Daffodil", "Rose", "Peony", "Primula", "Snowdrop",
                    "Poppy", "Primrose", "Petuna", "Pansy"];

      $("#acInput").autocomplete({
        source: flowers,
        response: filterResults
      });

      function filterResults(event, ui) {

```

```

        for (var i = 0; i < ui.content.length; i++) {
            if (ui.content[i].label == "Peony") {
                ui.content.splice(i, 1);
            }
        }
    });
</script>
</head>
<body>
    <form>
        <div class="ui-widget">
            <label for="acInput">Flower Name: </label><input id="acInput"/>
        </div>
    </form>
</body>
</html>

```

我定义了filterResults函数来响应response事件。这个函数中枚举了即将显示给用户看的结果（通过ui.content属性）。response事件的处理函数必须直接修改数组，因此使用splice方法来删除ui.content数组中的Peony项。

### 3. 覆盖默认选择行为

select事件的默认行为是使用弹出菜单里选中项的value属性替换文本框中用户输入的内容。绝大多数情况下，这个行为正是我们所需要的。不过，我们可以利用这个事件做一些补充工作，甚至完全阻止它的这一行为，做些别的事情。代码清单19-12是一个补充默认行为的例子，它额外设置了一个相关文本框的值。

代码清单19-12 覆盖select事件的默认选择行为

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
    <script type="text/javascript">
        $(document).ready(function() {

            var flowers = ["Aster", "Daffodil", "Rose"];

            var skus = { Aster: 100, Daffodil: 101, Rose: 102};

            $("#acInput").autocomplete({
                source: flowers,
                select: function(event, ui) {
                    $("#sku").val(skus[ui.item.value]);
                }
            });
        });
    </script>
</head>

```

```

<body>
  <form>
    <div class="ui-widget">
      <label for="acInput">Flower Name: </label><input id="acInput"/>
      <label for="sku">Stock Keeping Unit: </label><input id="sku"/>
    </div>
  </form>
</body>
</html>

```

**提示** 如果需要详细了解事件的默认行为，请阅读第9章。

当select事件发生时，处理函数使用ui参数得到当前选中项的值，并根据这个值设置关联文本框input#sku。在这个例子里，库存量（stock keeping unit）保存在skus对象中。以这种方式，我们可以基于最初的选择结果对其他字段提供默认值，以此帮助用户。在许许多多的场合中这种做法都价值巨大，特别是在选择收货地址的时候。这个例子的具体效果见图19-9，你最好在浏览器中运行一下这个例子，以使能得到完美的体验。这个页面的HTML代码与书中所有其他例子的源代码都可以从Apress.com的源代码/下载区免费下载。

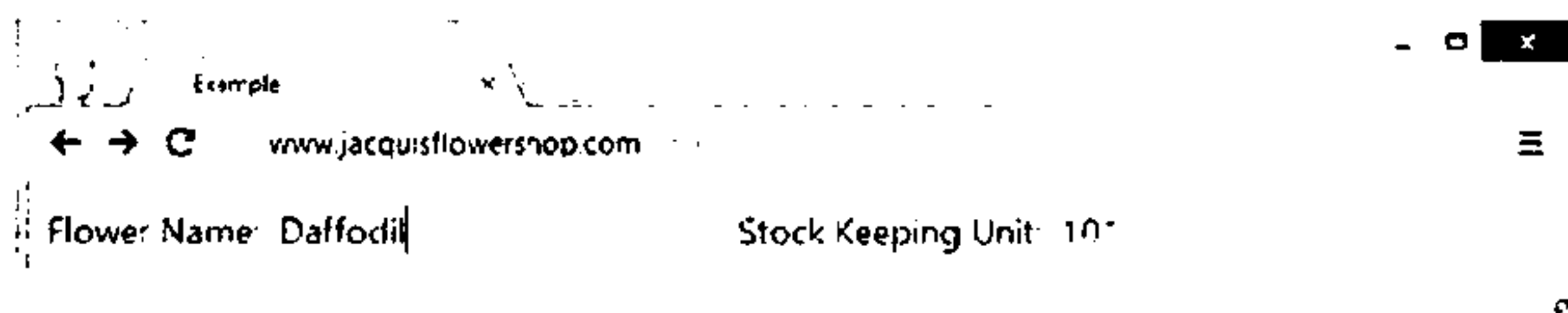


图19-9 利用select事件填充另一个文本框

## 19.2 jQuery UI 折叠菜单组件

折叠（accordion）菜单组件把多个内容块汇集在一起，用户最多同时只能看到一个内容。当用户选中另一块内容时，原来可见的内容就隐藏，这样可以营造出类似于手风琴上风箱的效果。

折叠菜单组件非常适合展示那些各节内容相对独立，而且我们不想让用户一览无余的场合。理想情况下，这些相对独立的内容都属于同一个主题，并且可以使用简单的标题来表达。

### 19.2.1 创建折叠菜单

可以调用accordion方法生成jQuery UI折叠菜单组件，参见代码清单19-13。

代码清单19-13 创建一个折叠菜单

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js"></script>

```



```

<script src="handlebars-jquery.js"></script>
<script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<style type="text/css">
    #accordion {margin: 5px}
    .dcell img {height: 60px}
</style>
<script id="flowerTpl" type="text/x-jquery-tmpl">
    {{#flowers}}
    <div class="dcell">
        
        <label for="{{product}}">{{name}}:</label>
        <input name="{{product}}" value="0" />
    </div>
    {{/flowers}}
</script>
<script type="text/javascript">
    $(document).ready(function () {
        var data = {
            flowers: [{ "name": "Aster", "product": "aster" },
                { "name": "Daffodil", "product": "daffodil" },
                { "name": "Rose", "product": "rose" },
                { "name": "Peony", "product": "peony" },
                { "name": "Primula", "product": "primula" },
                { "name": "Snowdrop", "product": "snowdrop" },
                { "name": "Carnation", "product": "carnation" },
                { "name": "Lily", "product": "lily" },
                { "name": "Orchid", "product": "orchid" }]
        };

        var elems = $("#flowerTpl").template(data).filter("*");
        elems.slice(0, 3).appendTo("#row1");
        elems.slice(3, 6).appendTo("#row2");
        elems.slice(6).appendTo("#row3");

        $("#accordion").accordion();

        $("#button").button();
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="accordion">
            <h2><a href="#">Row 1</a></h2>
            <div id="row1"></div>
            <h2><a href="#">Row 2</a></h2>
            <div id="row2"></div>
            <h2><a href="#">Row 3</a></h2>
            <div id="row3"></div>
        </div>
        <div id="buttonDiv"><button type="submit">Place Order</button></div>
    </form>
</body>
</html>

```

```

    </form>
</body>
</html>

```

例子中最重要的部分是div#accordion元素。

```

...
<div id="accordion">
  <h2><a href="#">Row 1</a></h2>
  <div id="row1"></div>

  <h2><a href="#">Row 2</a></h2>
  <div id="row2"></div>

  <h2><a href="#">Row 3</a></h2>
  <div id="row3"></div>
</div>
...

```

我重排了一下代码，以便让结构更明显。我们在最外层的div元素上调用accordion方法以生成折叠菜单。当调用accordion方法时，jQuery UI会在div元素的内容里寻找标题元素（h1到h6），并把内容分组，让每个标题与它的下一个元素关联。在本例中，我将h2元素用作标题，每个h2元素之后紧跟着一个div元素。我会使用数据模板为这些div元素生成产品内容（基于花店产品数据）。

注意，每个h2元素里包裹了一个a元素。这就是我们为每个内容块指定标题的用意。图19-10展示了经jQuery UI转换之后div#accordion的样子。

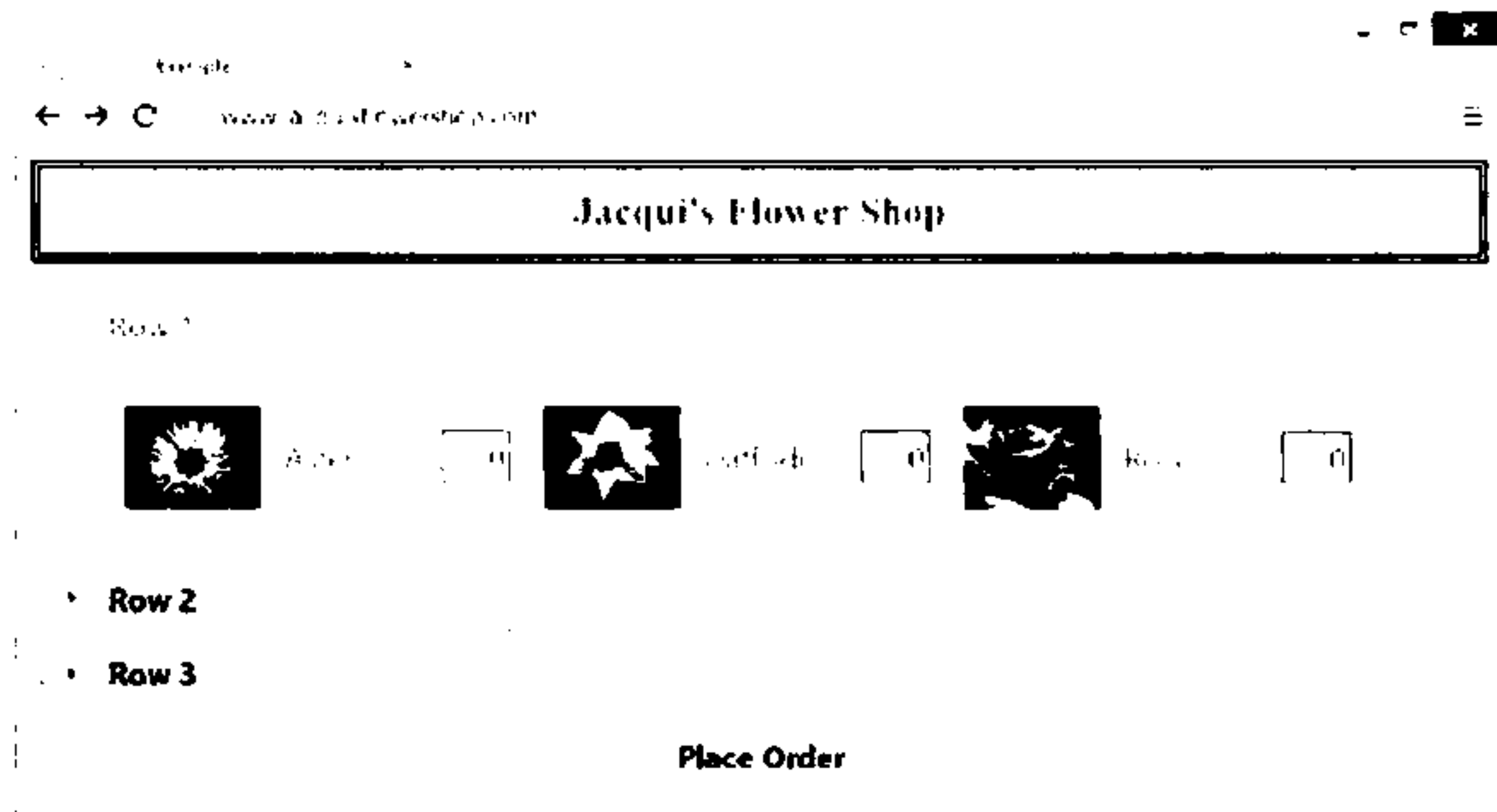


图19-10 jQuery UI折叠菜单

**提示** 把href属性设置为#是一种声明这个a元素仅被JavaScript使用的常见技术。<sup>(1)</sup>我把它用在这里纯粹是为了让例子更简单。我建议使用jQuery动态地插入这些a元素，这样就不会妨碍那些不使用JavaScript的用户。

(1) 作为一个开发者，在这种场合我更倾向于使用根本没有href的属性。这样就可以避免不必要的副作用。至于这样做造成鼠标放到这些a上不会变成可点击的小手形状这一问题，完全可以使用样式轻松解决。——译者注

折叠菜单创建之后，第一块内容显示出来，其他内容则处于隐藏状态。a元素的内容用作每块内容的标签，点击标签则关闭当前显示的内容，打开选中的内容（内容转换有着美妙的动画效果，很遗憾这些效果我没法用截图表现出来）。点击标题的效果见图19-11。

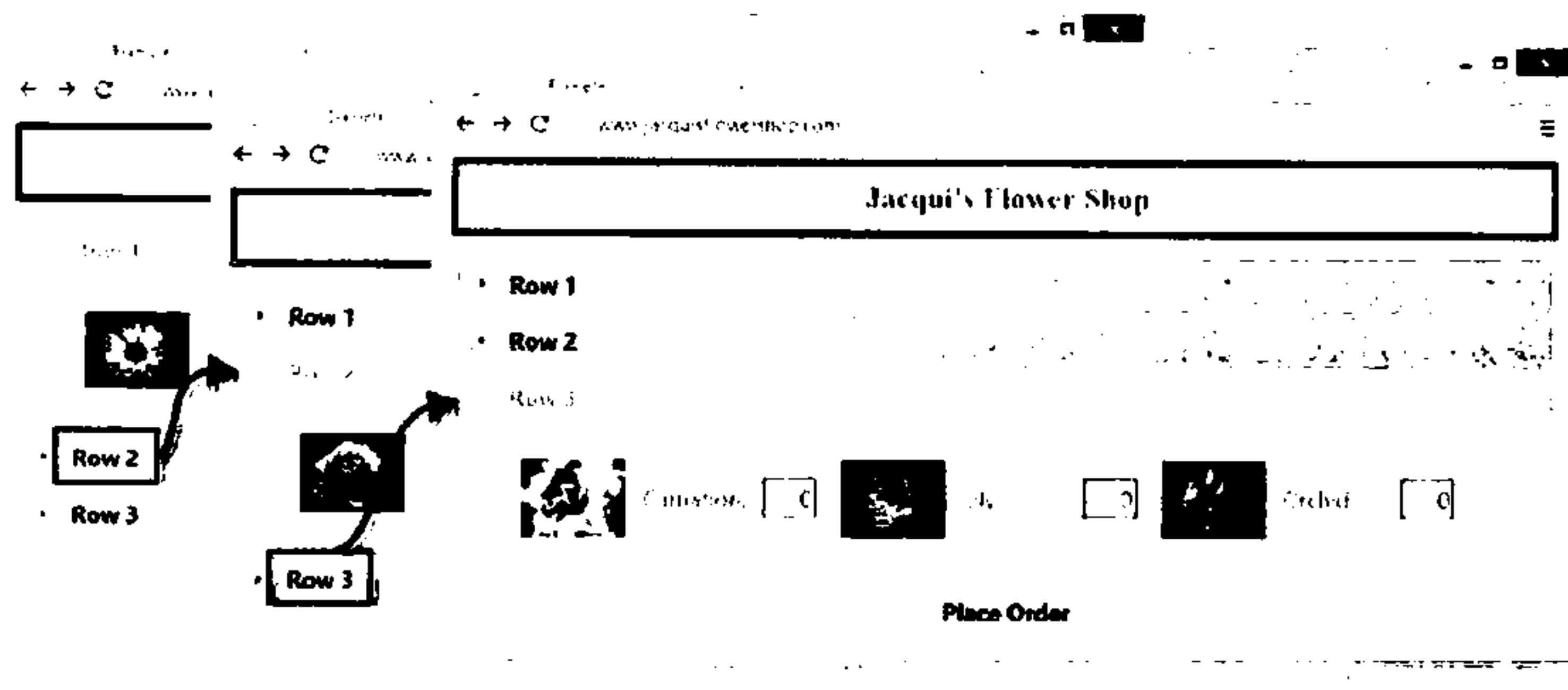


图19-11 切换折叠菜单

## 19.2.2 配置折叠菜单组件

折叠菜单组件定义了许多选项，我们可以利用这些选项微调组件的行为。表19-7列出了这些选项。在接下来的内容中，我将演示如何利用这些选项配置折叠菜单组件。

表19-7 折叠菜单组件选项

选 项	描 述
active	获取或者设置将要显示的内容元素。默认初始显示第一个内容元素
animated	指定从一个内容元素切换到另一个内容元素时使用的动画特效。参阅第35章以详细了解jQuery UI 动画特效
collapsible	默认值是false，如果设置为true，所有内容均可折叠
disabled	默认值是false，若设置为true，则折叠菜单组件被禁用
event	指定由标题元素触发内容元素切换的事件，默认值是click
header	指定哪些元素被当做标题元素
heightStyle	控制折叠菜单及其内容面板的高度
icons	指定折叠菜单使用的图标

**提示** 在jQuery UI 1.10中，animate选项取代了animated选项；heightStyle选项替代了旧版中的autoHeight选项、clearStyle选项和fillSpace选项；新添加的activeHeader选项，则用来指定激活面板所使用的图标（若设置了icons属性的话）。

### 1. 设置折叠菜单的高度

heightStyle属性用来设置折叠菜单及其面板的高度，它支持三个可取值，见表19-8。

表19-8 heightStyle属性可取值

名 称	描 述
auto	所有面板与最高的面板等高
fill	所有面板与父元素等高
content	每个面板与自身内容等高

有许多方法可以设置折叠菜单组件的高度，包括基于内容元素的高度或者父元素的高度进行设置。最常用的处理就是直接使用auto选项的默认值。它会把所有内容元素的高度设置为一个统一的值（高度最大元素的高度），并基于这个高度确定折叠菜单的尺寸。

这正是我在上一个例子里使用的技术。不过，如果内容区包含图片，特别是这些图片是使用jQuery插入到页面中去的，就要特别注意。问题发生在我们调用accordion方法时，这时很可能还有一些图片尚未被浏览器加载完毕，这样就会误导jQuery UI，从而在计算内容区高度时出错。在示例页面中，内容区高度在图片加载前是55 px，而图片加载完毕后的高度是79 px。如图19-12所示，当我们看到折叠菜单的内容区出现意外的滚动条时，就知道出问题了。

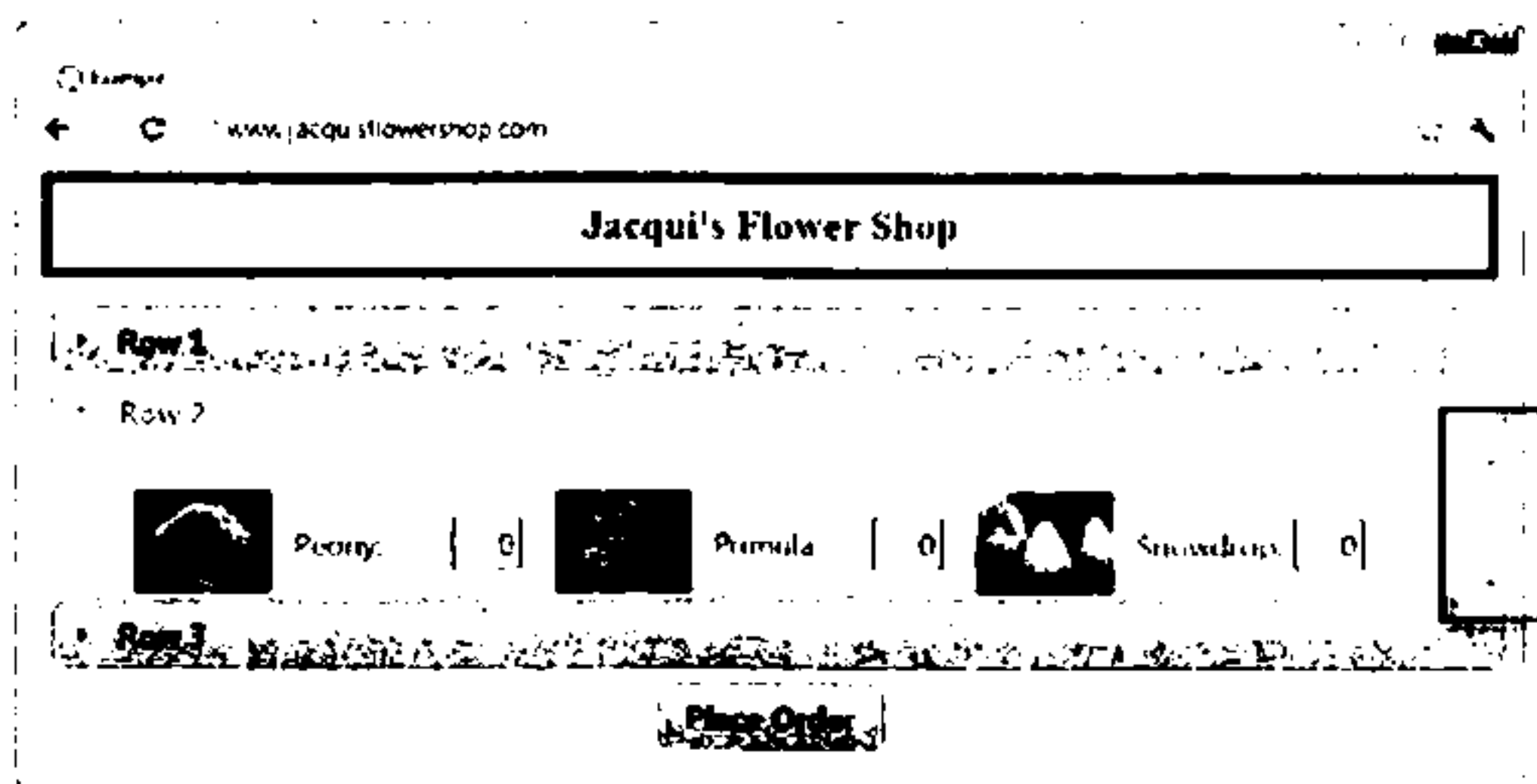


图19-12 高度错误导致页面问题

jQuery UI不能检测到内容区高度随着图片加载完成而发生的变化，因而会在显示内容时出错。为了解决这个问题，我们需要提供所有外部资源都加载完毕之后内容区的高度信息。有很多种方法做到这一点，在本例中，我选择为img元素设置高度：

```
...
<style type="text/css">
    #accordion {margin: 5px}
    .dcell img {height: 60px}
</style>
...
```

暂时撇开图片的问题，如果我们希望各个内容块高度一致，使用autoHeight选项就很合适。然而，当各内容块的尺寸差异过大时，这一设置就会导致怪异的视觉效果。代码清单19-14换用了一种稍微不同的产品元素插入方式。

代码清单19-14 内容区尺寸有着巨大差异的折叠菜单

```

...
<script type="text/javascript">
    $(document).ready(function () {

        var data = [{ "name": "Aster", "product": "aster" },
            { "name": "Daffodil", "product": "daffodil" },
            { "name": "Rose", "product": "rose" },
            { "name": "Peony", "product": "peony" },
            { "name": "Primula", "product": "primula" },
            { "name": "Snowdrop", "product": "snowdrop" },
            { "name": "Carnation", "product": "carnation" },
            { "name": "Lily", "product": "lily" },
            { "name": "Orchid", "product": "orchid" } ]];

        var elems = $("#flowerTpl").template(data).filter("*");
        elems.slice(0, 3).appendTo("#row1");
        elems.slice(3, 6).appendTo("#row2");
        elems.slice(6).appendTo("#row3");

        $("<h2><a href=#>All</a></h2><div id=row0></div>").prependTo("#accordion")
            .filter("div").append($("#row1, #row2, #row3").clone())

        $("#accordion").accordion();

        $("button").button();
    });
</script>
...

```

为了得到一个特别高的内容块，我使用jQuery的clone方法克隆已有内容，并把克隆好的div元素插入到一个新内容块。这样，这个内容块就包含所有的产品，它是其他内容块的3倍高，从而导致折叠菜单在显示较小的内容块时底部留有大块空白，如图19-13所示。

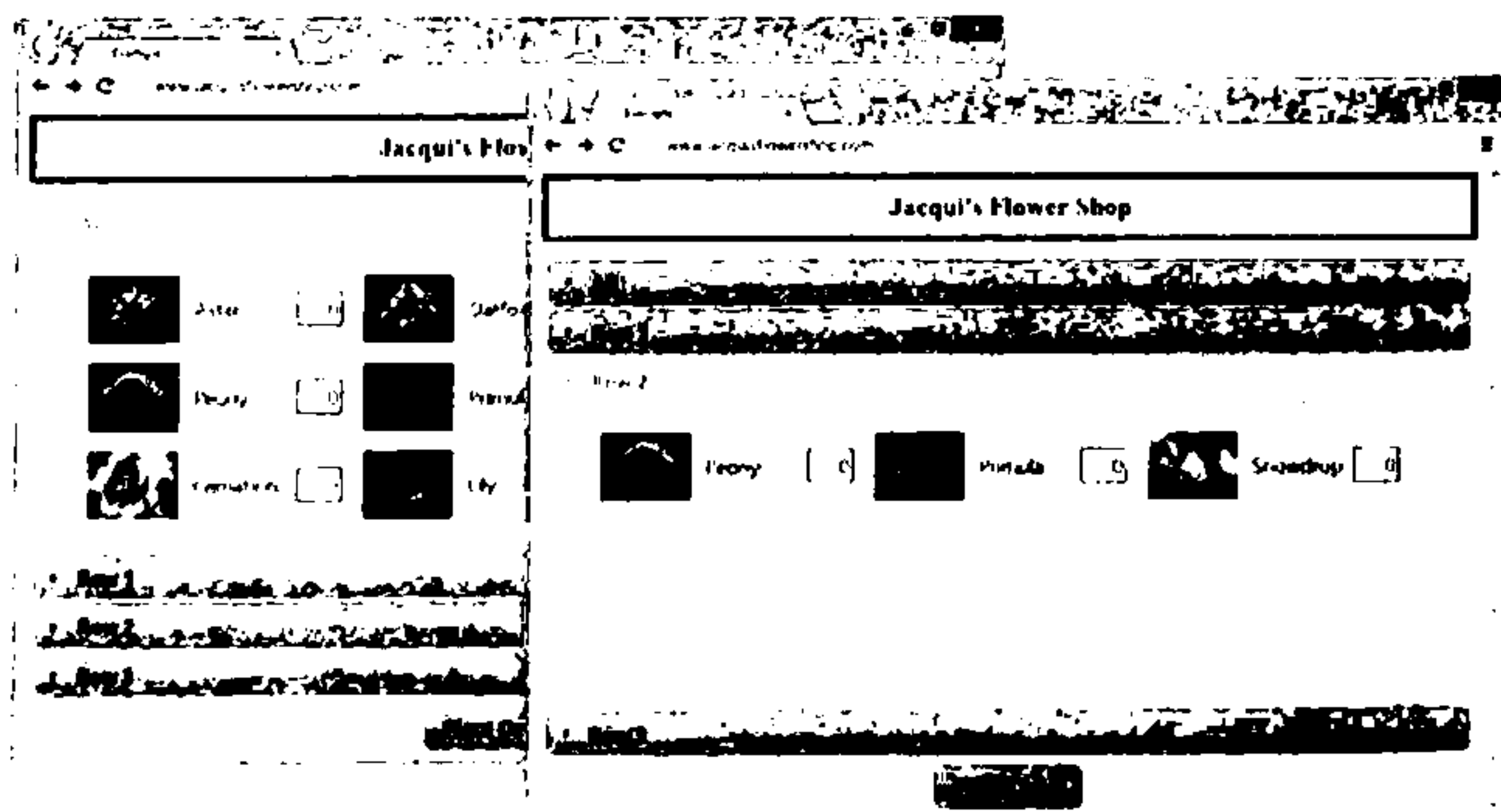


图19-13 高度差别过大时使用auto特性带来的问题

如果大量的空白不符合应用程序的风格，可以将heightStyle属性改为content（参见代码清单19-15）。

代码清单19-15 改变heightStyle设置

```
...
$("#accordion").accordion({
    heightStyle: "content"
});
...
```

如此一来，折叠菜单组件的高度会根据各部分内容的高度进行调整。图19-14展示了这种效果。

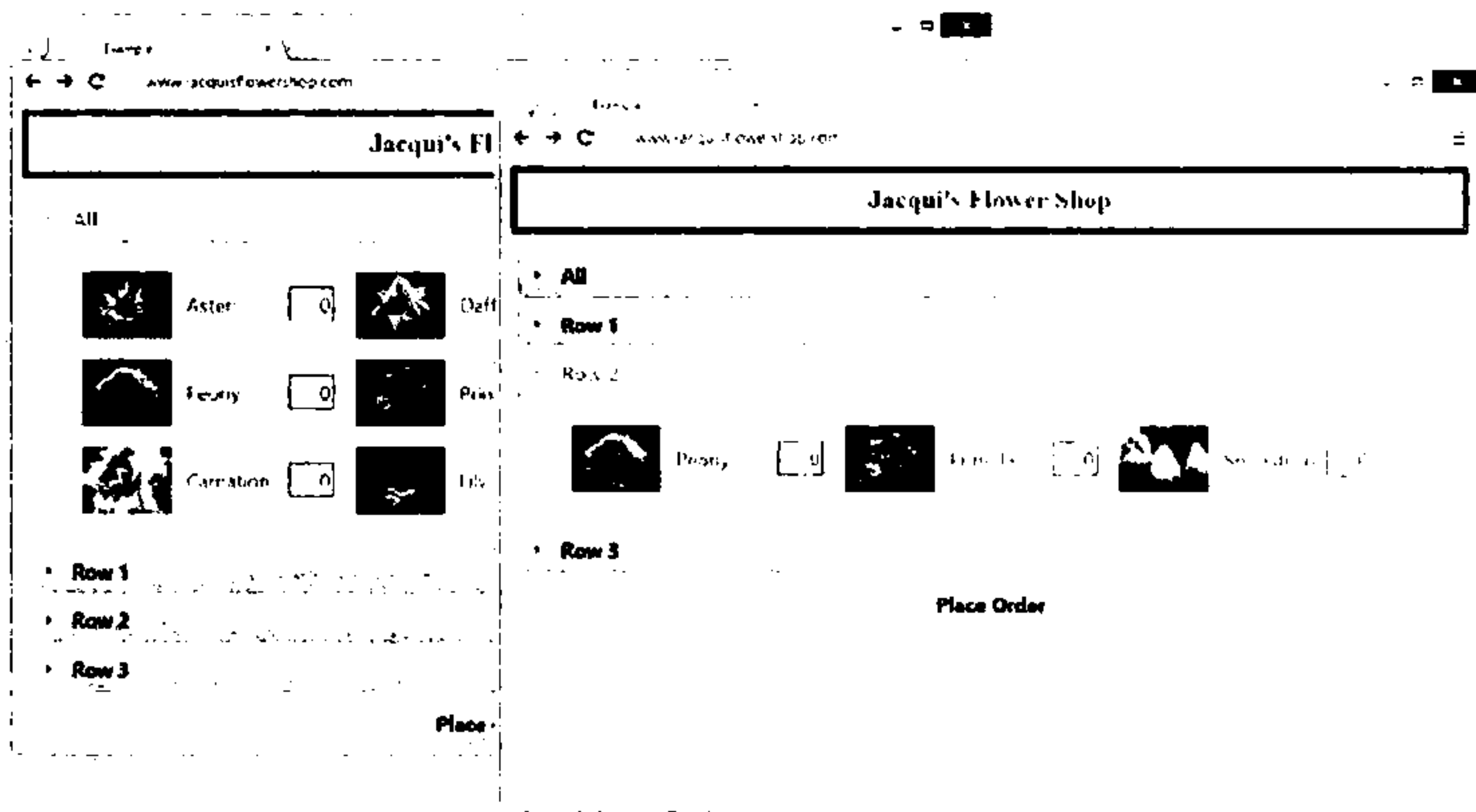


图19-14 折叠菜单组件根据内容块高度自动调整自身高度

采用这种方式显示内容非常巧妙，不过确实也会导致页面布局随着折叠菜单的高度变化而变化。这有可能会让用户感到困扰，尤其是当关键控件在屏幕上的位置变来变去时。

## 2. 利用父元素设置折叠菜单高度

一种完全不同的做法是设定折叠菜单的高度与其父元素相同。如果我们处理的内容是动态生成的（尺寸不一），因为不希望页面布局被破坏，又无法更好地处理折叠菜单高度，这种做法最合适。如代码清单19-16所示，我们可以利用fill选项达到这一目的。

代码清单19-16 设置折叠菜单的高度与父元素相同

```
...
<script type="text/javascript">
    $(document).ready(function () {

        var data = [{ "name": "Aster", "product": "aster" },
                    { "name": "Daffodil", "product": "daffodil" },
                    { "name": "Rose", "product": "rose" },
                    { "name": "Peony", "product": "peony" },
```

```

        { "name": "Primula", "product": "primula" },
        { "name": "Snowdrop", "product": "snowdrop" },
        { "name": "Carnation", "product": "carnation" },
        { "name": "Lily", "product": "lily" },
        { "name": "Orchid", "product": "orchid" } ]
    });

    var elems = $("#flowerTpl").template(data).filter("*");
    elems.slice(0, 3).appendTo("#row1");
    elems.slice(3, 6).appendTo("#row2");
    elems.slice(6).appendTo("#row3");

    $("<h2><a href=#>All</a></h2><div id=row0></div>").prependTo("#accordion")
        .filter("div").append($("#row1, #row2, #row3").clone());

    $("#accordion").wrap("<div style='height:300px'></div>");

    $("#accordion").accordion({
        heightStyle: "fill"
    });

    $("button").button();
});
</script>
...

```

在这个例子中，我使用一个新的div元素包住div#accordion元素，并把这个新元素的高度设置为300 px。调用accordion方法时，我把heightStyle选项设置为true。如果父元素的高度小于内容元素，就会出现滚动条。如果父元素的高度大于内容元素，就会多一些空白。图19-15展示的是一个带滚动条的折叠菜单。之所以出现滚动条，这是因为显示所有花卉的内容块高度超过了300 px（父元素高度）。

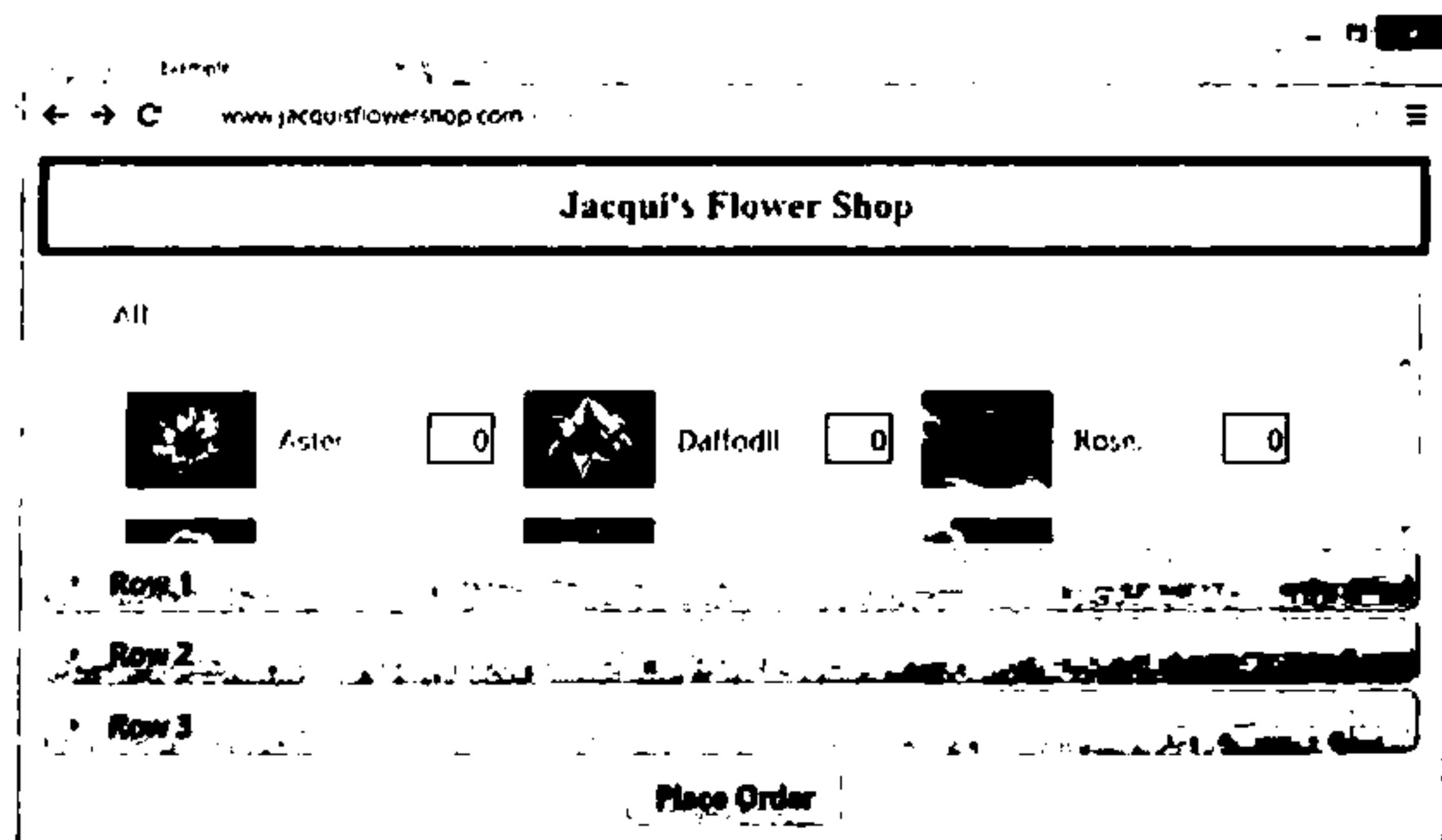


图19-15 让折叠菜单的高度适应父元素高度

### 3. 改变事件类型

默认情况下，用户通过点击标题来打开或者关闭一块内容。如代码清单19-17所示，我们可以利用event选项改变这一行为。

代码清单19-17 使用event选项

```

...
<script type="text/javascript">
    $(document).ready(function () {

        var data = [{ "name": "Aster", "product": "aster" },
            { "name": "Daffodil", "product": "daffodil" },
            { "name": "Rose", "product": "rose" },
            { "name": "Peony", "product": "peony" },
            { "name": "Primula", "product": "primula" },
            { "name": "Snowdrop", "product": "snowdrop" },
            { "name": "Carnation", "product": "carnation" },
            { "name": "Lily", "product": "lily" },
            { "name": "Orchid", "product": "orchid" } ]
        };

        var elems = $("#flowerTpl").template(data).filter("*");
        elems.slice(0, 3).appendTo("#row1");
        elems.slice(3, 6).appendTo("#row2");
        elems.slice(6).appendTo("#row3");

        $("#accordion").accordion({
            event: "mouseover"
        });

        $("button").button();
    });
</script>
...

```

在这个例子里，我利用event选项把激活内容的事件改为mouseover（详见第9章）。现在只要鼠标指针进入标题区域，jQuery UI就会立即展开该标题对应的内容区。我没法使用屏幕截图来展示这个效果，因此你最好能在浏览器中实际运行一下这个例子。这确实挺巧妙，不过我建议还是不要这么做。用户已经习惯点击展开图标以打开内容区，这种不必要的改变只会让用户惊讶和不安。

#### 4. 指定默认展开的内容块

折叠菜单组件的默认行为是展开第一个内容块。利用active选项可以改变这一行为。如代码清单19-18所示，可以把active设置为我们希望打开的内容元素的索引。

代码清单19-18 指定默认展开的内容块

```

...
<script type="text/javascript">
    $(document).ready(function () {

        var data = [{ "name": "Aster", "product": "aster" },
            { "name": "Daffodil", "product": "daffodil" },
            { "name": "Rose", "product": "rose" },
            { "name": "Peony", "product": "peony" },
            { "name": "Primula", "product": "primula" },
            { "name": "Snowdrop", "product": "snowdrop" },
            { "name": "Carnation", "product": "carnation" },

```



```

        { "name": "Lily", "product": "lily" },
        { "name": "Orchid", "product": "orchid" }}
    ];

    var elems = $("#flowerTpl").template(data).filter("*");
    elems.slice(0, 3).appendTo("#row1");
    elems.slice(3, 6).appendTo("#row2");
    elems.slice(6).appendTo("#row3");

    $("#accordion").accordion({
        active: 1
    });

    $("button").button();
});
</script>
...

```

如图19-16所示，折叠菜单现在初始打开的是index值为1对应的行（索引从0开始计数，因此索引值1即第二个内容面板）。

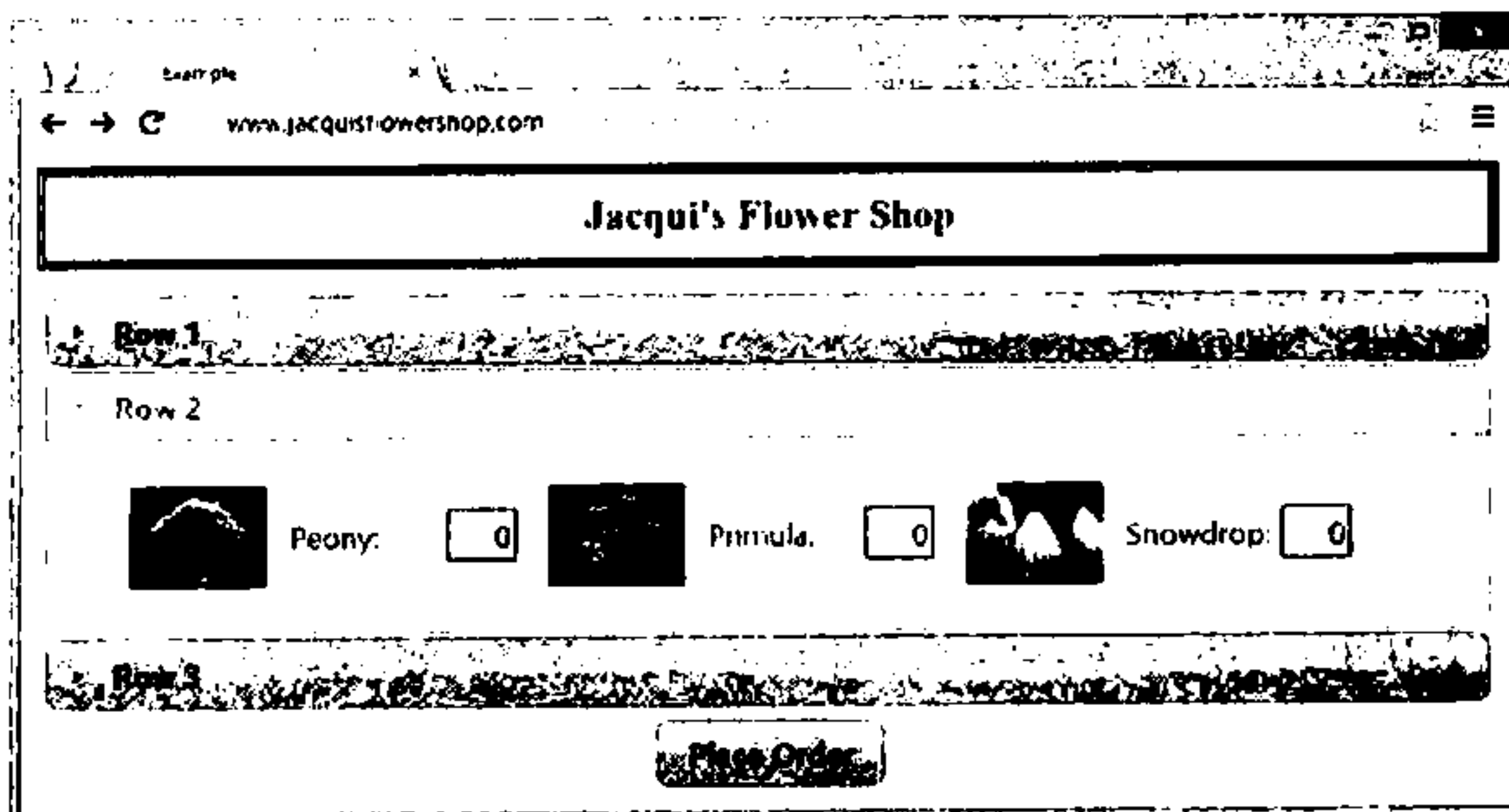


图19-16 指定默认展开的内容块

我们也可以把active选项设置为false，这样默认就不会展开任何内容。如果你真打算这么做，那就必须把collapsible选项设置为true。这样就改变了组件默认的至少显示一个内容块的策略。代码清单19-19展示了这两个选项的应用。

#### 代码清单19-19 禁用默认展开一个内容块的行为

```

...
$("#accordion").accordion({
    active: false,
    collapsible: true
});
...

```

以上代码产生的效果见图19-17。

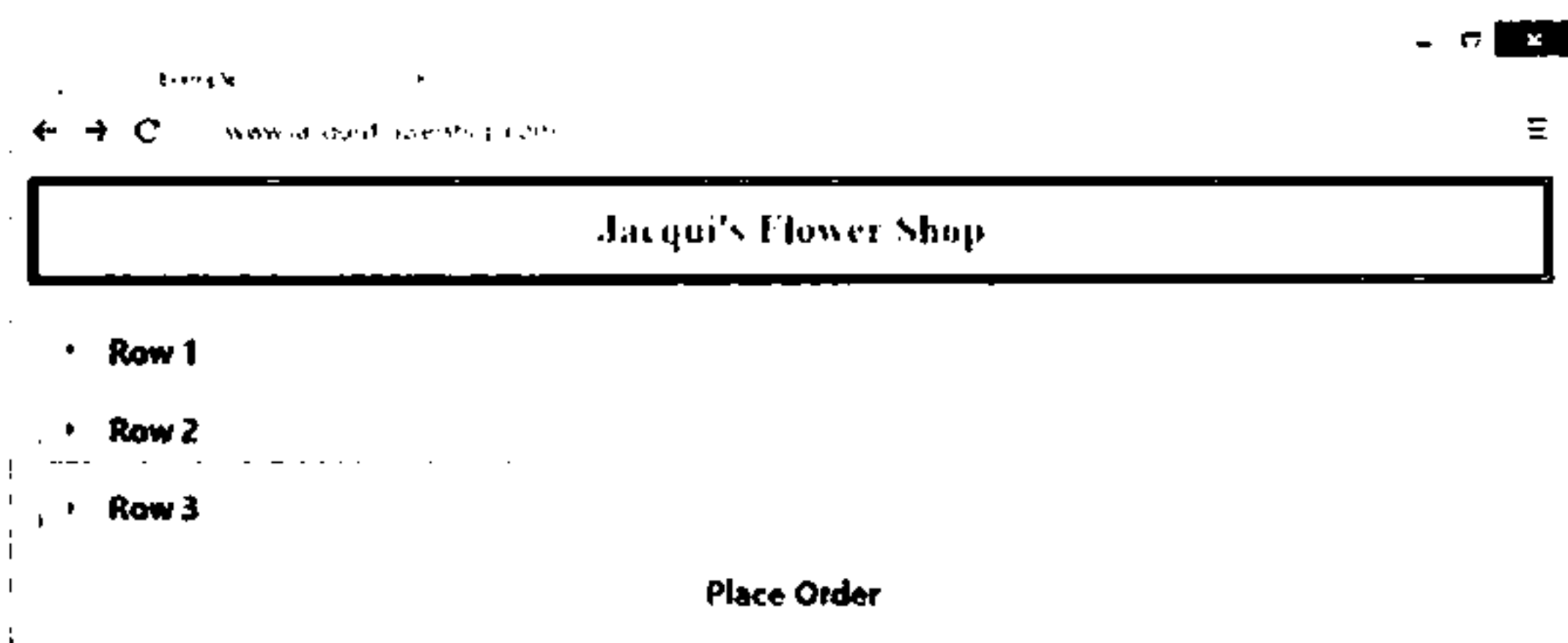


图19-17 完全折叠的折叠菜单（没有一个内容块是展开状态）

折叠菜单仍然照常工作，只是默认不会展开任何内容，所有的内容块都是折叠状态。在显示区域有限并且折叠菜单的内容并不那么重要的场合，这一技术非常有用。

### 5. 改变折叠菜单使用的图标

可以利用icons选项改变折叠菜单标题上使用的图标，具体例子见代码清单19-20

代码清单19-20 改变折叠菜单使用的图标

```
...
$("#accordion").accordion({
    collapsible: true,
    icons: {
        header: "ui-icon-zoomin",
        activeHeader: "ui-icon-zoomout"
    }
});
...
```

**提示** 在jQuery UI 1.10中，headerSelected属性被activeHeader属性代替。

icons选项的值是一个有着header和activeHeader两个属性的对象。header属性指定内容块折叠时使用的图示，而activeHeader属性指定的是内容块展开时使用的图标。我倾向于同时使用这个选项和collapsible选项，当使用图标提示用户点击能放大（或者缩小）的时候，这样显得特别自然。例子中指定图标的具体效果见图19-18。

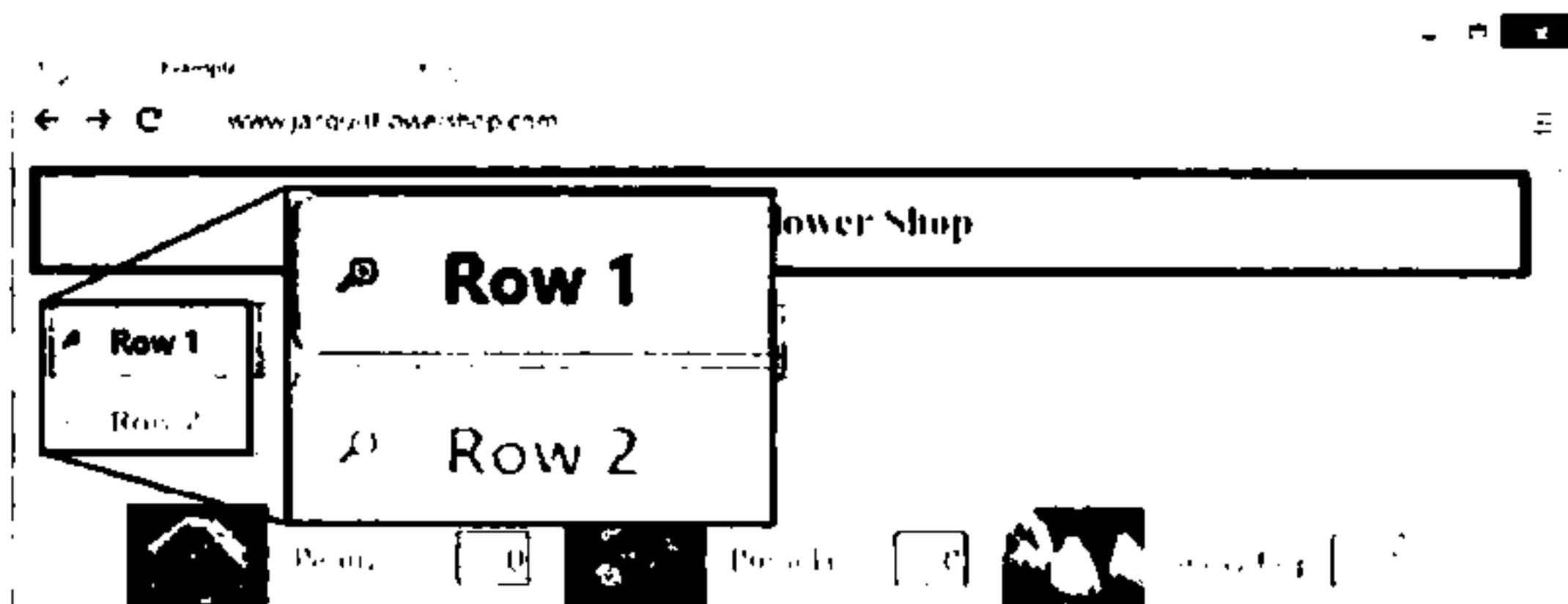


图19-18 在折叠菜单标题中使用自定义图标

### 19.2.3 折叠菜单方法

jQuery UI折叠菜单组件也定义了一些方法，参见表19-9。

表19-9 accordion方法

方 法	描 述
accordion("destroy")	移除组件功能，还原代码本来面目
accordion("disable")	禁用组件功能
accordion("enable")	启用组件功能
accordion("option")	设置组件选项
accordion("refresh")	刷新组件的尺寸

**提示** jQuery UI 1.10删除了activate方法，上一节中介绍的active选项可替代此方法。原resize方法被refresh方法替代。

refresh方法更新折叠菜单的尺寸以反映内容元素的变化，上一节中介绍过的heightStyle选项决定尺寸如何变化。其他的方法都是jQuery UI通用方法，适用于所有的jQuery UI组件。

**提示** jQuery UI 1.10中，调用refresh方法也会更新整套面板以反映底层元素的变化，允许添加或者删除元素。

### 19.2.4 折叠菜单组件事件

表19-10列出了jQuery UI折叠菜单组件定义的3个事件。

表19-10 折叠菜单组件事件

事 件	描 述
activate	内容面板被激活时触发此事件
beforeActivate	内容面板即将激活前触发此事件
create	组件创建成功之后触发此事件

**提示** jQuery UI 1.10修订了原折叠菜单组件定义的事件。原changestart事件被beforeActivate事件代替，而change事件被activate事件代替。与旧事件相比，传递给事件处理函数的ui对象的属性名称（见表19-11）也发生了变化。

表19-11 用于change和changestart事件中的ui对象定义的属性

属 性	描 述
newHeader	新激活内容块元素对应的标题元素

(续)

属 性	描 述
oldHeader	上一个激活内容块元素对应的标题元素
newPanel	新激活内容块元素
oldPanel	上一个激活内容块元素

如代码清单19-21所示，利用beforeActivate和active事件，我们可以监控内容元素的变化。

代码清单19-21 使用change事件

```
...
<script type="text/javascript">
    $(document).ready(function () {

        var data = [{ "name": "Aster", "product": "aster" },
            { "name": "Daffodil", "product": "daffodil" },
            { "name": "Rose", "product": "rose" },
            { "name": "Peony", "product": "peony" },
            { "name": "Primula", "product": "primula" },
            { "name": "Snowdrop", "product": "snowdrop" },
            { "name": "Carnation", "product": "carnation" },
            { "name": "Lily", "product": "lily" },
            { "name": "Orchid", "product": "orchid" } ]
        };

        var elems = $("#flowerTpl").template(data).filter("*");
        elems.slice(0, 3).appendTo("#row1");
        elems.slice(3, 6).appendTo("#row2");
        elems.slice(6).appendTo("#row3");

        $("#accordion").accordion({
            active: false,
            collapsible: true,
            activate: handleAccordionChange
        });

        function handleAccordionChange(event, ui) {
            if (ui.oldHeader.length) {
                console.log("Old header: " + ui.oldHeader[0].innerText);
            }
            if (ui.newHeader.length) {
                console.log("New header: " + ui.newHeader[0].innerText);
            }
        }

        $("button").button()
    });
</script>
...
```

我使用activate事件响应内容元素的变化。与自动完成组件类似，jQuery UI使用附加参数传递信息给事件处理函数。这个附加参数通常使用ui这个名字，它定义的属性见表19-11。

这些属性的值都是数组，所以我要先确定它的length属性大于0，才可以得到第一个HTMLElement对象，并把它的innerText属性输出到控制台。

## 19.3 小结

本章介绍了jQuery UI自动完成组件和折叠菜单组件。我们注意到，虽然这些组件都遵循同样的模式，但提供了更丰富的功能和更多的配置选项。这些选项便于我们定制组件以满足应用程序的需要。在下一章，我们将了解标签组件。

猛一看，标签组（Tab）件与我们在第19章中讲到的折叠菜单组件很相似。然而，它的功能比折叠菜单要强大得多，而且定制功能也更加强大。与前面介绍jQuery UI组件的几章一样，我们首先了解如何创建组件，然后是组件支持的选项、方法和事件。在本章的末尾，我准备了一个使用标签组件提供分节表单的例子，这是一个处理有着许多输入框的超长表单的实用技术。表20-1列出了本章概要。

表20-1 本章概要

问 题	解决方法	代码清单
如何创建标签组件	定义标签和内容元素结构，然后调用tabs方法	1
如何使用Ajax获取标签内容	把标签标题中a元素的href属性设置为内容面板的数据来源URL	2、3
如何获取或改变活跃标签	使用active选项	4
如何禁用某些标签	使用disabled选项	5
如何改变激活标签的事件	使用event选项	6
如何允许所有标签均可折叠	使用collapsible选项	7
如何添加或者删除标签	修改底层HTML元素，然后调用refresh方法	8
如何强制载入远程标签的内容	使用load方法	9
在发起Ajax请求之前如何配置Ajax请求，以及如何修改从远程服务器载入的内容	处理beforeLoad和load事件	10、11
如果让一个表单显示在多个标签中	把表单拆分到多个div元素中，并增加标题结构，然后调用tabs方法	12~14
如何验证显示在多个标签中的表单内容	处理beforeActivate和activate事件	15

## 新版jQuery UI中与本章有关的变化

在jQuery UI 1.10中，标签组件的底层API发生了巨大的变化。它的方法和配置选项变少了，与其他组件更加一致，可以更放心地在需要修改时直接修改底层元素。

fx、ajaxOptions、cache、spinner、selected、idPrefix、tabTemplate、panelTemplate和cookie选项被删除；url、abort、select、add、remove和length方法也被删除。标签组件事件也大幅简化。有关新事件的详细信息，可参阅20.5节。

标签组件发生的变化略显激进，却更加简单易用。先使用jQuery处理底层HTML元素，再调用新的refresh方法更新组件状态的工作方式，所有组件方法和选项均可使用，具体可见本章示例代码。

## 20.1 创建标签

我们使用tabs方法创建jQuery UI标签组件。与折叠菜单组件有点相似，标签组件也需要专门的HTML结构来实现，参见代码清单20-1。

20

代码清单20-1 创建jQuery UI标签

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script id="flowerTpl" type="text/x-jquery-tmpl">
    {{#flowers}}
    <div class="dcell">
      
      <label for="{{product}}">{{name}}:</label>
      <input name="{{product}}" value="0" />
    </div>
    {{/flowers}}
  </script>
  <script type="text/javascript">
    $(document).ready(function () {
      var data = {
        flowers: [{ "name": "Aster", "product": "aster" },
          { "name": "Daffodil", "product": "daffodil" },
          { "name": "Rose", "product": "rose" },
          { "name": "Peony", "product": "peony" },
          { "name": "Primula", "product": "primula" },
          { "name": "Snowdrop", "product": "snowdrop" },
          { "name": "Carnation", "product": "carnation" },
          { "name": "Lily", "product": "lily" },
          { "name": "Orchid", "product": "orchid" } ]
      };

      var elems = $("#flowerTpl").template(data).filter("*");
      elems.slice(0, 3).appendTo("#tab1");
      elems.slice(3, 6).appendTo("#tab2");
      elems.slice(6).appendTo("#tab3");

      $("#tabs").tabs();
      $("#button").button();
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
```

```

    <div id="tabs">
      <ul>
        <li><a href="#tab1">Row 1</a>
        <li><a href="#tab2">Row 2</a>
        <li><a href="#tab3">Row 3</a>
      </ul>
      <div id="tab1"></div>
      <div id="tab2"></div>
      <div id="tab3"></div>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
</html>

```

标签组件要求两种类型的元素。第一种是内容元素，就是点击标签对应要显示的元素。另一种是结构元素，它们为jQuery UI标签组件提供必要的信息，以方便tabs方法生成相应的结构。

我们使用div元素存放内容。在下面这个例子中，我使用了3个div元素，就像前面几章中的例子一样，每个div元素里存放着一行花卉产品信息。

```

...
<div id="tab1"></div>
<div id="tab2"></div>
<div id="tab3"></div>
...

```

每个内容元素都必须有一个id属性，这很重要。只有这样，jQuery UI的标签组件才可以正确地找到需要显示的内容。对于结构元素，我们使用的是li元素，每个li中必须有一个a元素，就像下面这样。

```

...
<ul>
  <li><a href="#tab1">Row 1</a>
  <li><a href="#tab2">Row 2</a>
  <li><a href="#tab3">Row 3</a>
</ul>
...

```

li元素的数目定义标签的数量，a元素的内容将用作标签文本，而href属性定义与该标签相关的内容元素。

---

**提示** 我使用了数据模板插件动态生成标签内容，因为这样能更清楚地表达需要的数据结构。内容元素当然可以定义成静态元素，或者如我在下一节中所要介绍的，从服务器上动态地获取。

---

图20-1展示了例子中的结构转换成一组标签组件的样子。

标签是用户熟悉的界面隐喻。点击一个标签，jQuery UI就显示和它相关的内容元素。类似折叠菜单组件，使用标签组件一样能够在相对紧凑的空间里显示大量内容，而且用户能够专注地查看他们认为重要的部分。这意味着我们必须谨慎地安排各标签内容之间的关系。我们的目标是对内容分组，让用户更容易找到自己关心的内容（切换标签的次数尽可能得少），同时保留内容元素的自然分组。和



其他任何用户界面一样，这要求对用户要完成的任务（用户需求）和他们的工作流（而不是系统的工作流）拥有深刻的理解。

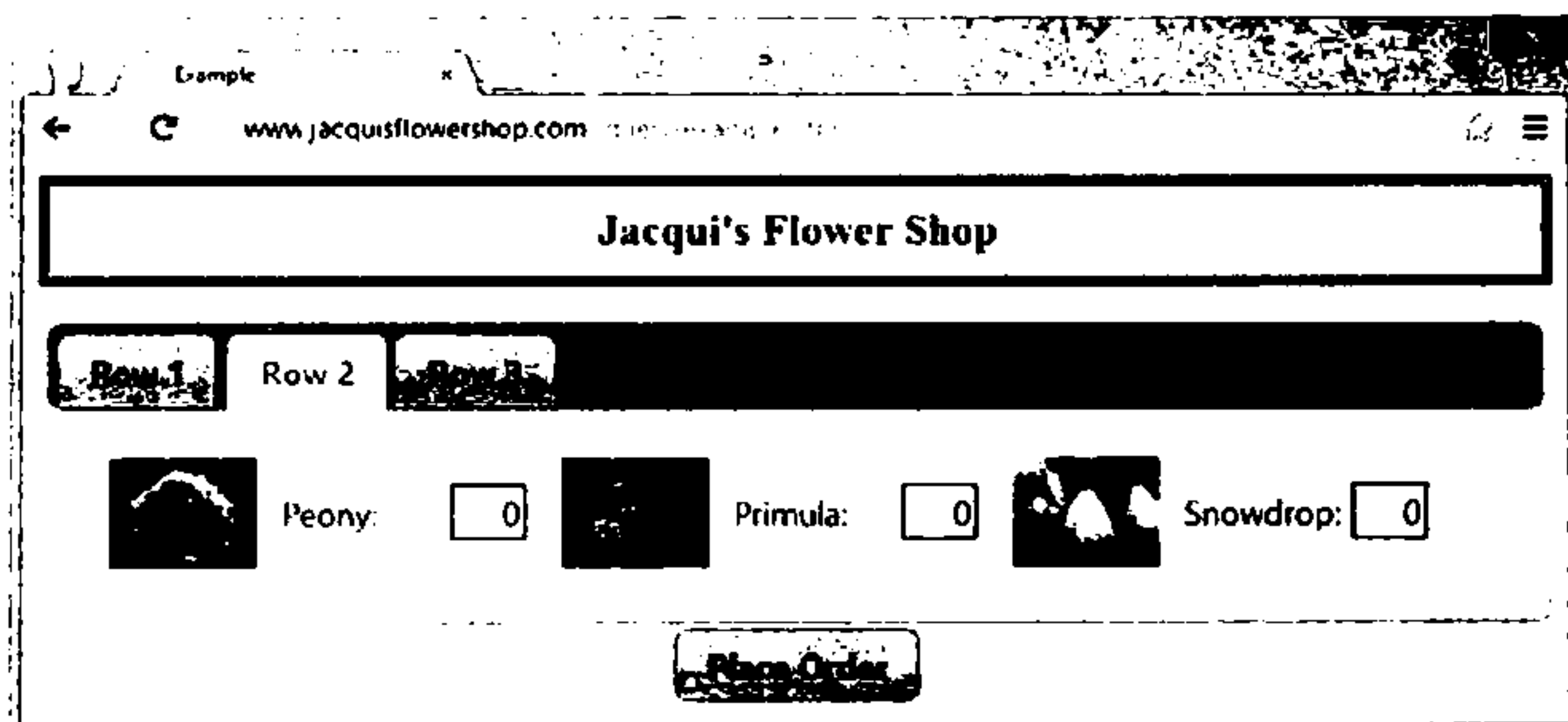


图20-1 创建jQuery UI标签

## 20.2 使用 Ajax 获取标签内容

标签组件一个不错的功能就是能够使用Ajax获取标签的内容。要实现这一点，只需把a元素的href属性指定为获取相应内容的URL。使用Ajax获取内容的标签又称远程标签。为演示这一功能，我添加了一个新文件tabflowers.html，代码清单20-2列出了它的内容。

代码清单20-2 tabflowers.html文件的内容

```
<div>
  <div class="dcell">
    <label for="aster">Aster:</label>
    <input name="aster" value="0" />
  </div>
  <div class="dcell">
    <label for="daffodil">Daffodil:</label>
    <input name="daffodil" value="0" />
  </div>
  <div class="dcell">
    <label for="rose">Rose:</label>
    <input name="rose" value="0" />
  </div>
</div>
<div>
  <div class="dcell">
    <label for="peony">Peony:</label>
    <input name="peony" value="0" />
  </div>
  <div class="dcell">
    <label for="primula">Primula:</label>
    <input name="primula" value="0" />
  </div>
</div>
```

```

</div>
<div class="dcell">
  <label for="snowdrop">Snowdrop:</label>
  <input name="snowdrop" value="0" />
</div>
</div>

```

为了让这个例子尽可能简单，我使用的结构与数据模板生成的内容一样。代码清单20-3中的这个例子演示了如何将tabflowers.html文件用作一个标签的内容。

### 代码清单20-3 使用Ajax获取标签内容

```

...
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="tabs">
      <ul>
        <li><a href="tabflowers.html">Ajax Content</a>
        <li><a href="#tab1">Row 1</a>
        <li><a href="#tab2">Row 2</a>
        <li><a href="#tab3">Row 3</a>
      </ul>
      <div id="tab1"></div>
      <div id="tab2"></div>
      <div id="tab3"></div>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
...

```

在这段脚本中，我们新加了一个名为Ajax Content的标签并指定了它的数据来源URL。这个例子的效果见图20-2。

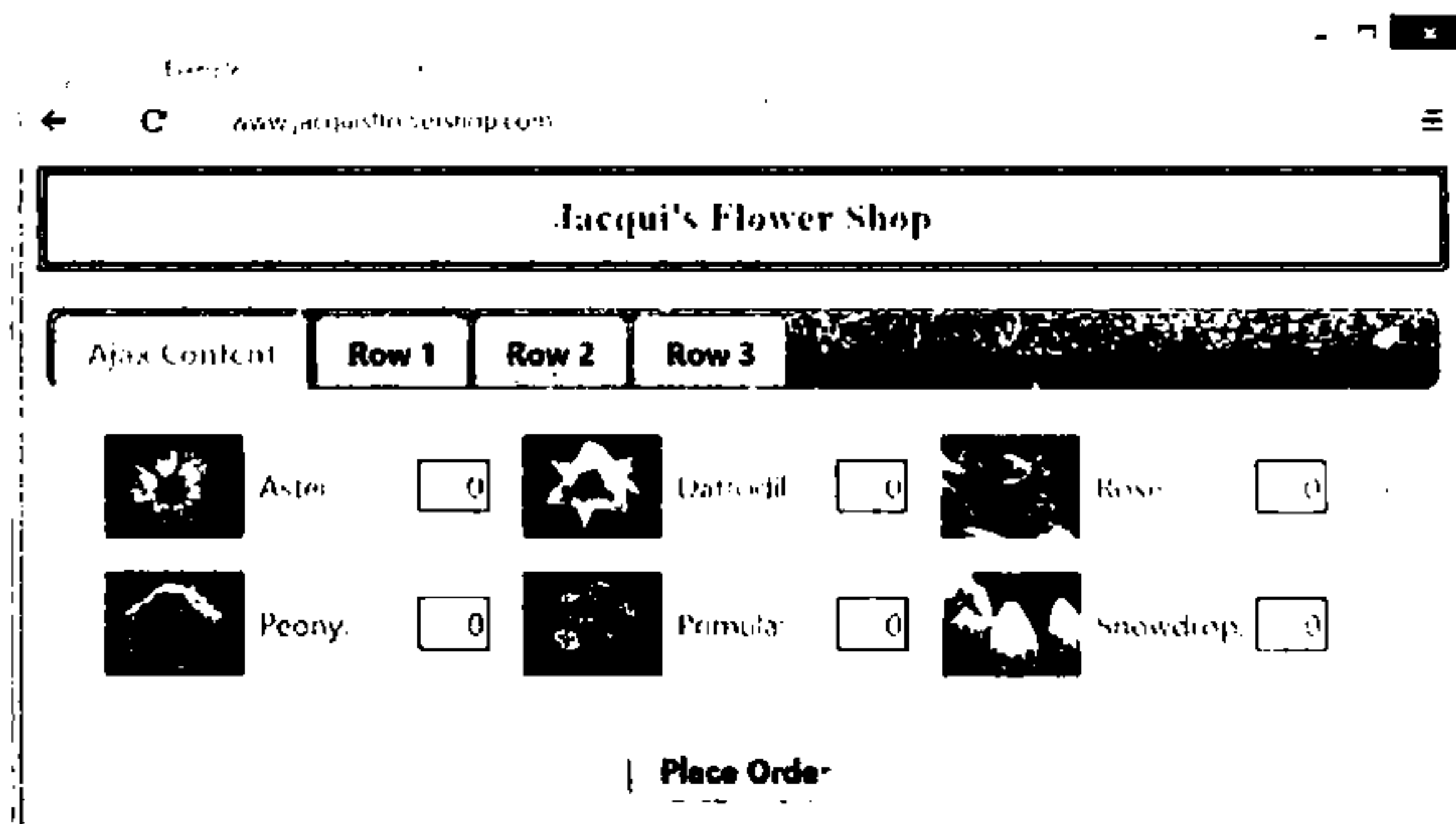


图20-2 使用Ajax获取标签内容

提示 不必手动为远程标签创建内容元素，标签组件会自动打理好这一切。

## 20.3 配置标签组件

乍一看，标签组件就像我们在第19章中讲过的垂直折叠菜单的变体。它们确实有一些共同的外观特征，不过标签组件支持更多的配置选项。表20-2列出了jQuery UI标签组件支持的选项。在接下来的几节中，我将着重介绍如何使用这些配置选项。

表20-2 标签组件选项

选 项	描 述
active	获取或设置当前显示标签，使用从零起始的编号指定面板。若设置为false，则所有面板均呈关闭状态（这一特性仅在collapsible选项设置为true时有效）
collapsible	若设置为true，则所有标签均呈关闭状态
disabled	用于启用 / 禁用单个标签
event	用来设置标签切换动画的事件
heightStyle	指定如何确定组件及诸标签高度
hide	指定标签关闭时的动画，关于jQuery UI动画详见本书第35章
show	指定标签展开时的动画，关于jQuery UI动画详见本书第35章

提示 jQuery 1.10 大幅修改了标签组件的配置选项。要详细了解变更细节，可阅读本章开头的提示部分。表20-2列出的这些新选项更加简单，与其他jQuery UI组件也更加一致。

### 20.3.1 选取活跃标签

如代码清单20-4所示，active选项可用来确定当前显示的是什么标签，并改变当前标签。

代码清单20-4 使用active选项获取及改变当前显示标签

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style>
    #radioDiv { text-align: center; margin-top: 10px;}
  </style>

  <script id="flowerTpl" type="text/x-jquery-tmpl">
```

```

    {{#flowers}}
    <div class="dcell">
        
        <label for="{{product}}">{{name}}:</label>
        <input name="{{product}}" value="0" />
    </div>
    {{/flowers}}
</script>
<script type="text/javascript">
    $(document).ready(function () {
        var data = {
            flowers: [{ "name": "Aster", "product": "aster" },
                { "name": "Daffodil", "product": "daffodil" },
                { "name": "Rose", "product": "rose" },
                { "name": "Peony", "product": "peony" },
                { "name": "Primula", "product": "primula" },
                { "name": "Snowdrop", "product": "snowdrop" },
                { "name": "Carnation", "product": "carnation" },
                { "name": "Lily", "product": "lily" },
                { "name": "Orchid", "product": "orchid" } ]
        };
        var elems = $("#flowerTpl").template(data).filter("*");
        elems.slice(0, 3).appendTo("#tab1");
        elems.slice(3, 6).appendTo("#tab2");
        elems.slice(6).appendTo("#tab3");

        $("#tabs").tabs();

        $("#radioDiv").buttonset().change(function (e) {
            $("#tabs").tabs("option", "active", e.target.value);
        });
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="tabs">
            <ul>
                <li><a href="tabflowers.html">Ajax Content</a>
                <li><a href="#tab1">Row 1</a>
                <li><a href="#tab2">Row 2</a>
                <li><a href="#tab3">Row 3</a>
            </ul>
            <div id="tab1"></div>
            <div id="tab2"></div>
            <div id="tab3"></div>
        </div>
        <div id="radioDiv">
            <input type="radio" name="tabNo" id="one" value="1" />
            <label for="one">1</label>
            <input type="radio" name="tabNo" id="two" value="2"/>
            <label for="two">2</label>
            <input type="radio" name="tabNo" id="three" value="3"/>

```

```

        <label for="three">3</label>
    </div>
</form>
</body>
</html>

```

我在页面中添加了一排jQuery UI按钮（参见第18章），然后调用change方法注册了change事件处理函数。这样只要点击这排按钮中的任何一个，都会调用这个处理函数。该处理函数使用option方法设定active属性，造就图20-3所示的效果。

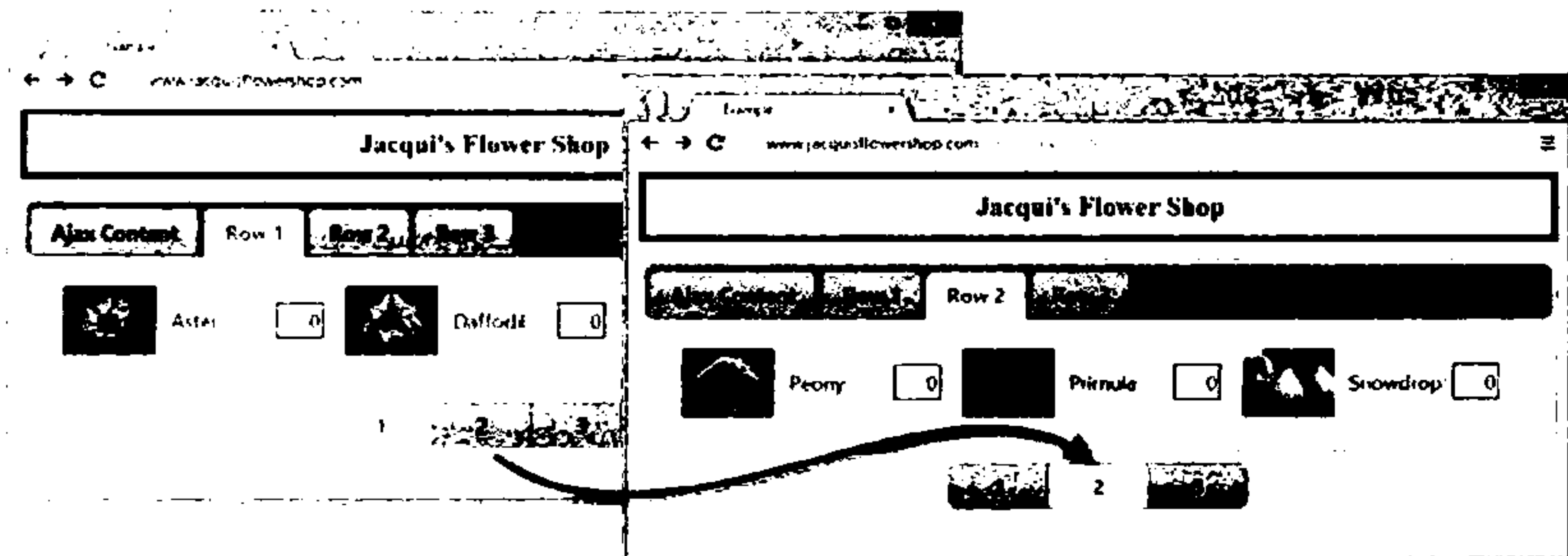


图20-3 使用active选项改变活跃标签

## 20.3.2 禁用某些标签

把disabled选项设置成布尔值true或者false，可以禁用或者启用标签组件。不过如果把它设置成一个由数字值组成的数组，也可以用来启用或禁用某些标签，如代码清单20-5所示。

### 代码清单20-5 禁用部分标签

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <style type="text/css">
        #buttonDiv {margin: 5px}
    </style>
    <script type="text/javascript">
        $(document).ready(function () {

            $("#tabs").tabs();

            $("input:checkbox").button().click(function () {
                var disabledPositions = [];

```

```

        $("input:checked").each(function (index, elem) {
            disabledPositions.push(Number(elem.value));
        })

        $("#tabs").tabs("option", "disabled", disabledPositions)
    });
});
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="tabs">
            <ul>
                <li><a href="#tab1">Tab 1</a>
                <li><a href="#tab2">Tab 2</a>
                <li><a href="#tab3">Tab 3</a>
            </ul>
            <div id="tab1">This is the content for tab 1</div>
            <div id="tab2">This is the content for tab 2</div>
            <div id="tab3">This is the content for tab 3</div>
        </div>
        <div id="buttonDiv">
            <label for="cb0">Tab 1</label><input type="checkbox" id="cb0" value=0>
            <label for="cb1">Tab 2</label><input type="checkbox" id="cb1" value="1">
            <label for="cb2">Tab 3</label><input type="checkbox" id="cb2" value="2">
        </div>
    </form>
</body>
</html>

```

我使用静态内容创建了几个标签，添加了几个复选框并把它们转换为jQuery UI反转按钮。点击某个按钮，则按钮对应的标签就会被禁用或者启用（如图20-4所示）。

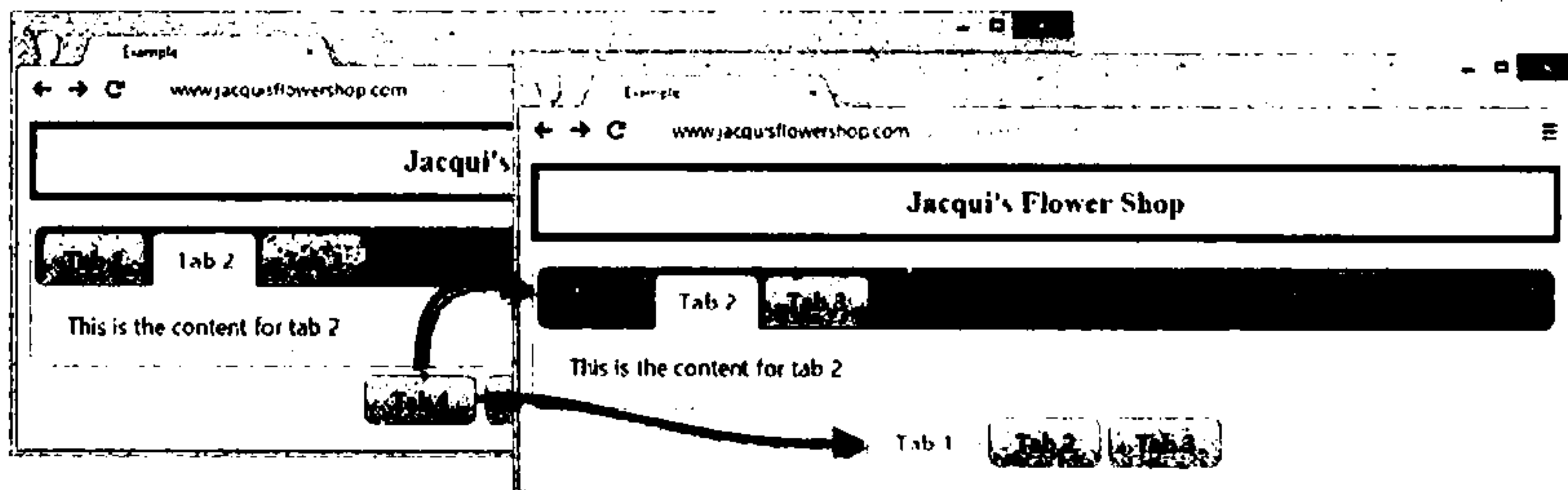


图20-4 点击反转按钮禁用或者启用对应的标签

### 20.3.3 改变激活标签的事件

标签组件默认使用click事件激活，也就是说用户必须点击标题才能激活一个标签。我们可以使

用event选项来指定使用别的事件激活标签。如代码清单20-6所示，如果希望鼠标移上去就激活标签，这个选项就太有用了。

代码清单20-6 改变激活标签的事件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css"> #buttonDiv {margin: 5px}</style>
  <script type="text/javascript">
    $(document).ready(function() {

      $("#tabs").tabs({
        event: "mouseover"
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="tabs">
      <ul>
        <li><a href="#tab1">Tab 1</a>
        <li><a href="#tab2">Tab 2</a>
        <li><a href="#tab3">Tab 3</a>
      </ul>
      <div id="tab1">This is the content for tab 1</div>
      <div id="tab2">This is the content for tab 2</div>
      <div id="tab3">This is the content for tab 3</div>
    </div>
  </form>
</body>
</html>
```

在这个例子里，我指定使用mouseover事件激活标签。也就是说，只要鼠标悬念在标签的标题上就会激活这个标签。

---

**提示** 我建议尽量不要这么做，前面折叠菜单组件那章已经说过这个观点。这么做视觉效果也许很炫，但它实际上很烦人。用户不得不小心，以避免把鼠标移出相关标签的标题之外。

---

## 20.3.4 可折叠的标签

如代码清单20-7所示，我们可以利用collapsible选项制作出一种介于折叠菜单与标签组件之间的“混血儿”。

## 代码清单20-7 使用collapsible选项

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $("#tabs").tabs({
            collapsible: true
        });
    });
</script>
...

```

当我们把collapsible选项设置为true时，点击当前激活菜单的标题就会把它折叠起来（如图20-5所示）。

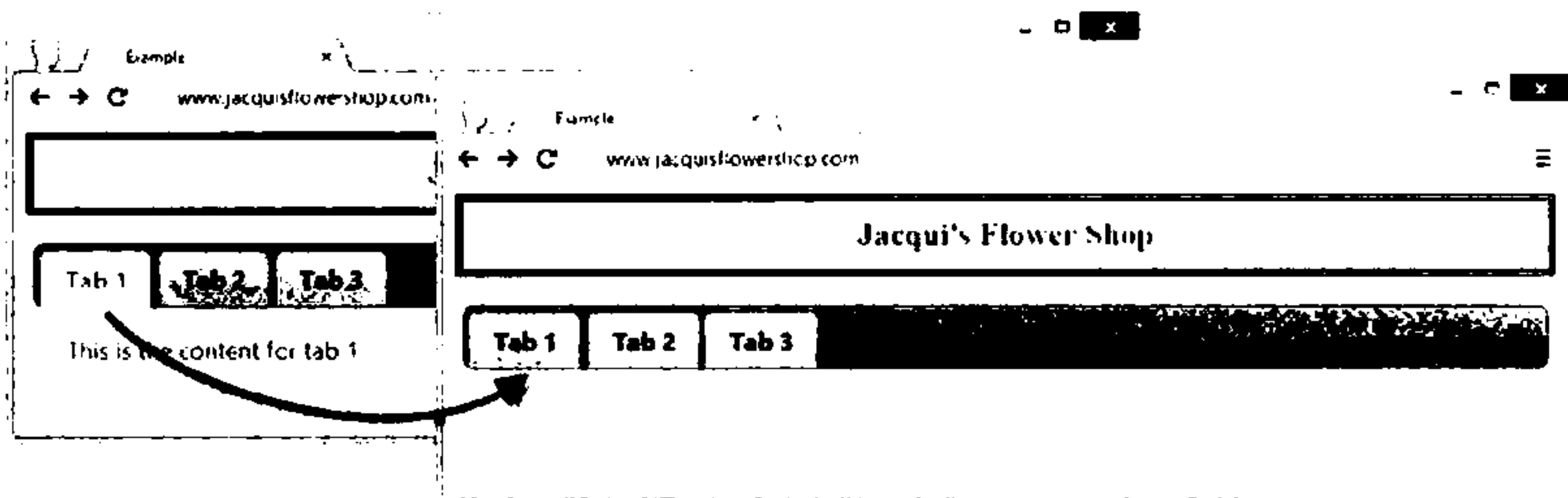


图20-5 收起当前活跃的标签

**提示** 仅仅为了使内容完整，我才在这里介绍这一选项。事实上，我从不使用这一选项，因为这一行为即不符合直观感觉又会令用户困惑。

## 20.4 标签组件方法

表20-3列出了jQuery UI标签组件支持的方法。

表20-3 tabs方法

方 法	描 述
tabs("destroy")	从底层元素上移除标签组件
tabs("disable")	禁用整个标签组件或者部分标签（可参阅20.3.4节中使用disable选项的例子）
tabs("enable")	启用整个标签组件或者部分标签
tabs("option")	修改一个或者多个选项，参阅18.1.1节了解配置jQuery UI组件的详细过程
tabs("load")	使用Ajax请求明确载入一个标签的内容
tabs("refresh")	更新组件状态以反映底层元素的变更



**提示** 在jQuery UI 1.10版本中，标签组件方法发生了一些变化。新版中删除了add、remove、select、url、length以及abort方法。其中add和remove方法被新方法refresh代替；利用active选项可以实现select方法的功能；利用新的beforeActivate事件，我们可以自己实现abort方法的功能；length方法没有替代方法。

## 20.4.1 增加或者删除标签

过去我使用jQuery UI定义的add和remove方法增加或删除标签。在jQuery 1.10中这些方法已被删除，代之以refresh方法。refresh方法依据底层元素发生的变化，刷新组件的状态。代码清单20-8展示了refresh方法的用法。

代码清单20-8 使用refresh方法更新标签组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    #buttons {margin: 5px 0}
  </style>
  <script type="text/javascript">
    $(document).ready(function () {

      $("#tabs").tabs();

      $("button").button().click(function (e) {
        var tabsElem = $("#tabs");
        if (this.id == "add") {
          var tabID = tabsElem.children("div").length + 1;
          tabsElem.children("ul").append($("<li>").append($("<a>")
            .attr("href", "#tab" + tabID).text("Tab " + tabID)));
          $("<div>").attr("id", "tab" + tabID)
            .text("This is the content for tab " + tabID).appendTo(tabsElem);
        } else {
          tabsElem.find("li").first().remove();
          tabsElem.children("div").first().remove();
        }
        tabsElem.tabs("refresh");
      })
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="buttons" class="ui-widget">
```

```

    <button id="add">Add Tab</button>
    <button id="remove">Remove Tab</button>
  </div>
  <div id="tabs">
    <ul>
      <li><a href="#tab1">Tab 1</a>
      <li><a href="#tab2">Tab 2</a>
      <li><a href="#tab3">Tab 3</a>
    </ul>
    <div id="tab1">This is the content for tab 1</div>
    <div id="tab2">This is the content for tab 2</div>
    <div id="tab3">This is the content for tab 3</div>
  </div>
</body>
</html>

```

添加一对button元素，并删除标签组件所需的li和div元素。当点击Add Tab按钮时，生成一些新元素并将它们添加到DOM。当点击Remove Tab按钮时，找出第一个li和div元素，并删除它们。

随后调用了refresh方法，注意观察标签组件随底层元素变化而发生的变化（图20-6）。

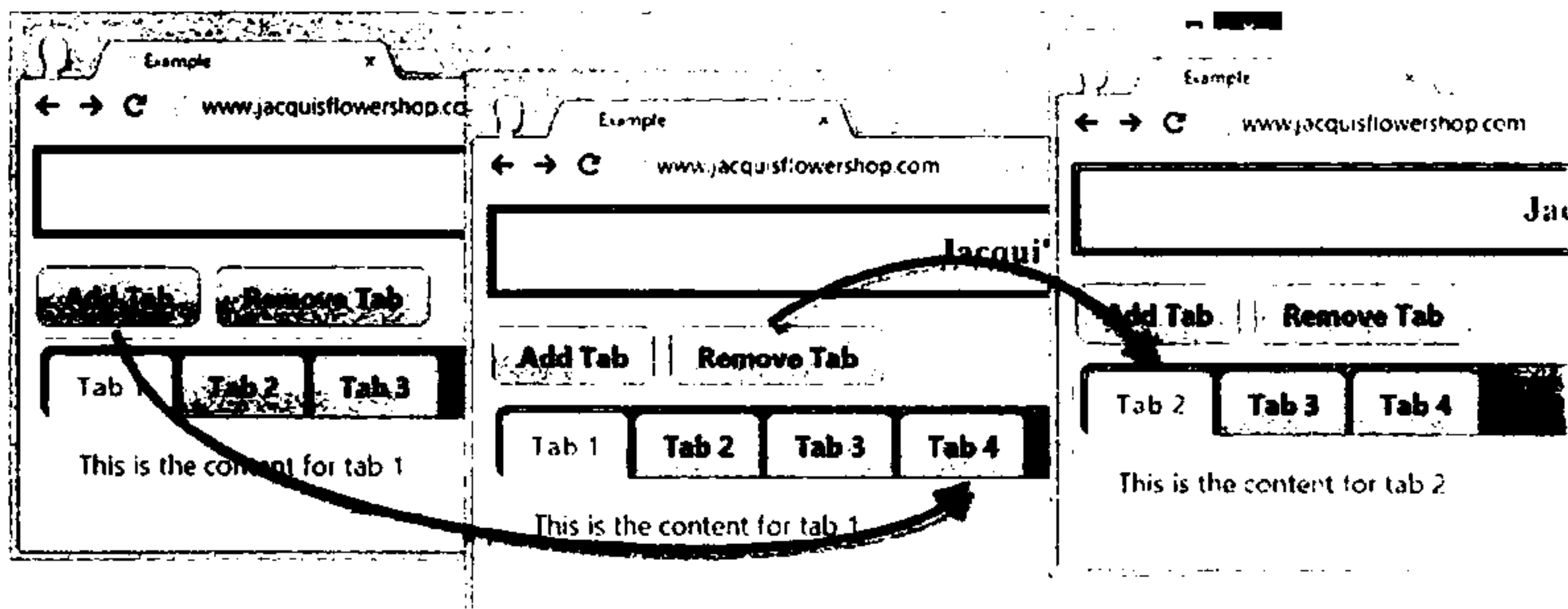


图20-6 增加或者删除标签

## 20.4.2 触发标签的Ajax请求

默认情况下，只有用户打开远程内容标签时才会发出Ajax请求。这能避免不必要的请求，但也会导致用户在打开一个标签时，内容显示延迟。可以利用load方法让远程标签主动载入内容，具体见代码清单20-9。

代码清单20-9 使用load方法明确载入远程标签的内容

```

<!DOCTYPE html>
<html>
  <head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>

```

```

<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<style type="text/css">
    #buttons {margin: 5px 0}
</style>
<script type="text/javascript">
    $(document).ready(function () {

        $("#tabs").tabs();

        $("#load").button().click(function (e) {
            var tabsElem = $("#tabs");
            tabsElem.find("a").each(function (index, elem) {
                if (elem.href.indexOf("example.html") == -1) {
                    tabsElem.tabs("load", index);
                }
            });
        });
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>

    <div id="buttons" class="ui-widget">
        <button id="load">Load</button>
    </div>

    <div id="tabs">
        <ul>
            <li><a href="#tab1">Tab 1</a>
            <li><a href="#tab2">Tab 2</a>
            <li><a href="#tab3">Tab 3</a>
            <li><a href="tabflowers.html">Ajax Content</a>
        </ul>
        <div id="tab1">This is the content for tab 1</div>
        <div id="tab2">This is the content for tab 2</div>
        <div id="tab3">This is the content for tab 3</div>
    </div>
</body>
</html>

```

load方法的参数是我们希望载入内容的远程标签的顺序号。在本例中，我在按钮按下时（click事件处理函数中）执行load方法。

## 20.5 标签组件事件

表20-4列出了jQuery UI标签组件内建的一些事件，在后面的内容中我会进行详细介绍。

表20-4 标签组件事件

事 件	描 述
create	当标签组件创建完成之后触发
beforeActivate	在标签面板显示之前触发
activate	在标签面板显示之后触发
beforeLoad	在载入远程标签的内容之前触发
load	当远程标签的内容成功载入之后触发

**提示** 在 jQuery UI 1.10 中，新版组件删除了 select、show、add、remove、enable 以及 disable 事件，添加了 beforeActivate、activate 和 beforeLoad 事件。这些被删除的事件之前饱受质疑，部分原因是标签组件现在依赖于底层元素的状态（和 refresh 方法）；而另一方面，对于这些被删掉的功能，利用新事件也能实现。

### 20.5.1 拦截 Ajax 请求

beforeLoad 事件在 Ajax 请求发出之前触发。事件处理函数会接到一个 jQuery 事件对象和一个附加对象参数——参数名通常使用 ui——这个 ui 对象定义的属性见表 20-5。

表20-5 传递给beforeLoad事件处理函数的ui参数定义的属性

名 称	描 述
tab	返回包含着远程标签元素的 jQuery 对象
panel	返回包含着远程标签面板元素的 jQuery 对象
jqXHR	返回将用于发起 Ajax 请求的 jqXHR 对象
ajaxSettings	返回将传递给 \$.ajax 方法、用于发起 Ajax 请求的参数对象

#### 代码清单20-10 包含表单的页面

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#tabs").tabs({
        beforeLoad: function (e, ui) {
          ui.ajaxSettings.url = "flowers.html";
        }
      });
    });
  </script>
```

```
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="tabs">
    <ul>
      <li><a href="#tab1">Tab 1</a>
      <li><a href="#tab2">Tab 2</a>
      <li><a href="#tab3">Tab 3</a>
      <li><a href="tabflowers.html">Ajax Content</a>
    </ul>
    <div id="tab1">This is the content for tab 1</div>
    <div id="tab2">This is the content for tab 2</div>
    <div id="tab3">This is the content for tab 3</div>
  </div>
</body>
</html>
```

在beforeLoad事件处理函数中，我改变了ui.ajaxSettings对象的url属性，从而改变了远程标签的内容来源：点击标签，不再载入tabflowers.html，而是载入flowers.html。

20.5.2 修改远程标签的数据

当远程标签需要的数据从远程服务器成功载入时，会触发load事件。在载入内容显示给用户前，我们可使用load事件修改载入内容。事件处理函数会收到两个参数：一个jQuery事件对象，一个ui对象。ui对象定义的属性见表20-6。

表20-6 load事件处理函数的ui参数所定义的属性

名 称	描 述
tab	返回包含远程标签元素的jQuery对象
panel	返回包含载入内容的面板元素的jQuery对象

注意表中的两个属性都没有直接引用从服务器上载入的内容，事件处理函数只能访问标签组件的标题（tab属性）和内容面板（panel属性）元素。代码清单20-11演示了如何使用这些元素修改标签的标题，以及如何修改从服务器载入的内容。

代码清单20-11 生成标签

```
...
<script type="text/javascript">
  $(document).ready(function () {
    $("#tabs").tabs({
      load: function (e, ui) {
        ui.tab.find("a").text("Loaded!");
        ui.panel.children().first().remove();
      }
    });
  });
</script>
...
```

使用`ui.tab`属性定位标签使用的元素,然后使用jQuery的`text`方法修改标签的标题。使用`ui.panel`属性定位从服务器载入的第一个内容子元素,并把它从DOM中删除。这个例子的实际效果见图20-7。

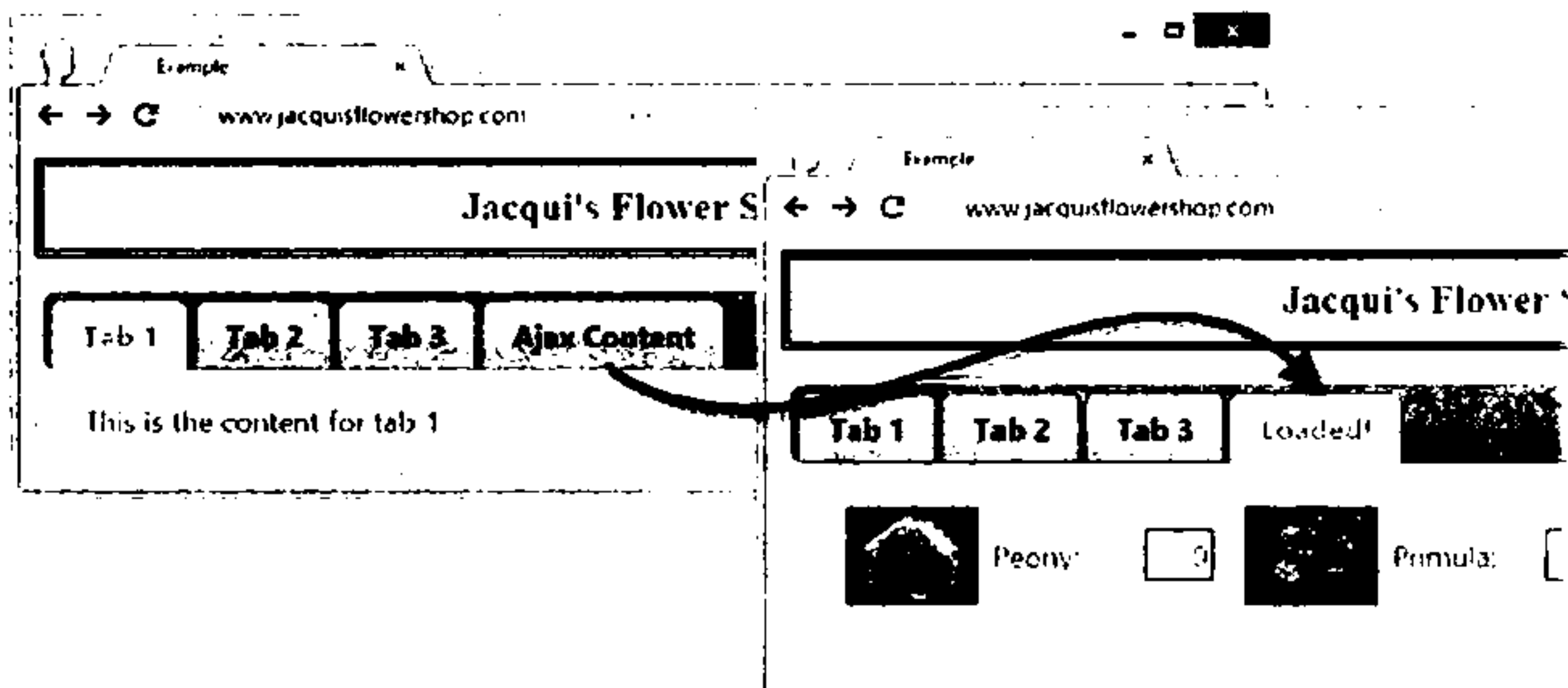


图20-7 处理load事件

**提示** 注意,此处不必调用`refresh`方法,即使我对标签组件所依赖DOM元素做了修改,也不必调用。这是因为在load事件处理函数执行之后, jQuery UI会自动调用`refresh`方法。

### 20.5.3 用标签显示表单

把一个长长的表单分解成更容易完成的小节是一项很有用的技术,这能让用户明确知道自己的进展。代码清单20-12中的页面包含了我们即将使用的表单。

代码清单20-12 包含表单的页面

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    #tab2 input, #tab3 input {width: 200px; text-align: left}
    #tab1, #tab2, #tab3 {padding: 10px}
    .fl {float: left}
    #buttonDiv {clear: both}
    #tabs, h1 {margin: 10px}
    .regLabel {width: auto}
  </style>
  <script id="flowerTpl" type="text/x-jquery-tmpl">
    {{#flowers}}
```

```

        <div class="dcell ui-widget">
            
            <label for="{{product}}">{{name}}:</label>
            <input name="{{product}}" value="0"/>
        </div>
    {{/flowers}}
</script>
<script id="detailsTpl" type="text/x-jquery-tmpl">
    {{#details}}
        <div class="ui-widget">
            <label for="{{name}}">{{name}}:</label>
            <input name="{{name}}" placeholder="{{hint}}"/>
        </div>
    {{/details}}
</script>
<script type="text/javascript">
    $(document).ready(function () {

        var data = [{ "name": "Aster", "product": "aster" },
                      { "name": "Daffodil", "product": "daffodil" },
                      { "name": "Rose", "product": "rose" },
                      { "name": "Peony", "product": "peony" }];

        var elems = $("#flowerTpl").template({ flowers: data }).filter("*");
        elems.slice(0, 2).appendTo("#row1");
        elems.slice(2, 4).appendTo("#row2");

        var detailsData = [{ name: "Name", hint: "Enter your name" },
                            { name: "Street", hint: "Enter your street" },
                            { name: "City", hint: "Enter your city" },
                            { name: "State", hint: "Enter your state" },
                            { name: "Zip", hint: "Enter your zip code" }];

        $("#detailsTpl").template({ details: detailsData }).filter("*")
            .appendTo("#tab2").clone().appendTo("#tab3")

        $("button").button();
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="tabs" class="ui-widget">
            <ul>
                <li><a href="#tab1">1. Select Products</a>
                <li><a href="#tab2">2. Your Details</a>
                <li><a href="#tab3">3. Your Shipping Address </a>
            </ul>
            <div id="tab1">
                <h2>1. Select Products</h2>
                <div id="row1"></div>
                <div id="row2"></div>
            </div>

```

```

        <div id="tab2" class="fl">
            <h2>2. Your Details</h2>
        </div>
        <div id="tab3" class="fl">
            <h2>3. Your Shipping Address</h2>
        </div>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
</form>
</body>
</html>

```

我在页面中添加了一些额外的内容和结构,以充实之前的示例页面。虽然页面中的花卉产品很少,但我在页面中添加了一个区域来捕获用户的个人信息及送货细节。这个表单最初的样子见图20-8。

The image shows a web browser window displaying a form titled 'Jacqui's Flower Shop'. The form is organized into three tabs: '1. Select Products', '2. Your Details', and '3. Your Shipping Address'. Under '1. Select Products', there are four items: 'Pink' (a pink flower), 'Purple' (a purple flower), 'Orange' (an orange flower), and 'Purple' (a purple flower). Each item has a checkbox and a 'Select' button. Under '2. Your Details', there are input fields for 'Name', 'Street', 'City', 'State', and 'Zip'. Under '3. Your Shipping Address', there are input fields for 'Name', 'Street', 'City', 'State', and 'Zip'. At the bottom of the form is a 'Place Order' button.

图20-8 准备使用标签组件呈现的多节表单

这个表单本身没有什么特殊之处,不过它更适用于jQuery UI标签组件。它非常纯粹,由几个独立区域组成,因此每一部分都适合放到一个标签中显示。

表单的大部分内容都是利用数据模板插件生成的。使用前面各章用过的jQuery技术,我使用数据生成元素,必要时克隆它们,把最后的产出添加到页面当中。我们并不是必须使用标签来显示表单,不过这既然是一本讲jQuery的书,我愿意尽可能地使用jQuery的核心功能。

我们看到,图中的ul元素和它包含的链接指向内容元素。我通常会隐藏这些链接元素,不过在这里我要演示一下这种结构的一个很好的副作用:它特别适合用作标题。我们生成了一个列表,而且列表的每一项都包含一个链接,因此可以单击链接直接跳到页面那一部分,而且如果链接指向的是另一个文件,浏览器就会重定向到那个页面。标签标题的这种行为,恰是我需要的。

### 1. 应用标签组件

现在我们已经做好了准备,可以生成标签了。代码清单20-13展示了script元素中必要的变更,除



此之外页面的其他部分都无需改变。

### 代码清单20-13 生成标签

```
...
<script type="text/javascript">

    $(document).ready(function () {

        var data = [{ "name": "Aster", "product": "aster" },
                     { "name": "Daffodil", "product": "daffodil" },
                     { "name": "Rose", "product": "rose" },
                     { "name": "Peony", "product": "peony" }];

        var elems = $("#flowerTpl").template({ flowers: data }).filter("");
        elems.slice(0, 2).appendTo("#row1");
        elems.slice(2, 4).appendTo("#row2");

        var detailsData = [{ name: "Name", hint: "Enter your name" },
                           { name: "Street", hint: "Enter your street" },
                           { name: "City", hint: "Enter your city" },
                           { name: "State", hint: "Enter your state" },
                           { name: "Zip", hint: "Enter your zip code" }];

        $("#detailsTpl").template({ details: detailsData }).filter("")
            .appendTo("#tab2").clone().appendTo("#tab3")

        $(".f1").removeClass("f1");
        $("#tabs").tabs().find("h2").remove();

        $("button").button();

    });
</script>
...
```

我删除了f1类，它本来负责细节内容元素和收货地址区域的定位。我还删除了原来用作分节标题的h2元素。如图20-9所示，最后我调用了tabs方法，它将内容元素用作标签的基础。

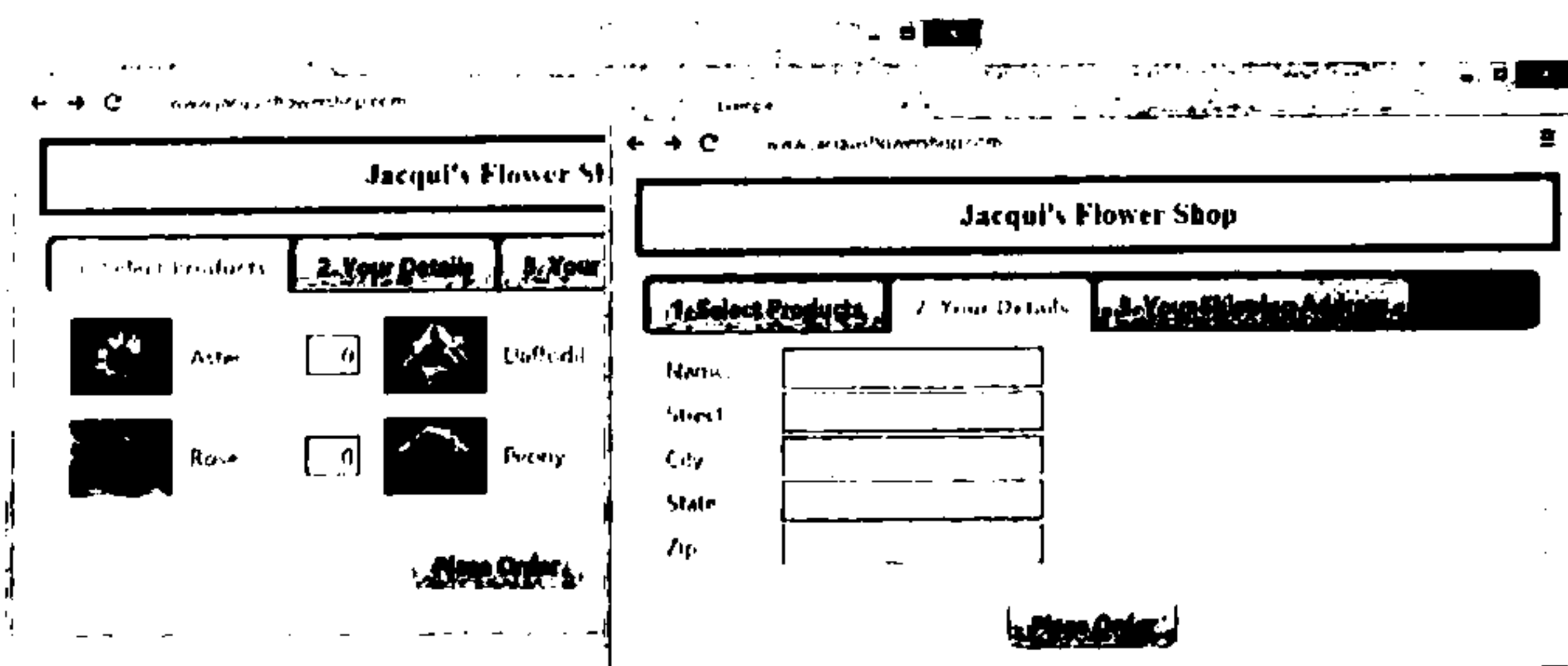


图20-9 在表单上调用tabs方法

## 2. 处理按钮单击

为了让使用标签的表单更容易填写,我为提交按钮注册了click事件处理函数。在这个处理函数中,我抑制了提交事件的默认行为(提交表单),让它顺序跳到下一个标签,直到到达最后一个标签。这时,单击提交按钮才会真的把表单提交到服务器。代码清单20-14中展示了新增加的脚本内容。

代码清单20-14 利用Submit按钮遍历表单

```
...
<script type="text/javascript">

    $(document).ready(function () {

        var data = [{ "name": "Aster", "product": "aster" },
                     { "name": "Daffodil", "product": "daffodil" },
                     { "name": "Rose", "product": "rose" },
                     { "name": "Peony", "product": "peony" }];

        var elems = $("#flowerTpl").template({ flowers: data }).filter("");
        elems.slice(0, 2).appendTo("#row1");
        elems.slice(2, 4).appendTo("#row2");

        var detailsData = [{ name: "Name", hint: "Enter your name" },
                           { name: "Street", hint: "Enter your street" },
                           { name: "City", hint: "Enter your city" },
                           { name: "State", hint: "Enter your state" },
                           { name: "Zip", hint: "Enter your zip code" }];

        $("#detailsTpl").template({ details: detailsData }).filter("")
            .appendTo("#tab2").clone().appendTo("#tab3")

        $(".fl").removeClass("fl");
        $("#tabs").tabs().find("h2").remove();

        $("button").button().click(function (e) {
            var tabsElem = $("#tabs");
            var activeTab = tabsElem.tabs("option", "active");
            if (activeTab < tabsElem.find("ul > li").length - 1) {
                tabsElem.tabs("option", "active", activeTab + 1)
                e.preventDefault();
            }
        });
    });
</script>
...
```

先使用active选项得到当前标签的序号,然后使用jQuery选择器得到所有的标签标题元素,从而算出共有几个标签。如果用户尚未抵达最后一个标签,就把active选项设置成下一个。只要不是最后一个标签,就调用preventDefault方法阻止表单提交,从而达到只允许在最后一个标签提交表单的目的。

## 3. 实施验证

此时此刻,用户完全可以直接跳到最后一个标签并提交表单。为了避免这种情况,我准备施加一点基本的表单验证。为了让这个例子保持简单,在这里我手工编写代码处理表单验证。不过对于真正

的项目，建议使用第13章中讲到的验证插件及其验证技术。代码清单20-15展示了脚本中变更的部分，这些代码实施了一些基本的验证措施，可成功阻止用户什么也不填写就直接跳到表单末尾的行为。

代码清单20-15 利用一些基本验证阻止用户直接跳标签的行为

20

```
...
<script type="text/javascript">

    $(document).ready(function () {

        var data = [{ "name": "Aster", "product": "aster" },
                     { "name": "Daffodil", "product": "daffodil" },
                     { "name": "Rose", "product": "rose" },
                     { "name": "Peony", "product": "peony" }];

        var elems = $("#flowerTpl").template({ flowers: data }).filter("*");
        elems.slice(0, 2).appendTo("#row1");
        elems.slice(2, 4).appendTo("#row2");

        var detailsData = [{ name: "Name", hint: "Enter your name" },
                           { name: "Street", hint: "Enter your street" },
                           { name: "City", hint: "Enter your city" },
                           { name: "State", hint: "Enter your state" },
                           { name: "Zip", hint: "Enter your zip code" }];

        $("#detailsTpl").template({ details: detailsData }).filter("*")
            .appendTo("#tab2").clone().appendTo("#tab3")

        var activePanel;

        $(".fl").removeClass("fl");
        $("#tabs").tabs({
            beforeActivate: function (e, ui) {
                validatePanel(e, ui.oldPanel);
            },
            activate: function (e, ui) {
                activePanel = ui.newPanel;
            }
        }).find("h2").remove();

        function validatePanel(e, panelElem) {
            var inputElems = panelElem.find("input");
            if (panelElem.attr("id") == "tab1" ?
                sumInputElems(inputElems) : countEmptyOrZeroValues(inputElems)) {
                alert("Validation Problem!");
                e.preventDefault();
            }
        }

        function sumInputElems(inputs) {
            var total = 0;
            inputs.each(function (index, elem) {
                total += Number($(elem).val());
            });
        }
    });
</script>
```

```

    });
    return total == 0;
}

function countEmptyOrZeroValues(inputs) {
    var count = 0;
    inputs.each(function (index, elem) {
        if (elem.value == null || elem.value == "") {
            count++;
        }
    });
    return count > 0;
}

$("#button").button().click(function (e) {
    var tabsElem = $("#tabs");
    var activeTab = tabsElem.tabs("option", "active");
    if (activeTab < tabsElem.find("ul > li").length - 1) {
        tabsElem.tabs("option", "active", activeTab + 1);
        e.preventDefault();
    } else {
        validatePanel(e, activePanel);
    }
});
});
</script>
...

```

我使用了两个标签组件事件实现所期望的效果。在用户从一个标签导航至下一个标签时，beforeActivate事件就派上用场了，因为利用该事件处理函数的第二个参数——ui对象，能得到即将关闭标签的内容面板的引用。ui对象定义各个属性见表20-7。

表20-7 beforeActivate和activate事件处理函数的ui参数对象定义的属性

名 称	描 述
newHeader	新激活标签的标题元素
oldHeader	上一个激活标签的标题元素
newPanel	新激活标签的面板元素
oldPanel	上一个激活标签的面板元素

使用oldPanel引用实施验证，并在必要时调用事件对象的preventDefault方法阻止标签组件前往下一个标签（详见第9章）。

注意这个方案里有一个技巧，当用户完成表单的最后一个标签后，点击按钮提交表单到服务器并不会触发beforeActivate事件。为此需要处理activate事件来保存当前显示的新标签的一份引用（利用ui对象的新Panel属性，该属性的定义见表20-7）。这样，在提交表单之前，就能利用这个引用来实施表单验证。

如图20-10所示，在这个方案里，我会在验证失败时，调用alert函数显示一条警告信息，告诉用户出错了。显然，在真实的项目中，我们更应该使用验证插件的问题摘要功能（参阅第13章）。

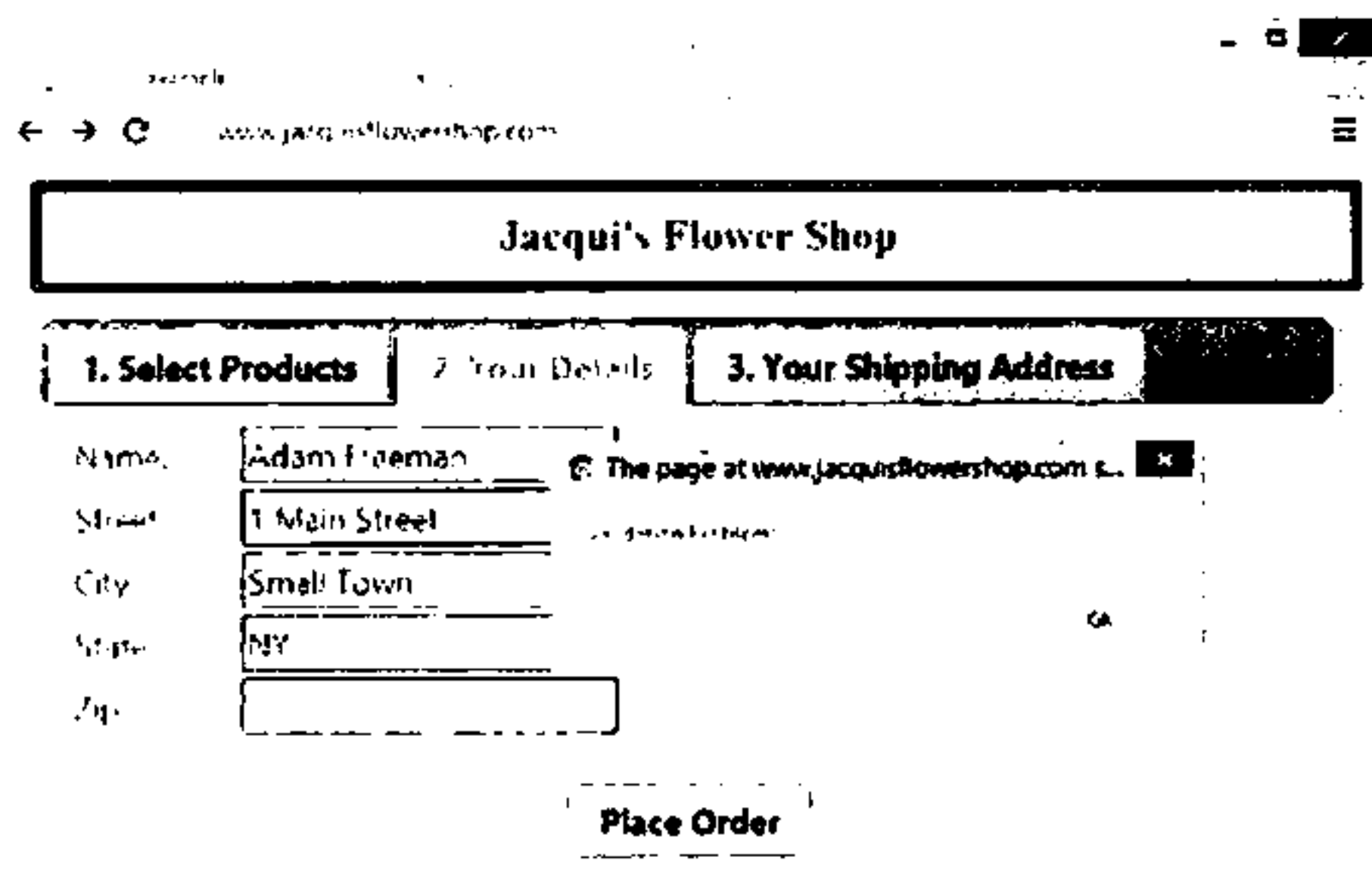


图20-10 检测到错误时弹出的警告对话框

## 20.6 小结

本章详细介绍了jQuery UI的标签组件。这个组件提供了丰富的功能，适用于许多场合。我经常使用这个组件。它相当灵活，支持全方位的定制，而且选取某个标签并显示相应的独立内容，是用户通常很熟悉的操作方式，一看就知道怎么用。折叠菜单组件就没有它这么容易上手了。在下一章，我将介绍日期拾取器组件的用法。

## 日期拾取器组件

本章的主题是jQuery UI的日期拾取器组件 (Datepicker)，它提供了一种很常见的日期选择界面，便于用户选取日期。由于表达日期的格式实在太多，从用户那里收集日期信息一直是令广大Web开发者头痛的问题。日期拾取器组件能帮助用户更容易地选择日期，这能够有效地保证数据的一致性，降低出错的可能性，而且使用的是标准input文本框。表21-1列出了本章概要。

表21-1 本章概要

问 题	解决方法	代码清单
如何生成日期拾取器	在input元素上调用datepicker方法	1
如何生成嵌入式日期拾取器 (不弹出)	在span或者div元素上调用datepicker方法	2
如何指定日期拾取器显示的默认日期	使用defaultDate选项	3
如果指定一个额外的input元素，在用户选择一个日期时更新它的值	使用altField选项	4
如何设置弹出式日期拾取器的条件	使用showOn选项	5
如何指定触发弹出式日期拾取器的按钮上的文本	使用buttonText选项	6
如何指定触发弹出式日期拾取器的按钮上的图片	使用buttonImage选项	7
如何限制日期选择范围	使用constrainInput、minDate和maxDate选项	8、9
如果在日期拾取器中一次显示多个月	使用numberOfMonths选项	10~12
如何允许用户直接使用下拉菜单直达某月或者某年	使用changeMonth和changeYear选项	13
如何在日期拾取器中显示星期信息	使用showWeek和weekHeader选项	14
如果在日期网格中显示上一月和下一月的部分日期	使用showOtherMonths和selectOtherMonths选项	15
如何在日期拾取器下方显示按钮面板	使用showButtonBar和gotoCurrent选项	16
如何显示一个有格式的提示信息给用户	使用appendText选项 (或HTML5的占位符功能)	17、18
如何使用脚本获取或者设置日期	使用getDate和setDate方法	19
如何利用脚本控制日期拾取器的显示或者隐藏	使用show和hide事件	20
如何响应用户进入到一个新的月或者一个新的年的事件	使用onChangeMonthYear事件	21
如何响应日期拾取器关闭事件	使用onClose事件	22
如何让日期拾取器使用当地语言文字	使用jQuery UI i18n功能	23

## 21.1 生成日期拾取器

日期拾取器一共有两种基本用法。最常见的用法是把日期拾取器组件附加到input元素上，也就是选中input元素，然后调用datepicker方法。input文本框并不会立即发生什么视觉上的变化，然而一旦input元素得到焦点（用户使用Tab键把焦点从其他元素移到input元素或者单击了这个文本框），日期拾取器就会弹出来以方便用户选择日期。代码清单21-1展示的就是这样一个弹出式日期拾取器。

代码清单21-1 生成弹出式日期拾取器

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    input {width: 200px; text-align: left}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#datep").datepicker();
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div class="ui-widget">
      <label for="datep">Date: </label><input id="datep"/>
    </div>
  </form>
</body>
</html>
```

图21-1展示了文本框得到焦点之后弹出式日期拾取器的情况。

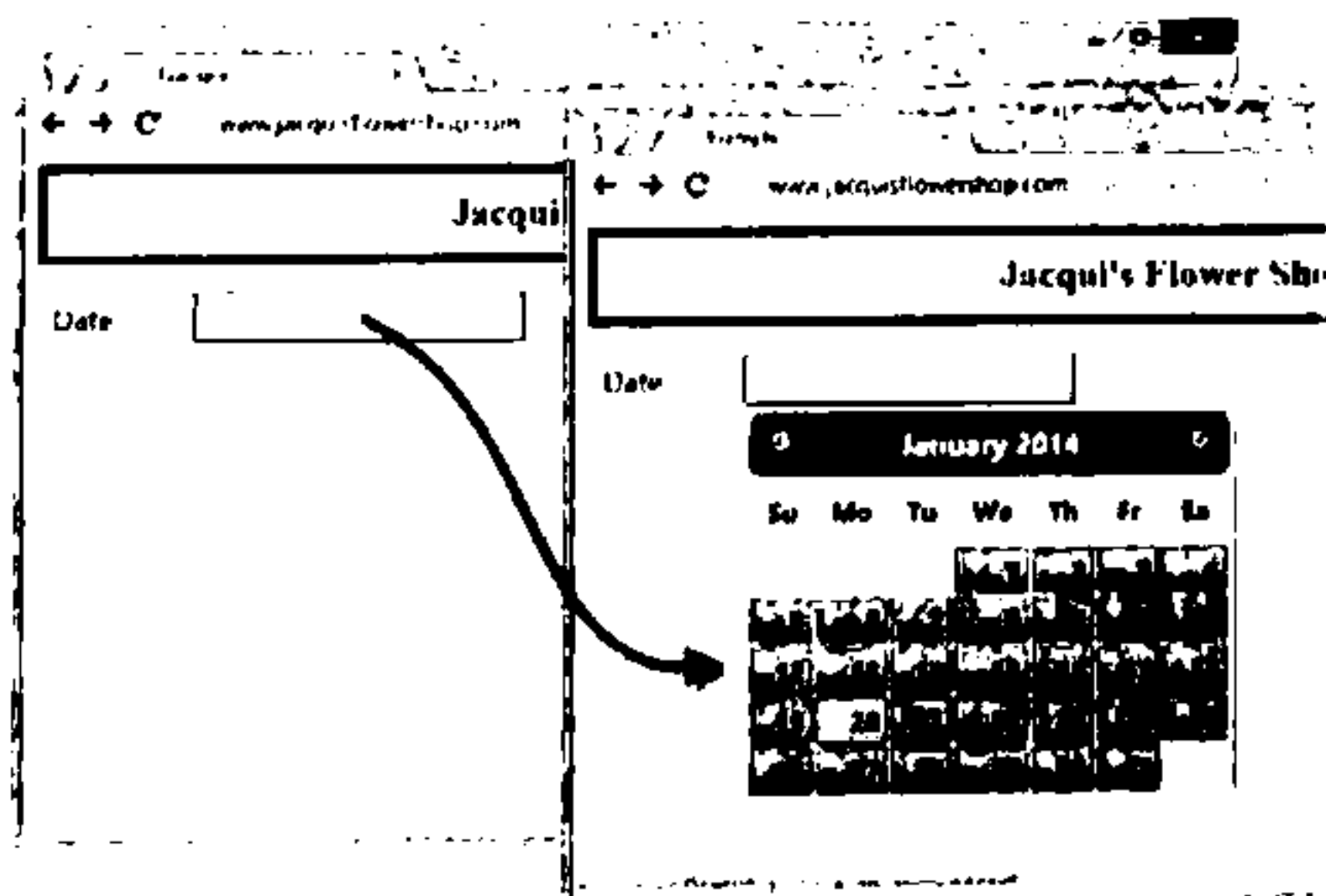


图21-1 文本框得到焦点，日期拾取器随之弹出

日期拾取器弹出之后，用户可以手工输入日期，也可以从弹出窗口中选择一个日期。当文本框失去焦点，或者用户敲了回车键或者Esc键时，日期拾取器会自动消失。

## 生成嵌入式日期拾取器

另一种使用方法是生成嵌入式（不弹出的）日期拾取器。要达到这一目的，我们需要选中一个div或者span元素，然后调用datepicker方法。对于嵌入式日期拾取器，只要底层元素可见，它就可见。嵌入式日期拾取器的具体示例见代码清单21-2。

代码清单21-2 生成嵌入式日期拾取器

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    input {width: 200px; text-align: left; margin-right: 10px}
    #wrapper > * {float: left}
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#inline").datepicker();
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="wrapper" class="ui-widget">
      <label for="datep">Date: </label>
      <input id="datep"/>
      <span id="inline"></span>
    </div>
  </form>
</body>
</html>
```

在这个例子里，我选中了页面中的一个span元素，然后调用datepicker方法。这个例子的具体效果见图21-2。

当我们不方便弹出日期拾取器的时候，就使用嵌入式日期拾取器。在某些应用程序中，日期可能非常重要，需要一直显示日期拾取器。不过更多的场合还是更适合默认隐藏，在需要时才弹出的日期拾取器。显示或者隐藏嵌入式日期拾取器的问题在于，页面布局必须有弹性以容纳日期拾取器，这可能会影响显示效果。我发现，在几乎所有的场合都应该选用弹出式日期拾取器。



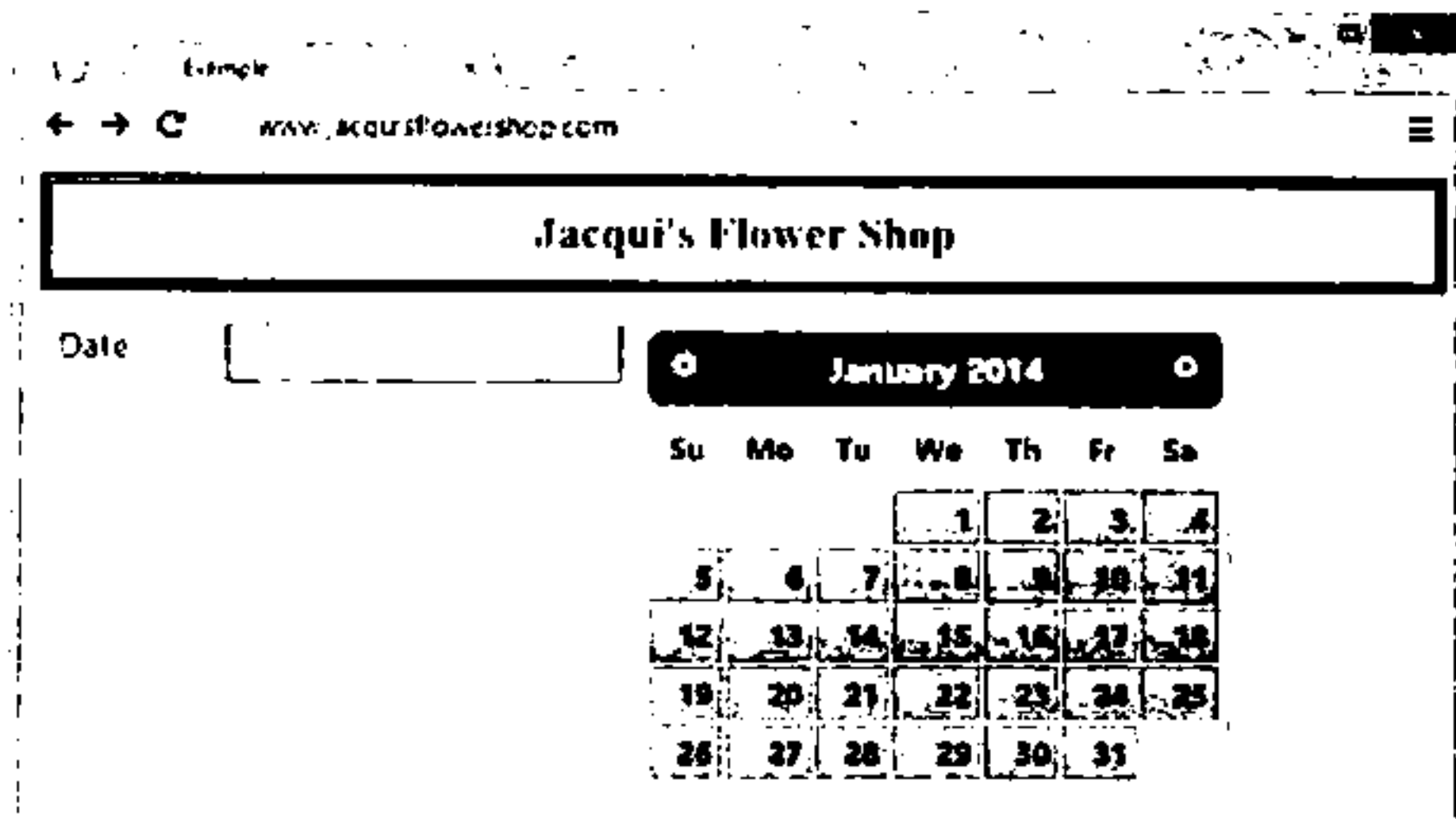


图21-2 嵌入式日期拾取器

## 21.2 配置日期拾取器

如果你以前处理过有关日期的事情，就会明白处理这类问题的复杂程度。日期拾取器组件支持非常多的选项，这也折射出了要解决问题的复杂度。在接下来的几节，我将向大家介绍一些与日期拾取器配置有关的选项。

### 21.2.1 基本设置

表21-2列出了日期拾取器组件的基础配置选项，在后面的内容中我会介绍如何使用这些选项。

表21-2 基本的日期拾取器选项

选 项	描 述
altField	额外指定一个input元素，当用户作出选择时，把这个input元素的值更新成用户选取的日期
buttonImageOnly	使用img元素而不是button元素显示buttonImage选项指定的图片，默认值为false
buttonImage	指定触发弹出式日期拾取器按钮图片的URL，默认不使用图片
buttonText	指定触发弹出式日期拾取器按钮上的文本，默认值为省略号 (...)
defaultDate	设置日期拾取器的默认日期（高亮的日期）
disabled	设置日期拾取器初始状态是否可用，默认值为false（表示可用）
showOn	设置弹出式日期拾取器的条件，默认值为focus

#### 1. 指定默认日期

defaultDate是最基本的设置，也是最有用的一个设置。这个选项指定日期拾取器最初显示的日期。如果我们没有设置defaultDate选项，那么日期拾取器就使用当前日期（也就是用户机器上当天的日期）。当然，日期是由用户的系统定义的。而由于可能的时区、国际日期变更线、错误配置，用户看到的日期很可能会与其期望中不同。

**提示** 这一选项仅适用于input元素没有设置value属性的情况。如果在页面代码中为input元素设置了value属性，或者在调用日期拾取器组件之前用户已经输入了一个日期，日期拾取器都会忽略该选项。

如果你不希望使用今天的日期，可以通过多种格式表达一个起始日期。表21-3列出了我们可以使用的各种格式。

表21-3 defaultDate选项的格式和值

值 / 格式	描 述
null	使用当前系统日期
Date对象	便当这个Date对象所表达的值
+days、-days	使用与今天相比的偏移量设置日期。比如说+3表示从今天开始算3天后的日期，而-2则表示2天前的日期
+1d +7w -1m +1y	使用与今天相比的相对值设置日期，可以是多少天前或多少天后 (d)，也可以是几个星期前后 (w)，还可以是几年前后 (y)。数字前的+号表示将来的日期，而-表示已经过去的日期。正值和负值还可以混合使用以表达某个日期。假设今天是2011年11月12日，那么-1d +1m 就表示2011年12月11日

代码清单21-3中的这个例子使用defaultDate选项指定了5年后的一天作为默认日期。

代码清单21-3 使用defaultDate选项

```
...
<script type="text/javascript">
  $(document).ready(function() {
    $("#datep").datepicker({
      defaultDate: "+5y"
    });
  });
</script>
...
```

我写这一章的时候是2013年的6月，从图21-3中可以看到，使用+5y作为defaultDate的值，其结果就是日期拾取器默认显示2018年6月。

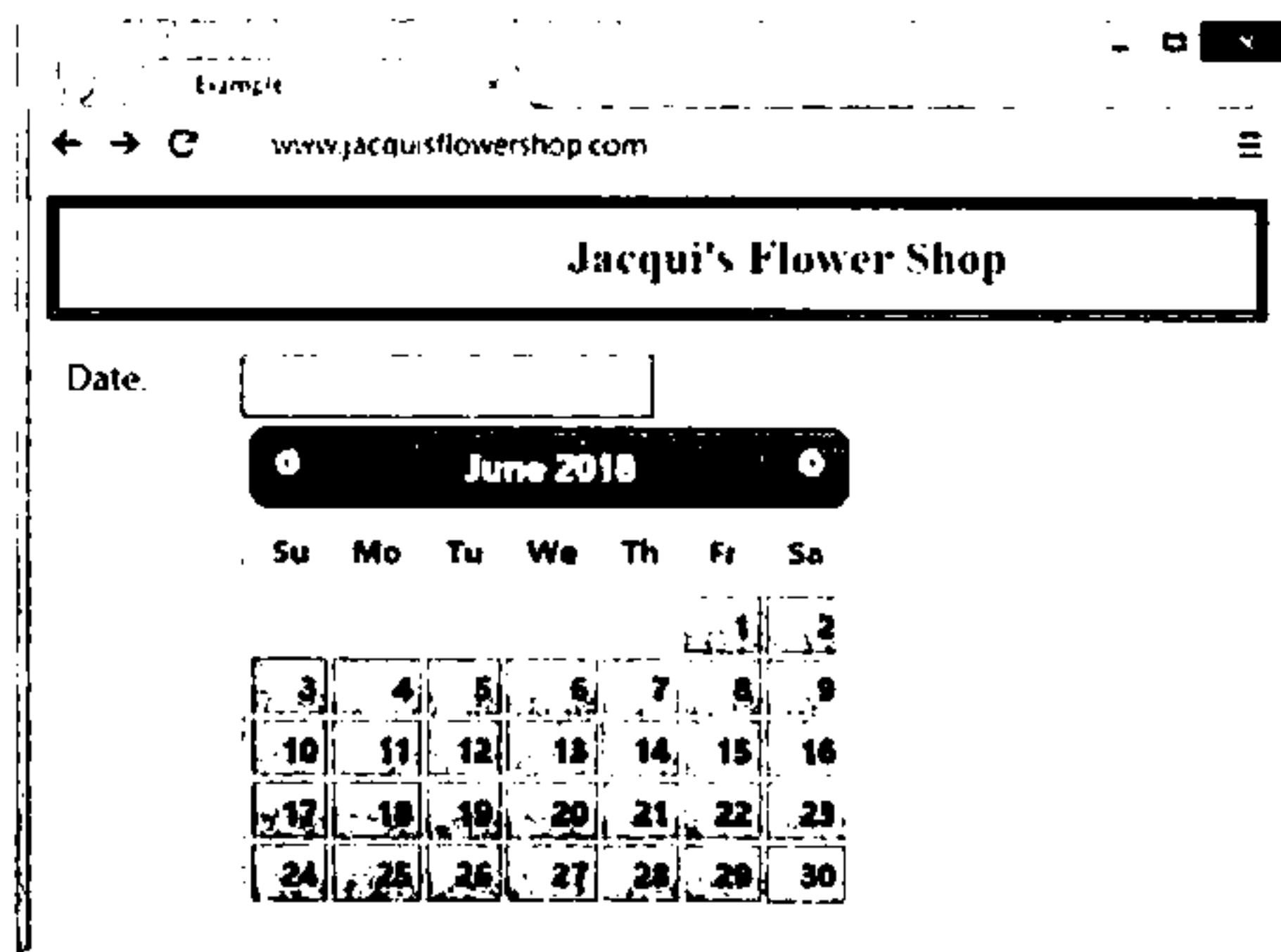


图21-3 使用defaultDate选项显示一个未来的日期

在这个例子中，我们省略了那些不想改变的时间间隔，直接使用+5y而不是+0d +0w +0m +5y。每

种时间间隔都支持正值、负值或者0值，可以根据需要自由选择，以得到所需要的日期。

## 2. 指定替代元素

利用altField选项，我们可以指定一个input元素。当用户选中一个日期时，日期拾取器组件就会更新该元素的值。这是把一个嵌入式日期拾取器与一个input元素关联起来的最简便方法。代码清单21-4演示了altField选项的一种用法，在这个示例中altField选项指定的元素用来显示用户选取的日期。

代码清单21-4 在嵌入式日期拾取器中使用altField选项

```
...
<script type="text/javascript">
  $(document).ready(function() {
    $("#inline").datepicker({
      altField: "#datep"
    });
  });
</script>
...
```

在这个例子中，我使用CSS选择器标识可替代元素。不过altField选项也接受jQuery对象或者HTMLElement对象作为它的值。这个例子产生这样一种效果：我在日期拾取器中的每一次选择，都会把选择结果显示在altField对应的那个input元素中（如图21-4所示）。

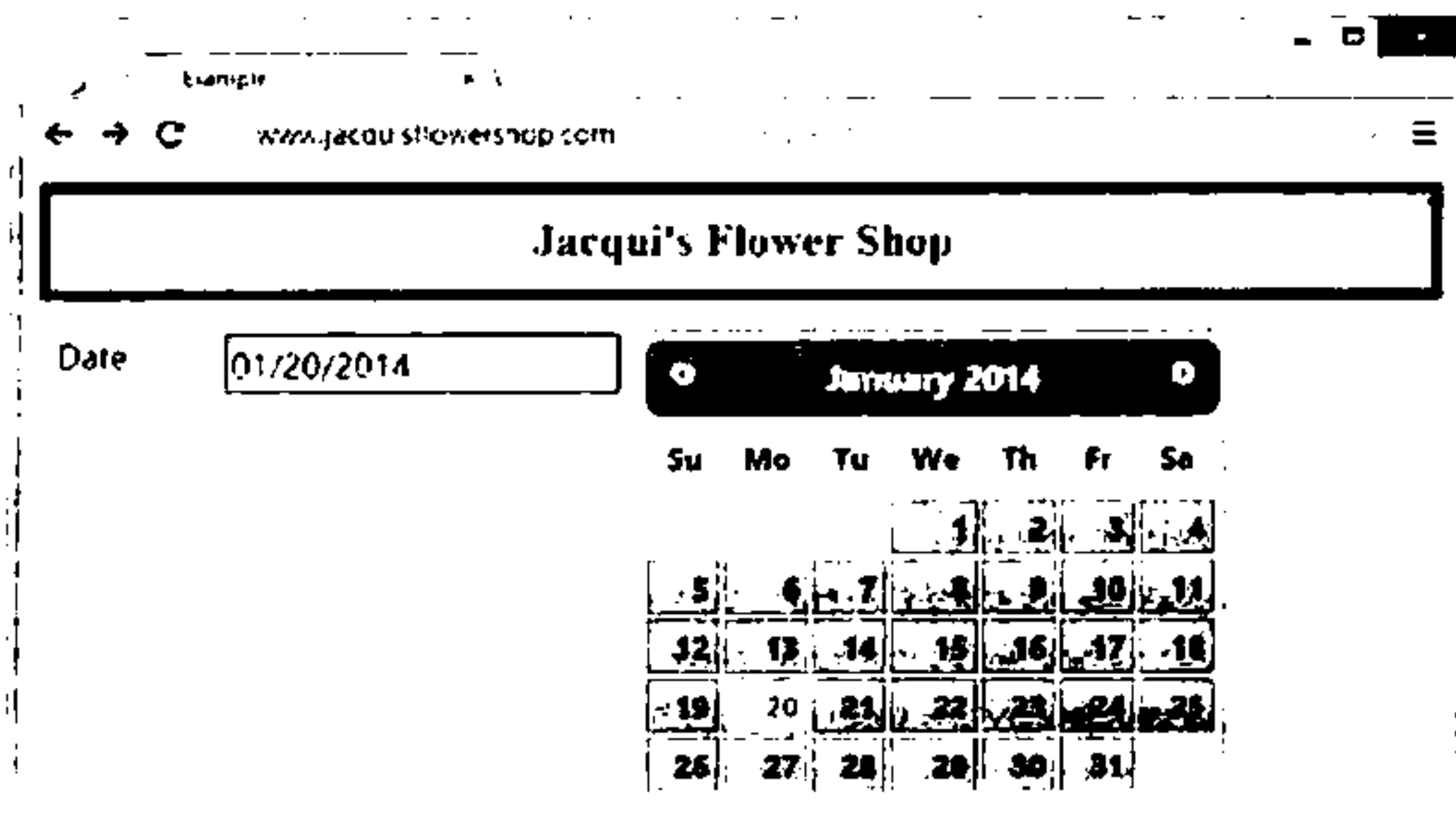


图21-4 为日期选择器指定对应的（底层）input元素

## 3. 设置弹出式日期拾取器的触发事件

利用showOn选项，我们可以控制弹出式日期拾取器的条件。这个选项支持以下3种值。

- focus: 当input元素得到焦点时弹出日期拾取器，它也是默认值。
- button: 单击按钮时弹出日期拾取器。
- both: input元素得到焦点，或者用户单击了按钮，两种情况下都会弹出日期拾取器。

如果我们把showOn选项设置为button或者both，日期拾取器组件就会创建一个button元素并把它立即插入到input元素之后。代码清单21-5演示了这个选项的用法。

## 代码清单21-5 使用showOn选项

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#datep").datepicker({
            showOn: "both"
        });
    });
</script>
...
```

我们可以在图21-5中看到这个按钮。由于示例中使用了both值，不论用户是单击这个按钮，还是单击文本框（使input元素得到焦点），都会触发弹出式日期拾取器。

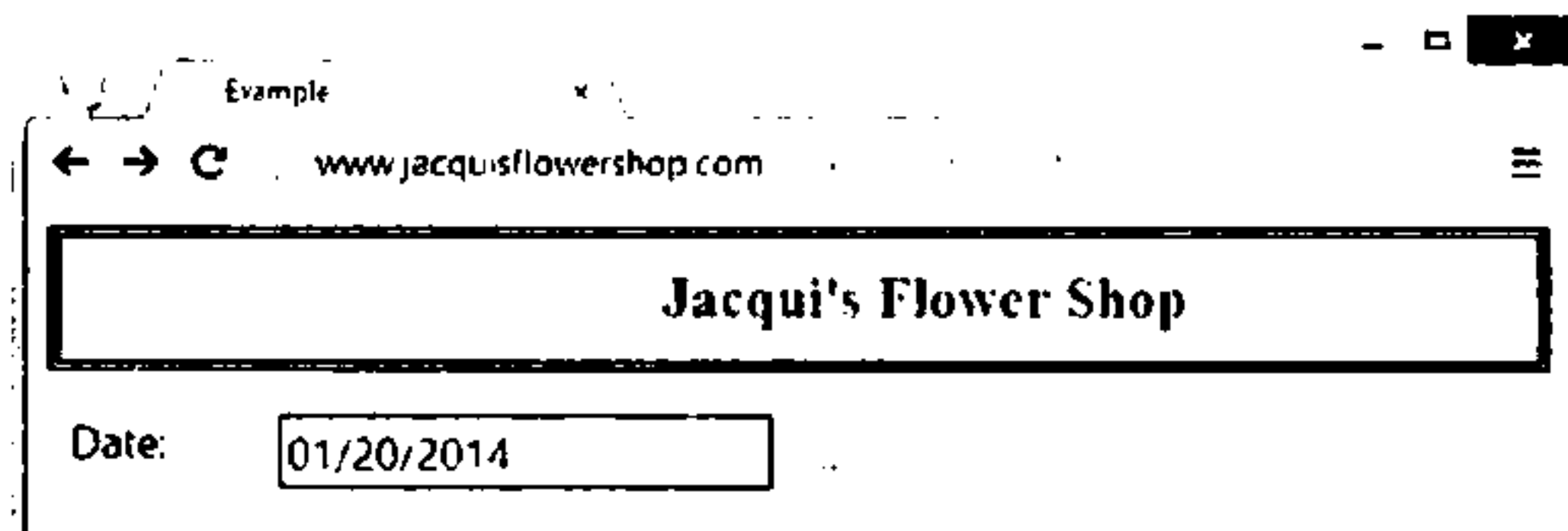


图21-5 由日期拾取器组件添加的用于响应showOn条件的按钮

**提示** 日期拾取器组件在这里添加的按钮并不是一个jQuery UI按钮。如果你希望保持按钮一致的外观，可以参考第18章中的内容，选中这个按钮元素并调用jQuery UI的button方法。

我们可以使用buttonImage或者buttonText选项来调整这个按钮元素的外观。如果把buttonImage选项的值设置为一个URL，日期拾取器组件就会在这个按钮上使用图片。相应地，如果为buttonText选项设置了一个短语，它就会替换默认内容（即...）。代码清单21-6演示了buttonText选项的用法。

## 代码清单21-6 使用buttonText选项

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#datep").datepicker({
            showOn: "both",
            buttonText: "Select"
        });
    });
</script>
...
```

如果我们同时使用buttonImage和buttonImageOnly选项，就在完全没有按钮的情况下实现了同样的功能。这时，日期拾取器组件会在页面中添加一个img元素（而不是button元素）。具体示例见代码清单21-7。

## 代码清单21-7 使用img元素，而不是button元素

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    input {width: 200px; text-align: left}
    #dpcontainer * {vertical-align: middle}
    #dpcontainer img {width: 35px;}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#datep").datepicker({
        showOn: "both",
        buttonImage: "right.png",
        buttonImageOnly: true
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="dpcontainer" class="ui-widget">
      <label for="datep">Date: </label><input id="datep"/>
    </div>
  </form>
</body>
</html>

```

我指定了一张图片（right.png）并把buttonImageOnly选项设置为true。我还在页面中添加了一些样式，用以控制与image相关的标签和input元素的样式。日期拾取器组件在生成img元素时并不是很智能，因此我们需要写一些样式来弥补图片与页面其他部分不搭的缺点（让图片与input元素处在同一行）。用图片替代按钮的具体效果见图21-6。

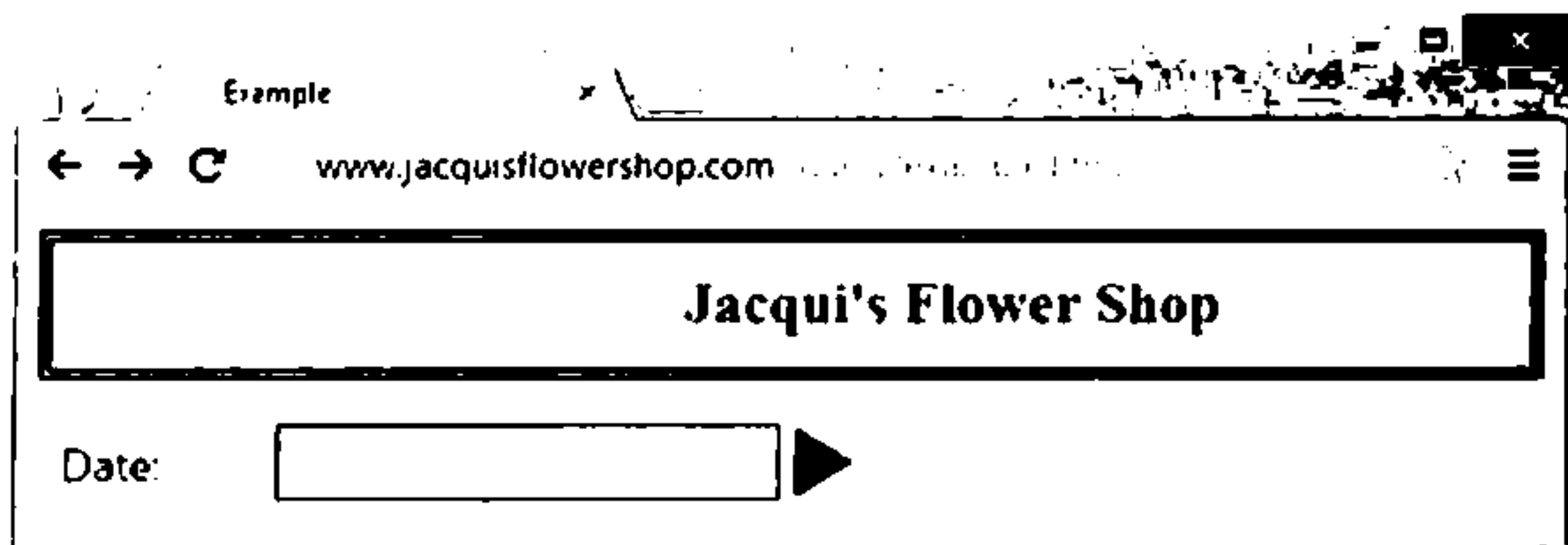


图21-6 在弹出式日期拾取器上使用图片，而不是按钮

## 21.2.2 管理日期选择范围

我们通常希望能控制用户通过日期拾取器组件选择日期的范围。表21-4列出了可以控制日期可选范围的选项。

表21-4 控制日期选择范围的选项

选 项	描 述
changeMonth	默认值是false。如果设置为true，日期拾取器组件就会显示一个下拉菜单，允许用户直接选择一个月份
changeYear	默认值是false。如果设置为true，日期拾取器组件就会显示一个下拉菜单，允许用户直接选择一个年份
constrainInput	默认值是true，表示日期拾取器组件会限制文本框里可输入的字符，以确保用户输入有效的日期
hideIfNoPrevNext	默认值是false。如果设置为true，当没有可选的上一组或下一组日期时，就把上一组（previous）和下一组（next）按钮隐藏起来（而不是禁用）
maxDate	指定用户可选日期的最大值，默认不对用户作任何限制
minDate	指定用户可选日期的最小值，默认不对用户作任何限制
numberOfMonths	指定日期拾取器一次显示几个月，默认值是1
showCurrentAtPos	如果日期拾取器一次显示多个月，指定第几个月是当前月（或者说默认月），默认值是0，表示第一个
stepMonths	当用户单击上一组（previous）或下一组（next）时跳跃的月数，默认值是1
yearRange	如果changeYear选项设置为true，那么该选项指定下拉菜单中年份的选择范围。默认范围是从当前年算起前后各10年

### 1. 限制输入字符及日期范围

如果把constrainInput选项设置为true，那么日期拾取器组件将限制只允许在文本框中输入可组成合法日期的字符。允许输入哪些字符与用户的区域设置（详见21.5节）有关。如果没有对日期拾取器组件进行任何本地化设置，那么input元素只允许用户输入数字和斜线（/）。

这个选项并不能保证用户输入一定是有效的日期（因为类似于99/99/99这样的数字也是被允许输入的），但它确实可以有效地降低出错机会。这一选项在showOn选项设置为button时尤其重要，因为文本框得到焦点并不会触发弹出式日期拾取器。只要日期拾取器弹出来，用户一般就会选择一个日期。但用户并非总能猜到那个按钮是用来显示日期拾取器的。加大用户直接输入日期的机会，这会直接大幅增加你处理用户输入坏数据的比例。代码清单21-8展示了constrainInput选项的用法。

代码清单21-8 应用最基础的日期选择范围选项

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    input {width: 200px; text-align: left; margin-right: 10px}
    #wrapper > * {float: left}
```

```

</style>
<script type="text/javascript">
  $(document).ready(function() {
    $("#datep").datepicker({
      constrainInput: true,
      minDate: "-3",
      maxDate: "+5"
    });
  });
</script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="wrapper" class="ui-widget">
      <label for="datep">Date: </label><input id="datep"/><span id="inline"></span>
    </div>
  </form>
</body>
</html>

```

constrainInput选项的默认值就是true。为了给这个例子增加一点趣味性，我为minDate和maxDate选项设置了相应的值。这里利用这两个选项指定用户可以选择的最小日期和最大日期。类似于我在21.2.1节中介绍过的defaultDate选项，minDate和maxDate选项可以接受null、Date对象、表示天数的数字、或者是一个相对于今天的日期字符串。在本例中，我使用了相对于今天的相对天数来设置这两个选项。在图21-7中我们可以看到，日期拾取器组件禁用了那些不允许用户选择的日期。

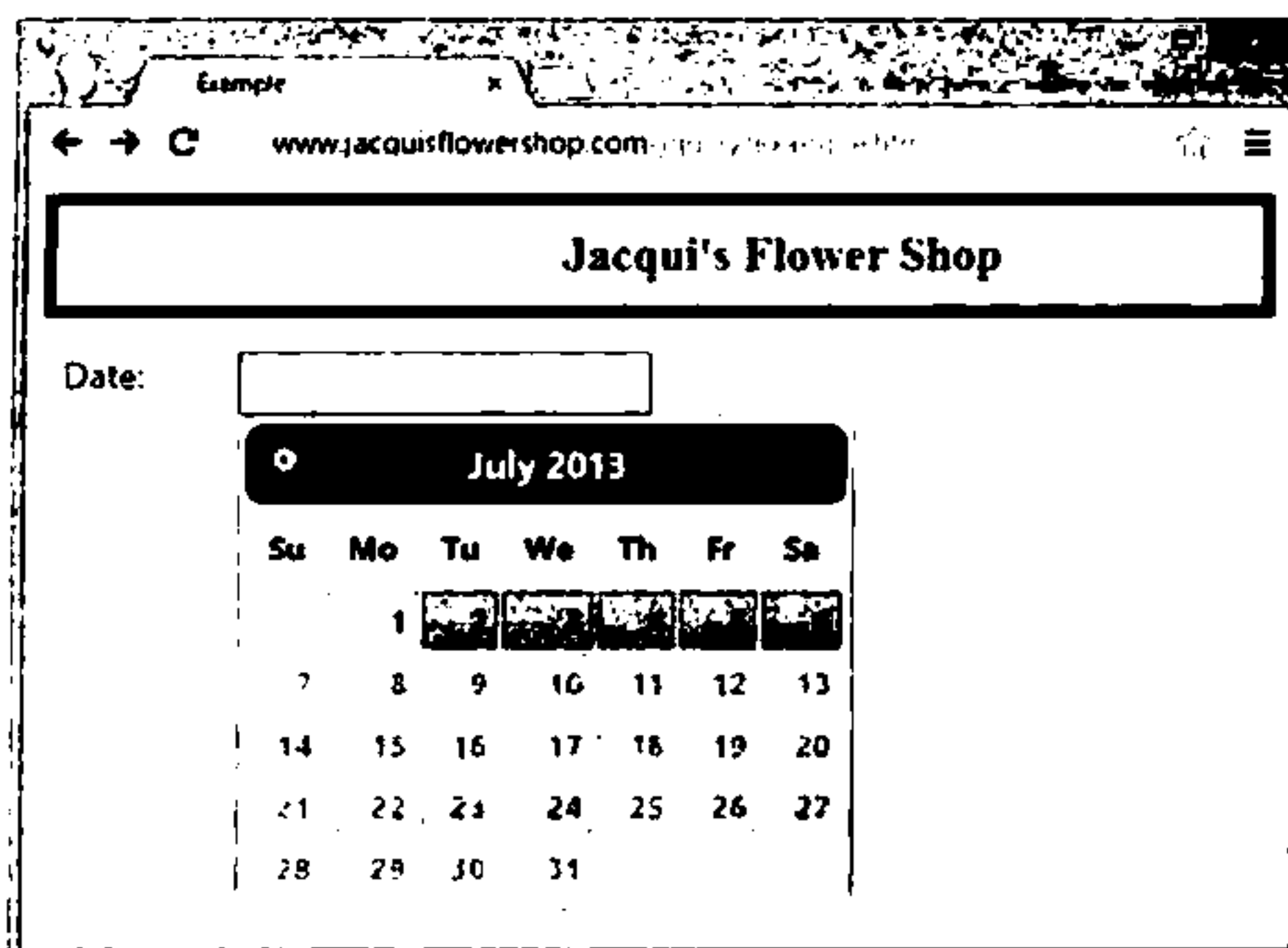


图21-7 限制用户可以选择的日期

**提示** 注意，如果发现没有必要，previous和next按钮就会被自动禁用。它们位于日期拾取器的左上角和右上角，允许用户前往上一个月或者下一个月。在图21-7中，用户只可以选择本月之中的日期，因此这两个按钮都被禁用了。把hideIfNoPrevNext选项设置为true，则在必要时会隐藏而不是禁用这两个按钮。



minDate不必是已经过去的日期，maxDate也不必是将来的日期，而且我们也不是一定要同时设置这两个选项。如代码清单21-9所示，如果我们筹备一件事情需要一段时间，因此需要用户选择一个未来的日期，可以把minDate选项设置成某个将来的日期（那时我们已经就绪），这样就能阻止用户输入一个让我们措手不及的日期。

代码清单21-9 通过一个日期限制生成未来时间窗口

```
...
<script type="text/javascript">
  $(document).ready(function() {
    $("#datep").datepicker({
      minDate: "+7"
    });
  });
</script>
...
```

在这个例子里，我把minDate选项设置为+7，这样用户无论如何也不可以选择从今天起未来7天中的任何一个日期。我并没有设置maxDate，也就是说一周之后任何日期都可以选择。这个例子的效果见图21-8。注意，图中next（下一组）按钮是可用的，而previous（上一组）按钮是禁用状态（这是因为minDate选项已经禁止用户选择过去的日期）。

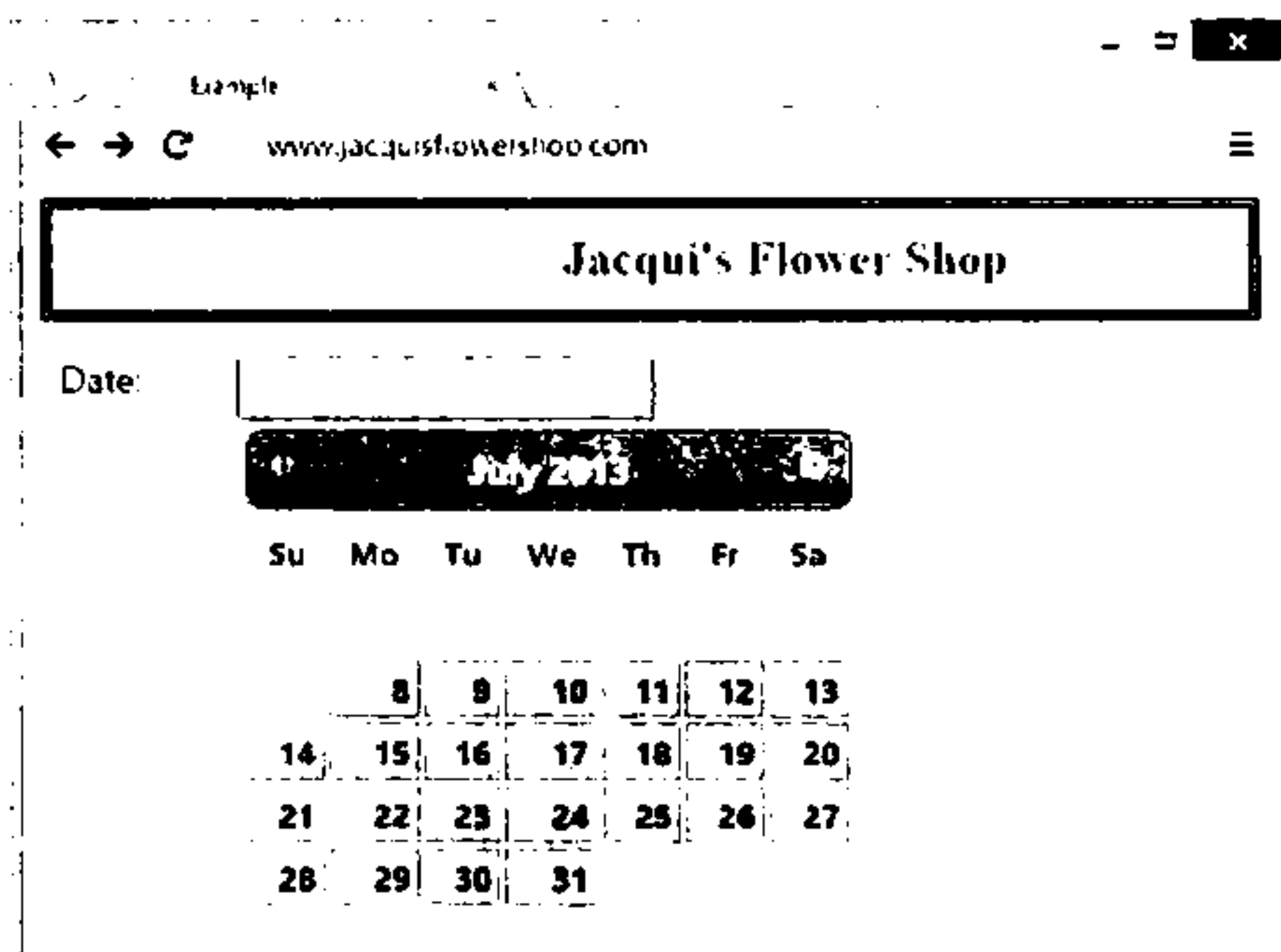


图21-8 一端封闭一端开放的日期区间

**提示** 综合使用minDate、maxDate和defaultDate选项，我们可以指定一段与今天毫无关系的日期区间

## 2. 一次显示多个月份

利用numberOfMonths选项，我们可以让日期拾取器一次显示多个月份。这个选项可以是一个表示月份个数的数字，也可以是一个仅包含两个数值元素（表示月份网格的宽高尺寸）的数组。代码清单21-10中使用了数组值，由于一次显示多个月份，使用弹出式日期拾取器时网格通常过大，所以特别适



合使用嵌入式日期拾取器。（稍后我会解释原因。）

#### 代码清单21-10 使用numberOfMonths选项

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#inline").datepicker({
            numberOfMonths: [1, 3]
        });
    });
</script>
...
```

在这个例子里，我把网格指定为1个月份高，3个月份宽。这个例子的具体效果见图21-9。

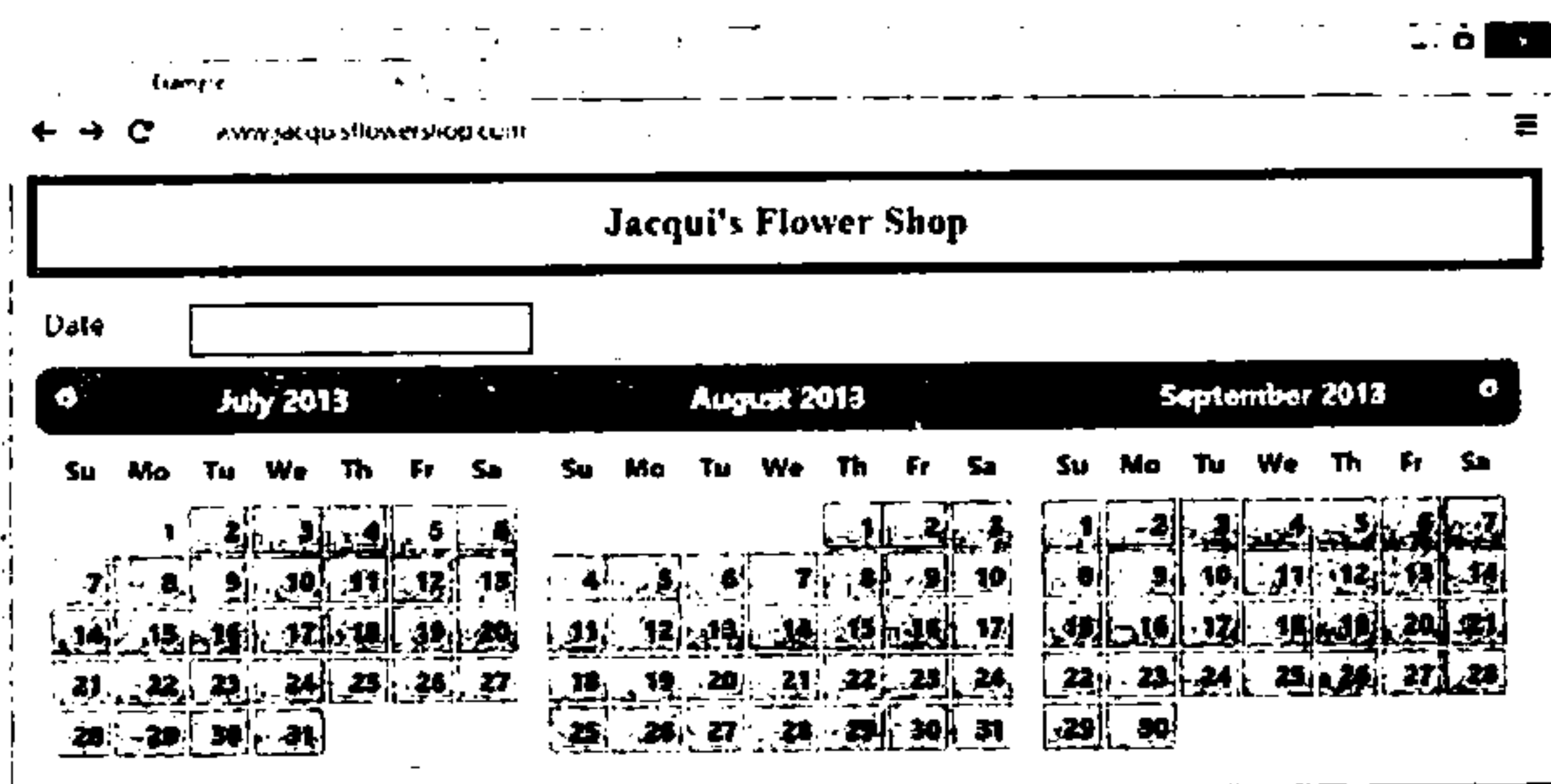


图21-9 显示多月份网格

**提示** 这里的数组值[1, 3]等同于数值3。当我们为numberOfMonths选项指定一个数值时，这个数字会被日期拾取器组件解释为一行内显示的月份数。

我很少在弹出式日期拾取器中使用这个选项，这是因为太大的网格需要假定用户浏览器窗口足够大，以便能够显示完整。弹出式日期拾取器不是一个操作系统对话框，而是一些小心格式化好的HTML元素，是页面的一部分。也就是说，如果在一个很小的浏览器空间里显示一个很大的日期拾取器，就会有一些信息显示不全。代码清单21-11展示了在弹出式日期拾取器中一次显示多个月份的情况。

#### 代码清单21-11 在弹出式日期拾取器上一次显示多个月份

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#datep").datepicker({
            numberOfMonths: [1, 3]
        });
    });
...
});
```

```
</script>
...
```

这段脚本的结果见图21-10。用户不仅仅无法看到一些日期，而且因为next按钮（用户需要单击它才能访问后面的月份）超出了屏幕范围而无法单击。

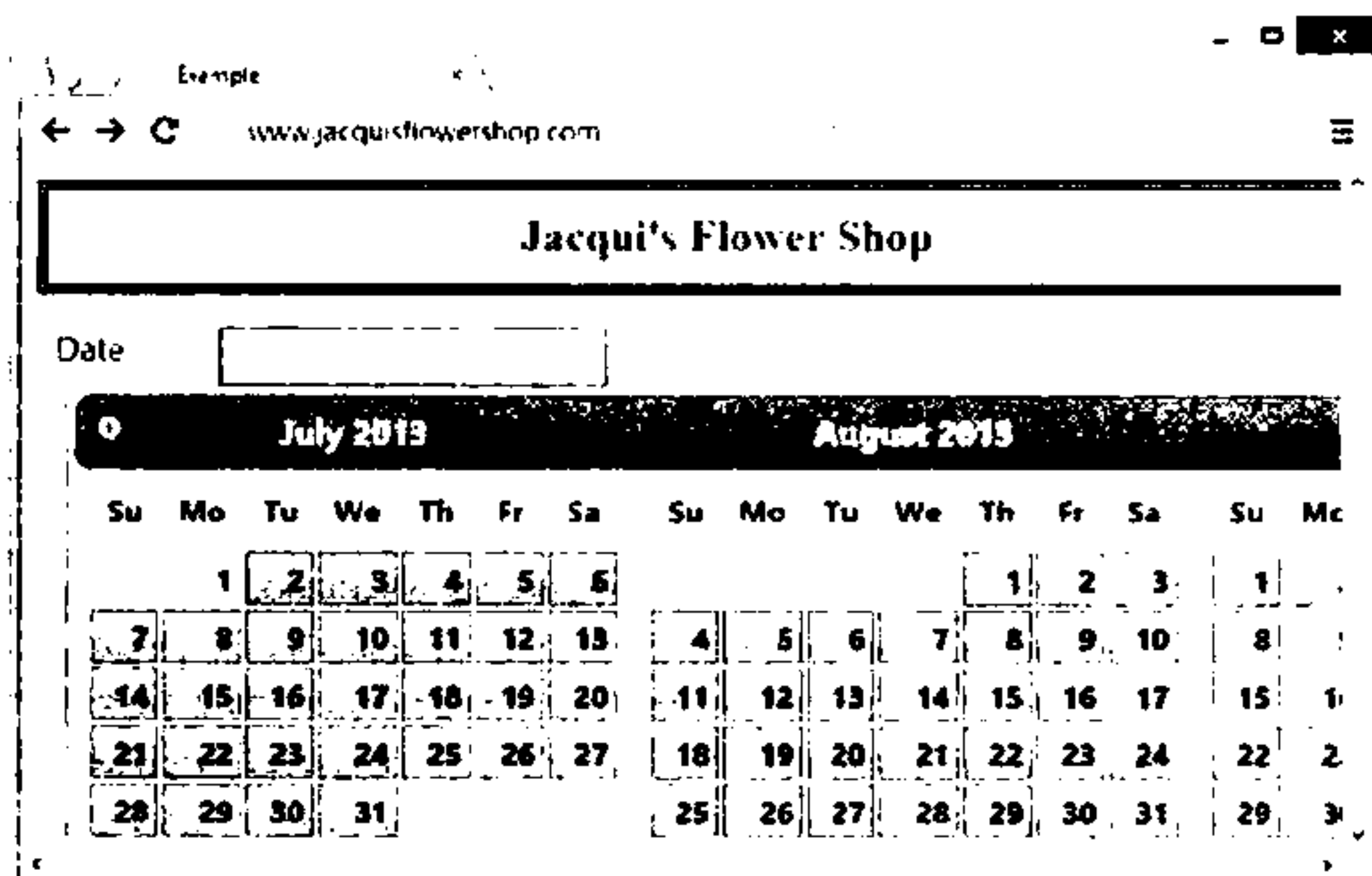


图21-10 一个巨大的弹出式日期拾取器

在一个多月份日期拾取器中，我们可以利用showCurrentAtPos选项改变当前日期的显示位置。在图21-9和图21-10中我们看到，默认情况下当前日期显示在第一个月份中，后面跟着未来的两个月份。showCurrentAtPos选项接受一个从0开始计数的序号，我们可以利用这个选项指定当前月份的显示位置。如果希望用户选择当前月前后的日期，这是一个非常方便的功能。代码清单21-12展示了这个选项的用法。

#### 代码清单21-12 使用showCurrentAtPos选项

```
...
<script type="text/javascript">
  $(document).ready(function(){
    $("#inline").datepicker({
      numberOfMonths: 3,
      showCurrentAtPos: 1
    });
  });
</script>
...
```

在这个例子中，我指定当前月份显示在日期拾取器的中间位置（参见图21-11）。

#### 3. 让用户直接选择某个月份和年份

我们可以把日期拾取器标题中的月份和年份替换为下拉菜单，这样用户就可以直接选择某个月份或者年份。如果可选日期范围很大的话，这对于用户来说是很方便的。changeMonth和changeYear选项用来控制这一特性。把这些选项设置为true，就会启用相应的下拉菜单。这两个菜单相互独立，互不影响。代码清单21-13展示了这两个选项的用法。

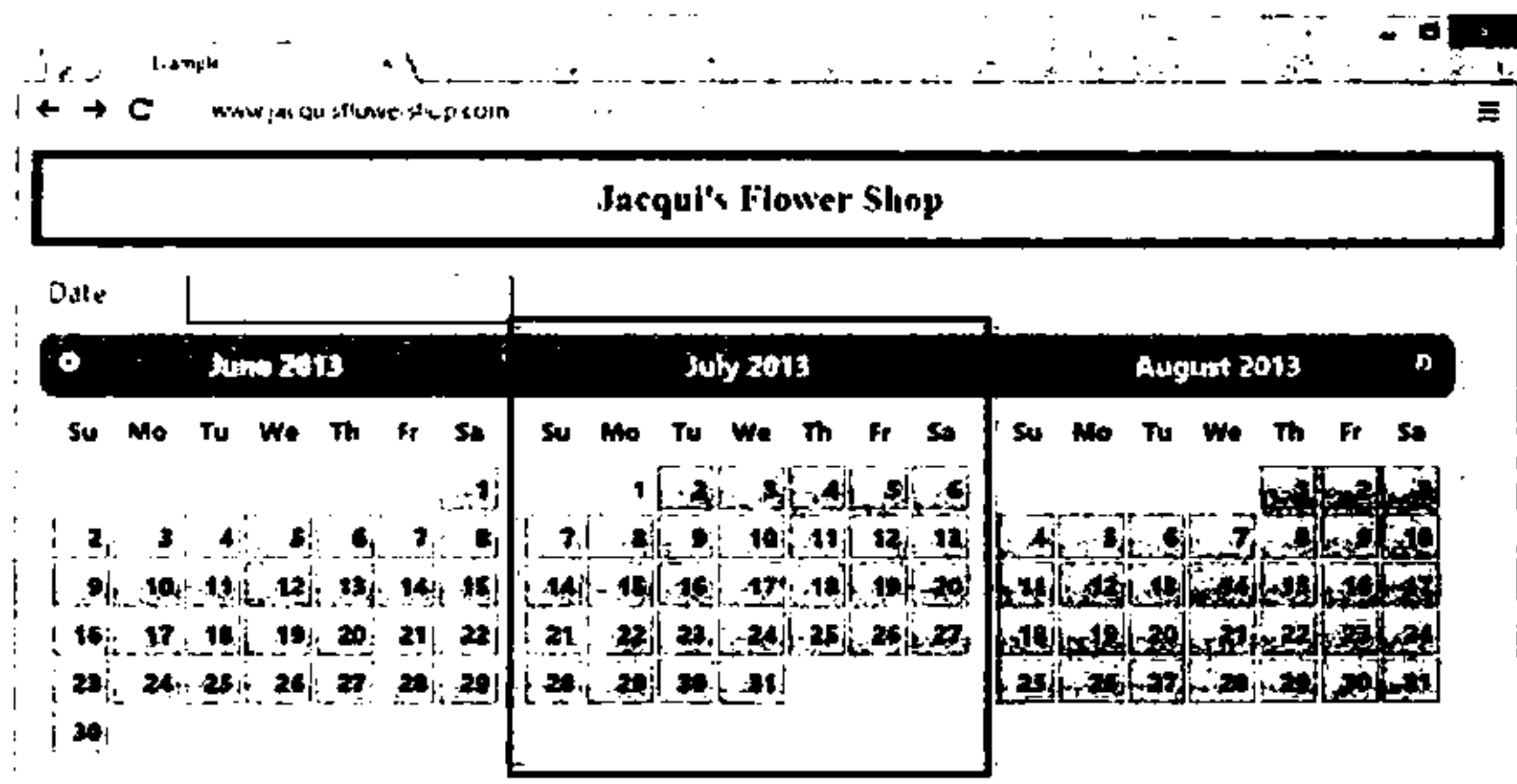


图21-11 在一个多月份日期拾取器中指定当前月份的显示位置

## 代码清单21-13 利用下拉菜单让用户直接选择某个月份

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#date").datepicker({
            changeMonth: true,
            changeYear: true,
            yearRange: "-1:+2"
        });
    });
</script>
...
```

在这个脚本中，我启用了两个下拉菜单。这里还使用yearRange选项限制了年份的选择范围。在这个例子里，我把yearRange选项的值设置为：-1:+2，这表示用户可以选择从今天起往前一年以及往后两年中的任何日期。我编写这一章的时间是2013年，因此用户看到的年份范围就是2012至2015。这些下拉菜单（以及年份的选择范围）参见图21-12。

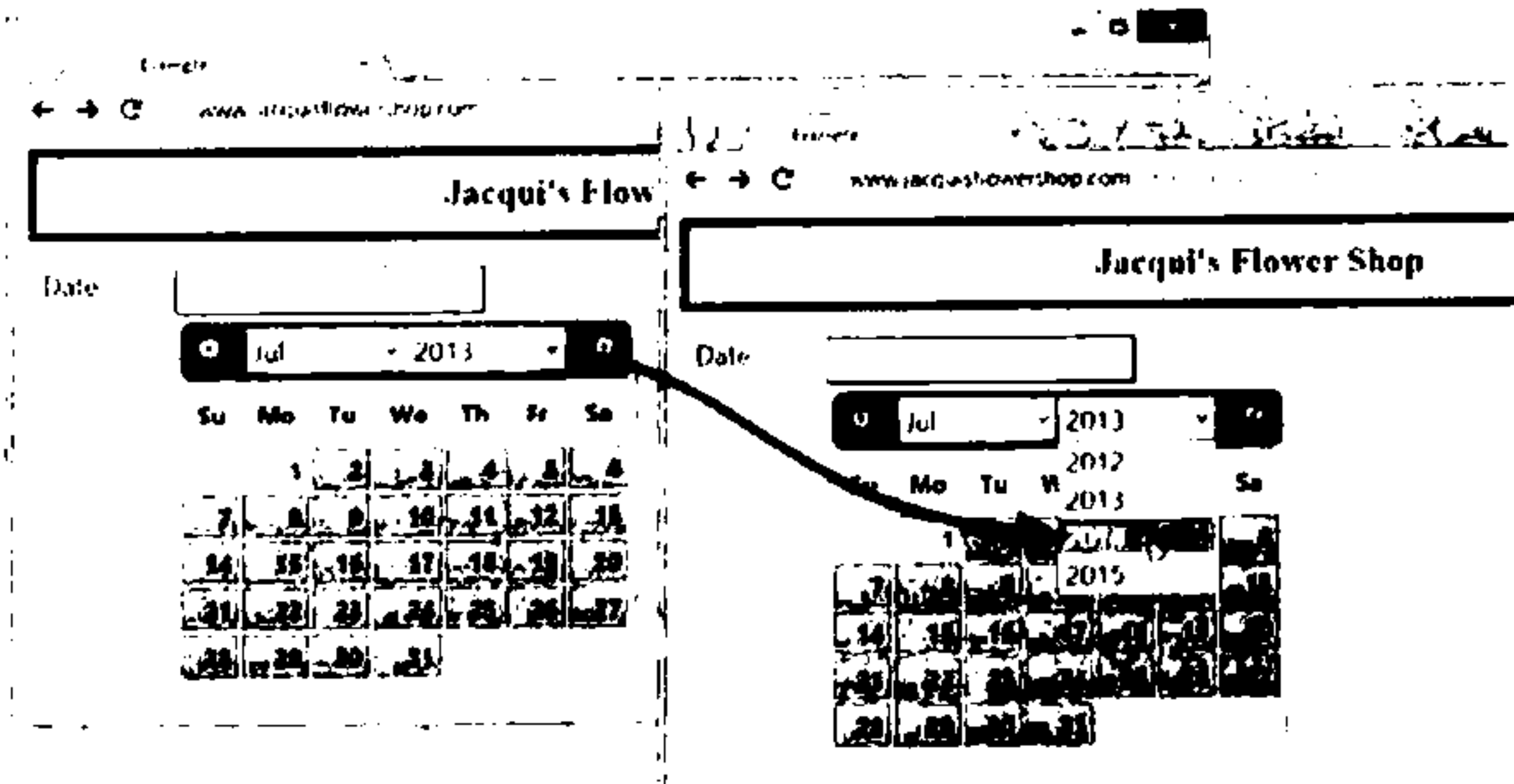


图21-12 让用户直达某个月份和年份

**提示** 我们也可以直接使用真实的年份来表示年份范围。比如说要达到和上面例子相同的效果（参见图21-12），我们也可以使用值2012:2015。

### 21.2.3 设置日期拾取器的外观

设置日期拾取器外观的选项很多。从前面的例子里我们知道，日期拾取器的默认外观已经能满足绝大多数选择日期的需求。不过，我们非常希望具有微调组件界面以使之满足某个Web应用程序需求的能力。代码清单21-5列出了日期拾取器的外观设置选项。

表21-5 日期拾取器的外观设置

选 项	描 述
appendText	在input元素之后增加一段文本，提示正确的日期格式
closeText	指定按钮面板负责关掉日期拾取器弹出窗口的按钮文本，默认值是Done
currentText	指定按钮面板负责返回当前日期的按钮文本，默认值是Today
duration	指定showAnim选项指定的动画效果的播放速度，默认值是normal。我会在第35章详细介绍jQuery UI动画特效
gotoCurrent	如果设置为true，那么按钮上的Today按钮将返回选中日期，而非今天的日期。默认值是false
selectOtherMonths	若设置为true，则通过showOtherMonths选项设置显示出来的其他月的日期也可以选中。默认值是false
showAnim	用来指定显示及隐藏日期拾取器时使用的动画。我会在第35章详细介绍jQuery UI动画特效。默认值是show
showButtonPanel	如果设置为true，那么在日期拾取器下方会出现一个按钮面板，允许用户返回当前日期及关闭日期拾取器窗口（如果是弹出式日期拾取器的话）。默认值是false
showOptions	为showAnim选项所设动画特效指定选项。我会在第35章详细介绍jQuery UI动画特效
showOtherMonths	默认值是false，如果设置为true，那么日期拾取器会使用上一个月和下个月的部分日期填充原本空白的网格
showWeek	默认值是false，如果设置为true，表示日期拾取器会增加一列以显示星期信息
weekHeader	如果showWeek选项为true的话，此选项用来设置星期信息列的标题。默认值是WK

#### 1. 显示星期信息

对于某些应用程序，知道一年的哪天属于这一年的第几个星期是很基础的需求，比如说那些预算管理类应用程序。如代码清单21-14所示，通过设置showWeek和weekHeader选项，jQuery UI日期拾取器能够显示星期信息。

代码清单21-14 在日期拾取器中显示星期信息

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#datep").datepicker({
            showWeek: true,
            weekHeader: "Week"
        });
    });
});
```

```
</script>
```

```
...
```

若showWeek选项的值为true, 则日期拾取器就会增加一列, 以显示星期信息。weekHeader选项用来改变这一列的默认标题(Wk)。在这个例子里, 我允许显示星期信息列, 并把这一列的标题改为Week。这个例子的结果见图21-13。

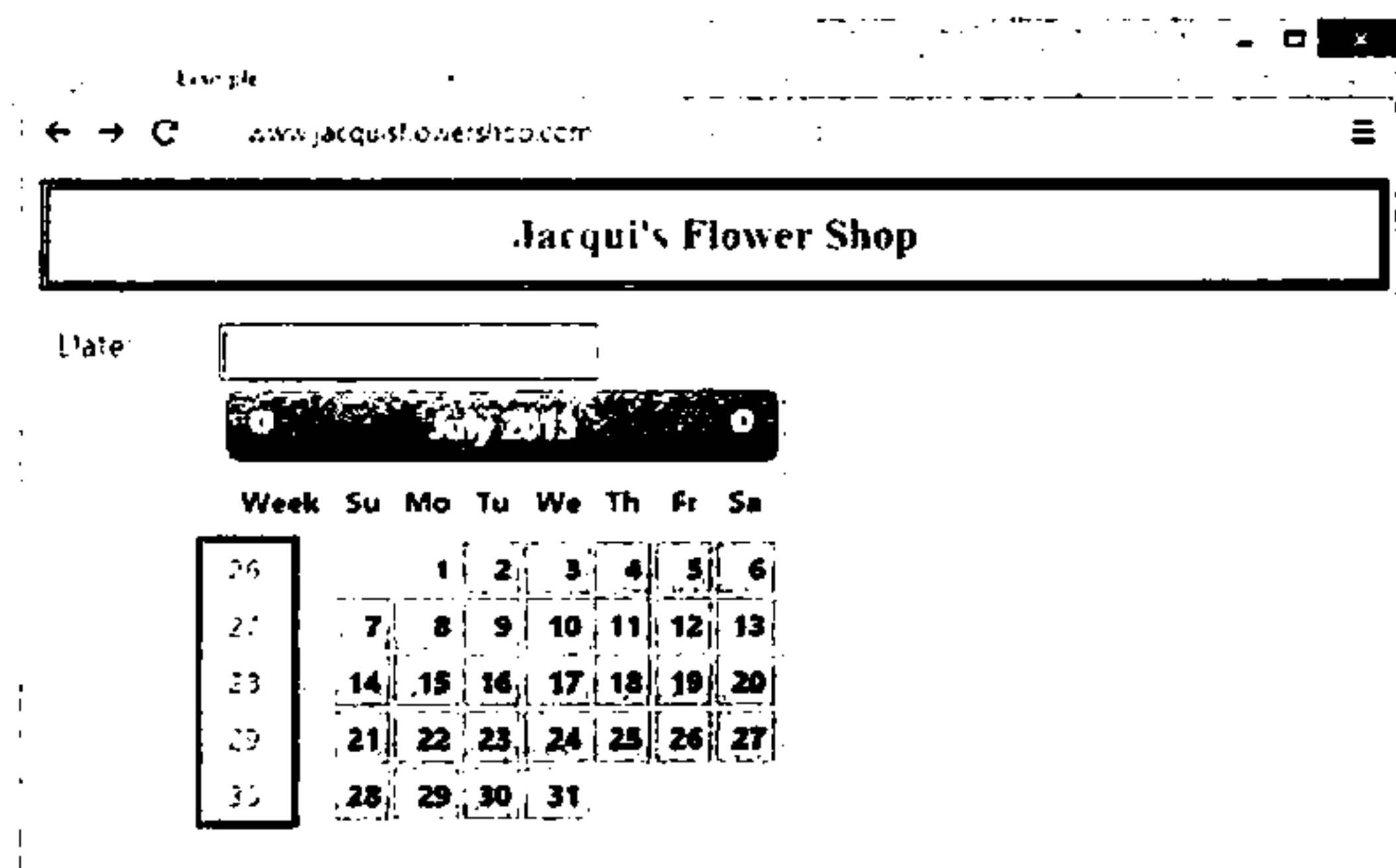


图21-13 在日期拾取器中显示星期信息

## 2. 让日期网格中的日期平滑衔接

默认情况下日期拾取器仅显示当月的日期, 也就是说在显示网格中, 本月起始日期之前和本月结束日期之后的网格都会保持空白。如代码清单21-15所示, 把showOtherMonths选项设置为true, 就可以让这些原本空白的网格显示出正确的日期(非当前月的)。

### 代码清单21-15 让日期网格中的日期平滑衔接

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#datep").datepicker({
            showOtherMonths: true
        });
    });
</script>
...
```

图21-14是这个例子的显示效果。其他月份的日期虽然能够看到, 但不能够选中(除非把selectOtherMonths选项设置为true)。

## 3. 按钮面板

当使用弹出式日期拾取器时, 如果把showButtonPanel选项设置为true, 那么日期拾取器窗口的下方会出现一排两个按钮(按钮面板): Today按钮和Done按钮。Today按钮用来返回当前日期, 而Done按钮会关掉弹出窗口。按钮面板的样子见图21-15。如果是嵌入式日期拾取器, 那么按钮面板中只显示Today按钮。

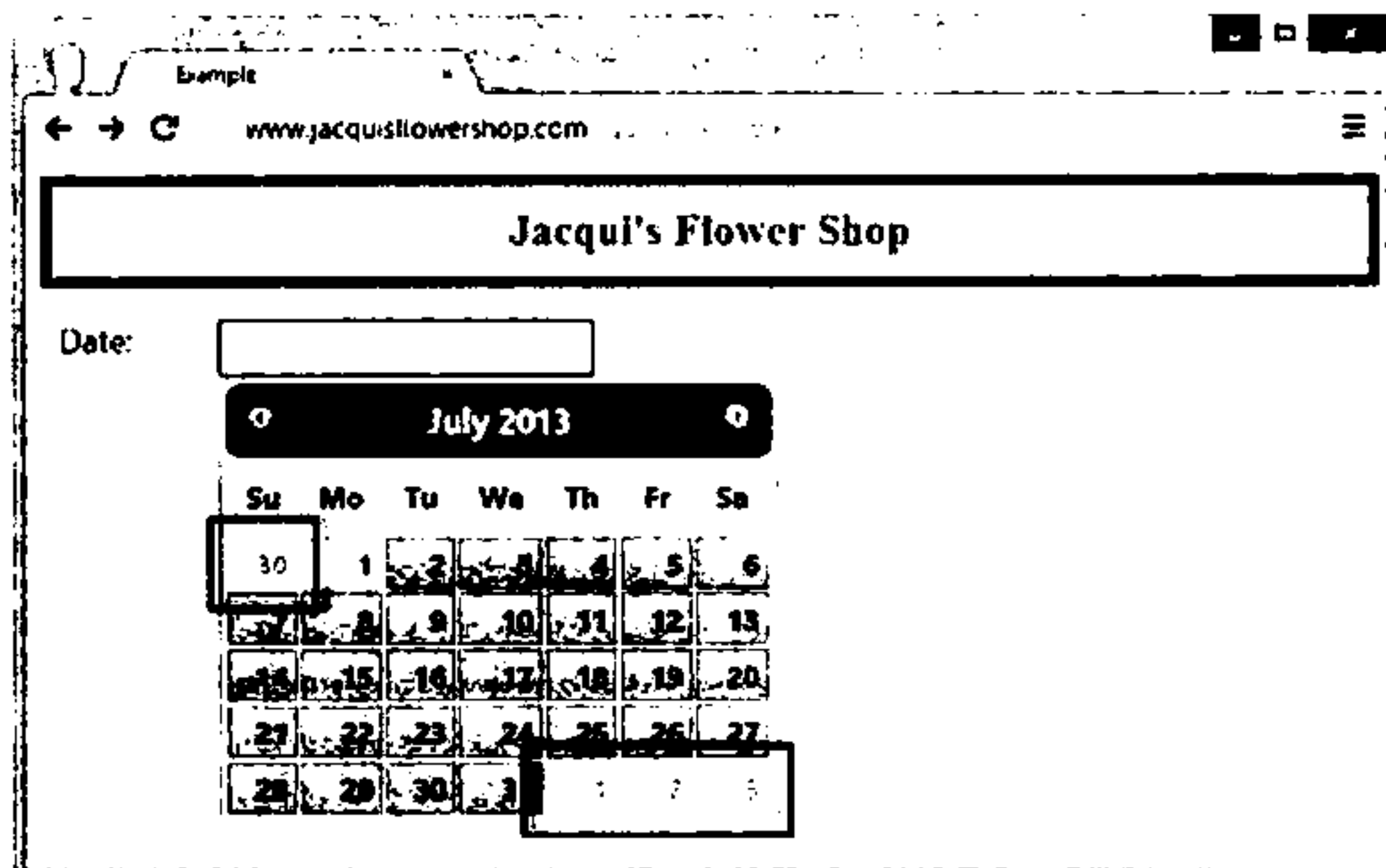


图21-14 显示上一个月及下一个月的部分日期

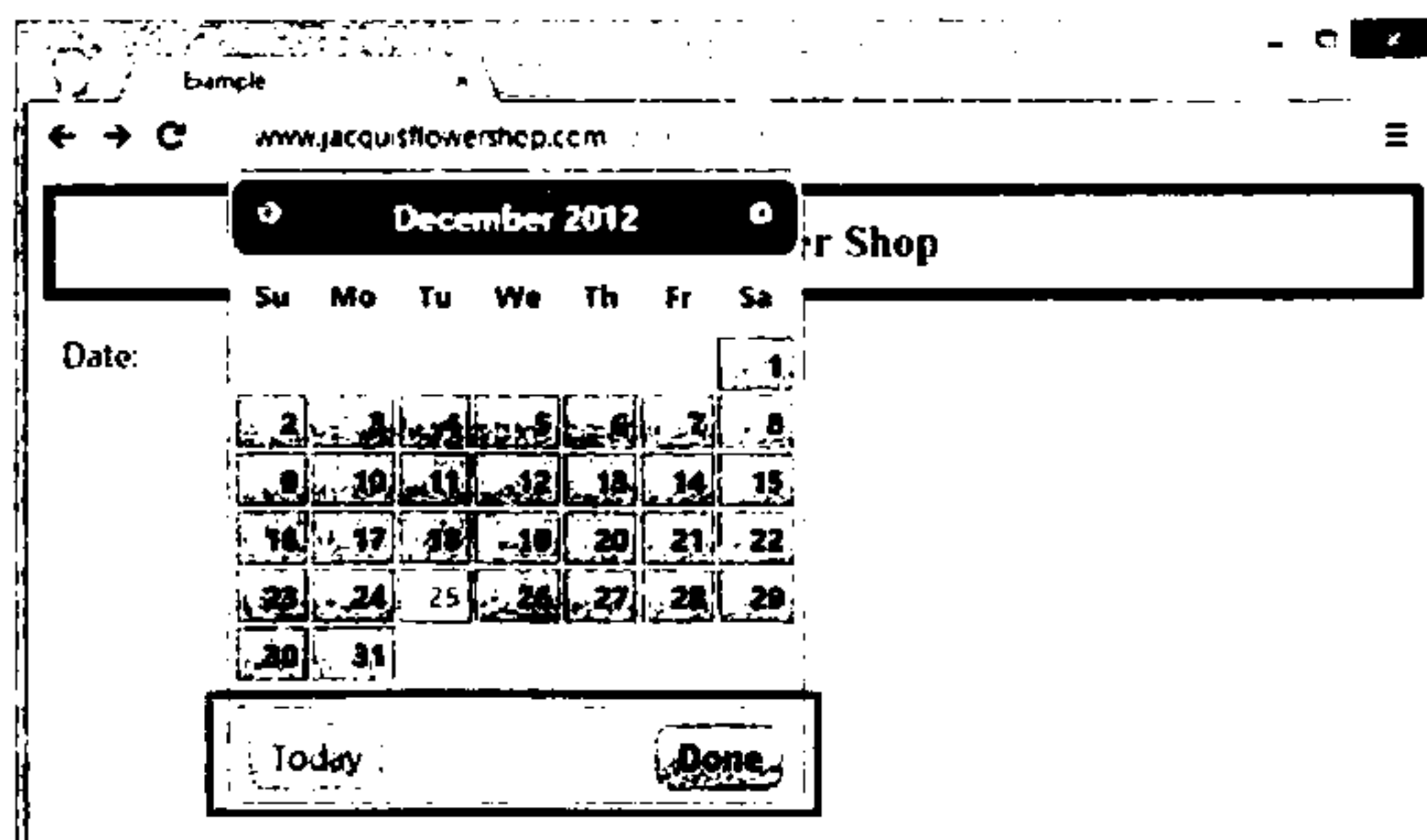


图21-15 按钮面板

**提示** 利用currentText选项和closeText选项，我们可以改变Today按钮和Done按钮上的文字。

如果把gotoCurrent选项设置为true，那么单击Today按钮将返回用户选中的日期，而不是今天的日期。如果我们设置了defaultDate，这个选项会比较有用，因为如果目的是让用户选择一个过去或者未来的日期，那么回到今天的日期就没什么意义。代码清单21-16给出了一个gotoCurrent选项的使用示例。

#### 代码清单21-16 使用gotoCurrent选项

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#datep").datepicker({
            gotoCurrent: true,
            showButtonPanel: true,
```

```

        gotoCurrent: true,
        defaultDate: "+1m +1y"
    }).val("12/25/2012");
    });
</script>
...

```

注意，gotoCurrent选项使得Today按钮回到用户选中的日期。在这个例子里，单击Today按钮会回到input元素值所代表的日期。不过，如果用户选择了另一个日期，并且稍后又再次打开日期拾取器，Today按钮将返回用户选择的那个日期，而不再是我们指定的默认日期。

#### 4. 提示用户正确的日期格式

我们可以利用appendText选项告诉用户正确的日期格式。具体的例子见代码清单21-17。

代码清单21-17 使用appendText选项设置显示格式的提示信息

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $("#datep").datepicker({
            appendText: '(mm/dd/yyyy)'
        });
    });
</script>
...

```

如图21-16所示，日期拾取器把我们指定的文本插入到页面当中。

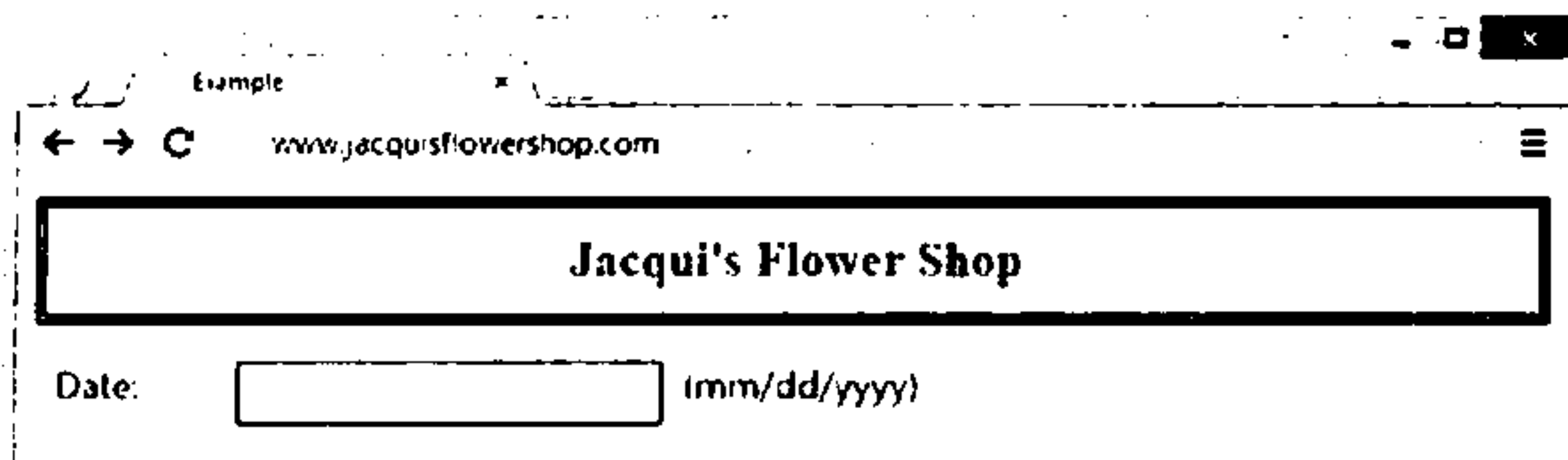


图21-16 利用appendText选项提示用户正确的日期格式

如果我们依赖用户单击按钮来弹出日期拾取器，这个选项就太有用了。因为当用户不使用日期拾取器自由输入日期时，提示信息将有助于提示他们填写正确的格式，从而有效地减少错误日期的数量。（这不仅对我们有帮助，而且能减少用户的挫败感。）最近，我发现HTML5的placeholder属性（input元素）是日期拾取器appendText选项的更好替代品。具体的例子见代码清单21-18。

代码清单21-18 利用HTML5的placeholder属性提供格式提示信息

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $("#datep").attr("placeholder", "mm/dd/yyyy").datepicker();
    });
</script>
...

```



没错，这要求用户必须使用支持HTML5的浏览器，不过它的效果更优雅。用户会看到灰色的提示文本，而一旦开始键入，这些提示文本就自动消失。我更喜欢这个，因为它比appendText选项提供的文本与input元素结合得更紧密，而且不需要额外的空间（不会破坏页面布局）。图21-17展示了placeholder属性中的文本在Google Chrome浏览器中的显示效果。

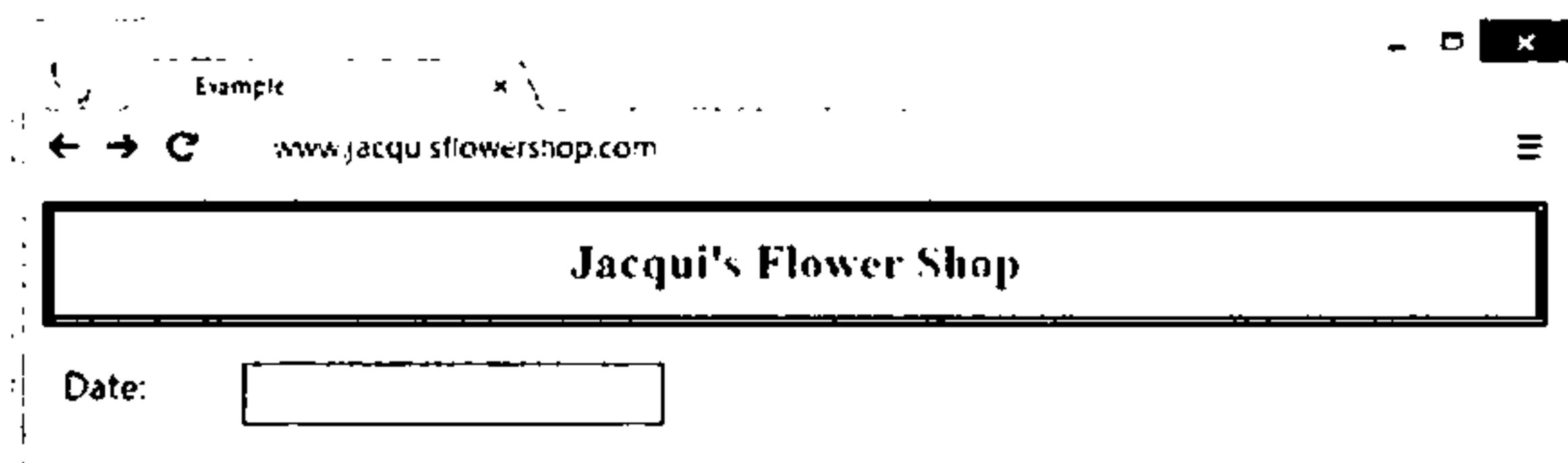


图21-17 利用HTML5 placeholder属性提示日期格式

## 21.3 日期拾取器方法

表21-6列出了日期拾取器支持的组件方法。

表21-6 日期拾取器方法

方 法	描 述
<code>datepicker("destroy")</code>	从底层元素上移除日期拾取器组件
<code>datepicker("disable")</code>	禁用日期拾取器
<code>datepicker("enable")</code>	启用日期拾取器
<code>datepicker("option")</code>	设置日期拾取器选项
<code>datepicker("isDisabled")</code>	如果日期拾取器已被禁用，返回true
<code>datepicker("hide")</code>	隐藏弹出式日期拾取器
<code>datepicker("show")</code>	显示弹出式日期拾取器
<code>datepicker("refresh")</code>	刷新日期拾取器以反映底层元素值的变化
<code>datepicker("getDate")</code>	从日期拾取器组件中得到当前选中日期
<code>datepicker("setDate", date)</code>	为日期拾取器设置选中日期

### 21.3.1 使用脚本获取和设置日期

在生成多个嵌入式日期拾取器以帮助用户选择一段日期时，我发现getDate和setDate方法最有用。在这种场合，我并不想把选中日期显示在input元素里，而是希望显示第一个日期和第二个日期之间的天数。代码清单21-19演示的就是这种用法。

代码清单21-19 使用两个日期拾取器实现选取日期区间的功能

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
```



```

<script src="jquery-2.0.2.js" type="text/javascript"></script>
<script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<style type="text/css">
    input {width: 200px; text-align: left; margin-right: 10px}
    #wrapper > * {float: left}
    #result {margin: auto; padding: 10px; width: 200px; clear: left}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $("#result").hide();

        $("#dateStart, #dateEnd").datepicker({
            minDate: "-7d",
            maxDate: "+7d",
            onSelect: function(date, datepicker) {
                if (datepicker.id == "dateStart") {
                    $("#dateEnd").datepicker("setDate", date)
                        .datepicker("enable").datepicker("option", "minDate", date)
                }

                if (!$("#dateEnd").datepicker("isDisabled")) {
                    var startDate = $("#dateStart").datepicker("getDate");
                    var endDate = $("#dateEnd").datepicker("getDate");
                    var diff = endDate.getDate() - startDate.getDate();
                    $("#dayCount").text(diff).parent().show();
                }
            }
        }).filter("#dateEnd").datepicker("disable");
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="wrapper" class="ui-widget">
            <label for="dateStart">Start: </label><span id="dateStart"></span>
            <label for="dateEnd">End: </label><span id="dateEnd"></span>
        </div>
        <div id="result" class="ui-widget">
            Number of Days: <span id="dayCount"></span>
        </div>
    </form>
</body>
</html>

```

在这个例子里有两个日期拾取器，第二个日期拾取器最开始（在页面第一次载入时）是被禁用的。我使用onSelect事件（当日期被选中时触发）响应用户选择日期的行为。当用户在第一个日期拾取器里作出选择之后，我使用setDate方法准备第二个日期拾取器，最后使用getDate方法得到两个日期拾取器的当前日期，然后计算出两个日期之间的天数。（为了使例子尽可能简单，在做比较时我假定两

个日期属于同一个月。) 浏览器里这个页面的显示效果见图21-18。

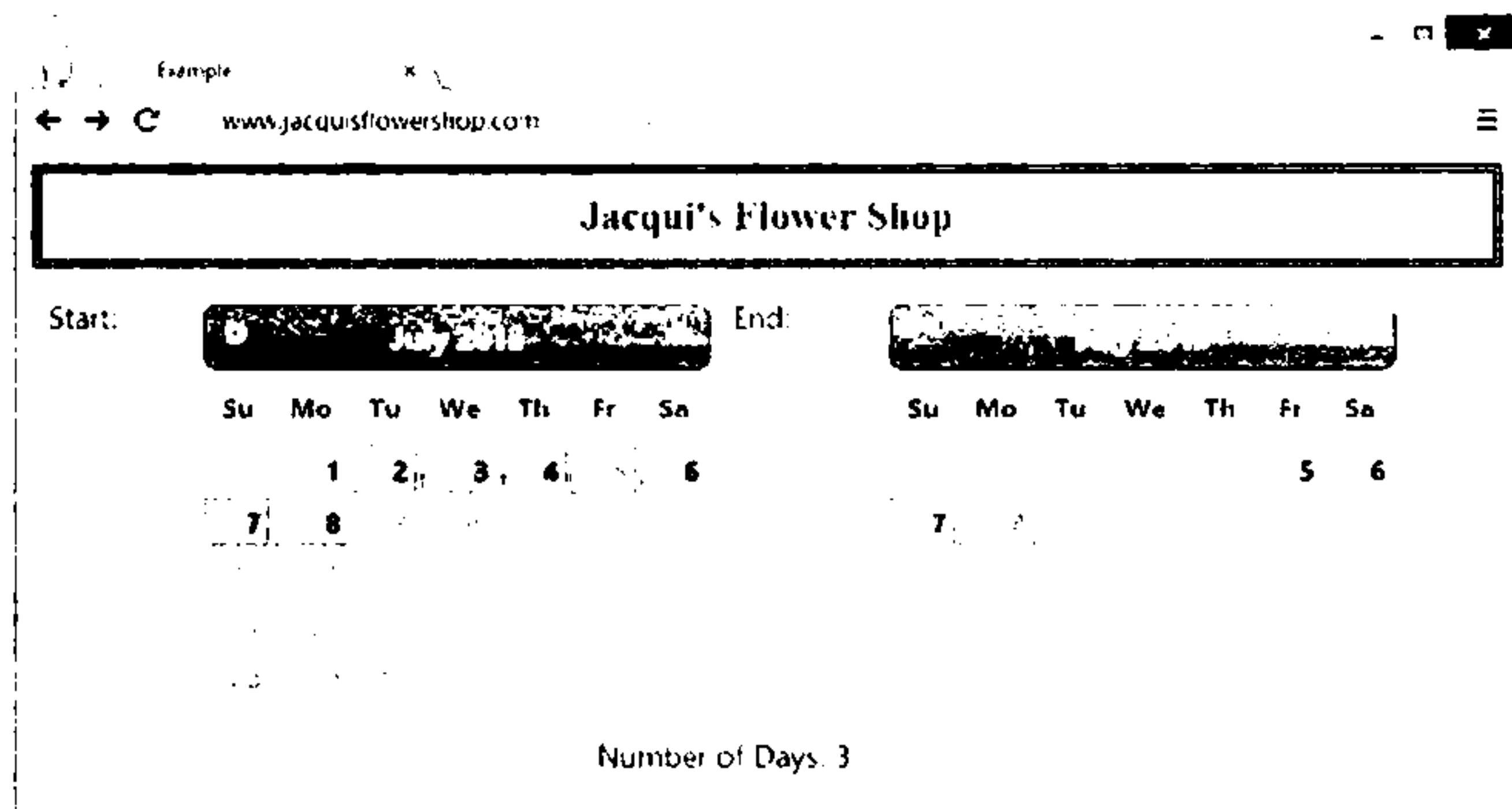


图21-18 使用getDate和setDate方法

### 21.3.2 使用脚本控制弹出式日期拾取器的显示和隐藏

在脚本中，我们可以使用show和hide方法控制弹出式日期拾取器的显示状态。如果不想利用input元素的focus事件或者日期拾取器生成的按钮，而是希望我们自己的某个什么东西来触发日期拾取器的弹出，这两个方法就可派上用场了。如代码清单21-20所示，我不是很喜欢使用日期拾取器生成的按钮来激活日期拾取器，因此有时候会自己在页面中添加一个按钮，然后使用这些方法控制日期拾取器的显示。

代码清单21-20 使用show方法和hide方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    input {width: 200px; text-align: left; margin-right: 10px}
    #wrapper > * {float: left}
    label {padding: 4px; text-align: right; width: auto}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      $("#datep").datepicker();

      $("#button").click(function(e) {
```

```

        e.preventDefault();
        $("#datep").datepicker("show");
        setTimeout(function() {
            $("#datep").datepicker("hide");
        }, 5000)
    });

});
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="wrapper" class="ui-widget">
            <label for="datep">Date: </label><input id="datep"/><span id="inline"></span>
            <button>Datepicker</button>
        </div>
    </form>
</body>
</html>

```

当单击按钮时，我调用日期拾取器的show方法。我较少使用hide方法，因为希望用户能够在选择一个日期后主动关闭拾取器。不过为了例子的完整性，如果用户单击了按钮，我使用setTimeout函数让弹出的日期拾取器在显示5秒钟之后自动关闭。

## 21.4 日期拾取器事件

类似于所有其他jQuery UI组件，日期拾取器组件也支持一系列事件，允许我们在重要的事情发生之后得到通知。表21-7列出了这些事件。

表21-7 日期拾取器事件

事 件	描 述
create	在日期拾取器组件创建之后触发
onChangeMonthYear	当用户跳到一个新的月或者新的年时触发
onClose	当弹出式日期拾取器被关闭后触发
onSelect	当用户选中一个日期时触发

我不想在这里演示onSelect事件，因为我们在前面的两个例子（包括21.3.1节中的一个例子）里已经使用过这个事件了。onSelect事件处理函数会接受两个参数，第一个是字符串表示的选中日期，第二个是触发此事件的日期拾取器对象。

### 21.4.1 响应月份或年份的变更

利用onChangeMonthYear事件，不论用户从下拉菜单（使用changeMonth和changeYear选项生成的）里选择一个月份或者年份，还是使用previous和next按钮改变月份或者年份，我们都可以响应用户行为。代码清单21-21演示了如何利用这一事件让两个日期拾取器保持一致。

## 代码清单21-21 使用onChangeMonthYear事件

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    input {width: 200px; text-align: left; margin-right: 10px}
    #wrapper > * {float: left}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      $("#dateStart, #dateEnd").datepicker({
        onSelect: function(date, datepicker) {
          if (datepicker.id == "dateStart") {
            $("#dateEnd").datepicker("setDate", date)
          }
        },
        onChangeMonthYear: function(year, month, datepicker) {
          if (datepicker.id == "dateStart") {
            var newDate = new Date();
            newDate.setMonth(month - 1);
            newDate.setYear(year);
            $("#dateEnd").datepicker("setDate", newDate);
          }
        }
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="wrapper" class="ui-widget">
      <label for="dateStart">Start: </label><span id="dateStart"></span>
      <label for="dateEnd">End: </label><span id="dateEnd"></span>
    </div>
  </form>
</body>
</html>

```

这个事件的处理函数会接受3个参数，分别是显示的年份、显示的月份以及触发此事件的日期拾取器对象。**this**变量被设置为与日期拾取器相关联的input元素。当用户在第一个日期拾取器里改变月份或者年份时，我使用**setDate**方法设置第二个日期拾取器的日期，以便使二者同步（保持一致）。

注意，在日期拾取器组件中1月表示为1，而在JavaScript的Date对象中却是用0表示1月。这就是我不不得不在第二个日期拾取器设置日期之前，写出下面这行丑陋代码的原因：

```
...
newDate.setMonth(month -1);
...
```

## 21.4.2 响应关闭弹出式日期拾取器事件

onClose事件用来响应弹出式日期拾取器的关闭。即使用户没有选择任何日期，这个事件也会被触发。这个事件的处理函数会接受一个字符串表示的当前日期（如果用户没有选择任何日期就关闭了日期拾取器，这个值就是一个空字符串）和触发此事件的日期拾取器对象。代码清单21-22中的示例简单响应了这样一个事件。

代码清单21-22 使用onClose事件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    input {width: 200px; text-align: left; margin-right: 10px}
    #wrapper > * {float: left}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#datep").datepicker({
        onClose: function(date, datepicker) {
          if (date != "") {
            alert("Selected: " + date);
          }
        }
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="wrapper" class="ui-widget">
      <label for="datep">Date: </label><input id="datep"/>
    </div>
  </form>
</body>
</html>
```

在这个例子里，如果用户选择了一个日期，示例会通过显示alert对话框显示用户选择的日期。必须承认，我从来没有在真实项目中使用过这一事件；在我看来，这些事件中的onSelect事件最有用。

## 21.5 本地化日期拾取器

jQuery UI的日期拾取器组件广泛支持世界上各式各样的日期格式。具体来讲，jQuery UI支持61个地区的使用格式。要利用这一特性，我们需要在页面中额外导入一个JavaScript脚本文件，并告诉日期拾取器组件我们打算支持的地区。具体的例子见代码清单21-23。

代码清单21-23 使用本地化的日期拾取器

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <script src="jquery-ui-i18n.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    input {width: 200px; text-align: left; margin-right: 10px}
    #wrapper > * {float: left}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#inline").datepicker($.datepicker.regional["es"]);
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="wrapper" class="ui-widget">
      <label for="datep">Date: </label><input id="datep"/><span id="inline"></span>
    </div>
  </form>
</body>
</html>
```

jquery-ui-i18n.js文件可以在第17章创建的自定义jQuery UI安装包的development-bundle/ui/i18n目录下找到。把这个文件复制到jQuery及jQuery UI库所在的目录，然后使用下面这行代码把它添加到页面当中：

```
...
<script src="jquery-ui-i18n.js" type="text/javascript"></script>
...
```

然后像下面这行语句一样设置创建日期拾取器时打算支持的地区：

```
...
$("#inline").datepicker($.datepicker.regional["es"]);
...
```

不得不说这种语法很差劲。不过，利用它我们确实能够指定期望的区域日期格式。在这个例子里，我指定的是es，它表示将使用西班牙人所用的日期格式。图21-19展示了这个例子的显示效果。

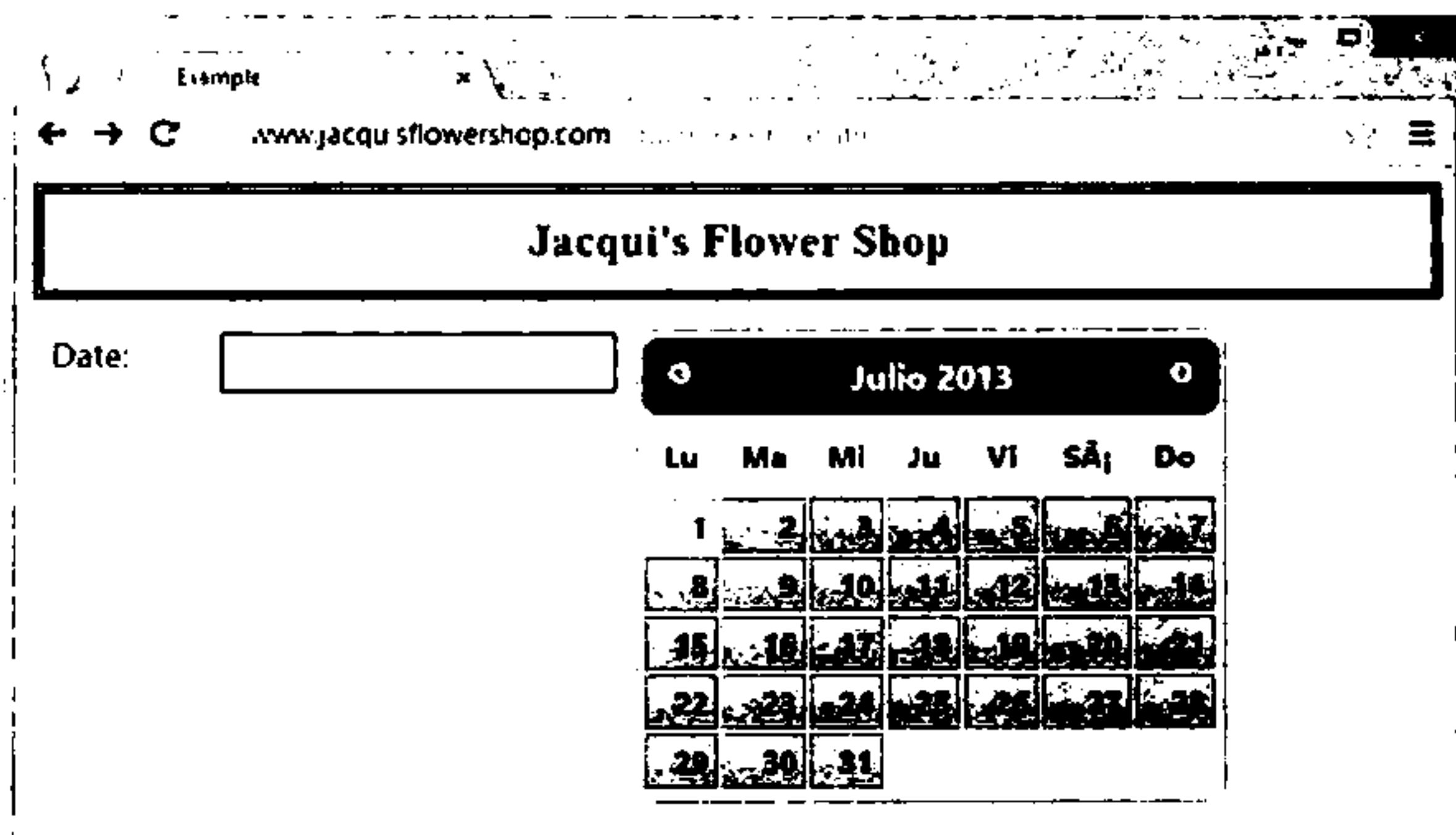


图21-19 显示本地化日期

我的建议是，本地化这个东西要么设置得当，要么干脆不设置。本地化的话题很大，不是日期格式这么简单，这意味着要完全遵守本地化的语言、地址、性别、货币单位、时间等其他方面的用户习惯。在一个Web应用程序中，只有一个部分实现了本地化，或者各个部分遵循的习惯不一致，只会使用户哭笑不得。要恰当地实现本地化，我们需要特别雇用某个人或者某个公司来做这项工作。如果缺少专业的支持，事情很容易就会被搞砸。

如果你正在使用Google Translate进行应用程序的本地化工作（这并不罕见），我建议使用美国英语或者美国地区的习惯部署应用程序。这虽然会将你的顾客限制为那些熟悉美国文化（拼写习惯、货币单位等）的人，但是至少避免了不恰当的本地化尝试几乎总会发生的灾难性事故。

## 21.6 小结

本章介绍了jQuery UI的日期拾取器组件，我们可以利用它方便用户选择日期。日期拾取器组件非常灵活，支持多种多样的选项，利用这些选项我们能够轻松配置选择日期的方式和日期拾取器的外观。就我个人经验来说，当不得不从用户那里获取日期信息时，在减少日期格式错误问题这一方面，日期拾取器“功德无量”。下一章的主题是jQuery UI的对话框组件和微调控制组件。

## 对话框组件和微调控制组件

本章主要介绍jQuery UI对话框组件和微调控制组件。表22-1列出了本章概要。

表22-1 本章概要

问 题	解决方法	代码清单
如何生成jQuery UI对话框	选中一个有着title属性的div元素，然后调用dialog方法	1
如何让对话框生成之后不显示	把autoOpen选项设置为false	2
如何阻止用户改变对话框大小	把resizable选项设置为false	3
如何在对话框中设置焦点	设置autofocus属性	4
如何改变对话框的初始显示位置	使用position选项	5
如何在对话框中添加一个或多个按钮	使用buttons选项	6
如何在对话框按钮上显示图标	使用icons选项	7
如何让对话框支持拖拽移动位置	使用draggable选项	8
如何生成一个模态对话框	把modal选项设置为true	9、10
如何使用脚本打开和关闭对话框	使用open、close及isOpen方法	11
如何阻止用户关闭对话框	让beforeClose事件处理函数返回false	12
如何响应用户移动或改变对话框尺寸的事件	响应dragStart、dragStop、drag、resizeStart、resizeStop和resize事件	13
如何生成微调控制组件	选中input元素并调用spinner方法	14~16
如何配置微调范围和行为	使用min、max、step以及page选项	17
如何改变微调按钮使用的图标	使用icons属性	18
长按微调按钮时，如何控制值改变的速度	使用incremental选项	19
如何改变微调数字显示的格式	使用culture和numberFormat选项	20
如何使用脚本改变微调控制组件的值	调用pageDown、pageUp、stepDown以及stepUp方法	21
如何响应微调控制组件值的变化	处理spinner事件	22

## 新版JQUERY UI与本章有关的变化

在编写本书第一版时，jQuery UI中还没有微调控制组件。随着jQuery UI升级到1.10版本，对话框组件的API（编程接口）也进行了更新，增加了一些新特性。其中，新增了appendTo选项，这让我们能够指定对话框元素插入DOM（文档对象模型）的位置；position选项改用我在第19章中讲过的格式；对话框中的按钮也开始支持显示图标。此外，对话框组件还删除了两个特性：stack选项和zIndex选项。



## 22.1 jQuery UI 对话框组件

利用jQuery UI对话框组件，我们可以得到一个有着标题区和内容区的平面窗口，它有点像我们熟悉的原生应用程序的对话框。当重要的事件发生，或者需要展示一条重要的信息时，它能够有效的吸引用户的注意力。不过，由于对话框总是会遮挡住部分页面内容，我们应该仅在必要时使用（即要显示的内容不适合显示在页面布局内时）。

### 22.1.1 创建对话框

使用jQuery选中一个div元素并调用dialog方法，就创建了一个对话框组件。尽管与折叠菜单组件相比，对话框组件的HTML结构要简单得多，但它也需要专门的HTML结构才能正常运转。代码清单22-1展示了创建对话框组件所需的HTML元素和结构。

代码清单22-1 使用jQuery UI生成对话框

```
<DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#dialog").dialog();
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="dialog" title="Dialog Box">
    This is the content that will be displayed in the dialog box. This <b>content</b>
    can be <em>styled</em>.
  </div>
</body>
</html>
```

对话框组件需要一个有title属性的div元素，对话框标题栏的文字就是title属性的值。div元素的内容将作为对话框的内容，如代码清单22-1所示，内容区还可以包含其他元素。就像在这个代码清单中所做的那样，当不带任何参数调用dialog方法时，对话框将会立即出现。对话框在浏览器中的显示效果见图22-1。

这个对话框由巧妙搭配的HTML元素生成，与操作系统无关，这意味着jQuery UI对话框与原生对话框的行为大不相同。例如，它不会显示在操作系统的窗口清单里，而且当我们改变浏览器窗口大小时，jQuery UI对话框可能会被部分甚至全部遮住。

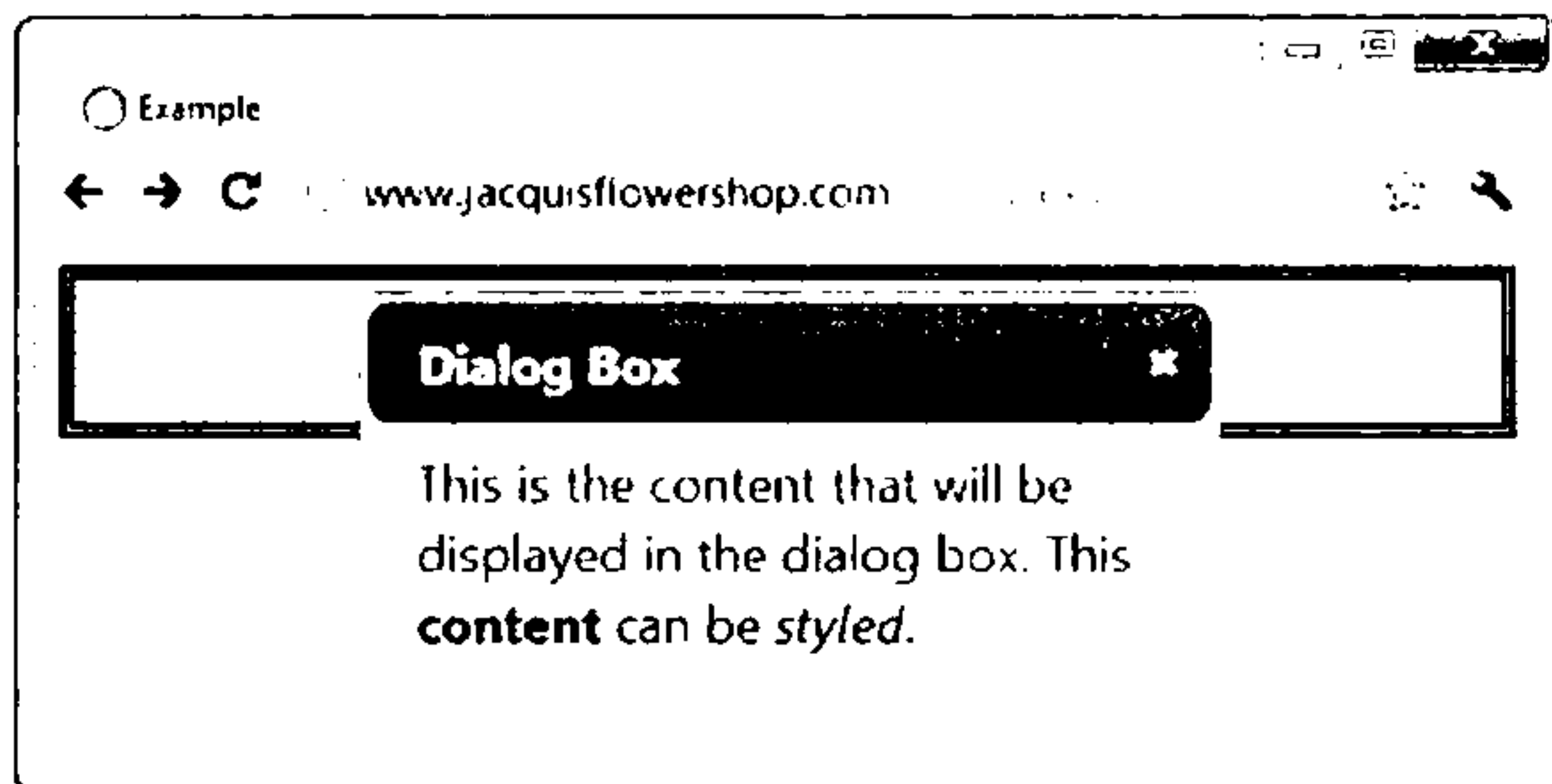


图22-1 一个简单的对话框

凭心而论，jQuery UI团队出色的工作使对话框尽几近完美。单击标题并拖曳，用户可以在浏览器窗口范围内改变对话框的位置。你可以改变对话框的大小，而若打算关闭对话框，只需单击右上角的关闭按钮。而且，由于jQuery UI对话框组件是由HTML构建而成，它的样式可以使用我们在第17章选择的jQuery UI主题，而且它能包含复杂且样式化的HTML内容。

在深入介绍对话框组件的选项、方法和事件之前，我要先演示一下对话框组件的一个常见用法和第一个例子相同，当不带任何参数调用dialog方法时，dialog对话框立刻就显示出来，不过这并不总是最好的行为。来看一个更常见的情况：我们在页面加载完成之后，创建一个对话框（但不显示它），然后等一件什么事情发生时再把它显示出来。代码清单22-2演示的就是这种用法。

#### 代码清单22-2 延迟jQuery UI对话框的显示时间

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#dialog").dialog({
        autoOpen: false
      });
      $("button").button().click(function(e) {
        $("#dialog").dialog("open");
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="dialog" title="Dialog Box">
    This is the content that will be displayed in the dialog box. This <b>content</b>
```

```
        can be <em>styled</em>.
```

```
    </div>
```

```
    <button>Show Dialog</button>
```

```
</body>
```

```
</html>
```

我们使用autoOpen阻止对话框立即显示。把这个选项设置为false，HTML元素就处于隐藏状态，用户看不到对话框。当我们需要显示对话框时，就调用open方法。图22-2演示的就是这种工作方式。

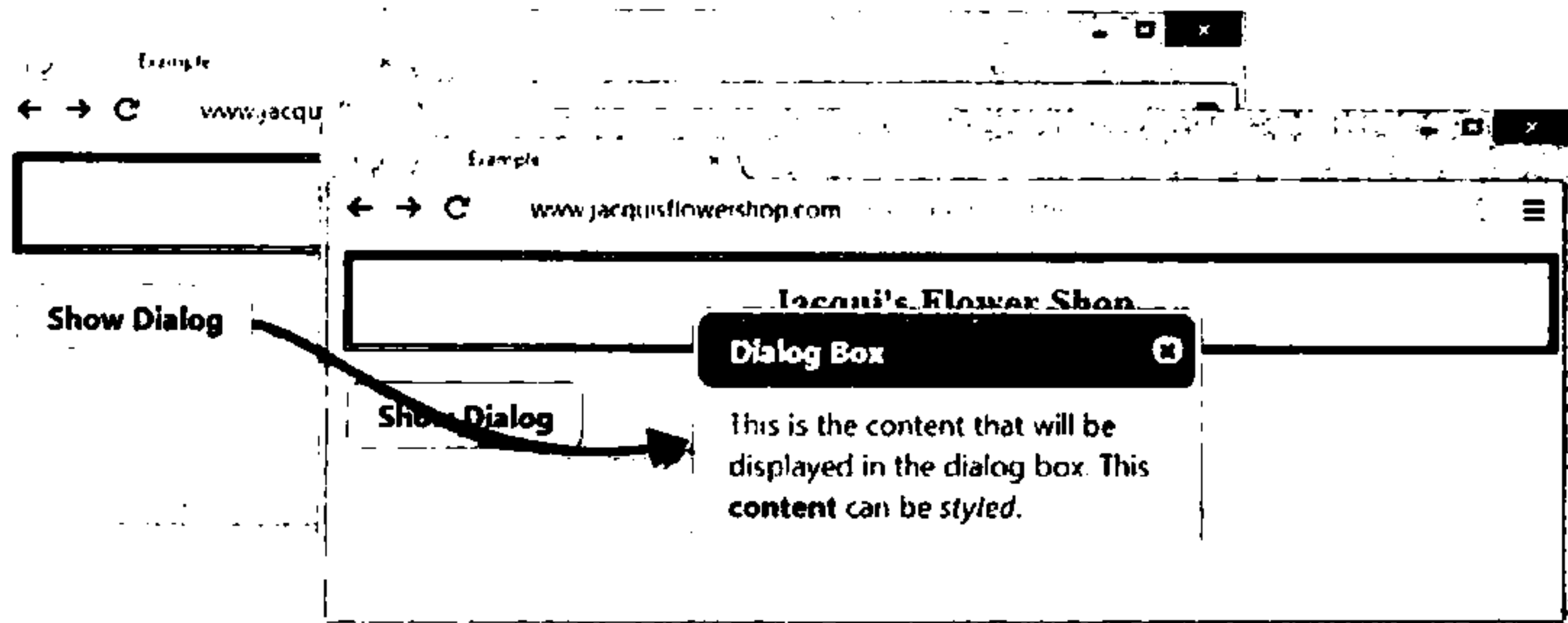


图22-2 创建对话框但不显示，直到用户单击Show Dialog按钮

### 22.1.2 配置对话框

对话框组件支持许多选项，这些选项可以用于定制对话框的显示方式。在上一节中，我已经展示过autoOpen选项，其他选项见表22-2。

表22-2 对话框组件选项

选 项	描 述
appendTo	指定对话框元素在DOM中的插入位置
autoOpen	默认值是true，表示对话框一旦创建就立即显示
buttons	指定要在对话框中添加的按钮以及单击这些按钮时需要调用的回调函数，默认没有任何按钮
closeOnEscape	默认值是true，表示用户按下Esc键会关闭对话框
draggable	默认值是true，用户可以单击标题区域并在浏览器窗口区域内拖曳对话框
height	指定对话框的初始高度（单位像素），默认值是auto，表示由对话框根据内容自动确定高度
hide	指定关闭对话框的动画效果。欲详细了解jQuery UI特效，请阅读第35章
maxHeight	指定对话框的最大高度，默认值是false，表示没有高度限制
maxWidth	指定对话框的最大宽度，默认值是false，表示没有宽度限制
minHeight	指定对话框的最小高度，默认值是false，表示没有最小高度限制
minWidth	指定对话框的最小宽度，默认值是false，表示没有最小宽度限制
modal	默认值是false，若设置为true，则表示对话框为模态窗口：在用户关闭窗口之前，不能与页面进行交互
position	指定对话框的初始位置

(续)

选 项	描 述
Resizable	默认值是true, 表示对话框会在右下角显示一个把手, 用户可以通过拖曳这个把手改变对话框的大小
show	指定显示对话框的动画效果。欲详细了解jQuery UI特效, 请阅读第35章
title	指定对话框的标题
width	指定对话框的初始宽度(单位像素), 默认值是auto, 表示由对话框根据内容自动确定宽度

**提示** jQuery UI 1.10 增加了appendTo选项, 从而允许我们指定对话框元素插入DOM的位置。另一个新特性是对话框组件现在支持让内容元素自动获得焦点。(稍后会演示这一特性。)除此之外, 还有一些别的变化: position选项值的格式有变化, stack和zIndex配置项则被删除。

### 1. 配置对话框的基本外观

利用title选项, 我们可以利用一个没有title属性的div元素生成对话框。在没有条件控制生成对话框的div元素时, 这个选项非常有用。代码清单22-3演示了title选项的用法。

**代码清单22-3 使用title选项**

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#dialog").dialog({
        title: "Hello",
        resizable: false
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="dialog">
    This is the content that will be displayed in the dialog box. This <b>content</b>
    can be <em>styled</em>.
  </div>
</body>
</html>
```

我在这个例子里还使用了resizable选项。这个选项阻止用户改变对话框的大小。我个人喜欢不带拖曳把手的对话框, 因为看起来比较清爽。不过, 我通常还是会让这个选项保留为true状态, 这样用户就有机会调整对话框的大小以使之适应较多或者较少的内容。图22-3展示了这个对话框在浏览器中的样子。

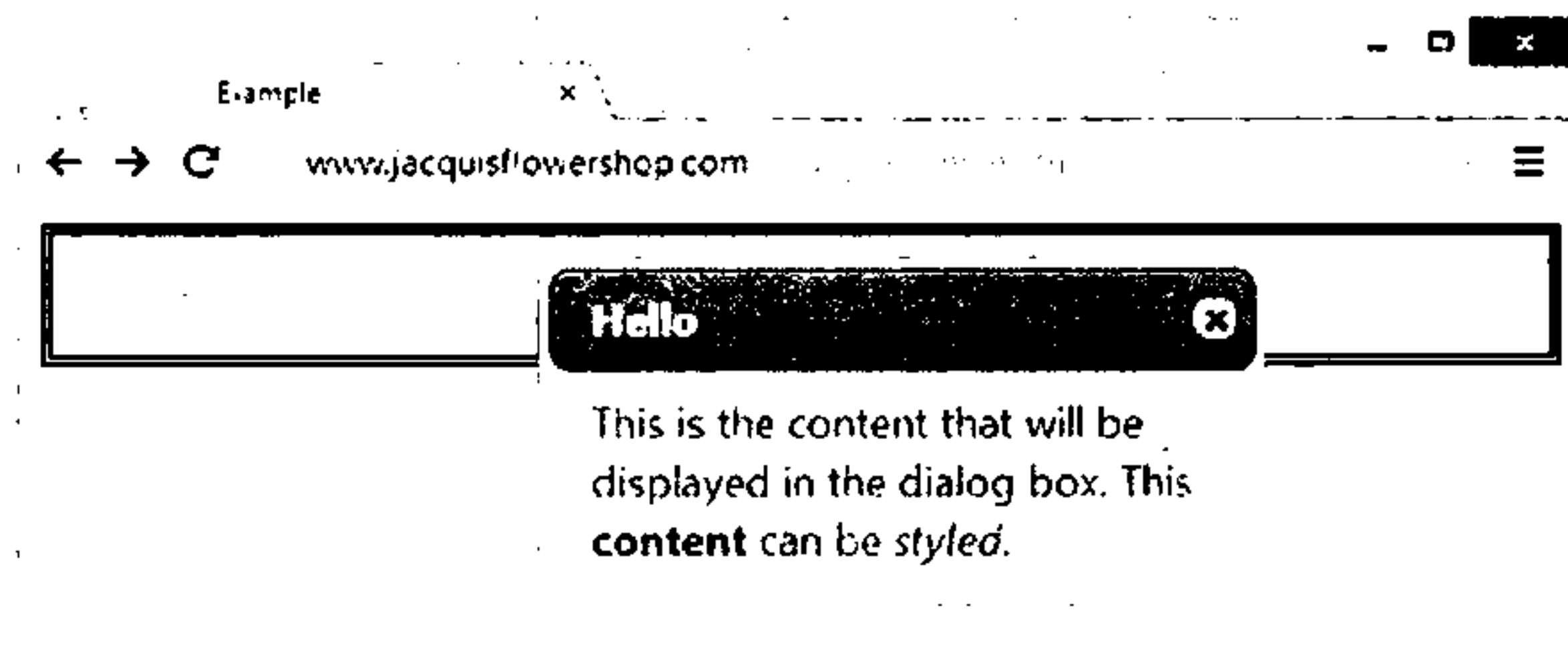


图22-3 一个有着自定义标题并且不能改变大小的对话框

对话框组件会自动为它找到的第一个具有autofocus属性的内容元素设置焦点,当显示一个对话框让用户输入数据时,这个特性非常有用。参见代码清单22-4。

#### 代码清单22-4 在内容元素上使用自动聚焦属性

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#dialog").dialog({
        title: "Hello",
        resizable: false
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="dialog">
    This is the content that will be displayed in the dialog box. This <b>content</b>
    can be <em>styled</em>.
    <p>Name: <input id="name"/> City: <input id="city" autofocus="autofocus" /></p>
  </div>
</body>
</html>
```

我在对话框中添加了一些input元素,并为第二个input元素添加了autofocus属性。在图22-4中,你可以清晰地看到,第二个input元素自动拿到了焦点。

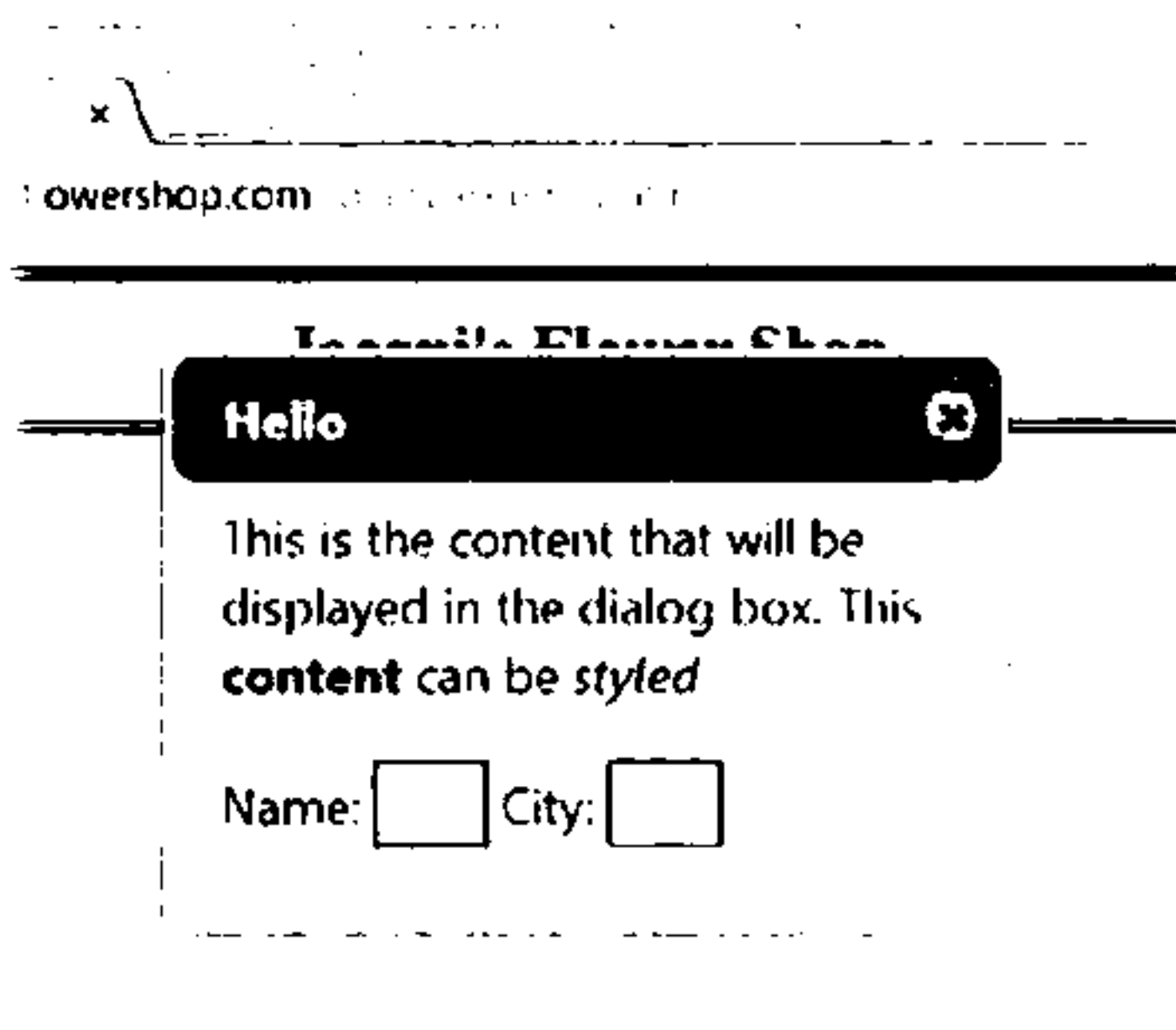


图22-4 autofocus属性的效果

## 2. 设置对话框的显示位置

`position`选项用来指定对话框在浏览器窗口中的显示位置。如表22-3所示, `position`选项使用了与自动完成弹出组件(第19章)完全相同的属性和值。

表22-3 position 属性

名 称	描 述
<code>my</code>	指定是对话框的什么位置相对于目标元素定位
<code>at</code>	指定 <code>my</code> 属性定义的对话框位置位于目标元素的什么位置
<code>of</code>	指定对话框的目标元素, 默认值为 <code>body</code>
<code>collision</code>	在对话框遮住窗口时, 指定调整对话框位置的方式

`my`和`at`属性使用空格分隔的两个值指定水平与垂直位置。水平位置可取值为: `left`、`right`和`center`, 垂直位置可取值为`top`、`bottom`和`center`。代码清单22-5中演示了这两个属性的用法。(关于其他位置属性的表示方法, 请参阅第19章。)

### 代码清单22-5 对话框定位

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#dialog").dialog({
        title: "Positioned Dialog",
        position: {
          my: "left top",
          at: "left top"
        }
      });
    });
  </script>
</head>
</html>
```

```

    }
  });
});
</script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="dialog">
    This is the content that will be displayed in the dialog box. This <b>content</b>
    can be <em>styled</em>.
  </div>
</body>
</html>

```

我把对话框的左上角放到了父元素（默认的body元素）的左上角（图22-5）。

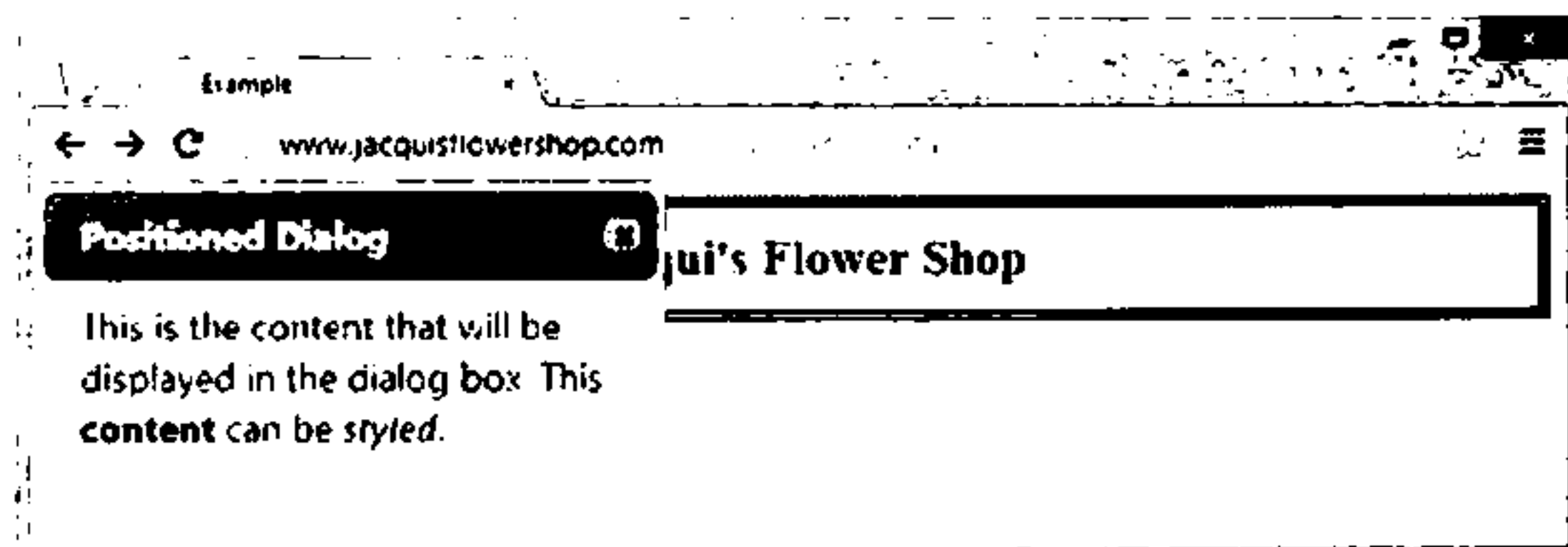


图22-5 指定对话框的显示位置

### 3. 在对话框中添加按钮

利用buttons选项，我们可以在对话框中添加按钮。这个选项的值是一个对象数组，其中的每一个对象都有两个属性：text和click。text属性的值是按钮文本，click属性的值是单击该按钮时要调用的回调函数。代码清单22-6展示了这个选项的用法。

#### 代码清单22-6 在对话框中添加按钮

```

...
<script type="text/javascript">
  $(document).ready(function() {
    $('#dialog').dialog({
      title: "Dialog",
      buttons: [{text: "OK", click: function() {/* do something */}},
               {text: "Cancel", click: function() {$(this).dialog("close")}}]
    });
  });
</script>
...

```

在这个脚本中，我添加了两个按钮。OK按钮不做任何事，而Cancel按钮会关掉对话框。注意，我在Cancel回调函数中使用的this变量，这个变量指向创建对话框所使用的div元素。这两个按钮在对话框中的显示位置见图22-6。这个例子使用了close方法，它会关掉对话框。我会在本章后面的部分介绍对话框组件的方法。

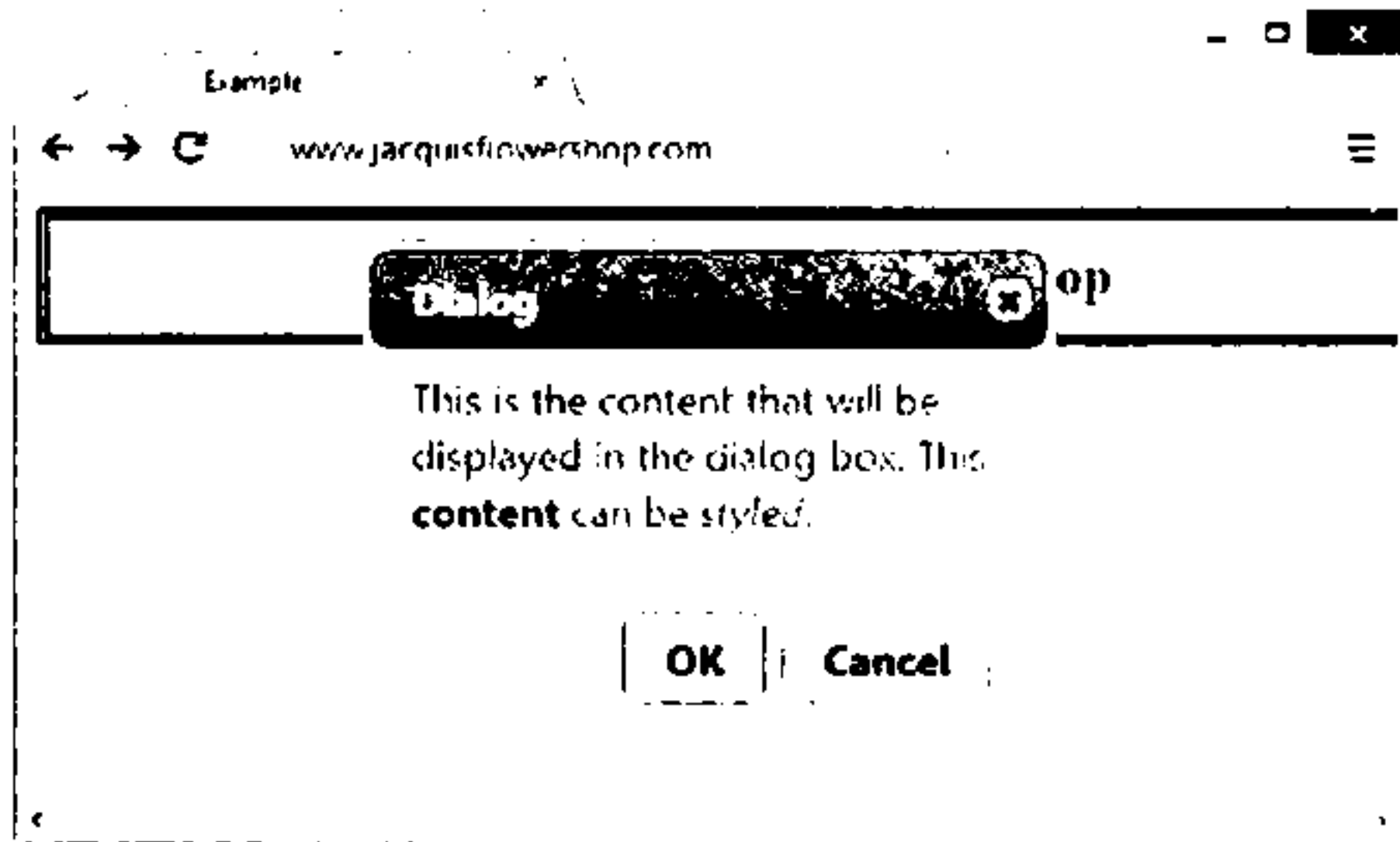


图22-6 在jQuery UI对话框组件中添加按钮

jQuery UI 1.10版本支持为对话框按钮指定图标，只需在定义每个按钮时为它指定icons属性即可，见代码清单22-7。

#### 代码清单22-7 为对话框组件中的按钮添加图标

```
...
<script type="text/javascript">
    $(document).ready(function () {
        $("#dialog").dialog({
            title: "Dialog",
            buttons: [{
                text: "OK",
                click: function () { /* do something */ },
                icons: {
                    primary: "ui-icon-star",
                    secondary: "ui-icon-circle-arrow-e"
                }
            }, { text: "Cancel", click: function () { $(this).dialog("close") } }
        ]);
    });
</script>
...
```

icons属性的值是一个对象，它有两个属性：primary和secondary，分别对应按钮文本左侧与右侧的图标。这两个属性指定图标的方式与按钮组件完全相同，具体参考第18章。代码清单中添加按钮图标代码之后的效果见图22-7。

**提示** 在定义按钮的对象（不是icons对象）上增加showText属性，并把值设置成false，可以禁止显示按钮上的文字。

#### 4. 拖曳对话框

draggable选项决定用户是否能够在浏览器窗口中拖曳对话框。这个选项的默认值是true，你最好不要修改这一选项，它使用户有机会查看被遮住的内容。这在使用对话框显示某种错误信息时或者系统出了什么问题时尤为重要。如果把draggable选项设置为false，用户就无法移动这个对话框了。



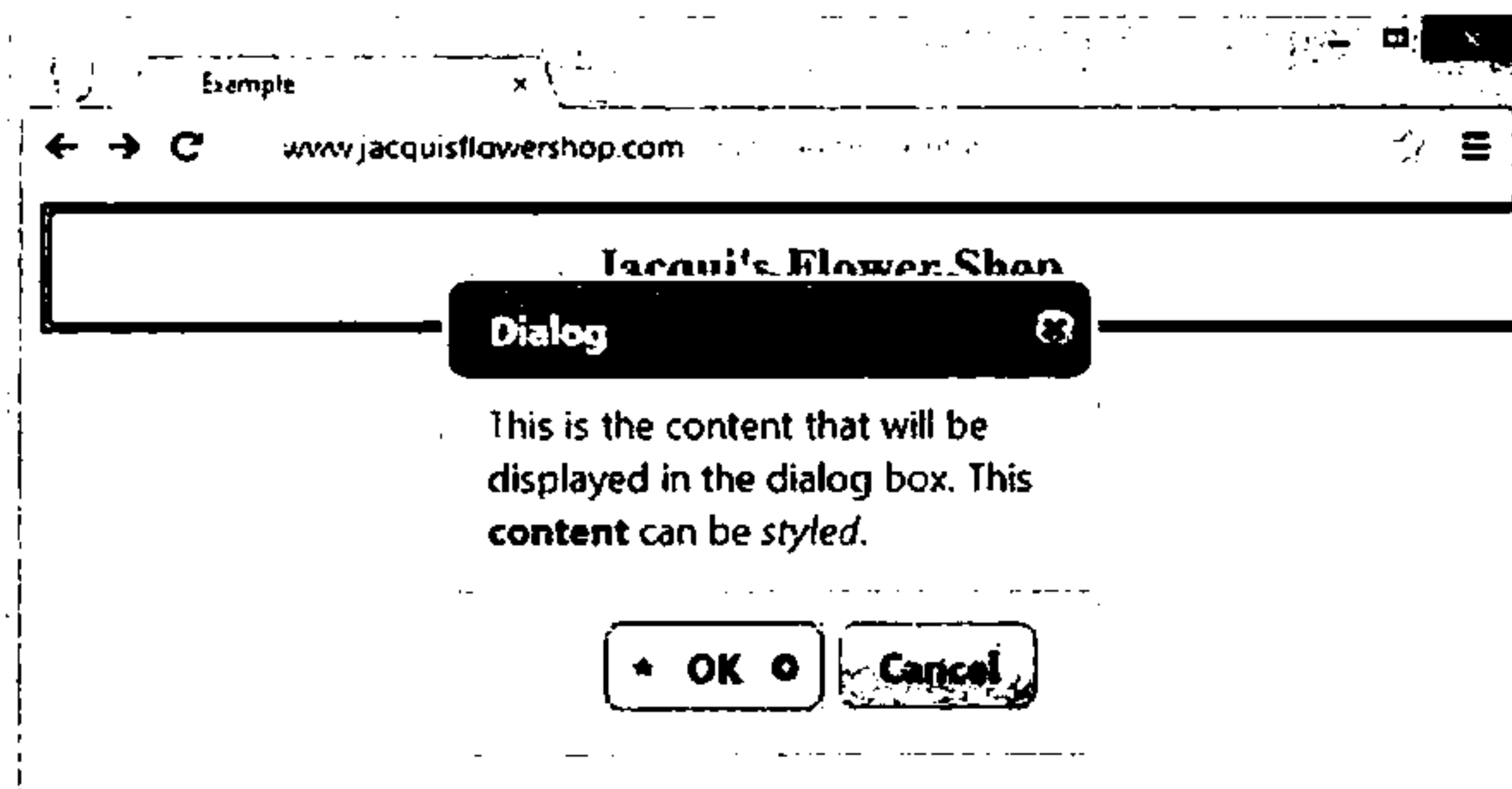


图22-7 在对话框内的按钮上使用图标

当我们在一个窗口中用到多个对话框时，`draggable`选项也就变得重要起来。我并不建议在一个窗口里同时显示多个对话框，不过如果不得不这么做，就得允许用户重新安排这些对话框的位置，以方便阅读它们。在小屏幕设备上，这些对话框往往堆在一起，参见代码清单22-8。

#### 代码清单22-8 使用`draggable`选项

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function () {
      $(".dialog").dialog({
        draggable: true
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="d1" class="dialog" title="First Dialog">
    This is the first dialog
  </div>
  <div id="d2" class="dialog" title="Second Dialog">
    This is the second dialog
  </div>
  <div id="d3" class="dialog" title="Third Dialog">
    This is the third dialog
  </div>
</body>
</html>
```

我在页面中放了3个对话框(使用位置默认值),使它们叠加在一起。将draggable选项设置为true,这样它们就支持拖拽移动位置了,如图22-8所示。

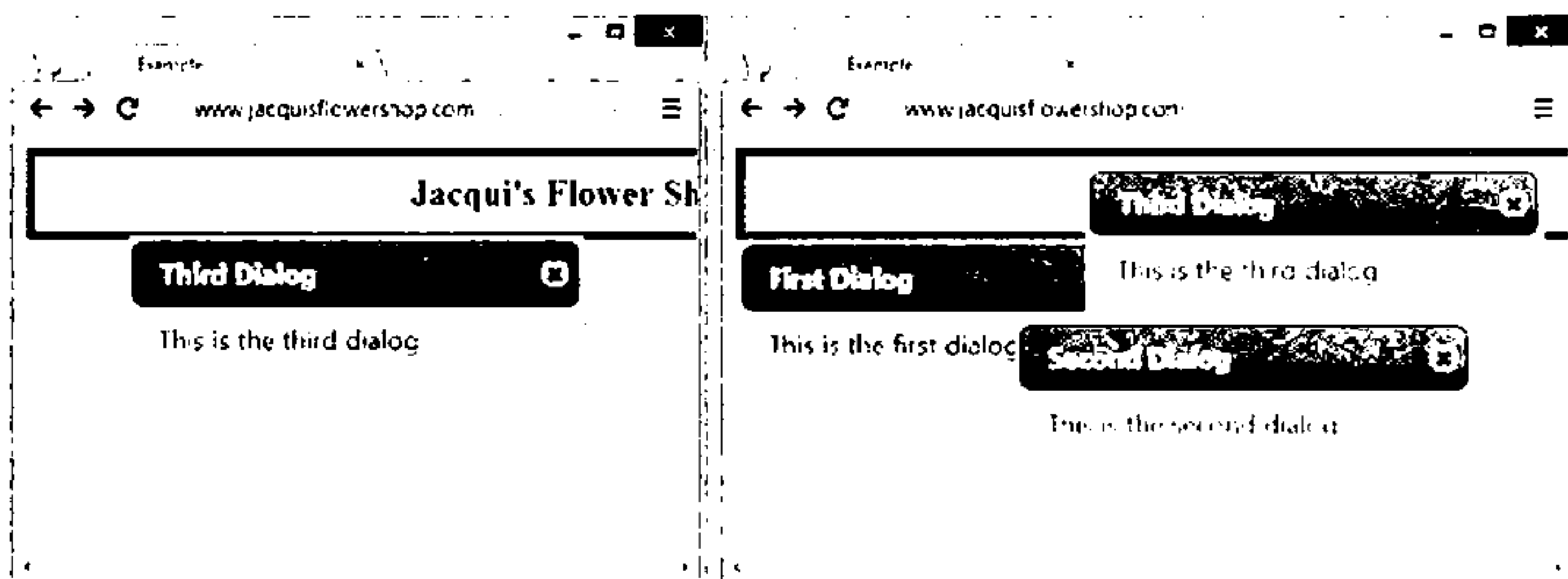


图22-8 拖拽对话框

### 5. 创建模态窗口

一个模态对话框在关掉之前,禁止用户与页面其他元素进行交互。把modal选项设置为true,我们就得到了一个模态对话框,参见代码清单22-9。

#### 代码清单22-9 创建模态窗口

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#dialog").dialog({
        buttons: [{text: "OK", click: function() {$(this).dialog("close")}}],
        modal: true,
        autoOpen: false
      });
      $("#show").button().click(function() {
        $("#dialog").dialog("open");
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="dialog" title="Modal Dialog">
    This is a modal dialog. Press OK to continue.
  </div>
  <button id="show">Show Dialog</button>
</body>
</html>
```

在这个例子里，我创建了一个模态对话框，但不让它立即显示。当用户单击Show Dialog按钮时，这个对话框就显示出来。模态对话框的实际效果见图22-9。这个例子依赖于open和close方法，我们分别用它显示和关掉对话框。在本章后面的部分，我会介绍对话框组件定义的所有方法。

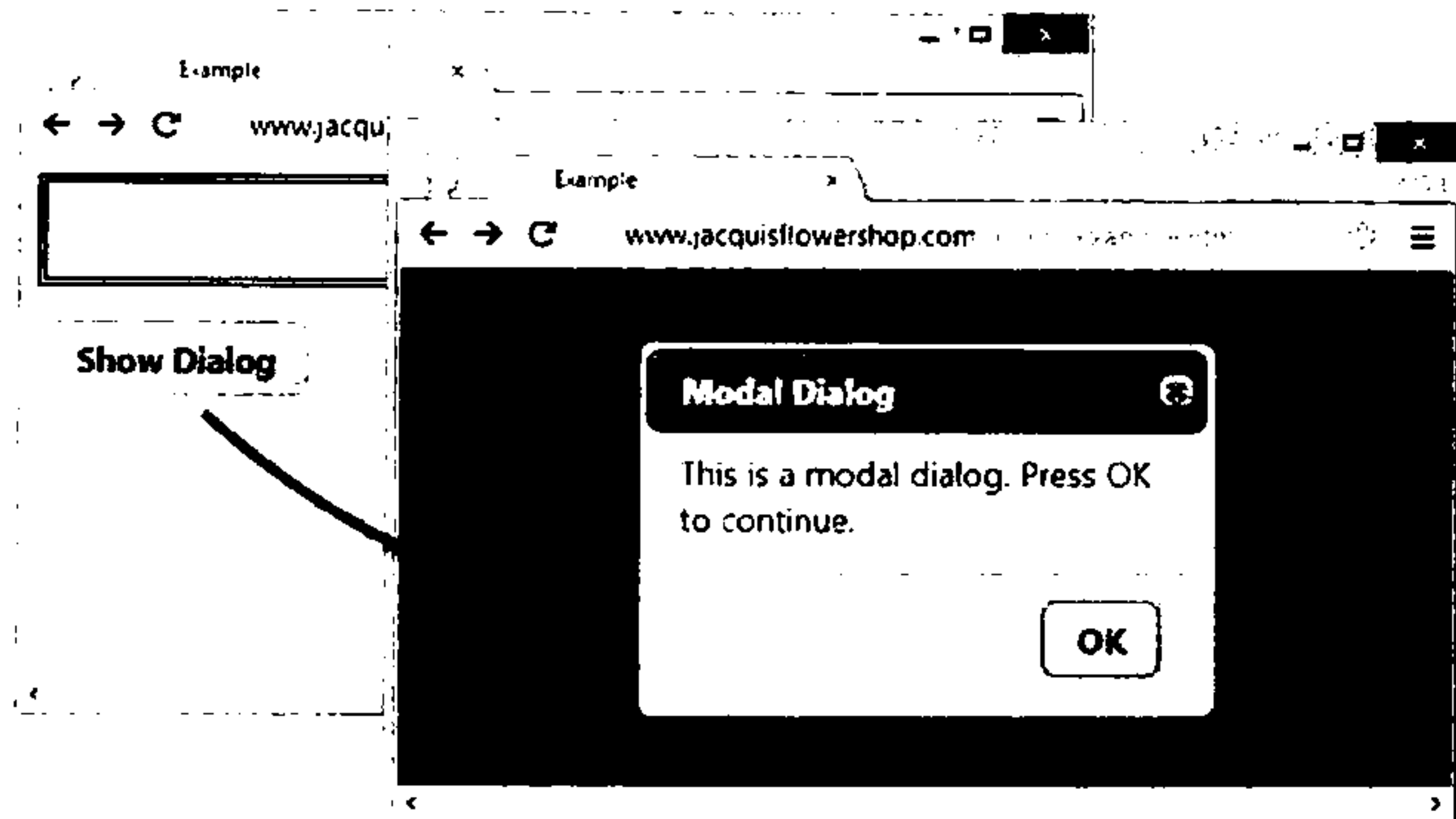


图22-9 一个模态对话框

在显示模态对话框时，jQuery UI会在对话框与页面其他部分之间放一黑色（半透明）的层。在对话框关闭之前，页面其他部分不会恢复正常状态。我在这个例子中添加了一个OK按钮，只要用户单击它就会关闭对话框。

**提示** 当我们需要在页面中选择用来显示对话框的按钮时，如果在对话框中也添加了按钮，那就不要使用\$("button")来选择。这个选择器匹配页面中所有的按钮（对话框中的和对话框外的），如果这样做就会导致单击对话框内的按钮时不仅执行计划执行的回调函数，也执行本来计划单击对话框外按钮才会执行的回调函数。

## 6. 在模态对话框中显示表单

模态对话框的优点是能充分吸引用户的注意力。如代码清单22-10所示，只要你愿意，就可以利用这一技术在模态对话框中显示简单的表单。

### 代码清单22-10 在模态对话框中显示表单

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
```

```

<link rel="stylesheet" type="text/css" href="styles.css"/>
<style type="text/css">
  #dialog input {width: 150px; margin: 5px; text-align: left}
  #dialog label {width: 100px}
  table {border-collapse: collapse; border: thin solid black; margin: 10px}
  #placeholder {text-align: center}
  #show {margin: 10px}
  td, th {padding: 5px; width: 100px}
</style>
<script id="rowTpl" type="text/x-handlebars-template">
  <tr><td>{{product}}</td><td>{{color}}</td><td>{{count}}</td></tr>
</script>
<script type="text/javascript">
  $(document).ready(function () {
    $("#dialog").dialog({
      buttons: [{text: "OK", click: addDataToTable}],
      modal: true,
      autoOpen: false,
      width: 340
    })

    $("#show").button().click(function () {
      $("#dialog").dialog("open");
    })

    function addDataToTable() {
      var data = {
        product: $("#product").val(),
        color: $("#color").val(),
        count: $("#count").val()
      }
      $("#placeholder").hide();
      $("#rowTpl").template(data).filter("*").appendTo("#prods tbody");
      $("#dialog").dialog("close");
    }
  });
</script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="dialog" title="Enter Details" class="ui-widget">
    <div><label for="product">Product: </label><input id="product" /></div>
    <div><label for="color">Color: </label><input id="color" /></div>
    <div><label for="count">Quantity: </label><input id="count" /></div>
  </div>
  <table id="prods" class="ui-widget" border="1">
    <tr><th>Product</th><th>Color</th><th>Quantity</th></tr>
    <tr id="placeholder"><td colspan=3>No Products Selected</td></tr>
  </table>
  <button id="show">Add Product</button>
</body>
</html>

```

在这个例子里，我在模态对话框所在的div中定义了几个input元素。当用户单击页面中的Add

Product按钮时，对话框就会显示出来请用户填写订货数据。当用户单击OK按钮时（使用buttons选项定义的），我就根据用户输入的信息使用数据模板在表格中添加一行数据。图22-10展示了这个过程。

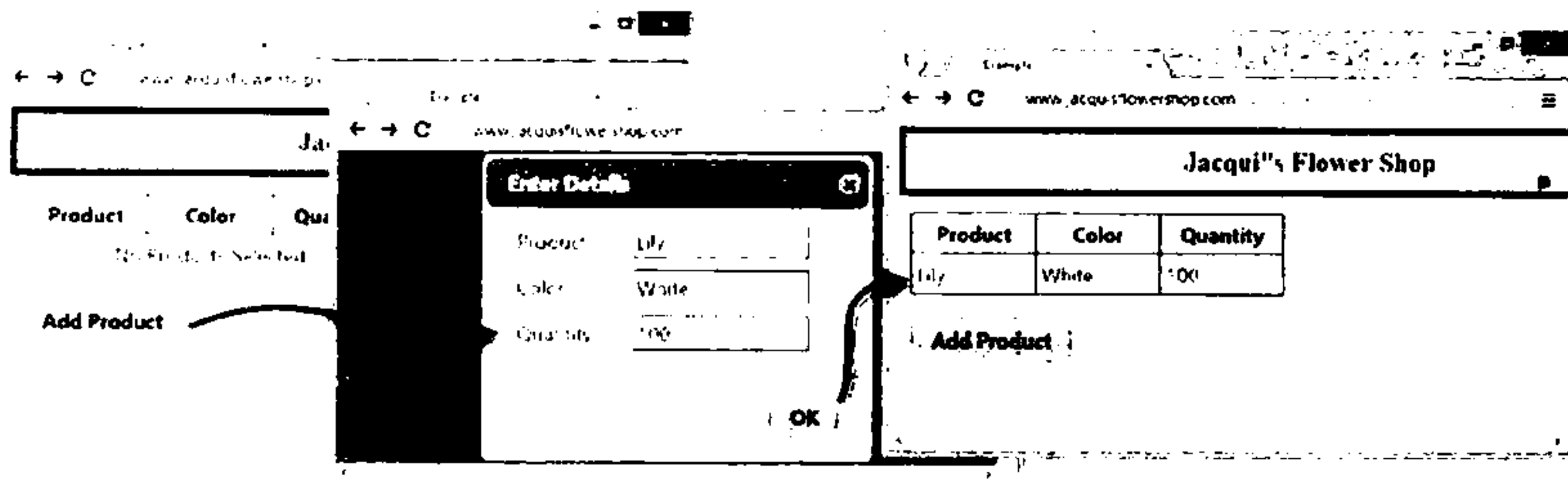


图22-10 使用模态对话框收集用户数据

我让这个例子尽量简单，不过你很容易在这个例子中加入第13章中讲到的验证技术来减少数据错误。而且，你也可以使用第14章和第15章讲到的Ajax技术把用户填写的数据提交到远程服务器。

**警告** 在模态对话框中展示表单这项技术只适用于简单的表单。如果你正试着把标签与折叠菜单塞进一个模态对话框，小心不要激怒你的用户，或者给他们造成困扰。如果表单需要花费大量的时间来填写，那么就不适合放进模态对话框，更合适的做法是把它直接整合到页面中。

### 22.1.3 对话框组件方法

jQuery UI对话框组件也定义了一些方法，参见表22-4。

表22-4 对话框组件方法

方 法	描 述
<code>dialog("destroy")</code>	恢复对话框组件底层元素到最初状态
<code>dialog("option")</code>	改变选项
<code>dialog("close")</code>	关闭对话
<code>dialog("isOpen")</code>	若对话框可见，返回true，否则返回false
<code>dialog("moveToTop")</code>	把对话框移到栈顶（使对话框完全可见）
<code>dialog("open")</code>	把对话框显示给用户

和你料想的一样，这些方法大都用于通过脚本控制对话框。就个人而言，我用得最多的方法是open和close。代码清单22-11展示了这些关键方法的使用。

#### 代码清单22-11 使用对话框组件方法

```
<!DOCTYPE html>
<html>
<head>
```

```

<title>Example</title>
<script src="jquery-2.0.2.js" type="text/javascript"></script>
<script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<script type="text/javascript">
    $(document).ready(function () {
        $("#d1, #d2").dialog({
            autoOpen: false,
        });

        $("#t1, #t2").button().click(function (e) {
            var target = this.id == "t1" ? "#d1" : "#d2";
            if ($(target).dialog("isOpen")) {
                $(target).dialog("close")
            } else {
                $(target).dialog("open")
            }
        });
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <div id="d1" class="dialog" title="First Dialog" class="ui-widget">
        This is the first dialog
    </div>
    <div id="d2" class="dialog" title="Second Dialog" class="ui-widget">
        This is the second dialog
    </div>
    <div>
        <button id="t1">Toggle Dialog 1</button>
    </div>
    <button id="t2">Toggle Dialog 2</button>
</body>
</html>

```

这个页面中有两个按钮，分别用来切换两个对话框的可见状态。我们使用`isOpen`方法判断这两个对话框的可见状态。图22-11展示了两个对话框（都可见）在浏览器中的样子。

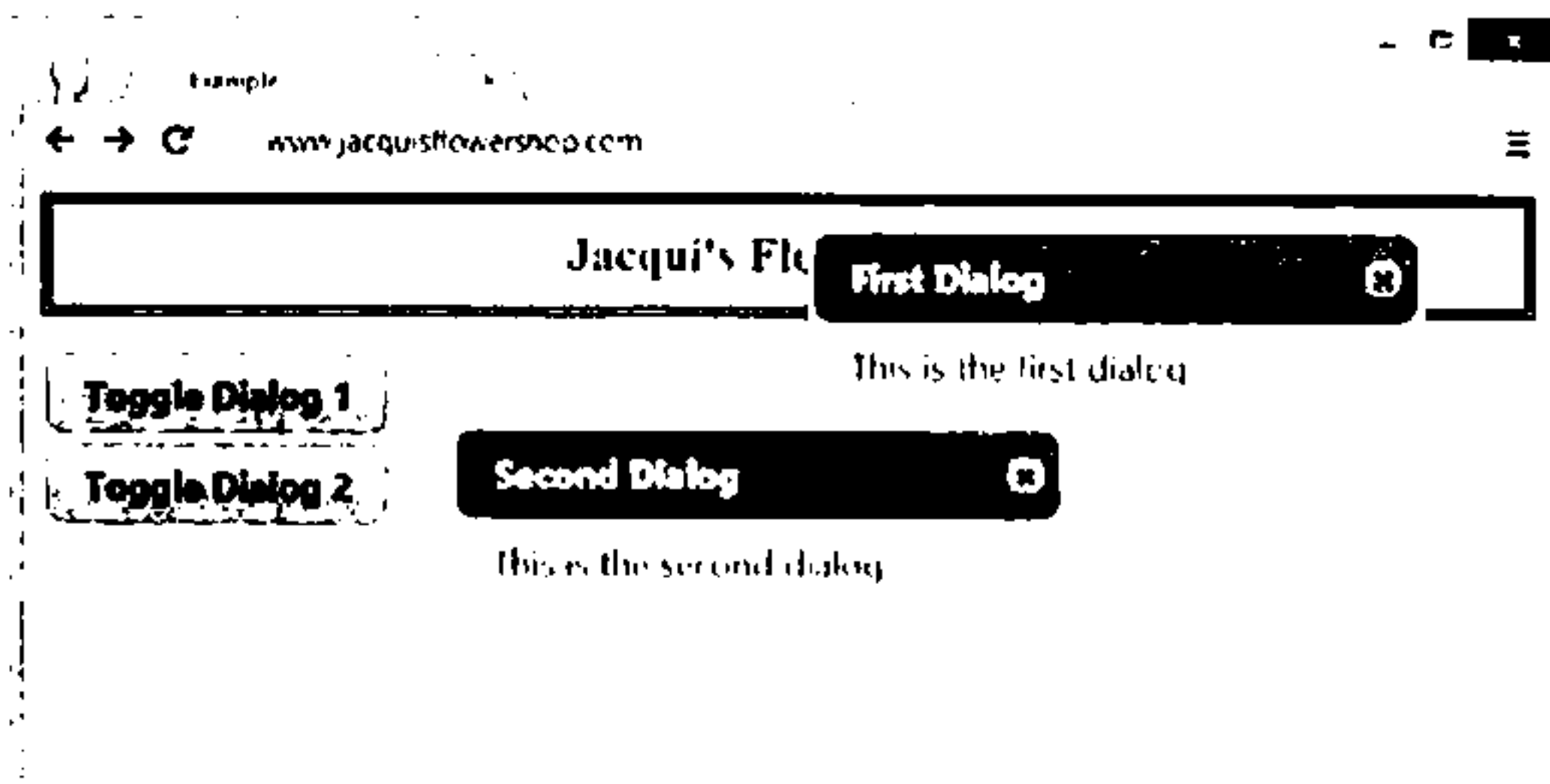


图22-11 使用组件方法切换对话框显示状态

## 22.1.4 对话框组件事件

jQuery UI对话框组件支持的事件见表22-5。我会在接下来的几节中讲解其中最有用的几个事件。

表22-5 对话框组件事件

事 件	描 述
create	当jQuery UI利用底层元素完成对话框创建时触发
beforeClose	当对话框即将关闭时触发。如果该事件的处理函数返回false，那么对话框将不会关闭
open	当对话框显示出来时触发
focus	当对话框得到焦点时触发
dragStart	当用户开始拖曳对话框时触发
drag	当用户拖曳对话框时每一次鼠标移动都触发
dragStop	当用户结束拖曳时触发
resizeStart	当用户开始改变对话框尺寸时触发
resize	当对话框尺寸正在改变时触发
resizeStop	当对话框尺寸停止改变时触发
close	当对话框关闭时触发

### 1. 阻止用户关掉对话框

利用beforeClose事件，我们能知道用户是否打算关闭对话框。这可能是当用户按下Esc键（如果closeOnEscape选项为true的话），单击了对话框右上角的关闭图标，或者单击了我们通过buttons选项添加的某个按钮。

绝大多数时候，我们都应该尊重用户的选择，允许他们关掉对话框。然而如代码清单22-12所示，确实有例外情况存在，比如用户必需使用对话框做一些事情才可以进入下一步。在这个例子里，对话框必须显示一定的时间，然后用户才可以继续。

代码清单22-12 阻止用户关掉对话框

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    input {width: 150px}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      var canClose = false;
      var delay = 15;

      $("#dialog").dialog({
        modal: true,
```

```

        autoOpen: false,
        beforeClose: function() {
            return canClose;
        },
        open: function() {
            var count = delay;
            var intID = setInterval(function() {
                count--;
                $("#time").text(count);
                if (count == 0) {
                    clearInterval(intID)
                    canClose = true;
                    $("#dialog").dialog("close")
                }
            }, 1000)
        }
    });

    $("#button").click(function(e) {
        $("#dialog").dialog("open");
    })

    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>

    <div class="ui-widget">
        <label for="user">Username: </label>
        <label for="pass">Password: </label>
        <button id="send">Login</button>
    </div>
    <div id="dialog" title="Wrong Password">
        The password you entered was incorrect. Please try again in
        <span id="time">15</span> seconds.
    </div>
</body>
</html>

```

在这个例子中，我定义了两个文本框供用户输入用户名和密码。用户输入了什么并不重要，但只要用户单击Login按钮，我就显示一个密码错误的模态对话框。

我在open事件处理函数中启动了一个时长15秒的倒计时定时器。在此期间，我使用beforeClose事件阻止用户关闭对话框。待15秒时间过后，我调用close方法来关闭这个对话框。通过组合使用open和beforeClose事件，我可以确保用户不会立即尝试其他的用户名和密码组合（没错，至少在用户不刷新页面的情况下管用）。

## 2. 响应用户改变对话框尺寸或者位置的行为

对话框组件定义了一整套事件跟踪对话框的尺寸和位置变化。通常情况下我们不必要理会这些事件，不过在极个别的场合，使用这些事件跟踪对话框的变化很方便。代码清单22-13就利用了dragStart和dragStop事件，在对话框被拖曳的过程中禁用了页面中的input和button元素。



代码清单22-13 响应对话框拖曳事件

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    input {width: 150px; text-align: left}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      $("#dialog").dialog({
        autoOpen: true,
        dragStart: function() {
          $("input, #send").attr("disabled", "disabled")
        },
        dragStop: function() {
          $("input, #send").removeAttr("disabled")
        }
      });

      $("button").click(function(e) {
        $("#dialog").dialog("open");
      });
    });
  </script>
</head>
<body>
<h1>Jacqui's Flower Shop</h1>

  <div class="ui-widget">
    <label for="user">Username: <input id="user"/>
    <label for="pass">Password: <input id="pass"/>
    <button id="send">Login</button>
  </div>
  <div id="dialog" title="Wrong Password">
    The password you entered was incorrect. Please try again in
    <span id="time">15</span> seconds.
  </div>
</body>
</html>

```

## 22.2 jQuery UI 微调控制组件

jQuery UI 1.9版本增加了微调控制组件，为input元素添加了一组向上向下的按钮，让用户可以在一定范围内点击按钮选择需要的值。选中需要的input元素，并调用spinner方法即可生成微调控制组件，如代码清单22-14所示。

代码清单22-14 创建基本的spinner组件

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    input {width: 150px;}
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#entry").spinner();
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div class="ui-widget">
    Enter value: <input id="entry" value="0" />
  </div>
</body>
</html>

```

在图22-12中，可以看到input元素的新面貌，以及点击jQuery UI所添加的按钮时的效果。

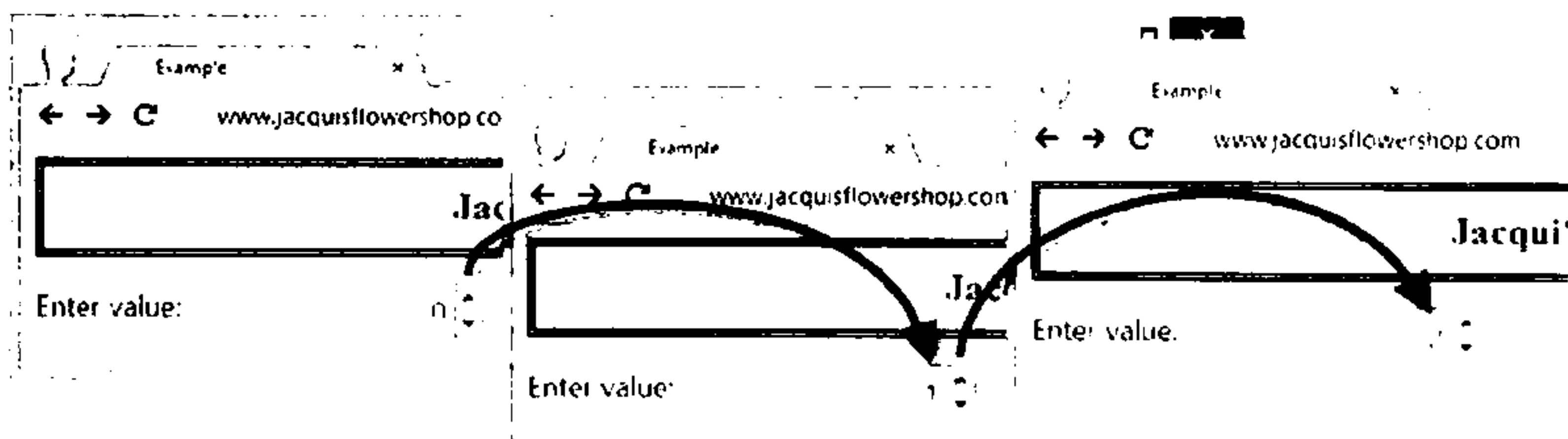


图22-12 创建并应用微调控制组件

### 22.2.1 在HTML5 input元素上应用微调控制组件

在HTML5标准中，input元素支持一些新type类型，其中一种类型的效果类似微调增强组件，这就是number类型。代码清单22-15就使用了这种input元素。

代码清单22-15 HTML5中number类型的input元素

```

<!DOCTYPE html>
<html>
<head>

```

```

<title>Example</title>
<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<style type="text/css">
    input {width: 150px;}
</style>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <div class="ui-widget">
        Enter value: <input id="entry" value="0" type="number" />
    </div>
</body>
</html>

```

在这个例子中，我拿走了所有的JavaScript代码，以强调浏览器对input元素新类型的内建支持。图22-13是该页面在Google Chrome浏览器中显示的样子。

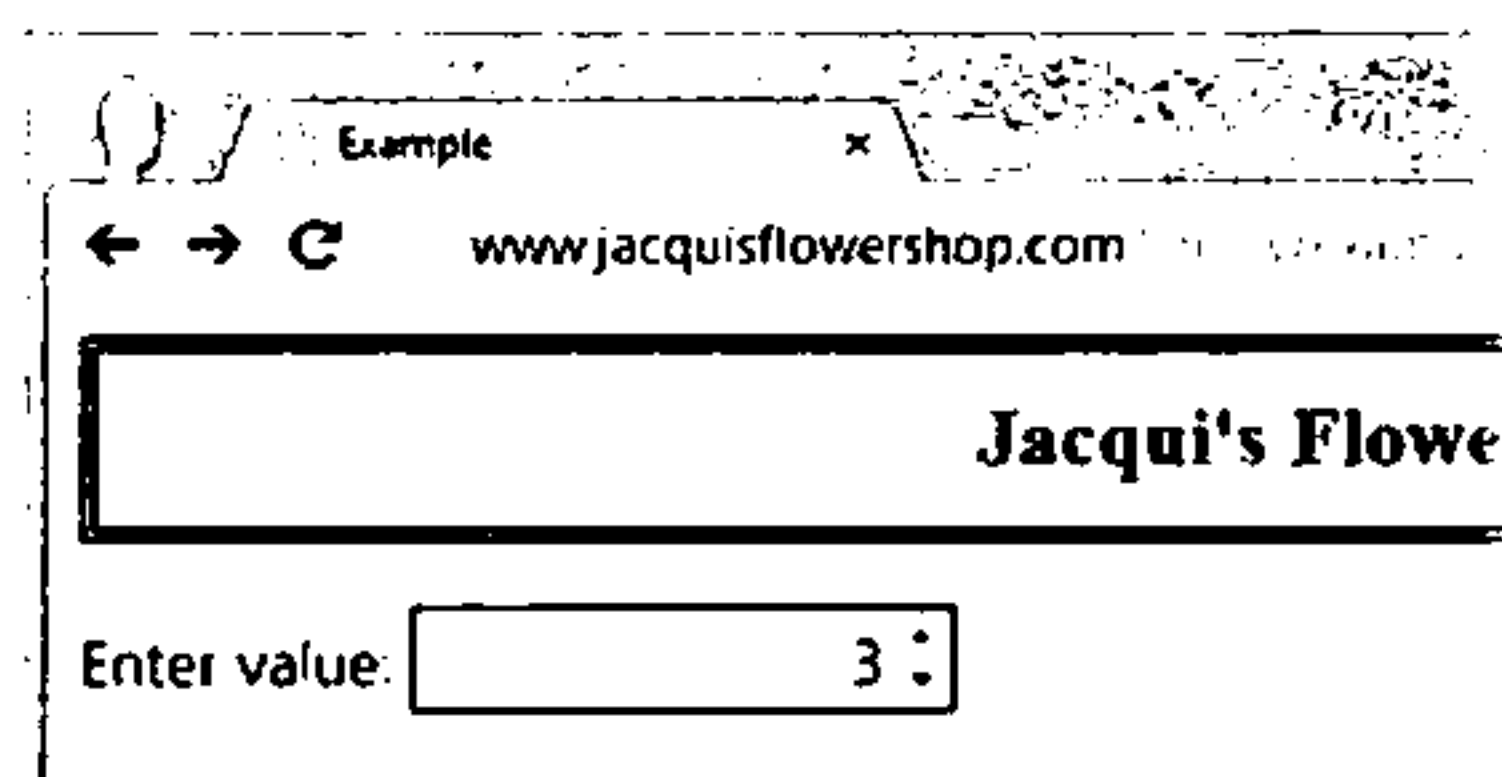


图22-13 Google Chrome浏览器中number类型的input元素

与jQuery UI微调控制组件相比，这个方案有一些优势。这是因为HTML5标准并未明确规定以什么方式控制值的改变，这样浏览器就能根据设备类型提供更合适的控制方式。jQuery UI微调控制组件总是使用向上和向下按钮，对使用智能手机等小屏触摸设备的用户来说，操作起来未免有些困难。

number类型input元素方案最大的问题，在于大部分浏览器会直接忽略它，即使这些浏览器支持一些别的HTML5特性。而且，同时使用HTML5方案与微调控制组件的方案也不可取，因为调用spinner方法时微调控制组件并不检查input元素的类型。代码清单22-16展示了一个在number类型的input元素上调用spinner方法的例子。

代码清单22-16 在HTML5 number类型的input元素上应用jQuery UI微调控制组件

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <style type="text/css">
        input {width: 150px;}
    </style>

```

```

<script type="text/javascript">
    $(document).ready(function () {
        $("#entry").spinner();
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <div class="ui-widget">
        Enter value: <input id="entry" value="0" type="number" />
    </div>
</body>
</html>

```

在这个例子中我恢复了所有JavaScript库和代码。在图22-14中可以看到，input元素有两套按钮可以改变元素的值。浏览器对新类型input元素的实现，以及jQuery UI对它们的处理还不是很成熟，因此我建议避免使用新类型input元素，尤其要避免同时使用新类型input元素与微调控制组件。

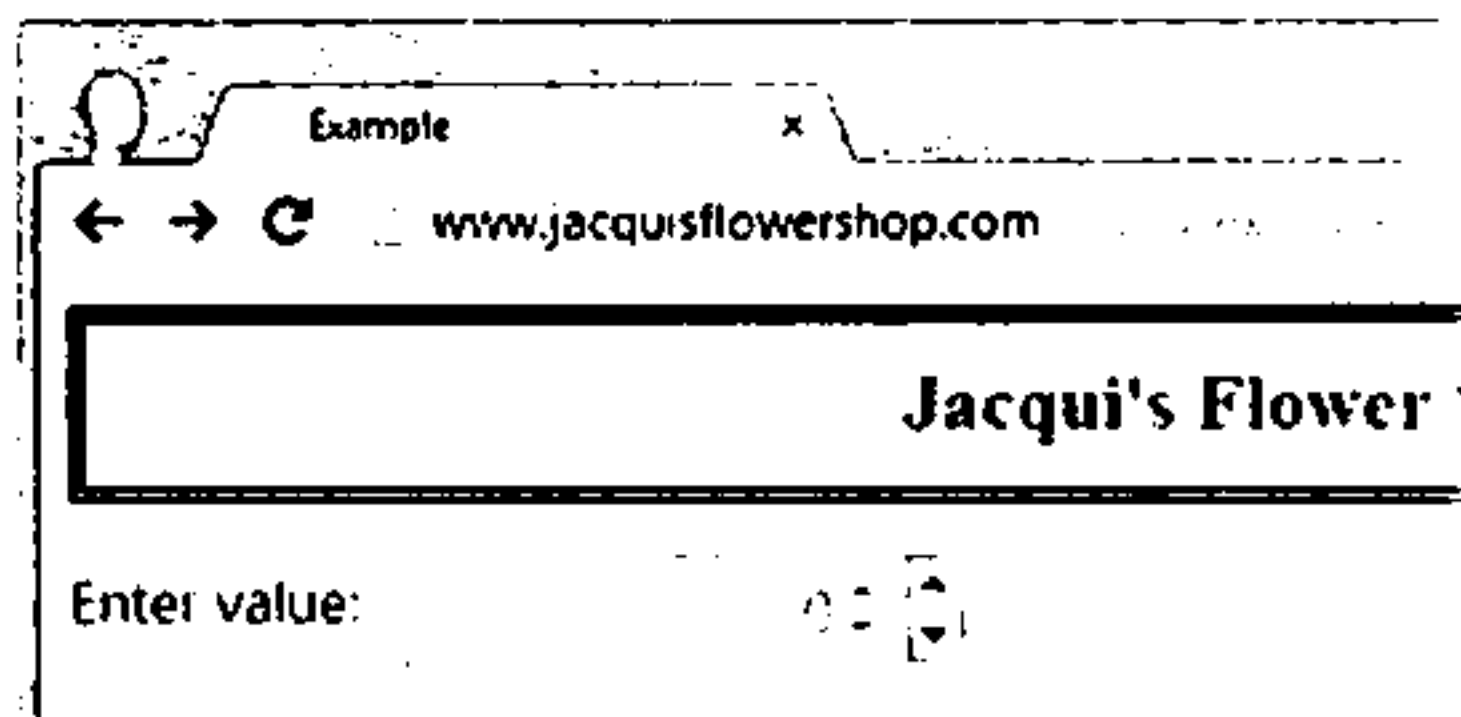


图22-14 在HTML5 number类型的input元素上应用jQuery UI微调控制组件

## 22.2.2 配置微调控制组件

微调控制组件支持许多选项，利用这些选项我们能够定制它的行为。表22-6列出了这些选项，在下面的内容中将演示这些选项的用法。

表22-6 微调控制组件选项

选 项	描 述
culture	指定按照什么地域解析值的格式（默认值为null）
disabled	若设置为true，则禁用组件（默认值为false）
icons	指定组件按钮图标（默认为向上的箭头和向下的箭头）
incremental	指定长按按钮时值变化的速度。若设置为true，按下按钮时持续的时间越长，值变化的越快。若设置为false，则值始终以恒定速度变化。也可以把值设置为一个函数，由这个函数动态定义值变化的速度。默认值为true
max	设置最大值（默认值为null，表示不限制）
min	设置最小值（默认值为null，表示不限制）
numberFormat	设置数字格式（默认值为null）
page	指定页大小，即用户调用pageUp或pageDown方法时值应变化多少（详见本章）。默认值为10
step	指定步大小，即用户点击上下按钮时值应变化多少（默认值为1）

### 1. 配置微调控制组件的行为

有四个选项决定微调控制组件的基本行为：`min`、`max`、`step`和`page`。代码清单12-17演示了如何使用这些属性配置组件支持的值，以及值如何增加和减小。

代码清单22-17 配置微调控制组件的行为

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    input {width: 150px;}
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#entry").spinner({
        min: 2,
        max: 100,
        step: 2,
        page: 5
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div class="ui-widget">
    Enter value: <input id="entry" value="0" />
  </div>
</body>
</html>
```

`min`和`max`属性并不对

**提示** 如果希望详细了解如何验证

`step`和`page`属性决定微调控制组件如何增加或者减少值。`step`属性指定用户按下向上或者向下按钮时，值应该变化多少。`page`属性指定当用户调用`pageUp`或`pageDown`方法时，值应该变化多少（详见本章后面部分）。

在这个例子中，我把最小值设置为2，把最大值设置为100。把`step`属性设置为2，这样每点击一次上下按钮值就会变化2；把`page`属性调协为5，则每次`pageUp`或`pageDown`调用，值就会变化5。具体效果见图22-15。

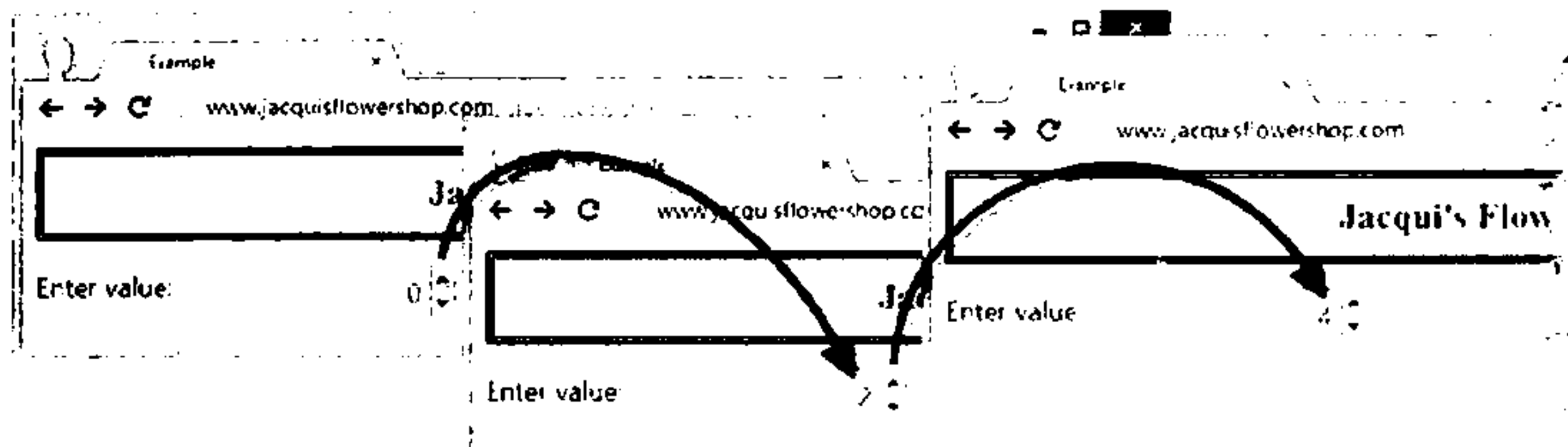


图22-15 控制微调控制组件的基本行为

**提示** 注意最小值设置并不影响input元素的初始值。在这个例子中我把input元素的值设置为0，它比微调控制组件的min属性还要小。

若用户输入超出允许范围的值之后，又点击向上或向下按钮，则值会被微调控制组件重新初始化为有效值的最小值。具体效果见图22-16。

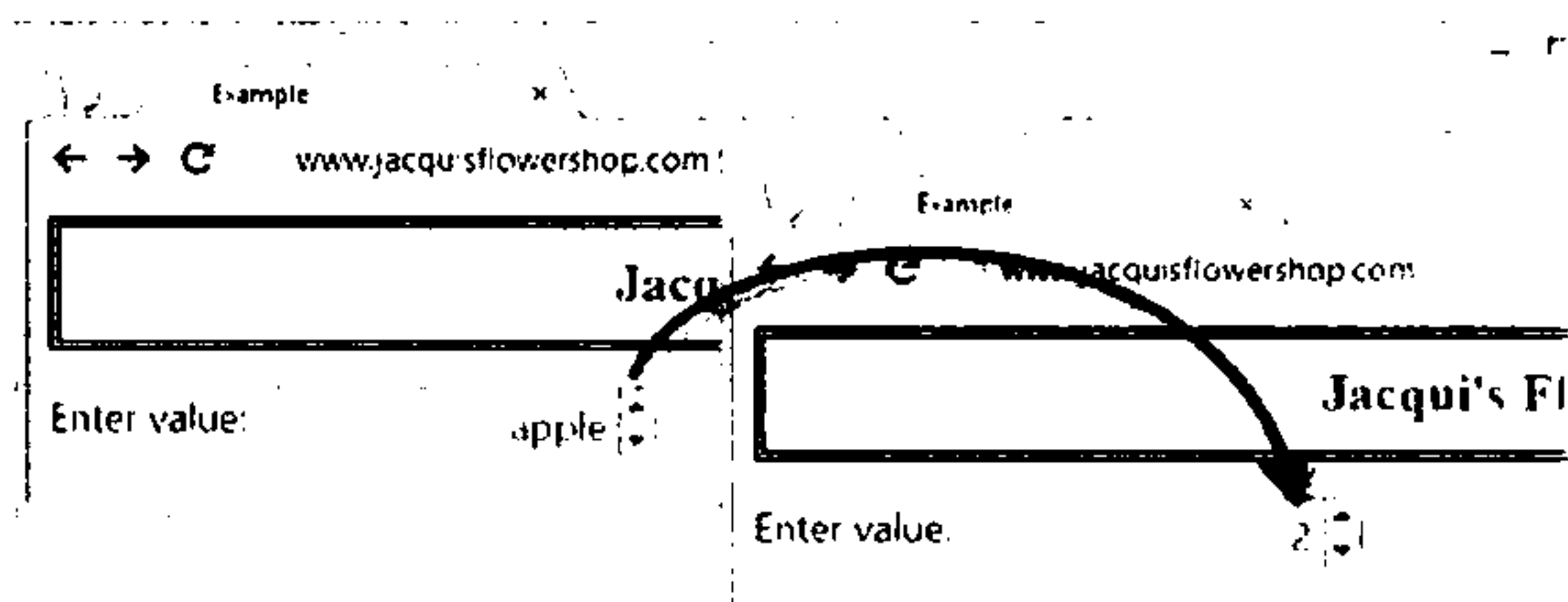


图22-16 微调控制组件重新初始化输入框

## 2. 改变微调控制组件所用图标

icons属性用来设置微调控制组件上下按钮的图标。如代码清单22-18所示，虽然组件的默认图标是向上和向下的箭头，但它们可以被设置为任意一个jQuery UI图标（详见第18章）。

### 代码清单22-18 改变微调控制组件所用图标

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    input {width: 150px;}
  </style>
```

```

<script type="text/javascript">
  $(document).ready(function () {
    $("#entry").spinner({
      min: 2,
      max: 100,
      step: 2,
      page: 5
      icons: {
        up: "ui-icon-circle-plus",
        down: "ui-icon-circle-minus",
      }
    });
  });
</script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div class="ui-widget">
    Enter value: <input id="entry" value="0" />
  </div>
</body>
</html>

```

icons属性的值是一个有着up和down属性的对象，这两个属性分别定义相应按钮上使用的图标。如图22-17所示，我把这两个图标分别设置成了加号和减号。

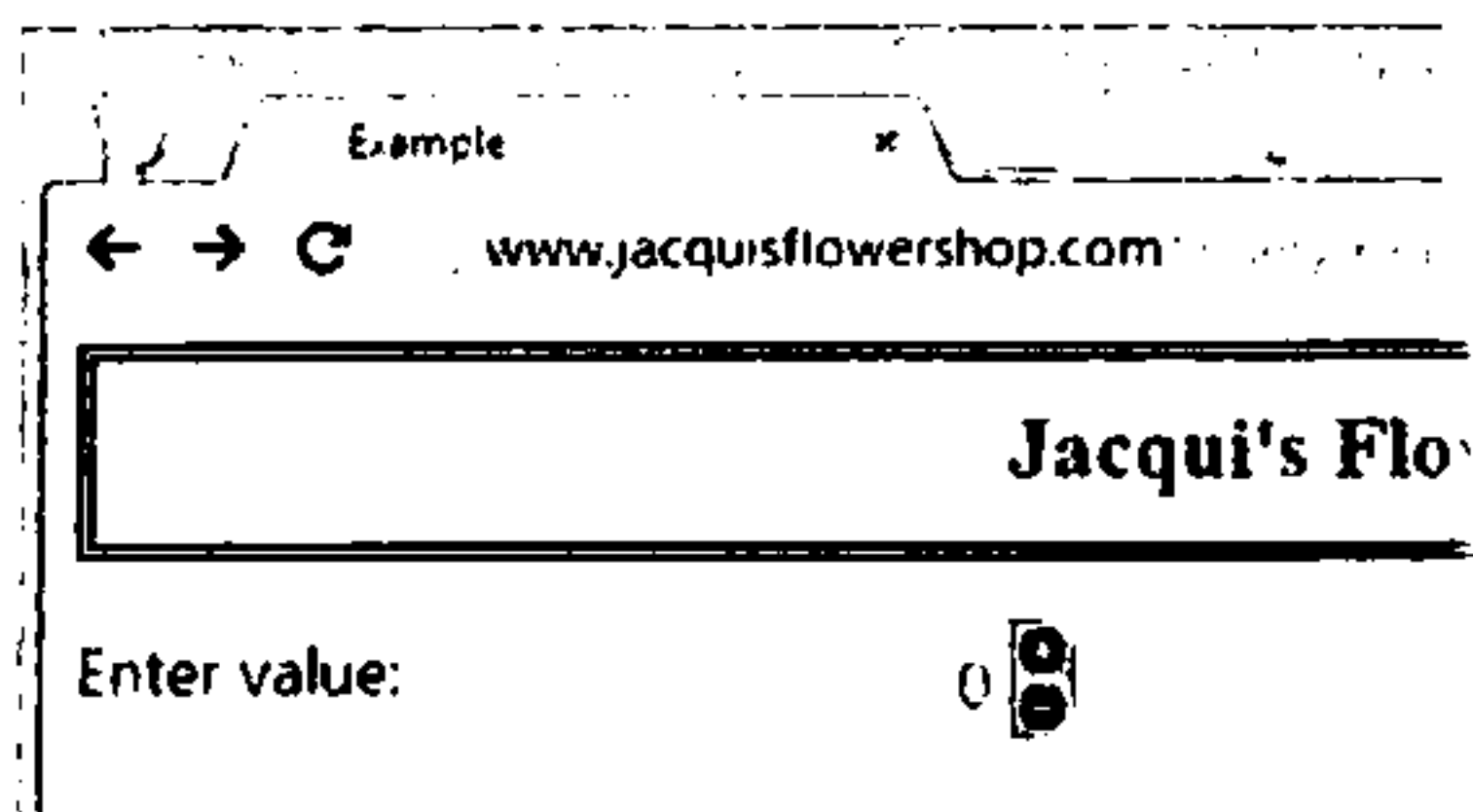


图22-17 改变微调控制组件所用图标

### 3. 控制值变化的速度

incremental属性用来设置当用户按住按钮不放时值变化的速度。若把此属性设置为true，则按住的时间越长，值变化的越快。若设置成false，则值始终以恒定的速度变化。如代码清单22-19所示，第三个选择则是把它设置成一个函数，用来动态地改变值的变化速度。

#### 代码清单22-19 把incremental选项设置为一个函数

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>

```

```

<link rel="stylesheet" type="text/css" href="styles.css"/>
<style type="text/css">
    input {width: 150px;}
</style>
<script type="text/javascript">
    $(document).ready(function () {
        $("#entry").spinner({
            incremental: function (spins) {
                return Math.pow(spins, 2);
            }
        });
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <div class="ui-widget">
        Enter value: <input id="entry" value="0" />
    </div>
</body>
</html>

```

传递给函数的参数，是按照按钮按下时间推算出的已发生的微调次数。在这个例子中，函数的返回值是微调次数的平方。

#### 4. 设置数值的显示格式

微调组件数值的显示格式，由numberFormat和culture两个属性控制，这两个属性依赖名为Globalize的第三方库。

访问<https://github.com/jquery/globalize>可下载Globalize库。点击其中的Releases链接，然后选择希望下载的版本（在我写作本书时，最新版是0.1.1）。可以选择下载zip或者tar.gz格式的文件。对下载文件解包并把解包得来的以下文件复制到包含example.html和JavaScript库的目录中。

□ lib/globalize.js.

□ lib/cultures/globalize.cultures.js

第一个文件包含处理地域的JavaScript代码，第二个文件包含着Globalize库内建的各地域的完整设置数据。

---

**提示** 在lib/cultures目录下还有一些较小的文件，它们只处理一个地域。如果只需要处理很少的地域，可以使用这些文件以减少库的体积。

---

微调控制组件的numberFormat属性指定数值应该显示成什么格式，而culture属性则指定应该遵守哪个地域的格式化习惯。在代码清单22-20中，演示了这两个属性的用法。

#### 代码清单22-20 应用numberFormat和culture属性

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>

```



```

<script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<script src="globalize.js"></script>
<script charset="utf-8" src="globalize.cultures.js"></script>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<style type="text/css">
    input {width: 150px;}
</style>
<script type="text/javascript">
    $(document).ready(function () {
        $("#entry").spinner({
            culture: "fr-FR",
            numberFormat: "C"
        });
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <div class="ui-widget">
        Enter value: <input id="entry" value="0" />
    </div>
</body>
</html>

```

globalize.js必须先于globalize.cultures.js文件加载，这样地域信息才能正确注册到主库。配置微调控制组件时，我把culture设置为fr-FR（即法国的法语区），把numberFormat属性设置为C，即把数值视为货币金额。实际效果见图22-18。

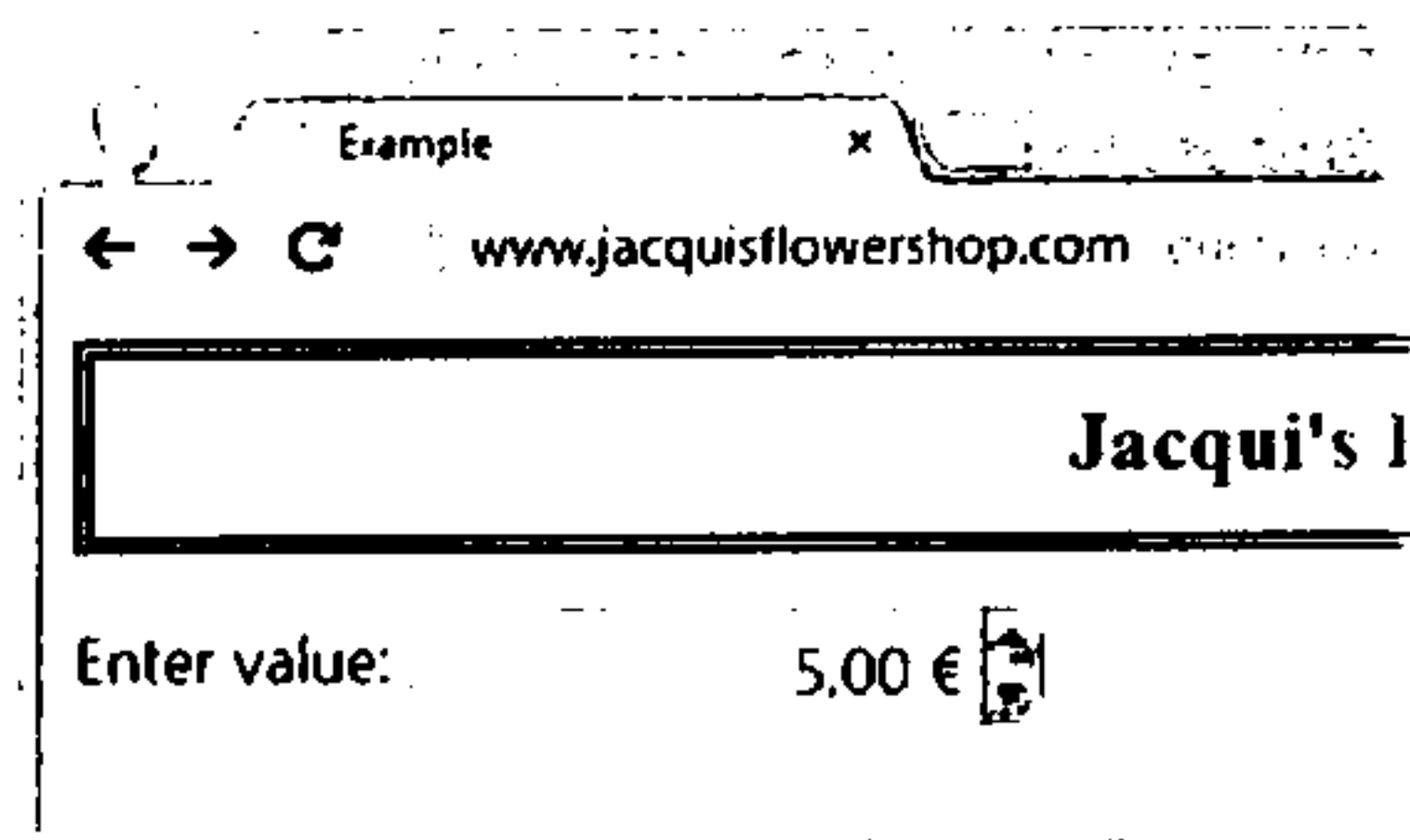


图22-18 调整微调控制组件的地域及数值格式

**提示** 如果忽略了culture选项，则地域默认为美国。

法国的流通货币是欧元，因此图中数字后面跟着欧元的符号，并且整数与小数之间使用逗号分隔。注意为正确处理区域设置，我在script标签上使用了charset属性，如下：

```

...
<script charset="utf-8" src="globalize.cultures.js"></script>
...

```

可以用meta标签设置整个HTML文档的字符集，如果忘记这么做——前面的示例页面中就没有涉及，就要明确告诉script元素应该使用什么字符集。若缺少这一属性，绝大多数浏览器就会使用别的地域设置显示货币单位。如图22-19所示，本该显示成欧元符号的字符，现在却显示成了乱码。

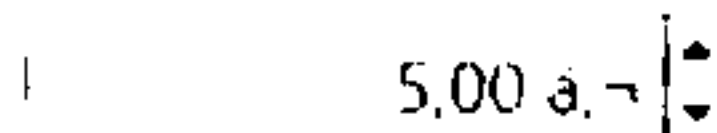


图22-19 省略设置地域的script标签的charset属性所带来的问题

### 22.2.3 微调控制组件方法

jQuery UI微调控制组件提供了以下方法（表22-7）。

表22-7 spinner方法

方 法	描 述
spinner("destroy")	从底层元素上移除微调控制组件
spinner("disable")	禁用微调控制组件
spinner("enable")	启用微调控制组件
spinner("option")	获取组件选项，或者设置组件选项
spinner("pageDown", count)	按指定页数减小组件的值
spinner("pageUp", count)	按指定页数增加组件的值
spinner("stepDown", count)	按指定步数减小组件的值
spinner("stepUp", count)	按指定步数增加组件的值
spinner("value")	获取组件的值，或者设置组件的值

代码清单22-21中演示了微调控制组件几个特有方法的用法。

#### 代码清单22-21 微调控制组件的方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <script src="globalize.js"></script>
  <script src="globalize.cultures.js"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    input {width: 150px;}
    button { margin-top: 10px; }
  </style>
  <script type="text/javascript">
    $(document).ready(function () {

      $("#entry").spinner({
        culture: "en-US",
```

```

        numberFormat: "C",
        step: 2,
        page: 10
    ));

    $("#button").button().click(function () {
        $("#entry").spinner(this.id);
        console.log("Value: " + $("#entry").spinner("value"));
    });

    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <div class="ui-widget">
        Enter value:
    </div>
    <div>
        <button id="pageDown">Page Down</button>
        <button id="stepDown">Step Down</button>
        <button id="stepUp">Step Up</button>
        <button id="pageUp">Page Up</button>
    </div>
</body>
</html>

```

我添加了一些button元素，点击它们调用微调控制组件的方法。数字增加多少或者减少多少，由初始化微调控制组件时所设置的page及step选项的值决定。这个例子的实际效果见图22-20。

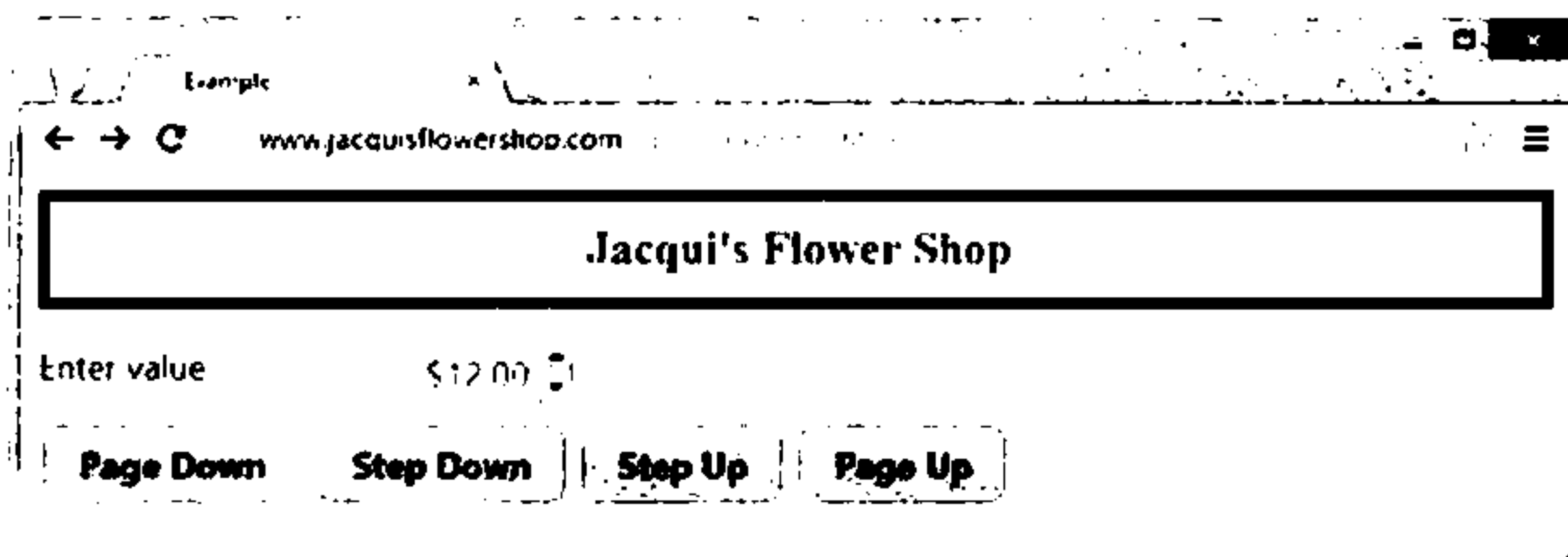


图22-20 用脚本控制微调控制组件的值

如果你注意观察浏览器控制台，会看到以下输出：

```

Value: 8
Value: 10
Value: 12

```

微调控制组件返回未格式化的值，它不受culture和numberFormat选项影响。使用jQuery选中底层的input元素，然后调用val方法，即可得到格式化之后的值。

## 22.2.4 微调控制组件事件

jQuery UI微调控制组件支持以下事件（表22-8）。

表22-8 微调控制组件事件

事 件	描 述
create	组件创建完成时触发此事件
change	组件的值发生变化，且底层input元素失去焦点时触发此事件
spin	组件的值增加或者减少时触发此事件
start	组件的值增加或者减少之前触发此事件
stop	组件的值增加或者减少之后触发此事件

在代码清单22-22中我改写了上一个例子：当stop事件发生时，把微调控制元素的值输出到控制台。

代码清单22-22 使用stop事件

```
...
<script type="text/javascript">
    $(document).ready(function () {
        $("#entry").spinner({
            culture: "en-US",
            numberFormat: "C",
            step: 2,
            page: 10,
            stop: function () {
                console.log("Value: " + $("#entry").spinner("value"));
            }
        });

        $("#button").button().click(function () {
            $("#entry").spinner(this.id);
            //console.log("Value: " + $("#entry").spinner("value"));
        });
    });
</script>
...
```

**提示** 取消spin和start事件（让事件处理函数返回false），可以阻止微调控制组件的值发生变化。要详细了解jQuery事件，请参阅第9章。

## 22.3 小结

本章的主题是jQuery UI对话框及微调控制组件。按照jQuery UI组件的固有模式，我重点介绍了两个组件的选项、方法和事件。在下一章，我们将一起了解菜单组件和提示说明组件。

本章的主题是jQuery UI菜单组件和提示说明组件。表23-1列出了本章概要。

表23-1 本章概要

问 题	解决方法	代码清单
如何创建菜单组件	选中专门结构的底层元素并调用menu方法，底层元素结构通常由ul、li和a元素构成	1
如何添加菜单分隔或禁用状态的菜单项	添加只包含虚线或空白的元素，并给它加上ui-state-disabled类	2
如何实现最基本的导航功能	为菜单结构中的a元素添加href属性	3
如何定制菜单结构	使用menus选项	4、5
如何为子菜单添加自定义图标	使用icons选项	6
如何为菜单项添加图标	添加span元素，并使用ui-icon类指定适当的jQuery UI图标	7
如何设置子菜单的位置	使用position选项	8
如何响应菜单项激活或者选中事件	处理blur、focus和select事件	9
如何创建提示说明组件	选中带有title属性的元素，然后调用tooltip方法	10、11
如何设定提示说明内容	使用content和item选项	12~15
如何为提示说明组件定制样式	使用tooltipClass选项和ui-tooltip-content类	16、17
如何让提示说明组件跟随鼠标移动	使用track选项	18
如何设置提示说明内容的位置	使用position选项	19
如何使用脚本控制提示说明组件	调用open和close方法	20
如何响应提示说明内容的显示和隐藏	处理open和close事件	21

#### 新版jQuery UI与本章有关的变化

我在编写本书第一版时，menu和tooltip组件还未问世。随着jQuery UI不断“开疆拓土”，它们被逐渐添加到jQuery UI组件大家庭。

## 23.1 jQuery UI 菜单组件

顾名思义，菜单组件用来提供菜单功能，让用户能够在选项树中导航。当我们需要展示深层结构的内容树时，比如在线商店的产品目录，这个组件非常有用。

### 23.1.1 创建菜单

如代码清单23-1所示，菜单组件依赖底层专门的HTML元素结构。选中专门结构的HTML元素，并调用menu方法，就生成了一个菜单。

代码清单23-1 创建菜单组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style>
    .ui-menu { width: 200px; }
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#menu").menu();
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>

  <ul id="menu">
    <li><a>Bouquets</a></li>
    <li><a>Heirloom Blooms</a></li>
    <li><a>Summer Color</a>
      <ul>
        <li><a>Aster</a></li>
        <li><a>Rose</a></li>
        <li><a>Orchid</a></li>
      </ul>
    </li>
    <li><a>Wedding Classics</a></li>
    <li><a>Exotica</a></li>
  </ul>
</body>
</html>
```

菜单的结构使用ul元素定义，其中每个li定义一个菜单项，内含一个a元素，a元素的链接文本则被提取出来作为菜单的标题。在li元素内嵌套定义ul元素，就能得到子菜单。上面例子的具体效果见图23-1。

从图中可以看到，我打开了一个子菜单，把鼠标放到一个突出显示状态的菜单项上。这个例子展示了菜单组件的两大重要特征。首先，菜单总是可见的，它不是弹出菜单，而是网页固有的一部分，就像亚马逊网上商店的顶级商品目录。

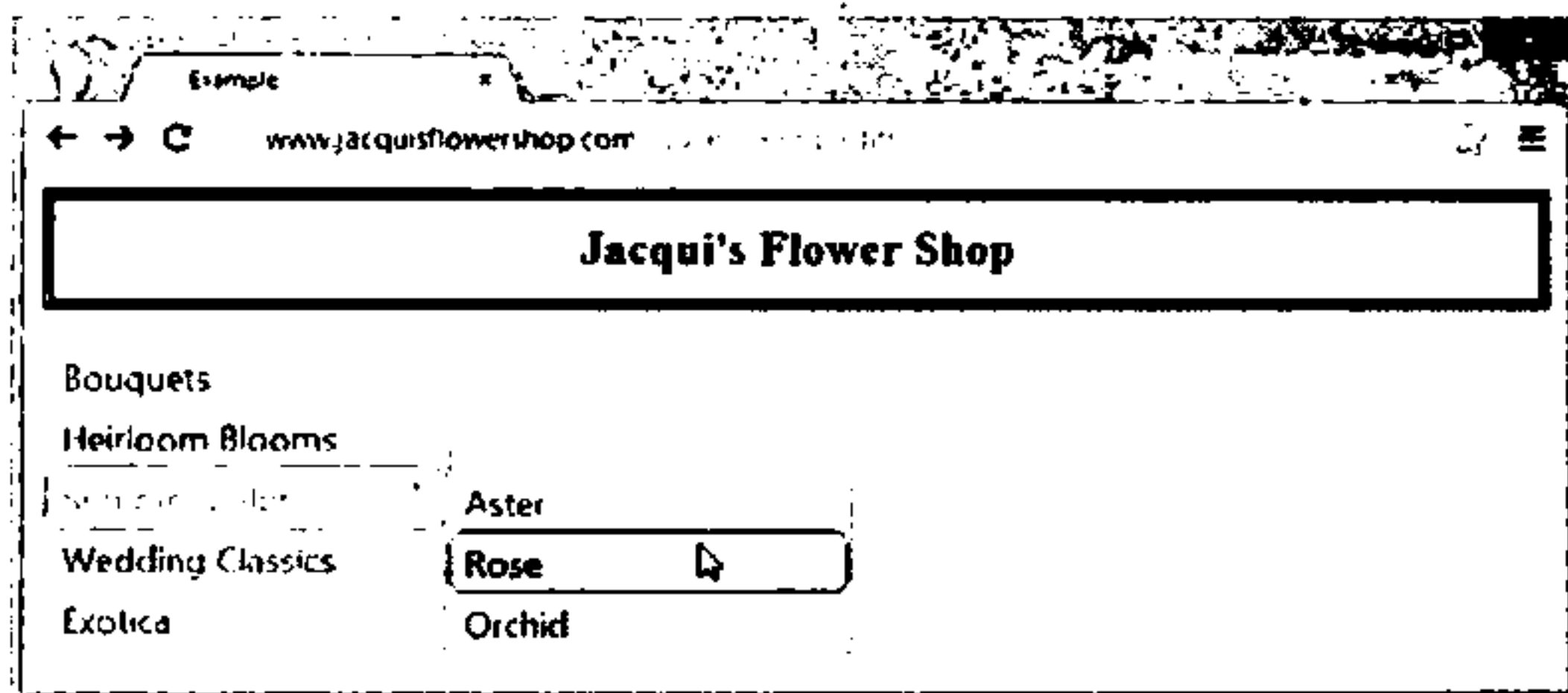


图23-1 创建一个简单的菜单组件

其次，菜单组件与底层元素的结构保持一致——菜单项不按名字或者其他语义排序。在图23-1中我们可以看到，底层元素怎么排列，菜单就怎么排列，子菜单也是如此。

**提示** 注意我使用css把顶级ul元素的宽度设成了200px。默认情况下，水平方面有多大空间，菜单就会占用多大空间。

### 1. 设置菜单项格式

除嵌套子菜单之外，元素结构还可用来设置菜单项的格式，生成禁用状态的菜单项，或者分隔有关的菜单组。代码清单23-2演示了这两种用法。

#### 代码清单23-2 使用元素结构设置菜单格式

```
...
<body>
  <h1>Jacqui's Flower Shop</h1>
  <ul id="menu">
    <li><a>Bouquets</a></li>
    <li><a>Heirloom Blooms</a></li>
    <li>-</li>
    <li><a>Summer Color</a>
      <ul>
        <li><a>Aster</a></li>
        <li><a>Rose</a></li>
        <li><a>Orchid</a></li>
      </ul>
    </li>
    <li><a>Wedding Classics</a></li>
    <li>-</li>
    <li class="ui-state-disabled"><a>Exotica</a></li>
  </ul>
</body>
...
```

全是空格或者短划线的菜单项被解释成菜单分隔符，在上面的例子中，我就添加了两个这样的菜单项。给li元素添加ui-state-disabled类，则通知菜单组件该菜单项被禁用。添加以上代码后的效果见图23-2。

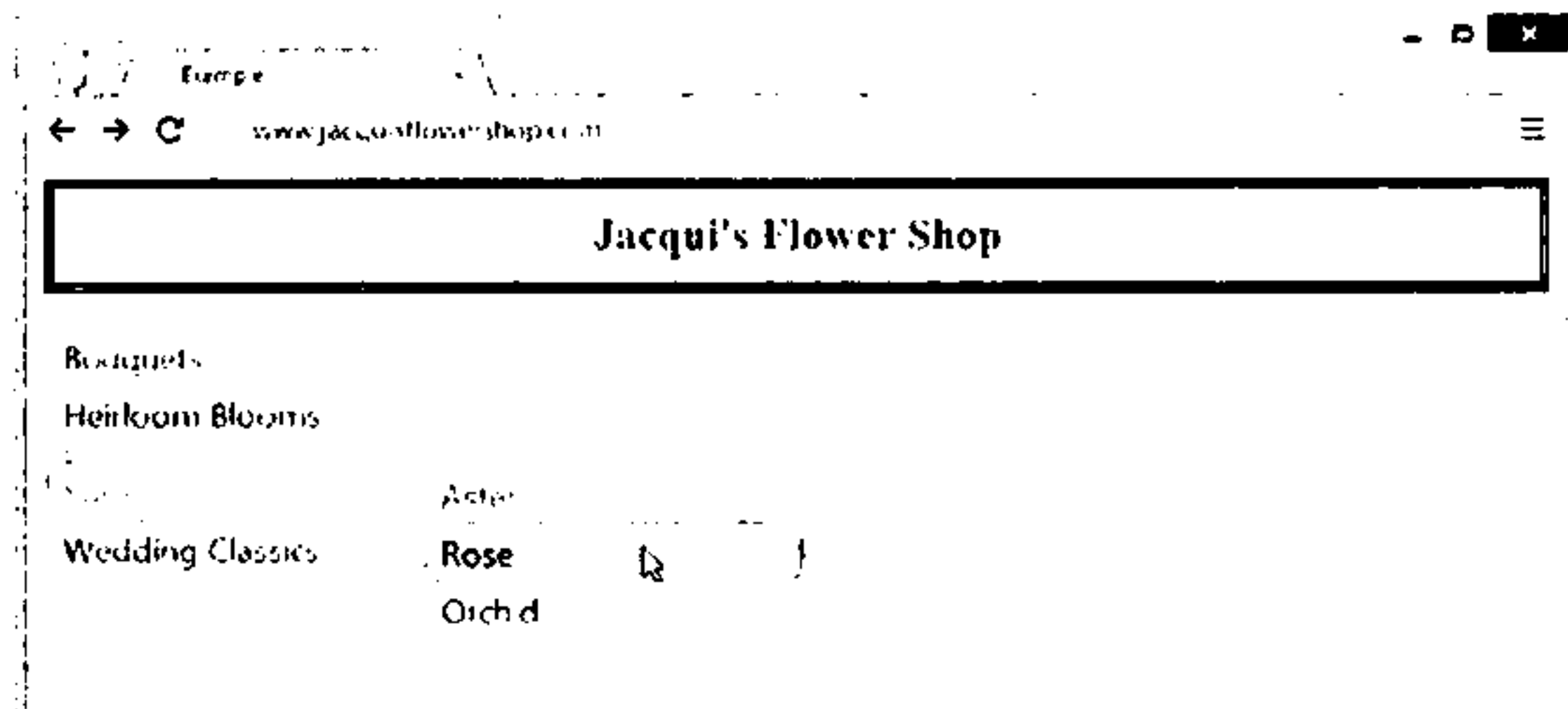


图23-2 添加菜单分隔符和禁用菜单项

## 2. 导航

在Web应用中，菜单组件最常用于网页间的导航。而实现导航最简单的方法，是为HTML底层元素结构中的a元素定义href属性。当用户选中一个菜单项，并且该菜单项的a元素定义了href属性，浏览器就会访问指定的URL。在代码清单23-3中，我就为一个菜单项添加了href属性。

代码清单23-3 为菜单中的a元素添加href属性

```
...
<body>
  <h1>Jacqui's Flower Shop</h1>

  <ul id="menu">
    <li><a>Bouquets</a></li>
    <li><a>Heirloom Blooms</a></li>
    <li>-</li>
    <li><a>Summer Color</a>
      <ul>
        <li><a href="http://apress.com">Aster</a></li>
        <li><a>Rose</a></li>
        <li><a>Orchid</a></li>
      </ul>
    </li>
    <li><a>Wedding Classics</a></li>
    <li>-</li>
    <li class="ui-state-disabled"><a>Exotica</a></li>
  </ul>
</body>
...
```

为一个菜单项定义了href属性，当我们选中Summer Color > Aster菜单时，浏览器将导航到Apress.com。

**提示** 并非所有的菜单都用来切换网页。当用户选中一个菜单项，我们可以对select事件执行任何操作。23.1.4节中将详细介绍这种用法。



## 23.1.2 配置菜单组件

菜单组件支持许多选项，利用这些选项我们可以定制菜单的呈现方式。表23-2列出了这些选项。

表23-2 菜单选项

选 项	描 述
disabled	若设置为true，则禁用菜单组件
icons	指定子菜单使用什么图标
menus	指定菜单底层结构使用什么元素
position	指定子菜单相对主菜单的位置
role	为提高易用性而指定ARIA角色 <sup>①</sup>

### 1. 使用另一种元素结构

尽管使用ul、li以及a元素定义菜单已成为一种标准方式，但是菜单组件能够支持任何能清晰定义父子结构的元素。代码清单23-4演示了如何使用div元素重新定义例子菜单。

代码清单23-4 使用div元素搭建菜单结构

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style>
    .ui-menu { width: 200px; }
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#menu").menu();
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="menu">
    <div><a>Bouquets</a></div>
    <div><a>Heirloom Blooms</a></div>
    <div></div>
    <div><a>Summer Color</a>
      <div>
        <div><a>Aster</a></div>
        <div><a>Rose</a></div>
      </div>
    </div>
  </div>
```

① ARIA（无障碍富互联网应用套件）是WAI（无障碍网页倡议）的一部分，它定义了一整套让Web内容及应用更易于访问的方法，主要用于提高动态内容，以及使用Ajax、HTML、JavaScript等相关技术开发的高级用户界面组件的易用性。——译者注

```

        <div><a>Orchid</a></div>
    </div>
</div>
<div><a>Wedding Classics</a></div>
<div></div>
<div><a>Exotica</a></div>
</div>
</body>
</html>

```

当你需要不使用任何已有HTML元素生成菜单时（典型情况：菜单由模板生成，或者是从远程服务利用Ajax技术获取的），这一技术非常有用。问题是菜单组件并不知道哪一个div元素代表了子菜单，如图23-3所示，所有的div元素都被菜单组件当成了主菜单。

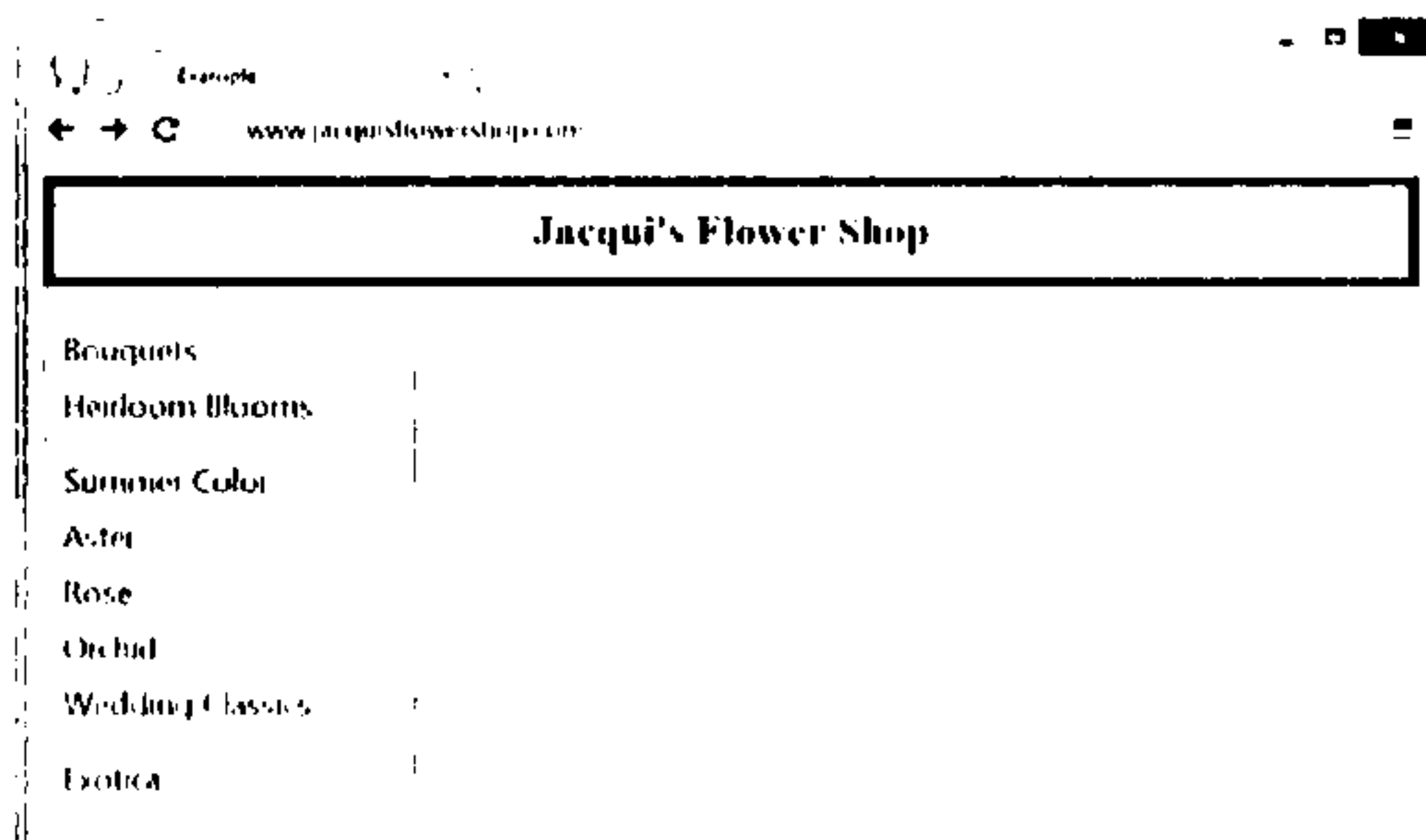


图23-3 菜单组件视所有div元素为菜单项

如代码清单23-5所示，只需把menus属性的值设置为匹配子菜单的选择器，即可告知菜单组件哪个div是子菜单。

#### 代码清单23-5 配置menus属性

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <style>
        .ui-menu { width: 200px; }
    </style>
    <script type="text/javascript">
        $(document).ready(function () {
            $("#menu").menu();
            menus: "div.subMenu"
        });
    </script>

```

```

    </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>

  <div id="menu">
    <div><a>Bouquets</a></div>
    <div><a>Heirloom Blooms</a></div>
    <div>-</div>
    <div><a>Summer Color</a>
      <div class="subMenu">
        <div><a>Aster</a></div>
        <div><a>Rose</a></div>
        <div><a>Orchid</a></div>
      </div>
    </div>
    <div><a>Wedding Classics</a></div>
    <div>-</div>
    <div><a>Exotica</a></div>
  </div>
</body>
</html>

```

在上面的例子中，我为代表子菜单的div元素添加了subMenu类，并在创建菜单组件时把menus属性配置为div.subMenu选择器。

---

**提示** 不是必须给子菜单元素添加一个类，也不是必须把这个类的名字叫subMenu。只要选择器能匹配子菜单元素，任何形式都可以。

---

## 2. 菜单图标

把icons属性的值设置成任意一个在第18章中提到的jQuery UI图标，即可为子菜单指定图标。代码清单23-6演示了icons选项的用法。

**代码清单23-6** 设定子菜单所用图标

```

...
<script type="text/javascript">
  $(document).ready(function () {
    $("#menu").menu({
      menus: "div.subMenu",
      icons: { submenu: "ui-icon-circle-plus" }
    });
  });
</script>
...

```

icons属性的值是一个对象。它有一个submenu属性，用于定义使用哪种图标表示子菜单，和以前一样，我们可以把它的值设为想要使用的图标名。如图23-4所示，在上面的例子里，我把submenu属性的值设置为一个圆圈中的加号。

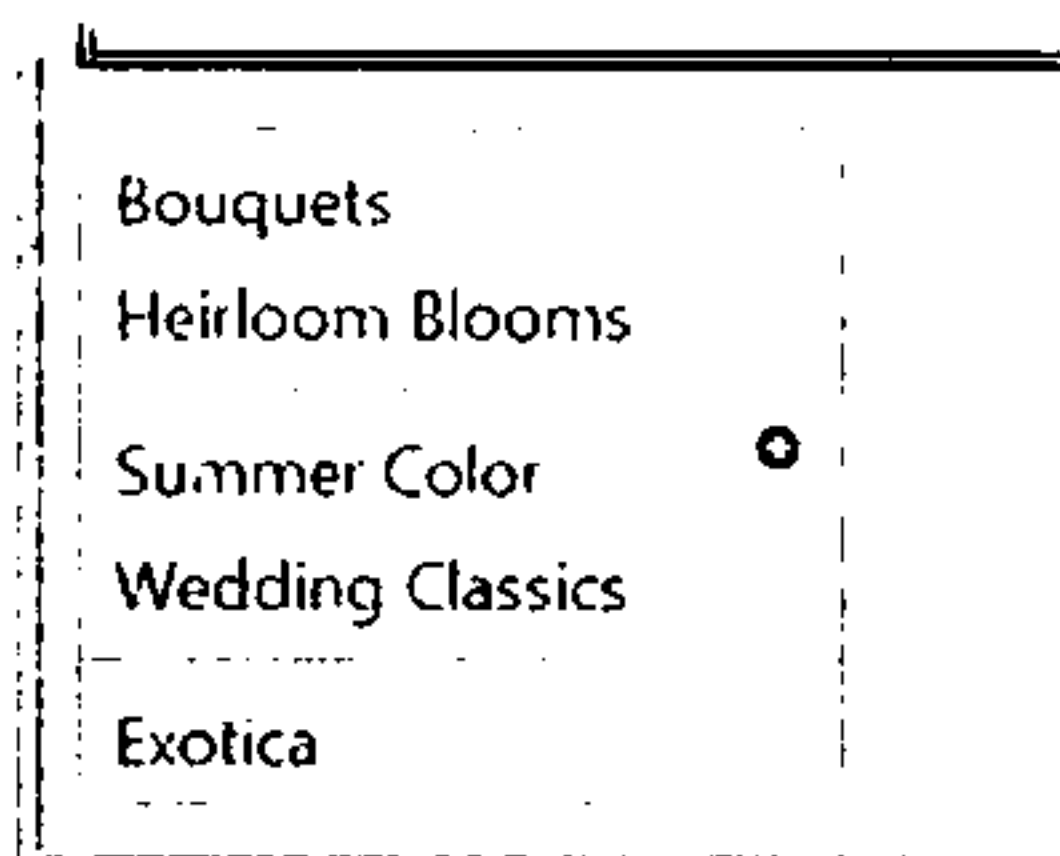


图23-4 改变表示子菜单的图标

如代码清单23-7所示，只需在菜单结构里为菜单项添加<span>元素，并为span元素指定图标类，即可为菜单项指定图标。

#### 代码清单23-7 为菜单项指定图标

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style>
    .ui-menu { width: 200px; }
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#menu").menu({
        icons: { submenu: "ui-icon-circle-plus" }
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>

  <ul id="menu">
    <li><a>Bouquets</a></li>
    <li><a>Heirloom Blooms</a></li>
    <li></li>
    <li><a>Summer Color</a>
      <ul>
        <li><a>Aster</a></li>
        <li><a>Rose</a></li>
        <li><a>Orchid</a></li>
      </ul>
    </li>
    <li>
      <a><span class="ui-icon ui-icon-circle-check"></span>Wedding Classics</a>
    </li>
  </ul>
```

```

        </li>
        <li>-</li>
        <li class="ui-state-disabled"><a>Exotica</li>
    </ul>
</body>
</html>

```

span元素必须位于a元素内，且拥有ui-icon类以及控制图标显示的类。在上面的例子中，ui-icon-circle-check类决定显示哪个图标。这段代码的实际效果见图23-5。

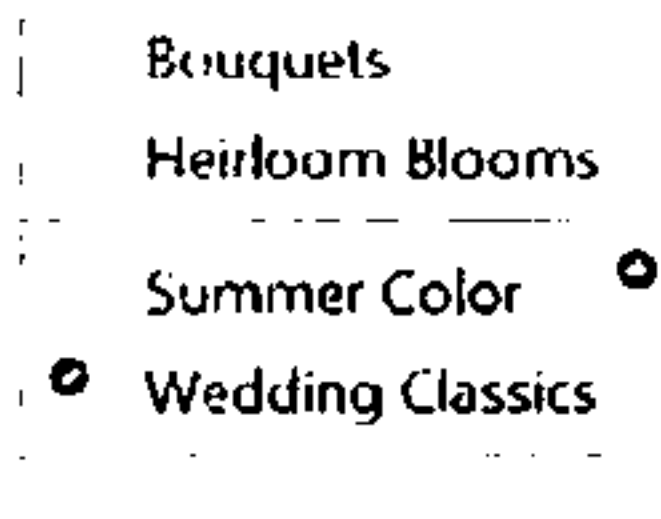


图23-5 为菜单项指定图标

### 3. 子菜单的弹出位置

position选项用于设定子菜单的弹出位置，这个选项值的格式与我们在第19章中介绍的位置格式相同。建议由菜单组件自动安排子菜单的位置，因为弹出菜单是透明的，即便弹出菜单遮挡住了主菜单，我们仍可以看到底下的元素。在代码清单23-8当中，我使用position选项把子菜单的显示位置定义为父菜单的中间位置。

#### 代码清单23-8 子菜单的弹出位置

```

...
<script type="text/javascript">
    $(document).ready(function () {
        $("#menu").menu({
            icons: { submenu: "ui-icon-circle-plus" },
            position: {
                my: "left center",
                at: "center center"
            }
        });
    });
</script>
...

```

这段代码的实际效果见图23-6，虽然弹出菜单遮住了主菜单，我们仍然可以看见底下的菜单项。

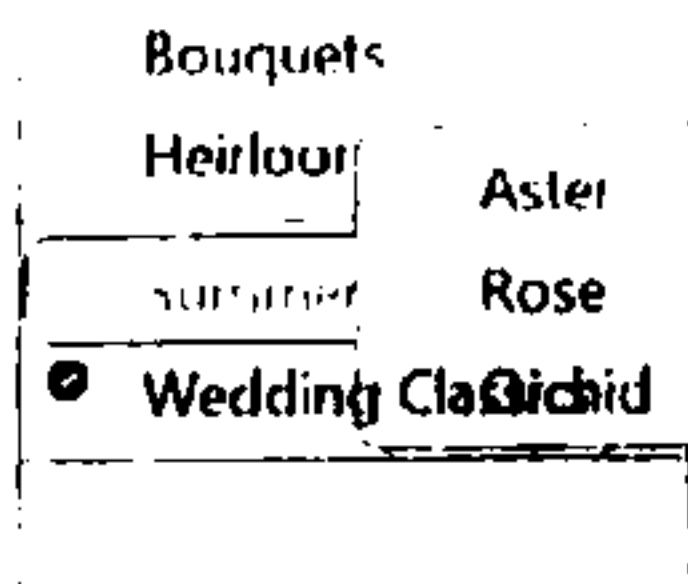


图23-6 指定子菜单的弹出位置

### 23.1.3 菜单组件的方法

jQuery UI菜单组件支持的方法见表23-3。我认为这些方法聊胜于无，并没有多少实际用途。菜单组件最大的功能在于导航，这些方法的用途最好还是留给读者自行探究。使用这些事件能够轻易营造出“扎眼”的菜单效果，我的建议是少用为佳。

表23-3 菜单组件支持的方法

方 法	描 述
<code>menu("blur")</code>	触发blur事件（稍后会介绍），使菜单失去焦点
<code>menu("collapse")</code>	关闭当前展开的子菜单
<code>menu("collapseAll")</code>	关闭所有展开的子菜单
<code>menu("destroy")</code>	从底层元素上移除菜单组件
<code>menu("disable")</code>	禁用菜单组件
<code>menu("enable")</code>	启用菜单组件
<code>menu("expand")</code>	展开当前菜单的子菜单
<code>menu("focus")</code>	让一个菜单项获得焦点
<code>menu("isFirstItem")</code>	如果当前选中菜单项是第一个菜单项，返回true
<code>menu("isLastItem")</code>	如果当前选中菜单项是最后一个菜单，返回true
<code>menu("next")</code>	把焦点移到下一个菜单项
<code>menu("option")</code>	改变一个或多个选项
<code>menu("previous")</code>	把焦点移到上一个菜单项
<code>menu("refresh")</code>	更新菜单组件的状态以反映底层元素的变化
<code>menu("select")</code>	选中激活菜单项，关闭所有打开的子菜单，并触发 select事件（稍后会介绍）

演示这些方法如何使用是一件困难的事，因为菜单组件的行为与当前选中菜单项密切相关。试图用按钮来控制菜单注定会失败，因为当你点击按钮时，焦点就会离开当前菜单。建议不要使用这些方法，而是依托标准的用户交互，使用下一节将要介绍的技术处理交互产生的事件。

**注意** 菜单组件还定义了一些方法支持滚动菜单的导航，但这一功能还相当不稳定（在本书写作时），因此我在上表中略去了这些方法。

### 23.1.4 菜单组件事件

jQuery UI菜单组件支持的事件见表23-4。

表23-4 菜单组件事件

方 法	描 述
<code>blur</code>	在菜单失去焦点时触发此事件（调用blur方法可以主动触发）
<code>create</code>	在组件创建完成时触发
<code>focus</code>	当菜单项获得焦点，或者被激活时触发此事件（调用focus方法可以主动触发）
<code>select</code>	当菜单项被选中（由用户选中或者调用了select方法）时触发此事件

在代码清单23-9中，我利用blur和focus事件跟踪用户激活的每一个菜单项，并在用户点击菜单项时使用select事件做出回应。

代码清单23-9 菜单组件的blur、focus、select事件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style>
    .ui-menu { width: 200px; }
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#menu").menu({
        focus: function (e, ui) {
          console.log("Focus: " + ui.item.find("a").first().text());
        },
        blur: function () {
          console.log("Blur");
        },
        select: function (e, ui) {
          console.log("Select: " + ui.item.find("a").first().text());
          e.preventDefault();
        }
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>

  <ul id="menu">
    <li><a>Bouquets</a></li>
    <li><a>Heirloom Blooms</a></li>
    <li>-</li>
    <li><a>Summer Color</a>
      <ul>
        <li><a href="http://apress.com">Aster</a></li>
        <li><a>Rose</a></li>
        <li><a>Orchid</a></li>
      </ul>
    </li>
    <li><a>Wedding Classics</a></li>
    <li>-</li>
    <li class="ui-state-disabled"><a>Exotica</a></li>
  </ul>
</body>
</html>
```

三个处理函数均支持jQuery event对象参数（详见第9章）和ui对象（它的item属性是一个jQuery对象）参数。这个jQuery对象包含着事件对应的HTML元素。理论上是这样，但是写作本书时，blur事件发生时ui对象的item属性并没有被设置成正确的值。

鉴于这种情况，在focus和select事件发生时，我输出了（触发事件的）jQuery对象中第一个元素的文本内容。而当blur事件发生时，我只是简单地记录了“Blur”。运行这个例子，并随便点击菜单项，就能在控制台看到类似下面内容的输出。

---

```
Focus: Bouquets
Blur
Focus: Heirloom Blooms
Blur
Focus: Summer Color
Select: Summer Color
Blur
Focus: Aster
Blur
Focus: Aster
Select: Aster
Blur
Focus: Aster
Blur
```

---

在输出到控制台的信息中，我特意加粗了其中一行，即那个包含子菜单的菜单项。这个菜单项同样可以被选中，在本例中，即便它拥有一个子菜单，我仍然能够点击Summer Color菜单项。若在处理select事件时不考虑到这一点，很可能会发生意料之外的行为。

---

**提示** 处理select事件时，我调用了preventDefault方法，以避免浏览器跳转到a元素href属性定义的链接。

---

## 23.2 jQuery UI 提示说明组件

提示说明组件主要用于向用户弹出具有帮助意义的上下文信息。它们是一柄“双刃剑”，使用得当能引导用户完成复杂操作，然而更多时候，提示说明被滥用，最终对用户造成干扰。

最常见的误区是告诉了用户一些他们已经知道的事，并且这样一来，反而失去了提供有用信息的机会，这种情况在Web表单要求复杂数据时尤为常见。我见过的最近一个例子是一个在线税务表单。税务是一件极其复杂的事，开发者认为给每个输入项提供说明信息会有帮助。这本是一个善意的举动，然而每个说明都只是简单重复表单标题里已经说明的内容。例如，标题为Birth Date的输入框提示信息是“输入你的生日具体日期”，却不告诉用户应该使用什么格式，是月/日/年，日/月/年，还是年/月/日。这样的例子在网上随处可见，每一个都白白浪费了向用户展示有用信息的机会。另一个与提示有关的问题是，它会阻止用户完成任务。在税务表单上，每一次我在关键栏目上猜错需要的格式，弹出的提示信息都会遮住输入框，这让表单的输入变得更慢。



建议谨慎使用提示说明，认真考虑所提供信息的价值。如果只是简单重复用户已经知道的信息，就没有必要弹出提示信息。

### 23.2.1 创建提示说明组件

如代码清单23-10所示，jQuery UI提示说明组件可用于任何HTML元素，只需在目标元素上调用tooltip方法即可。默认情况下，它会显示目标元素title属性的值。

**提示** 主流浏览器无需使用jQuery UI也会弹出信息显示title属性的值。提示说明组件的长处在于它让我们能够自定义弹出信息的样式，以保持整站的风格一致，并能精细控制提示信息如何显示，同时支持内容的范围更广。

23

代码清单23-10 创建提示说明组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style>
    [title] { font-weight: bold; font-style:italic }
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("[title]").tooltip();
    });
  </script>
</head>
<body class="ui-widget">
  <h1>Jacqui's Flower Shop</h1>

  <h3>Color and Beauty to Your Door</h3>
  <p>We are pleased to announce that we are starting a home delivery service for
  your flower needs. We will deliver within a
  <span title="We are at 200 Main St">20 mile radius</span>
  of the store for free and $1/mile thereafter.</p>
</body>
</html>
```

在这个例子里，我定义了一个拥有title属性的span元素，我选中了所有拥有title属性的元素，然后调用tooltip方法。为了突出显示这些拥有title属性的元素，我用同样的选择器为这些元素设置了样式。最终，jQuery UI为这些元素创建了提示说明组件，鼠标悬停在span元素上的效果见图23-7。

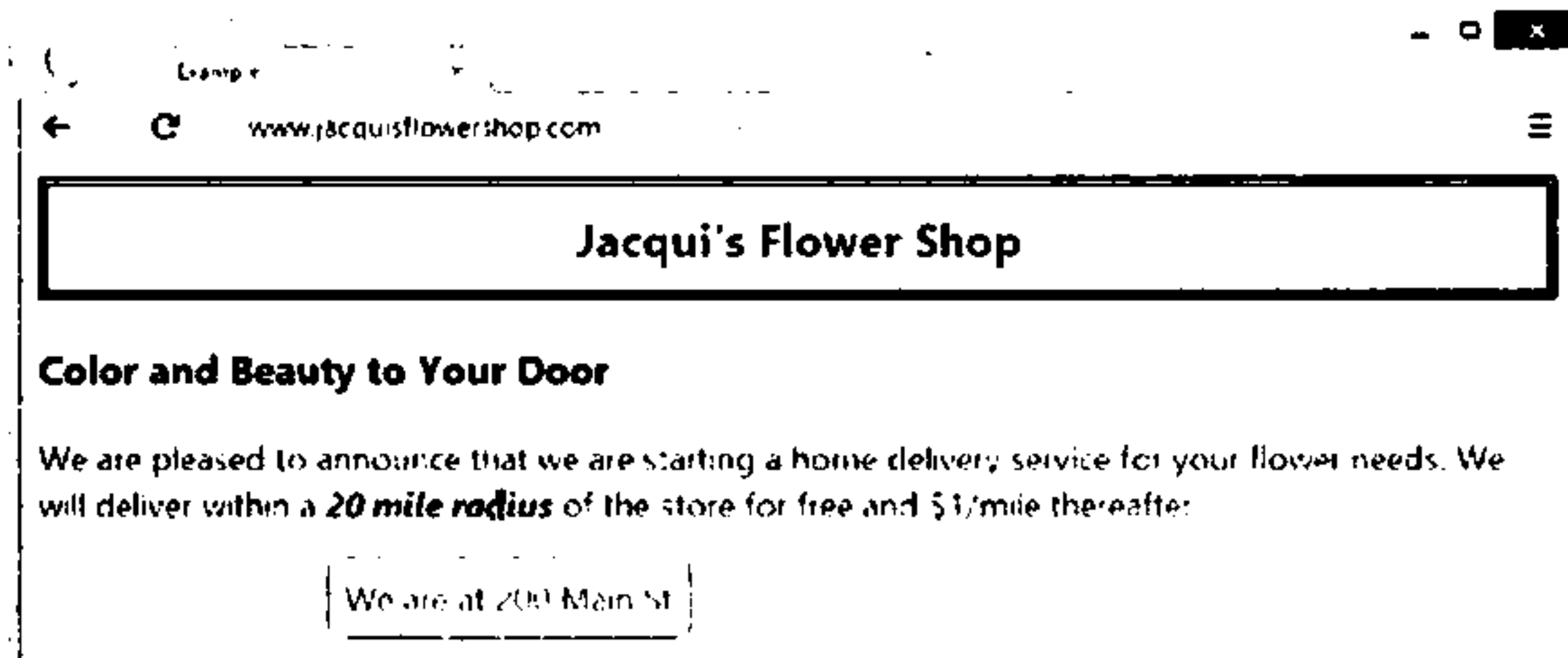


图23-7 jQuery UI提示

### 在input元素上使用提示说明组件

当提示信息所关联的元素获得焦点时，组件提示信息也会显示——在某种程度上，这对input元素会很有用。当用户在表单里切换到某个输入框时，相应的提示就会自动出现。如代码清单23-11所示，在input元素上使用提示说明组件的方式与其他元素并无二致。

#### 代码清单23-11 在input元素上使用提示说明组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style>
    input { width: 150px; }
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("input").tooltip();
    });
  </script>
</head>
<body class="ui-widget">
  <h1>Jacqui's Flower Shop</h1>
  <div><label>Name:</label><input title="Use lastname, firstname" /></div>
  <div><label>City:</label><input title="Don't include your state" /></div>
</body>
</html>
```

在上面的例子中，共有两个input元素有title属性，在这两个输入框之中使用TAB键切换焦点的显示效果见图23-8。

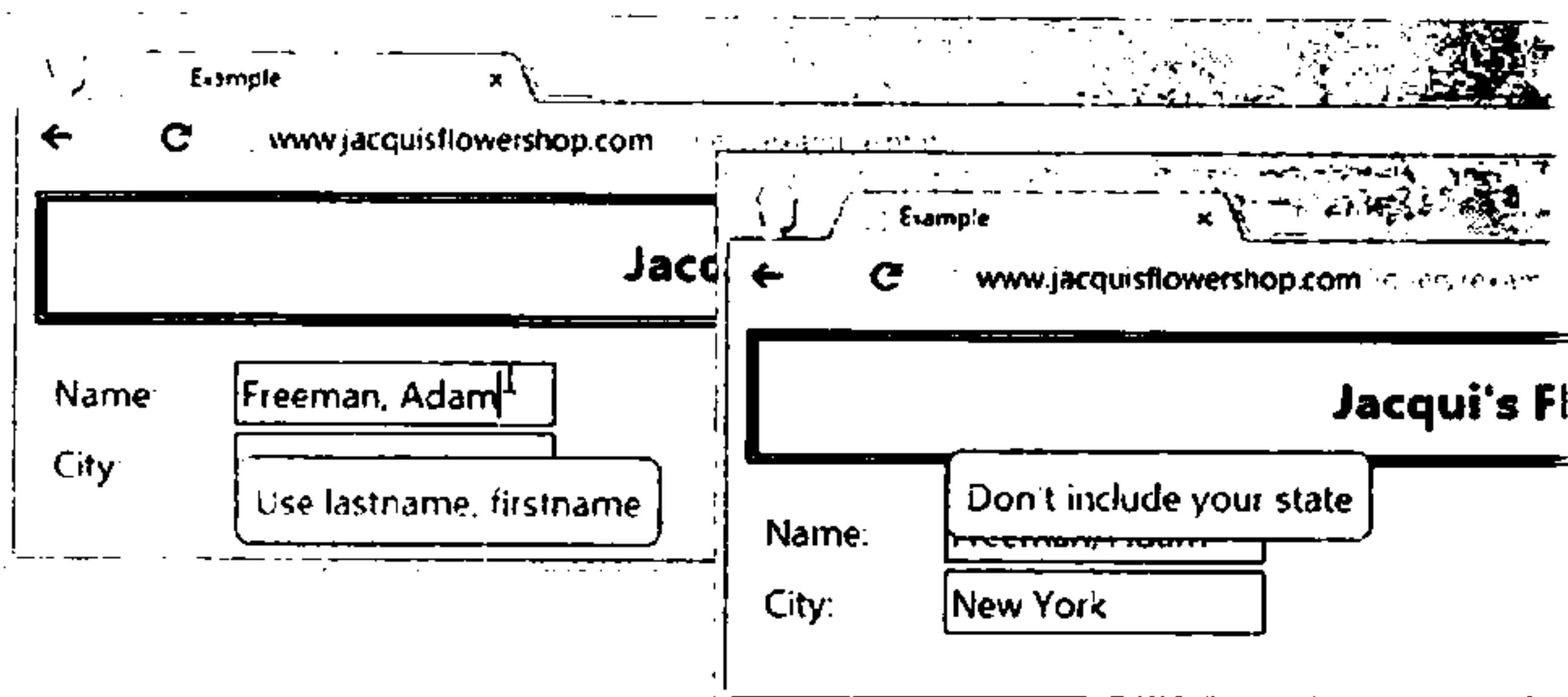


图23-8 在input元素上使用提示说明组件

### 23.2.2 配置提示说明组件

提示说明组件支持许多配置选项，利用这些选项我们可以定制提示信息的呈现方式。表23-5列出了这些选项，在后面的内容中我们将演示这些选项的用法。

表23-5 提示说明组件的配置选项

选 项	描 述
content	设置提示内容，可以是一段HTML字符串，也可以是一个函数
disabled	若设置为true，则禁用组件
hide	指定提示信息如何隐藏（参阅第34章详细了解jQuery UI对动画的支持）
items	指定一个选择器，该选择器用来进一步限制需要提示说明组件的元素范围
position	指定提示信息相对于关联元素的显示位置
show	指定提示信息如何显现出来（参阅第34章详细了解jQuery UI对动画的支持）
tooltipClass	为提示信息元素指定一个需要添加的额外的类，以支持不同类型的提示泡泡（例如出错泡泡）
track	若设为true，当鼠标放在底层元素之上时，提示层会跟随鼠标的位置联动

#### 1. 设置提示说明的显示内容

jQuery UI提示说明组件最强大的地方，就在于它可以支持格式内容，即我们可以直接指定HTML片段做为提示内容，也可以指定一个JavaScript函数，来动态生成提示内容。在代码清单23-12中，可以看到第一种用法。在这个例子里，我指定一段带格式的HTML文本做为提示说明的内容。

代码清单23-12 利用content选项显示格式文本

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
</head>
```

```

<style>
    span.toolTip { font-weight: bold; font-style:italic }
</style>
<script id="tipContent" type="text/html">
    We are at <b>200</b> Main Street
</script>
<script type="text/javascript">
    $(document).ready(function () {
        $("span.toolTip").tooltip({
            content: $("#tipContent").html(),
            items: "span.toolTip"
        });
    });
</script>
</head>
<body class="ui-widget">
    <h1>Jacqui's Flower Shop</h1>

    <h3>Color and Beauty to Your Door</h3>
    <p>We are pleased to announce that we are starting a home delivery service for
    your flower needs. We will deliver within a
    <span class="toolTip">20 mile radius</span>
    of the store for free and $1/mile thereafter.</p>
</body>
</html>

```

在这个例子里我定义了一个新的script标签，但是把它的type属性设置成了text/html，这样浏览器就不会把这个标签的内容当成JavaScript脚本，而是作为HTML内容来解释。这一技术同样适用于JavaScript数据模板（参见第12章），不过在这里，我的目的仅仅是把一段与网页主体无关的HTML片段暂存在这个标签里，直到需要它为止。

使用content选项把一段HTML传递给提示说明组件。注意，必须选中那个包含HTML片段的script元素，然后调用html方法得到它的内容，因为提示说明组件并未与jQuery良好整合，它只会把传进来HTML当成字符串。

同时必须使用items选项，因为在这里我不再使用title属性，而提示说明组件会默认查找title属性。如果没有items选项，调用tooltip方法时，jQuery UI就会直接忽略span元素。我已经为span元素添加了toolTip类，这个类构成了目标选择器，用于items选项。

script元素内的HTML片段，用作提示说明组件的显示内容，具体效果见图23-9，注意图中我用b标签添加的用粗体显示的信息。

---

**提示** 我在这个例子中展示的是格式文本，不过你可以使用jQuery UI提示说明组件显示任何HTML内容，包括图像在内。

---

## 2. 使用函数生成提示内容

也可以使用函数生成提示内容。如代码清单23-13所示，为了方便就地生成提示内容，我让函数直接返回HTML片段。

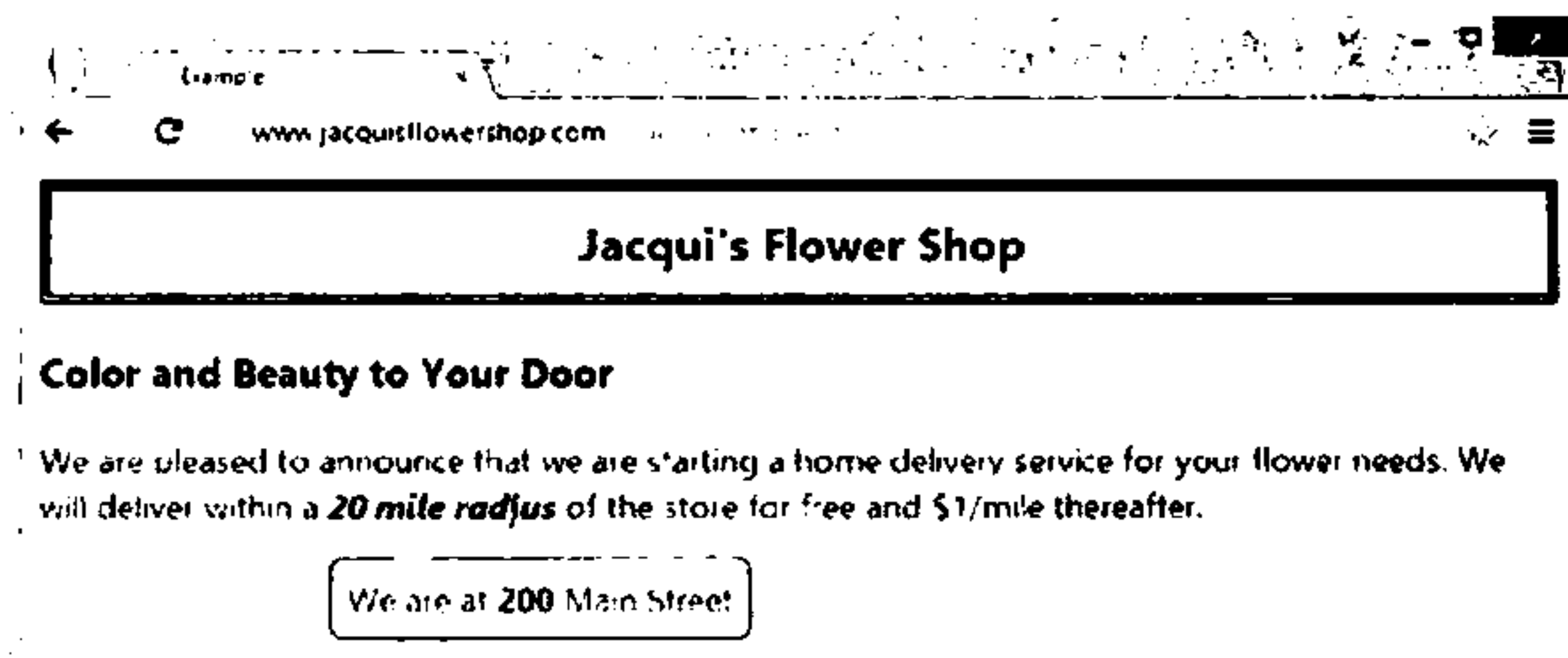


图23-9 设置提示信息的样式

## 代码清单23-13 使用函数生成提示内容

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style>
    span.toolTip { font-weight: bold; font-style:italic }
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("span.toolTip").tooltip({
        content: function () {
          if (this.id == "address") {
            return "We are at <b>200</b> Main Street";
          } else {
            return "Fee capped at <i>$20</i> during June!"
          }
        },
        items: "span.toolTip"
      });
    });
  </script>
</head>
<body class="ui-widget">
  <h1>Jacqui's Flower Shop</h1>

  <h3>Color and Beauty to Your Door</h3>
  <p>We are pleased to announce that we are starting a home delivery service for
  your flower needs. We will deliver within a
  <span id="address" class="toolTip">20 mile radius</span> of the store for free and
  <span id="maxPrice" class="toolTip">$1/mile thereafter.</span></p>
</body>
</html>

```

在这个例子里，我定义了两个拥有toolTip类的span元素，我会使用函数为这两个span元素提供不同的提示内容。调用我们定义的函数时，this的值被设置为触发显示提示内容动作的元素（即鼠标位置对应的元素）。利用id属性会得到该元素id属性的值，并根据这个值返回不同的HTML片段。实际效果见图23-10。

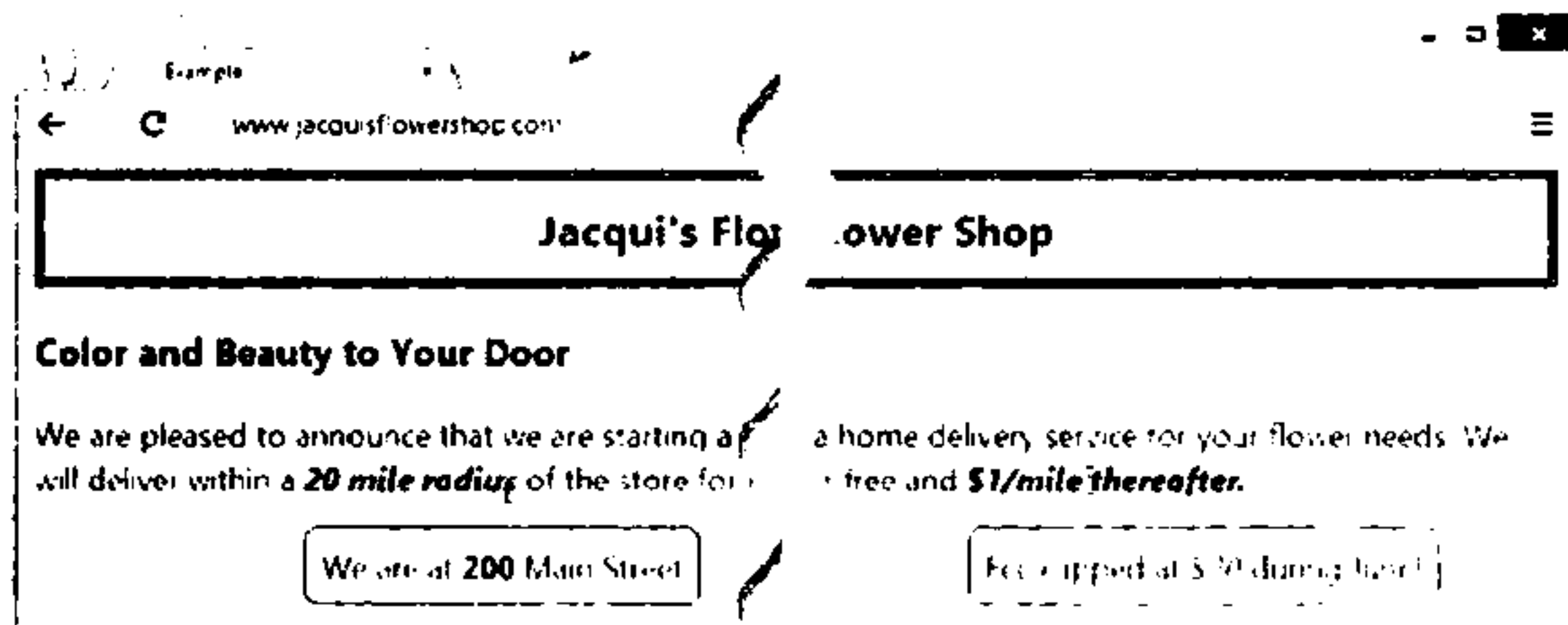


图23-10 使用函数生成提示内容

### 3. 从远程服务器获取提示内容

我们也可以使用Ajax技术以异步方式得到提示内容，使用回调函数为jQuery UI提示说明组件提供HTML片段。为了演示这种用法，我添加了一个新文件tooltips.json，并把这个文件与其他例子文件放到同一目录之下。代码清单23-14列出了tooltips.json文件的内容：

#### 代码清单23-14 tooltips.json文件内容

```
{"address": "We are at <b>200</b> Main Street",  
  "maxPrice": "Fee capped at <i>$20</i> during June!"}
```

在实际应用中，服务器通常会根据目标元素有针对性地返回提示信息。为了简单起见，我的JSON文件包含了示例需要的所有提示信息。在代码清单23-15中，你会看到如何获取这些数据，并把它作为提示内容传递给提示说明组件。

#### 代码清单23-15 从远程服务器获取内容

```
...  
<script type="text/javascript">  
  $(document).ready(function () {  
  
    var tooltipData;  
  
    $("span.toolTip").tooltip({  
      content: function (callback) {  
        if (tooltipData != null) {  
          console.log("Requested serviced locally: " + this.id);  
          return tooltipData[this.id];  
        } else {  
          var elemID = this.id;  
          $.getJSON("tooltips.json", function (data) {  
            tooltipData = data;  
            console.log("Requested serviced remotely: " + elemID);

```

```

        callback(tooltipData[elemID]);
    });
}
},
items: "span.toolTip"
));
});
</script>
...

```

由于JSON文件包含例子需要的所有数据，我只需向服务器请求一次，并把数据储存起来，就能应对后续的提示需要，这也正是例子中代码所做之事。示例代码的要点在于除非用于设置content选项的函数执行完成，否则传递给getJSON方法（参见第14章）的回调函数就不会执行。这意味着我们不能只返回HTML片段给提示说明组件。

为此，content选项函数接受一个回调函数做为参数，需要准备HTML片段时，就调用这个函数。

```

...
content: function (callback) {
...

```

当Ajax请求完成时，我会以需要显示的数据为参数调用callback函数。

```

...
callback(tooltipData[elemID]);
...

```

除显示内容是利用Ajax获取之外，这个例子的最终效果与前面的例子并无显著区别。

**提示** 将此特性纳入本章只是为了尽量完整地介绍组件功能。我建议尽量不要使用这一特性，提示信息只在能够即时提供时才有价值，然而这一特性在Ajax请求需要一段时间才能完成时，用户就要等待很久才能看到提示信息。

#### 4. 为提示信息添加额外的类

利用tooltipClass选项，可以为提示信息添加一个或多个类，从而能够随心所欲地控制提示信息的样式。代码清单23-16演示了这个选项的用法。

#### 代码清单23-16 tooltipClass选项

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <style>
        span.toolTip { font-weight: bold; font-style:italic }
        *.customTip { border-color: red; }
    </style>
    <script type="text/javascript">

```

```

$(document).ready(function () {
    $("span.toolTip").tooltip({
        content: function () {
            if (this.id == "address") {
                return "We are at <b>200</b> Main Street";
            } else {
                return "Fee capped at <i>$20</i> during June!"
            }
        },
        items: "span.toolTip",
        tooltipClass: "customTip"
    });
});
</script>
</head>
<body class="ui-widget">
    <h1>Jacqui's Flower Shop</h1>
    <h3>Color and Beauty to Your Door</h3>
    <p>We are pleased to announce that we are starting a home delivery service for
    your flower needs. We will deliver within a
    <span id="address" class="toolTip">20 mile radius</span> of the store for free and
    <span id="maxPrice" class="toolTip">$1/mile thereafter.</span></p>
</body>
</html>

```

我在样式表中定义了一个customTip类，并使用tooltipClass选项把这个类应用到提示信息元素上。customTip类把提示信息的边界颜色定义为红色，产生的实际效果见图23-11。

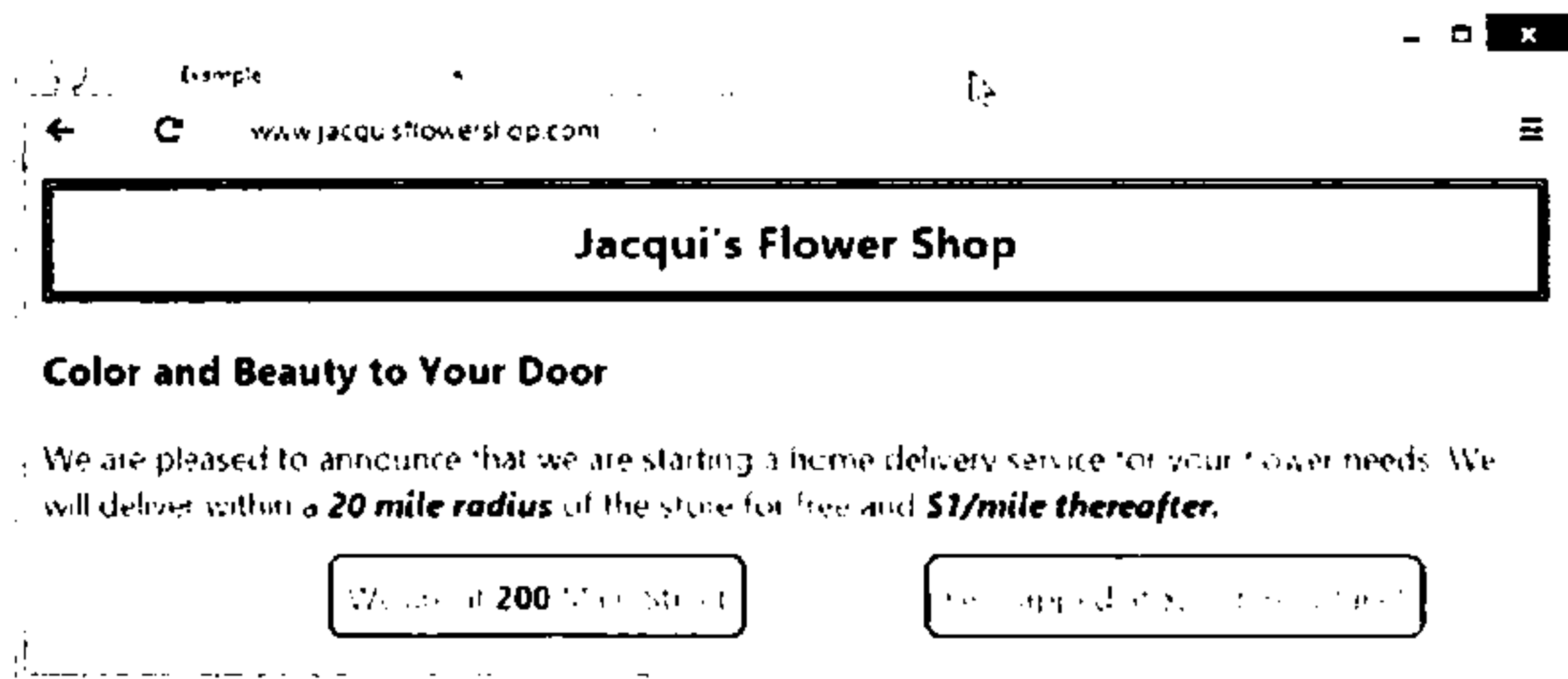


图23-11 为提示信息应用自定义类

由jQuery UI提示信息组件生成的HTML中，包含着可用于定制样式的另一个类。下面的示例代码就源自上例中提示信息组件的某个实例：

```

...
<div id="ui-tooltip-3" role="tooltip" class="ui-tooltip ui-widget ui-corner-all
    ui-widget-content customTip" style="top: 210.15625px; left: 161px; display: block;">
    <div class="ui-tooltip-content">We are at <b>200</b> Main Street</div>
</div>
...

```

tooltipClass选项对应的类，是应用于提示信息的外层div元素，而提示信息本身是内层div元素



的内容，即定义了ui-tooltip-content类的那个div。如代码清单23-17所示，可以同时为组件“壳”元素和提示内容元素分别定义样式，这算比较高阶的用法。

**提示** 这个例子中引用的另外一些类，比如ui-widget和ui-corner-all，都来自jQuery UI CSS框架（详见本书第35章）。

#### 代码清单23-17 为组件和内容分别定义样式

```
...
<style>
  span.tooltip { font-weight: bold; font-style: italic; }
  *.customTip { border-color: red; }
  *.ui-tooltip-content { border: thick solid black; margin: 10px; padding: 10px;
                        background-color: white; }
</style>
...
```

上面代码的实际效果见图23-12。

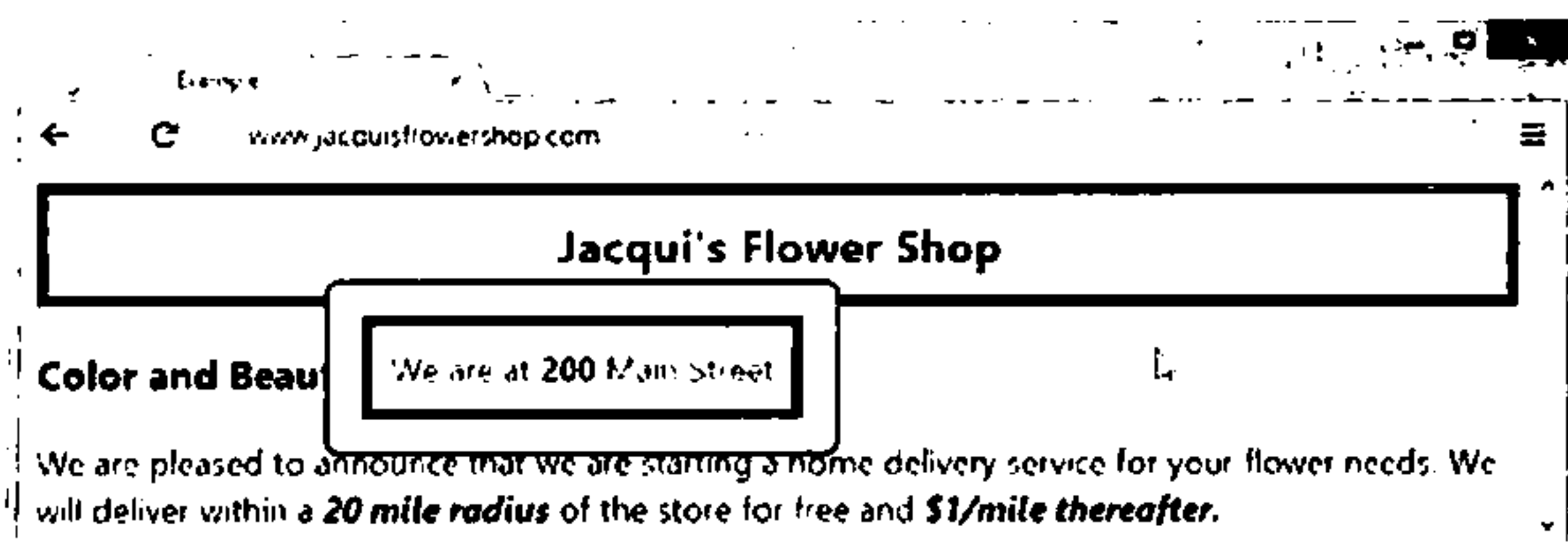


图23-12 分别定义组件样式和提示内容样式

**提示** 如果希望定义tooltip的背景颜色，需要在tooltipClass选项指定的类中把background-image属性设置成none。jQuery UI组件使用图片而非颜色设置背景。

### 5. 鼠标跟随

如果把track选项设置为true，提示信息将在组件的有效提示范围内跟随鼠标移动。这在我们需要根据鼠标位置及时改变提示内容的时候会很有用，代码清单23-18就是一个这样的例子：提示信息及时反映了鼠标位于底层元素的位置。

#### 代码清单12-18 让提示信息鼠标跟随

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
```

```

<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<style>
    span.toolTip { font-weight: bold; font-style:italic }
</style>
<script type="text/javascript">
    $(document).ready(function () {
        $("span.toolTip").tooltip({
            content: "Move the mouse",
            items: "span.toolTip",
            track: true
        }).mousemove(function(e) {
            $(this).tooltip("option", "content",
                "X: " + e.pageX + " Y: " + e.pageY);
        });
    });
</script>
</head>
<body class="ui-widget">
    <h1>Jacqui's Flower Shop </h1>
    <h3>Color and Beauty to Your Door </h3>
    <p>We are pleased to announce that we are starting a home delivery service for
    your flower needs. We will deliver within a
    <span id="address" class="toolTip">20 mile radius</span> of the store for free and
    <span id="maxPrice" class="toolTip">$1/mile thereafter.</span></p>
</body>
</html>

```

把track选项设置为true,从而打开鼠标跟随特性。然后在mousemove事件处理函数中使用option方法随时更新提示信息的内容。这是一个唯有亲自运行才会感受深刻的例子,图23-13是这一新奇行为的一个快照。

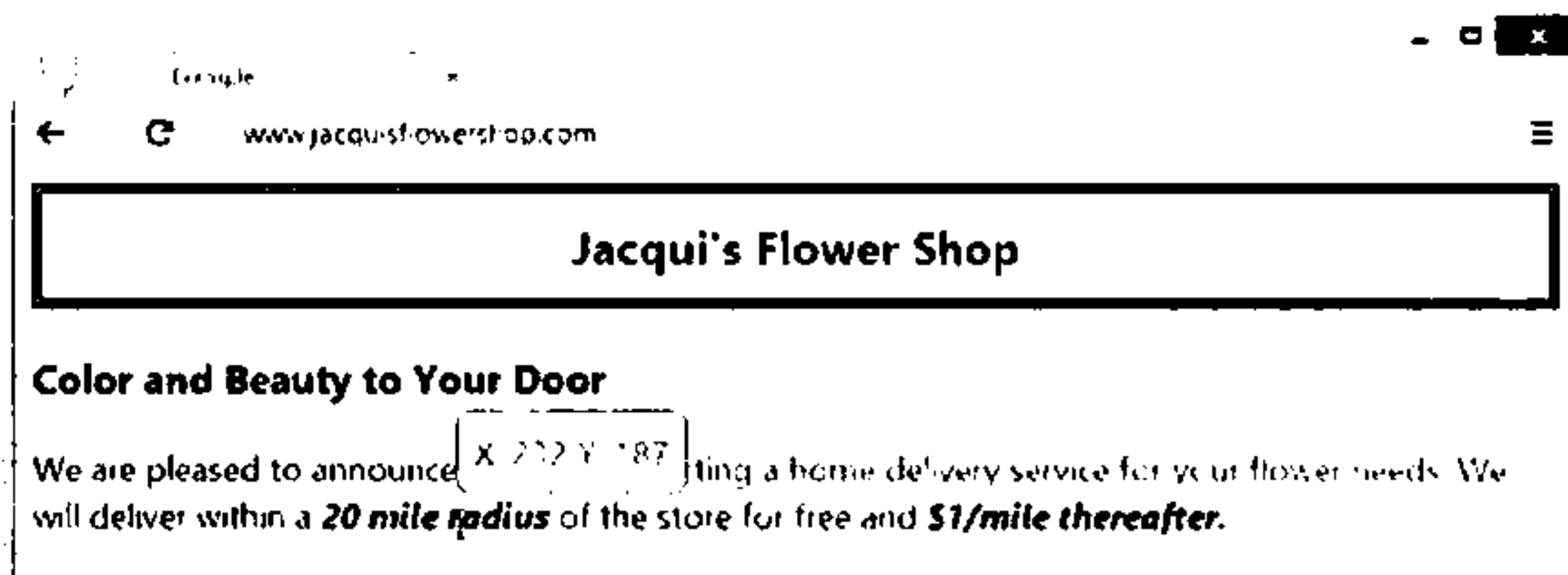


图23-13 提示信息的位置和内容随鼠标位置的变化而变化

## 6. 设置提示信息的显示位置

position选项用于指定提示信息相对于关联元素的显示位置,指定位置的数据格式与我们在第19章中讲过的格式完全相同。具体见代码清单23-19。

### 代码清单23-19 设置提示信息的显示位置

```

<!DOCTYPE html>
<html>

```

```

<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style>
    span.toolTip { font-weight: bold; font-style:italic }
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("span.toolTip").tooltip({
        content: function () {
          if (this.id == "address") {
            return "We are at <b>200</b> Main Street";
          } else {
            return "Fee capped at <i>$20</i> during June!"
          }
        },
        items: "span.toolTip",
        position: {
          my: "center bottom",
          at: "center top"
        }
      });
    });
  </script>
</head>
<body class="ui-widget">
  <h1>Jacqui's Flower Shop</h1>
  <h3>Color and Beauty to Your Door</h3>
  <p>We are pleased to announce that we are starting a home delivery service for
  your flower needs. We will deliver within a
  <span id="address" class="toolTip">20 mile radius</span> of the store for free and
  <span id="maxPrice" class="toolTip">$1/mile thereafter.</span></p>
</body>
</html>

```

在这个例子中，我指定提示信息的显示位置为关联元素的正上方，显示效果见图23-14。

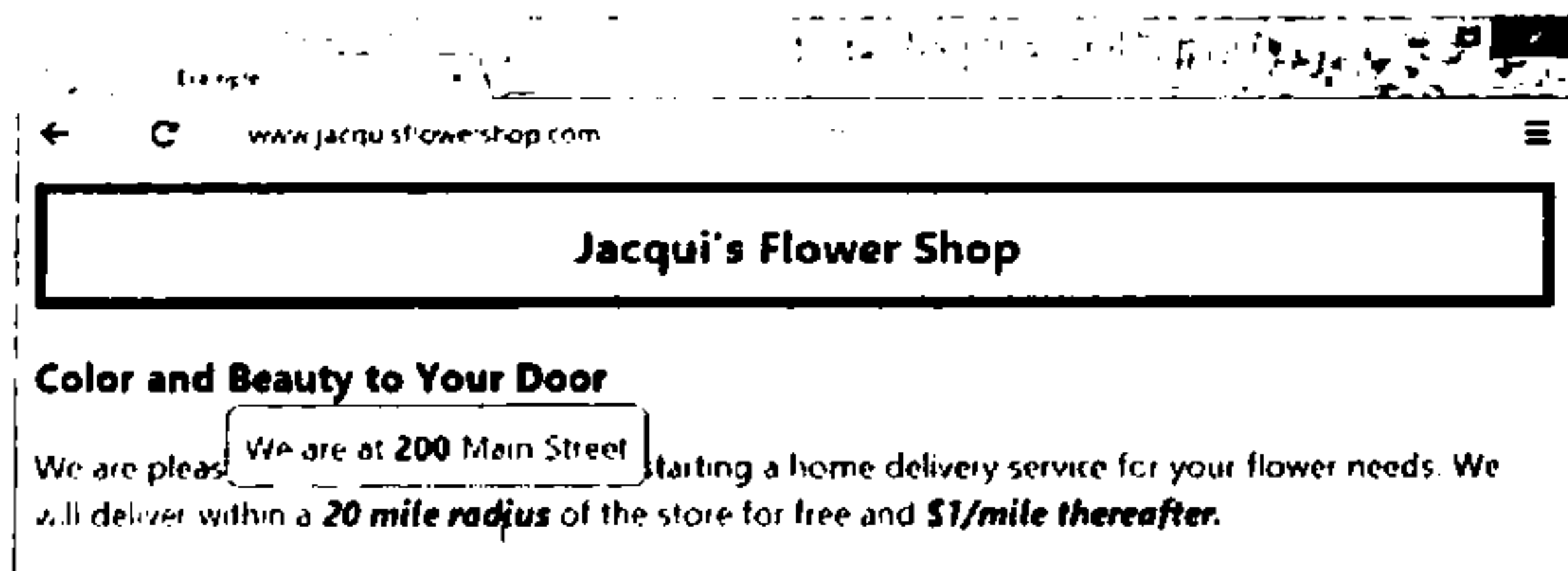


图23-14 设置提示信息的显示位置

### 23.2.3 提示说明组件的方法

jQuery UI提示说明组件支持的方法见表23-6。

表23-6 提示说明组件的方法

方 法	描 述
tooltip("close")	如果提示信息层处于显示状态，则关闭它
tooltip("destroy")	从底层元素中移除组件
tooltip("disable")	禁用组件，阻止提示信息显示
tooltip("enable")	启用组件，允许提示信息显示
tooltip("open")	如果提示信息层处于关闭状态，打开它
tooltip("option")	设置配置选项

值得一提的方法是open和close。利用这两个方法，我们可以用脚本控制提示信息的显示。在代码清单23-20中，你会看到如何使用按钮利用这些方法来显示或者隐藏页面中所有的提示信息。

代码清单23-20 open方法和close方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style>
    span.toolTip { font-weight: bold; font-style:italic }
  </style>
  <script type="text/javascript">
    $(document).ready(function () {
      $("span.toolTip").tooltip({
        content: function () {
          if (this.id == "address") {
            return "We are at <b>200</b> Main Street";
          } else {
            return "Fee capped at <i>$20</i> during June!"
          }
        },
        items: "span.toolTip",
        position: {
          my: "center bottom",
          at: "center top"
        }
      });
      $("button").button().click(function (e) {
        $("span.toolTip").tooltip(this.id);
      });
    });
  </script>
```

```
</head>
<body class="ui-widget">
  <h1>Jacqui's Flower Shop</h1>
  <h3>Color and Beauty to Your Door</h3>
  <p>We are pleased to announce that we are starting a home delivery service for
  your flower needs. We will deliver within a
  <span id="address" class="toolTip">20 mile radius</span> of the store for free and
  <span id="maxPrice" class="toolTip">$1/mile thereafter.</span></p>
  <div>
    <button id="open">Open</button>
    <button id="close">Close</button>
  </div>
</body>
</html>
```

该脚本产生的效果见图23-15。

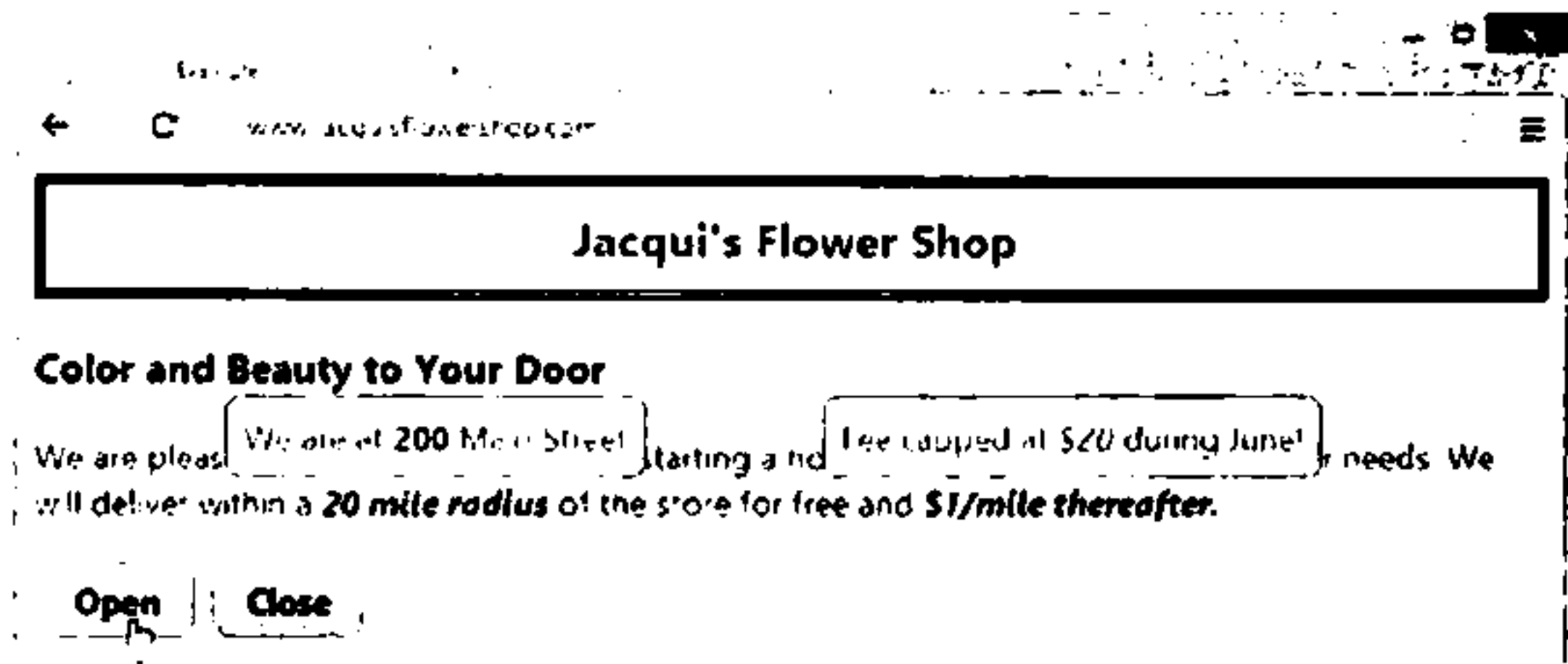


图23-15 利用脚本控制提示信息层的显示和隐藏

23.2.4 提示说明组件事件

jQuery UI提示说明组件支持的事件见表23-7。

表23-7 提示说明组件事件

事 件	描 述
close	当提示信息层关闭时触发
create	当组件创建完成时触发
open	当提示信息层显示完成时触发

代码清单23-21中演示了open和close事件的用法。事件处理函数会接到一个jQuery事件对象参数（详见第9章）和一个ui对象参数，这个ui对象参数的tooltip属性是包含提示信息元素的jQuery对象。

代码清单23-21 处理open事件和close事件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
```

```

<script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<style>
    span.toolTip { font-weight: bold; font-style:italic }
    span.active { border: medium solid red; padding: 10px; background-color: white }
</style>
<script type="text/javascript">
    $(document).ready(function () {
        $("span.toolTip").tooltip({
            content: function () {
                if (this.id == "address") {
                    return "We are at <b>200</b> Main Street";
                } else {
                    return "Fee capped at <i>$20</i> during June!"
                }
            },
            items: "span.toolTip",
            open: function (e, ui) {
                $(this).toggleClass("active");
            },
            close: function (e, ui) {
                $(this).toggleClass("active");
            }
        });
    });
</script>
</head>
<body class="ui-widget">
    <h1>Jacqui's Flower Shop</h1>
    <h3>Color and Beauty to Your Door</h3>
    <p>We are pleased to announce that we are starting a home delivery service for
    your flower needs. We will deliver within a
    <span id="address" class="toolTip">20 mile radius</span> of the store for free and
    <span id="maxPrice" class="toolTip">$1/mile thereafter.</span></p>
</body>
</html>

```

在这个例子中，我通过在提示信息关联元素上切换active类（利用this变量）来处理open和close事件。如图23-16所示，当提示信息显示出来时，关联元素就会突出显示。

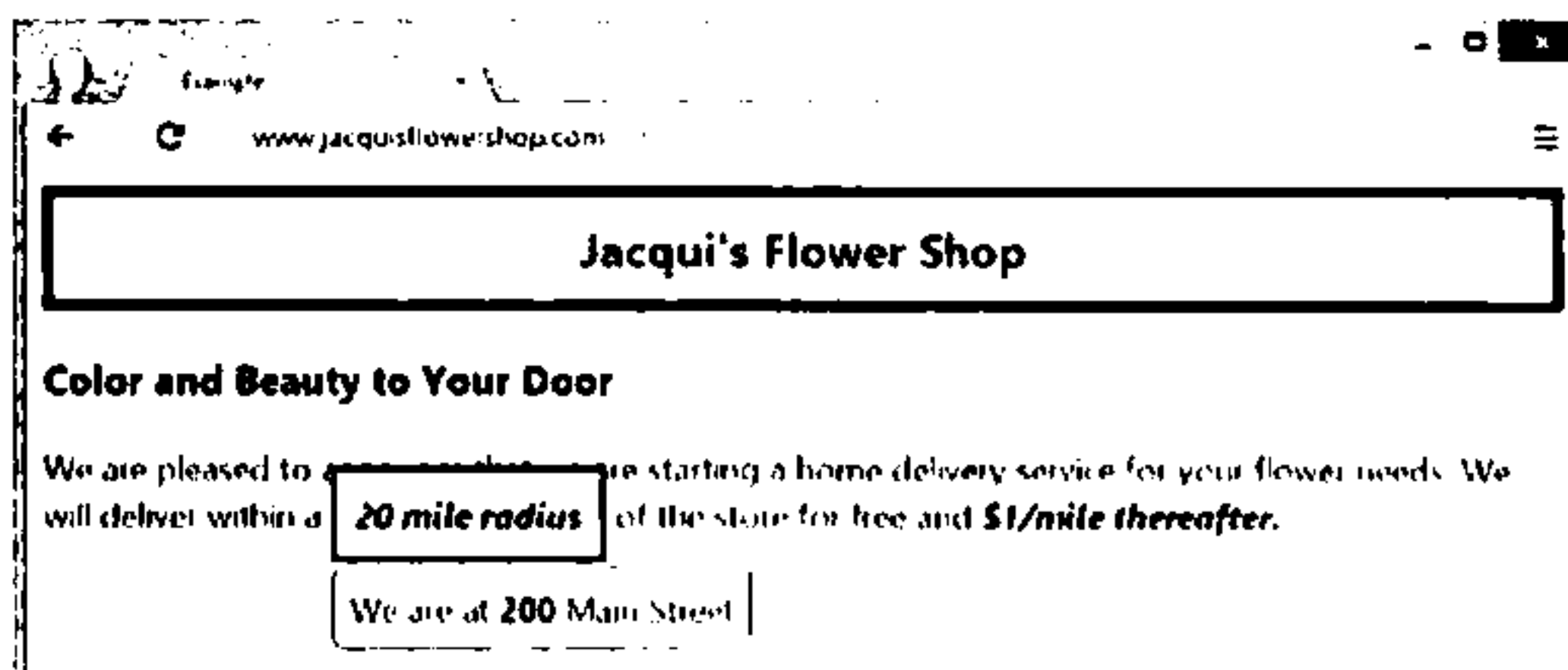


图23-16 处理open事件和close事件

## 23.3 小结

在本章中，我向大家展示了菜单组件和提示说明组件的工作原理。与介绍jQuery UI其他组件的章节类似，我们重点介绍了这些组件的选项、方法和事件。在下一章，我将向大家介绍第一个jQuery UI交互组件。

作为我们在第18章~23章学过的jQuery UI组件的补充，jQuery UI还有一些交互组件。它们是一些较底层的功能模块，可用于增强Web应用程序的界面。本章的主题是拖放（拖曳组件和接收组件），我们可以使用这一功能让页面元素支持拖放。

交互组件与其他组件具有同样的基本结构。它们也都有选项、方法和事件。我会使用同样的模式讲解交互组件，不过这里先讲一讲交互组件的特别之处。

交互效果很难通过屏幕截图展示。顾名思义，交互组件依赖于交互。我会尽力展示交互背后的本质，不过实话实说，要真正弄懂交互，你还是要要在浏览器里亲身体验一下。本章所有的例子（以及其他每章的例子）都可以在Apress网站的源代码/下载区免费下载。表24-1列出了本章概要。

表24-1 本章概要

问 题	解决方法	代码清单
如何应用拖曳行为	使用draggable方法	1
如何设置允许拖曳的方向	使用axis（轴）选项	2
如何限制拖曳的范围	使用containment（容器）选项	3
如何限制为单元格拖曳	使用grid（单元格）选项	4
如何限制延迟一段时间或拖曳一定距离后才开始拖曳	使用delay（延迟）和distance（距离）选项	5
如何响应被拖曳的元素	使用start、drag和stop事件	6
如何设置接收行为	使用droppable方法	7
如何在一个元素被拖曳时让接收元素高亮	使用activate和deactivate事件	8
被拖曳的元素重叠在接收元素之上时如何响应	使用over和out事件	9
如何指定哪些元素可以接收拖曳元素	使用accept选项	10
当拖曳开始或与接收元素重叠时如何自动设置接收元素的样式	使用activeClass和hoverClass	11
修改触发over事件所需的遮盖范围临界值	使用tolerance选项	12
如何分组拖曳元素和接收元素	使用scope选项	13
如何在拖放过程及拖放之后让拖曳元素保持不动	使用helper选项	14、15
如何在droppable事件中处理辅助元素	使用ui.helper属性	16
如何设置以使拖曳元素能够“吸附”到其他元素的边框	使用snap、snapMode和snapTolerance	17



## 24.1 创建拖曳元素

对于调用了draggable方法的元素，我们可以在浏览器窗口内移（拖）动它们。起初，文档中的这些元素像往常一样，处于正常的位置，然而当我们在这些元素上按下鼠标按钮并移动鼠标时，它们的位置就会变化。代码清单24-1展示了一个使用拖放组件的简单例子。

代码清单24-1 使用draggable方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    #draggable {font-size: x-large; border: thin solid black;
      width: 5em; text-align: center}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#draggable").draggable();
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="draggable">
    Drag me
  </div>
</body>
</html>
```

在这个例子中，我选择了一个div元素并调用了draggable方法，结果便是该元素支持拖放。如图24-1所示，这个元素起初位于正常位置，但可以被拖曳到浏览器窗口的任意位置。注意一点：文档中的其他元素，也就是h1，并没有受到draggable方法的影响。

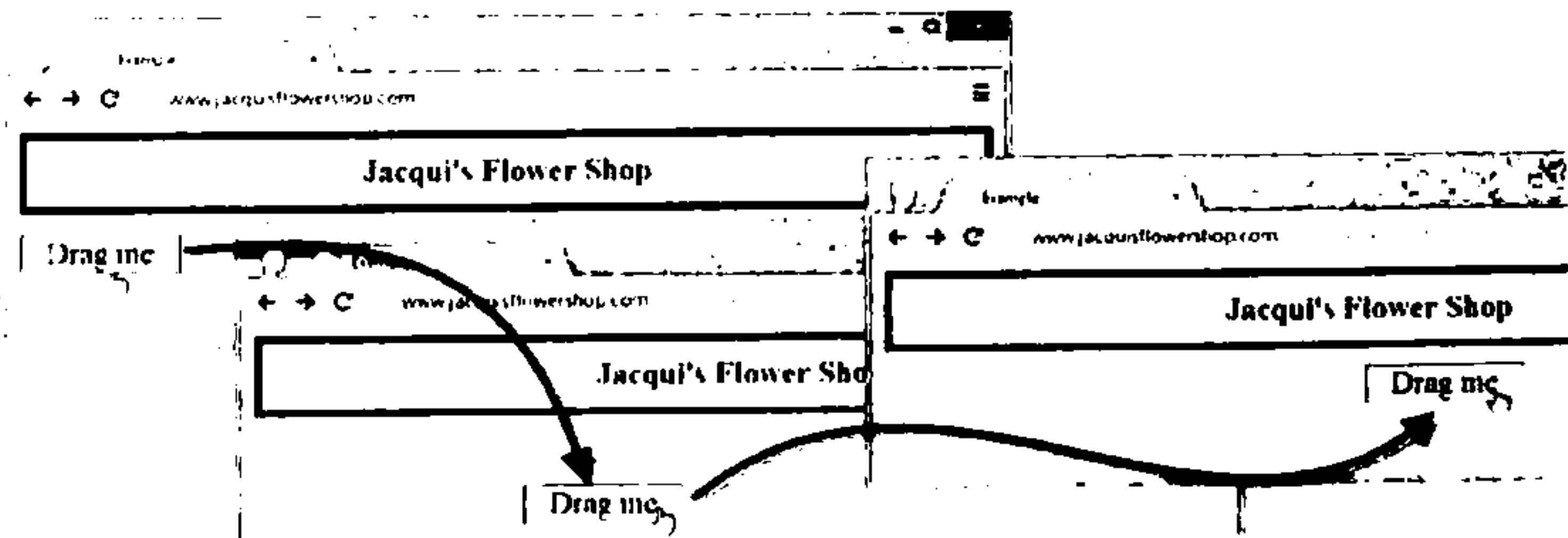


图24-1 在浏览器窗口内拖放一个元素

**提示** 能够支持拖曳本身已经很有用，而与droppable方法（在本章后面我会讲解）联用功能更强大

拖曳功能使用了非常智能的HTML和CSS实现。也就是说，这一功能在几乎所有浏览器中都能工作，不过拖曳行为仅限于浏览器中，支持拖曳的元素无法与原生操作系统的拖放功能一起工作。

**提示** HTML5确实包含了拖放功能，该功能通常使用了原生拖曳方案。我在《HTML5权威指南》一书中详细讲解了HTML5方案的细节，并给出了示例。那本书同样由Apress出版社出版。如果你使用jQuery UI的拖放方案，我建议你禁用HTML5的这一功能以避免干扰。为此，你只需要在页面的body元素中设置draggable属性为false。

### 24.1.1 配置拖曳组件

配置拖曳组件的方法有很多。表24-2列出了其中最重要的一些选项，这些选项的功能将在以下各节逐个演示。

表24-2 拖曳选项

设 置	描 述
axis	限定只允许在指定方向上拖曳。默认值为false，表示不加限制，我们可以设置为x（X轴）和y（Y轴）
Containment	限制元素的拖曳范围。参考表24-3可以详细了解支持范围。默认值为false，表示没有限制
delay	指定在元素移动之前用户必须拖曳的时间（防止误拖）。默认值为0，表示不加延迟
distance	指定在元素移动之前用户必须拖曳的距离（与原始位置相比）。默认值为1px
grid	强制拖曳元素以单元格的形式拖动。默认值为false，表示不使用单元格

**提示** 在24.3节，我讲解了一些能改变拖曳元素和接收元素关系的附加选项。

#### 1. 限制拖放方向

有好几种方式可以限制元素拖曳的方式。第一种是使用axis选项，设置拖曳方向为X轴或Y轴。代码清单24-2展示了一个例子。

代码清单24-2 使用axis选项限制拖曳行为

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    div.dragElement {font-size: large; border: thin solid black;
      width: 5em; text-align: center; background-color: lightgray; margin: 4px }
  </style>
  <script type="text/javascript">
```

```

$(document).ready(function() {

    $(".dragElement").draggable({
        axis: "x"
    }).filter("#dragV").draggable("option", "axis", "y");

});
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <div id="dragV" class="dragElement">
        Drag Vertically
    </div>
    <div id="dragH" class="dragElement">
        Drag Horizontally
    </div>
</body>
</html>

```

在本例中，我定义了两个div元素，用jQuery选中它们，然后调用draggable方法。这里先使用选项对象指定两个div元素都只能在X轴上拖动，然后使用filter方法选中dragV元素，修改选项为只能在Y轴拖放，这避免了jQuery又一次搜索整个文档的行为。结果第一个div元素只能垂直拖动，而第二个元素只能横向拖动。图24-2演示了这种效果。

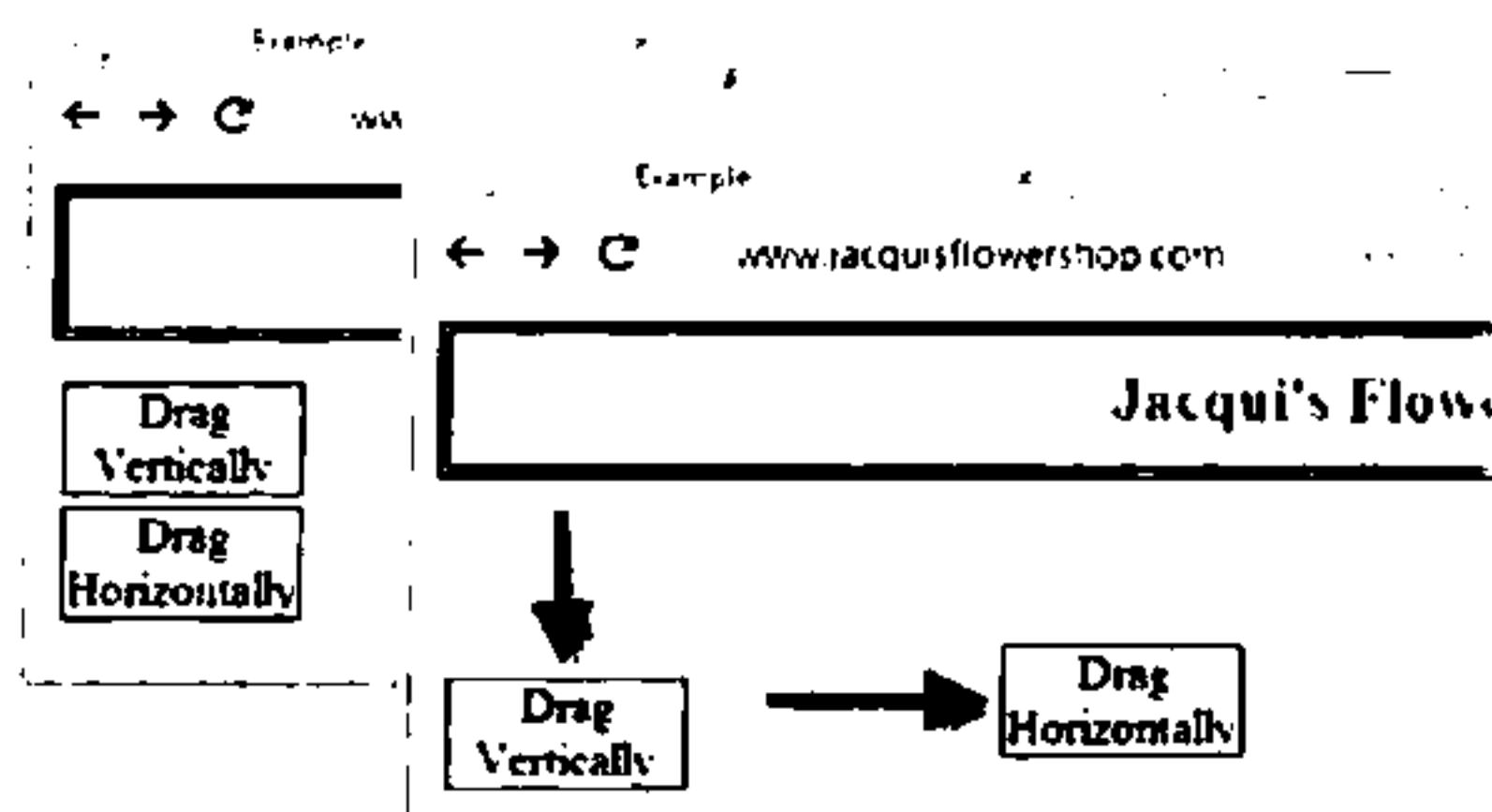


图24-2 限制元素的拖曳方向

## 2. 限制拖曳区域

我们可以限制元素拖曳的屏幕范围。这一点可以通过使用containment选项做到。如表24-3所示，这一选项支持多种格式的数值值。

表24-3 containment参数值格式

值	描述
Selector	当我们指定一个选择器字符串，被拖曳元素的拖放范围就是匹配该选择器的第一个元素
HTMLElement	被拖曳元素的拖曳范围仅限于指定元素
string	可选值有parent、document和window
数值数组	以[x1, y1, x2, y2]这样的格式指定一个数值数组，限制拖曳的范围

代码清单24-3展示了containment选项的用法。

## 代码清单24-3 使用containment选项

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    div.dragElement {font-size: large; border: thin solid black; padding: 4px;
      width: 5em; text-align: center; background-color: lightgray; margin: 4px }
    #container { border: medium double black; width: 400px; height: 150px}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $(".dragElement").draggable({
        containment: "parent"
      }).filter("#dragH").draggable("option", "axis", "x");
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="container">
    <div id="dragH" class="dragElement">
      Drag Horizontally
    </div>
    <div class="dragElement">
      Drag within Parent
    </div>
  </div>
</body>
</html>

```

在本例中，我限制两个div元素只能在父元素的范围内拖曳。那是一个固定尺寸的div元素。我还设置其中的可拖曳元素dragH只能在父元素的范围内垂直拖放。图24-3演示了这种效果。

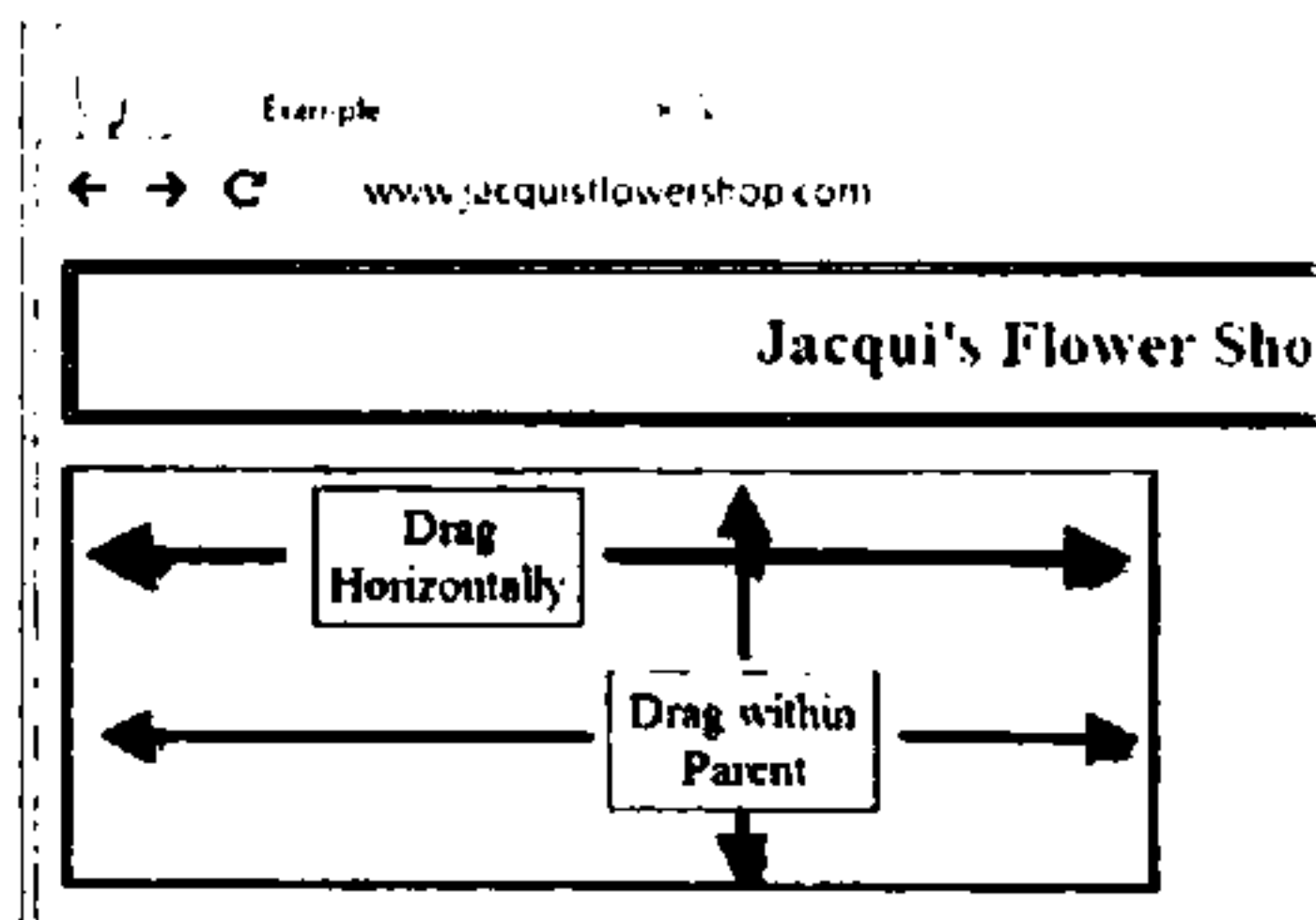


图24-3 限制只能在父元素范围内拖曳

### 3. 限制单元格方式拖动

grid选项用来使拖曳元素在拖放时自动对齐单元格。这项设置的值是一个两元素数组，这两个元素是两个数字，分别表示以px为单位的单元格宽度和高度。代码清单24-4展示了grid选项的用法。

代码清单24-4 使用grid选项

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    #draggable {font-size: large; border: thin solid black; padding: 4px;
      width: 100px; text-align: center; background-color: lightgray; margin: 4px;
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#draggable").draggable({
        grid: [100, 50]
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="draggable">
    Drag Me
  </div>
</body>
</html>
```

在这个例子中，我指定单元格宽100 px且高50 px。如图24-4所示，当我们拖曳元素时，它会自动从一个单元格（尽管看不见）跳到下一个单元格。

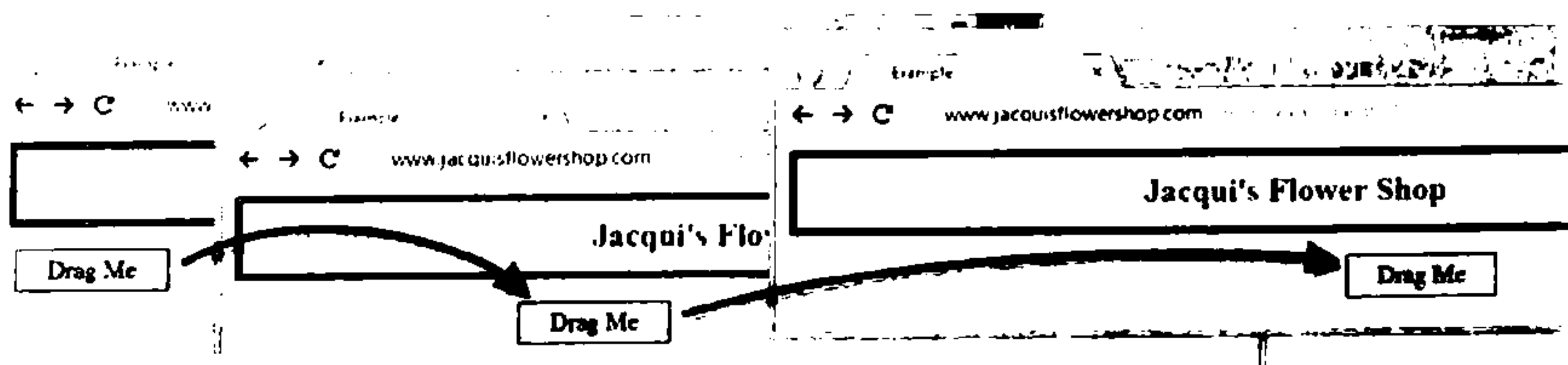


图24-4 以单元格方式拖曳元素

尽管我们很难使用屏幕截图表现出单元格自动对齐的拖曳效果，但这是一个特别能体现交互组件利于好的用户体验的例子。

**提示** 只要把一个方向的拖曳步长设置为1, 你就可以允许在那个方向上自由移动。举例来说, 把参数设置为[100, 1]就可以强制在X轴上以自动对齐100 px宽单元格的方式拖曳, 而在Y轴上却可以自由拖动。

#### 4. 延时拖动

有两个选项可以设置延时拖曳。我们可以使用delay选项, 要求用户必须拖住元素达到指定的时间, 然后元素才会移动位置。我们也可以使用distance选项, 要求用户必须拖住元素离开一定的距离, 然后元素才会跟着鼠标移动。代码清单24-5展示了这两个选项的用法。

代码清单24-5 使用delay和distance选项

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    #time, #distance {font-size: large; border: thin solid black; padding: 4px;
      width: 100px; text-align: center; background-color: lightgray; margin: 4px; }
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#time").draggable({
        delay: 1000
      });

      $("#distance").draggable({
        distance: 150
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="time">Time Delay</div>
  <div id="distance">Distance</div>
</body>
</html>
```

在上例中有两个可拖曳的div元素, 其中一个我设置了delay参数, 另一个则设置了distance参数。使用delay选项时, 用户必须持续拖住元素达到指定的时间, 然后元素才会移动。上面的例子中设置的delay参数是1000 ms (1 s)。在此期间不必一直移动鼠标, 但是在元素开始移动之前, 用户必须按住鼠标不放。在满足指定的延迟时间之后, 被拖曳的元素就会在满足前面提到过的单元格、范围和轴参数的前提下, 移动到相应的鼠标位置。

distance选项有着类似的效果, 只是用户必须移动鼠标指针离开起始点指定的距离 (单位px, 任意方向)。当条件满足时, 被拖曳的元素就会立刻移动到鼠标位置。

**提示** 如果在同一个元素上同时设置了这两个条件，那么只有两个条件都满足之后元素才会移动位置。也就是说，用户必须按下鼠标指定的时间并移动指定的距离。

## 24.1.2 使用draggable方法

拖曳组件一共只有一个方法，就是我们刚才用过的draggable方法，组件并未定义专用的拖曳方法。表23-4描述了draggable方法的一些典型用法。

表23-4 draggable方法

调用方式	描 述
draggable("destroy")	删除拖曳功能
draggable("disable")	禁用拖曳功能
draggable("enable")	允许拖曳功能
draggable("option")	修改一个或多个选项

## 24.1.3 使用拖曳事件

拖曳组件支持一套简单的事件，在元素被拖曳时通知用户。表24-5列出了这些事件。

表24-5 拖曳事件

事 件 名	描 述
create	为元素添加拖曳功能时触发
start	拖曳开始时触发
drag	拖曳期间鼠标移动触发
stop	拖曳停止时触发

我们可以随时响应这些组件事件。代码清单24-6演示了如何处理start和stop事件。

### 代码清单24-6 使用拖曳组件的start和stop事件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    #draggable {font-size: large; border: thin solid black; padding: 4px;
      width: 100px; text-align: center; background-color: lightgray; margin: 4px; }
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#draggable").draggable({
```

```

        start: function() {
            $("#draggable").text("Dragging...")
        },
        stop: function() {
            $("#draggable").text("Drag Me")
        }
    });
});
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <div id="draggable">
        Drag Me
    </div>
</body>
</html>

```

在这个例子中，我使用start和stop事件修改拖曳元素的内容。这得益于拖曳组件（和其他所有jQuery UI组件）仅使用HTML和CSS的实现方式：我们可以使用jQuery修改被拖曳的元素，即使该元素正在被拖曳。图24-5展示了这个例子的显示效果。

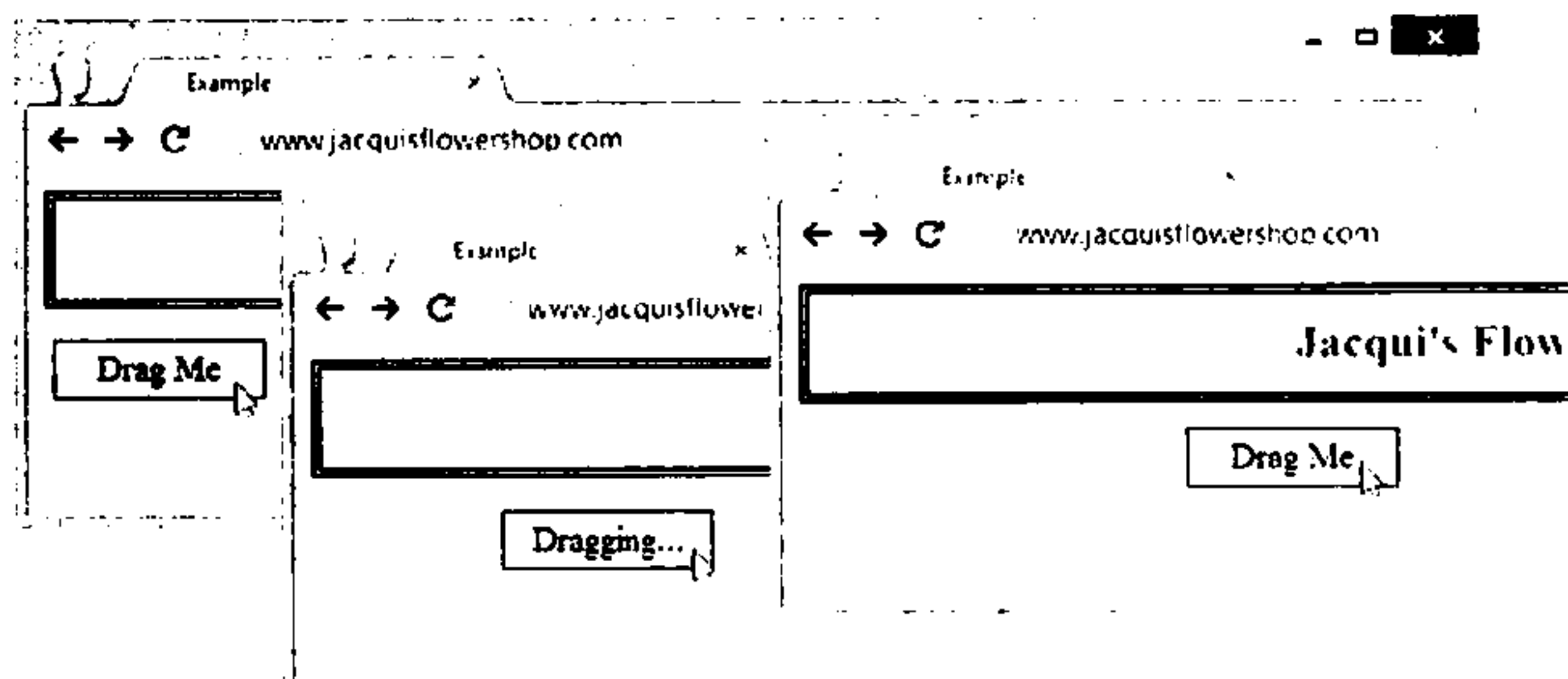


图24-5 在元素被拖曳期间使用拖曳事件修改元素

## 24.2 使用接收组件

拖曳功能要结合接收功能才能真正发挥功用。我们可以使用droppable方法创建一个接收元素（droppable element），然后若要实现任何功能，我们需要为交互组件定义的事件提供处理函数。表24-6列出了这些事件。

表24-6 接收事件

事 件	描 述
create	在设置接收功能时触发
activate	当用户开始拖曳元素时触发



(续)

事 件	描 述
deactivate	当用户停止拖曳元素时触发
over	当拖曳元素叠加到接收元素上方, 但还没有松开鼠标按键时触发
out	当拖曳元素移出(叠加→不叠加)接收元素时触发
drop	当用户将被拖曳的元素叠加到接收元素上并释放鼠标按键时触发

如代码清单24-7所示, 我们仅仅使用drop事件就能生成一个简单的接收元素。

代码清单24-7 实现基本的接收功能

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    #draggable, #droppable {font-size: large; border: thin solid black; padding: 4px;
      width: 100px; text-align: center; background-color: lightgray; margin: 4px;}
    #droppable {padding: 20px; position: absolute; left: 5px; bottom: 5px}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#draggable").draggable();

      $("#droppable").droppable({
        drop: function() {
          $("#draggable").text("Dropped")
        }
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="droppable">
    Drop Here
  </div>
  <div id="draggable">
    Drag Me
  </div>
</body>
</html>
```

在该例中我增加了一个内容为Drop Here的div元素。我使用jQuery选中该元素并调用droppable方法, 在参数对象中定义有drop事件的处理函数。一旦drop事件发生, 我就使用text方法修改拖曳元素文本。你可能觉得这个例子很无趣, 但它的确实现了拖放交互, 为讲解拖放原理奠定了适当的基础。图24-6展示了该例子各步骤的截图。

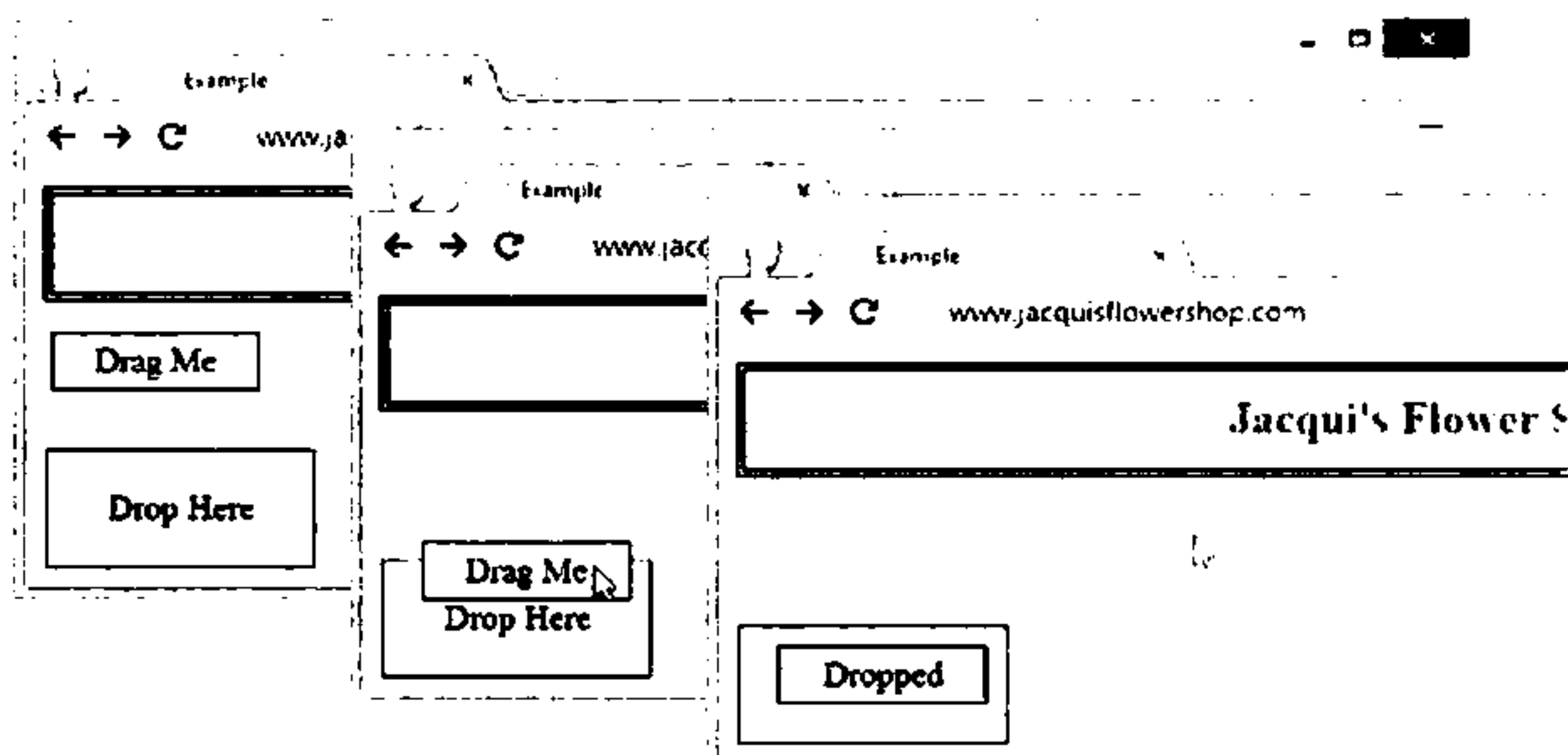


图24-6 简单拖放

这是一个相当简单的例子。我把一个元素拖到一个接收元素上，然后松手。被拖曳的元素仍然呆在我放手的地方，而它的文本已经被drop事件处理函数改变了。在下面几节中，我们将一起使用接收组件的其他事件来改进用户体验。

### 24.2.1 高亮接收元素

当用户开始拖曳操作时，我们可以使用activate和deactivate事件高亮接收元素。通常这都是一个好主意，因为会给用户发出一个明确信号：这个高亮的元素是拖放交互的一部分。代码清单24-8给出了一个这样的例子。

代码清单24-8 响应activate和deactivate事件

```
...
<script type="text/javascript">
  $(document).ready(function() {
    $("#draggable").draggable();

    $("#droppable").droppable({
      drop: function() {
        $("#draggable").text("Dropped");
      },
      activate: function() {
        $("#droppable").css({
          border: "medium double green",
          backgroundColor: "lightGreen"
        });
      },
      deactivate: function() {
        $("#droppable").css("border", "").css("background-color", "");
      }
    });
  });
</script>
...
```

当用户开始拖曳元素时，会触发接收元素的activate事件，activate事件的处理函数随之使用css方法改变放置元素的边框和背景色，造成接收元素高亮的显示效果，提示用户拖曳元素与接收元素之间具有某种关系。当用户松开鼠标按钮时，我使用deactivate事件删除接收元素的高亮样式，让接收元素恢复到拖曳之前。（这个事件在拖曳停止时触发，并不关心用户是否已经将元素拖曳到接收元素上方。）图24-7展示的就是这种效果。

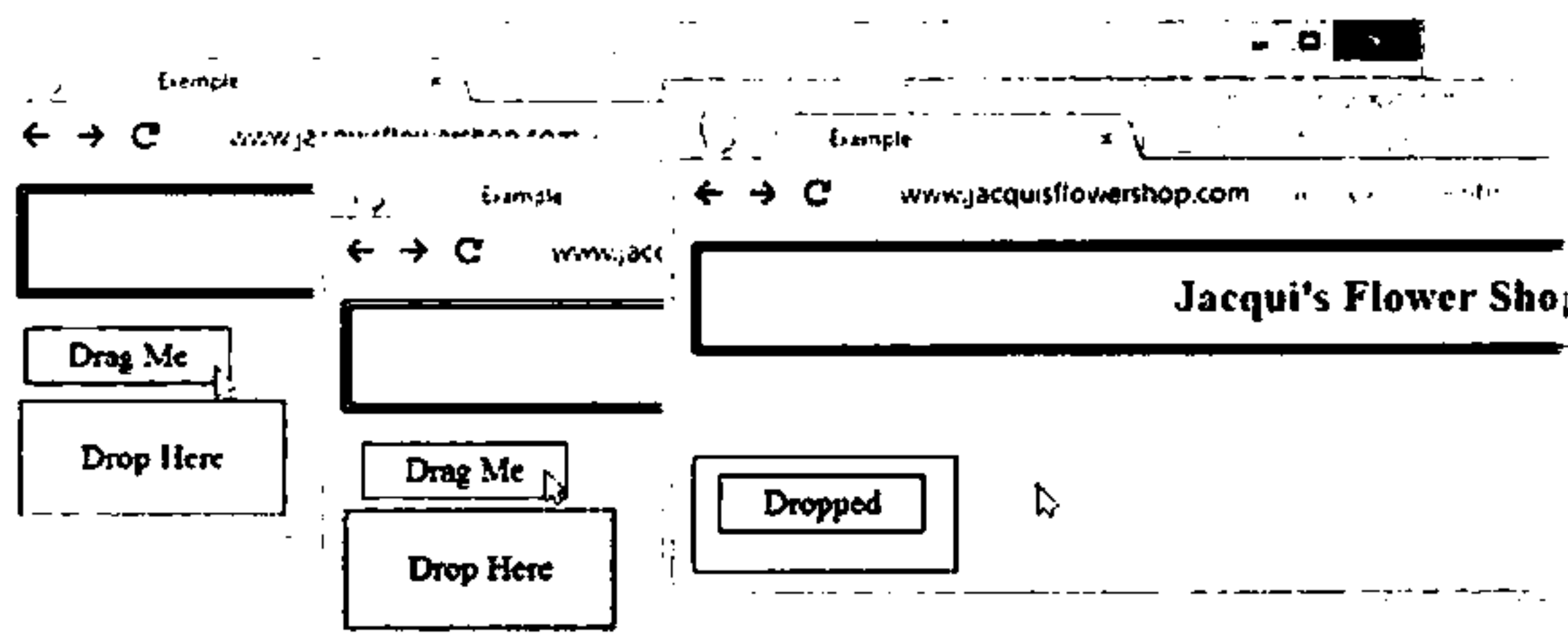


图24-7 使用activate和deactivate事件

### 24.2.2 处理遮盖元素

通过处理over和out事件，我们能够进一步改进拖放技术。当被拖曳元素的一半覆盖接收元素的任意部分时，over事件发生。当拖曳元素不再遮盖接收元素时，out事件发生。代码清单24-9演示了响应这些事件的方法。

代码清单24-9 使用over和out事件

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#draggable").draggable();

        $("#droppable").droppable({
            drop: function() {
                $("#draggable").text("Dropped");
            },
            over: function() {
                $("#droppable").css({
                    border: "medium double green",
                    backgroundColor: "lightGreen"
                });
            },
            out: function() {
                $("#droppable").css("border", "").css("background-color", "");
            }
        });
    });
</script>
...
```

在这个例子中，我使用了上一个例子中的事件处理函数，只不过这次将它们分别关联到over和out事件。如图24-8所示，当至少50%的拖曳元素遮盖住接收元素时，接收元素就会高亮。

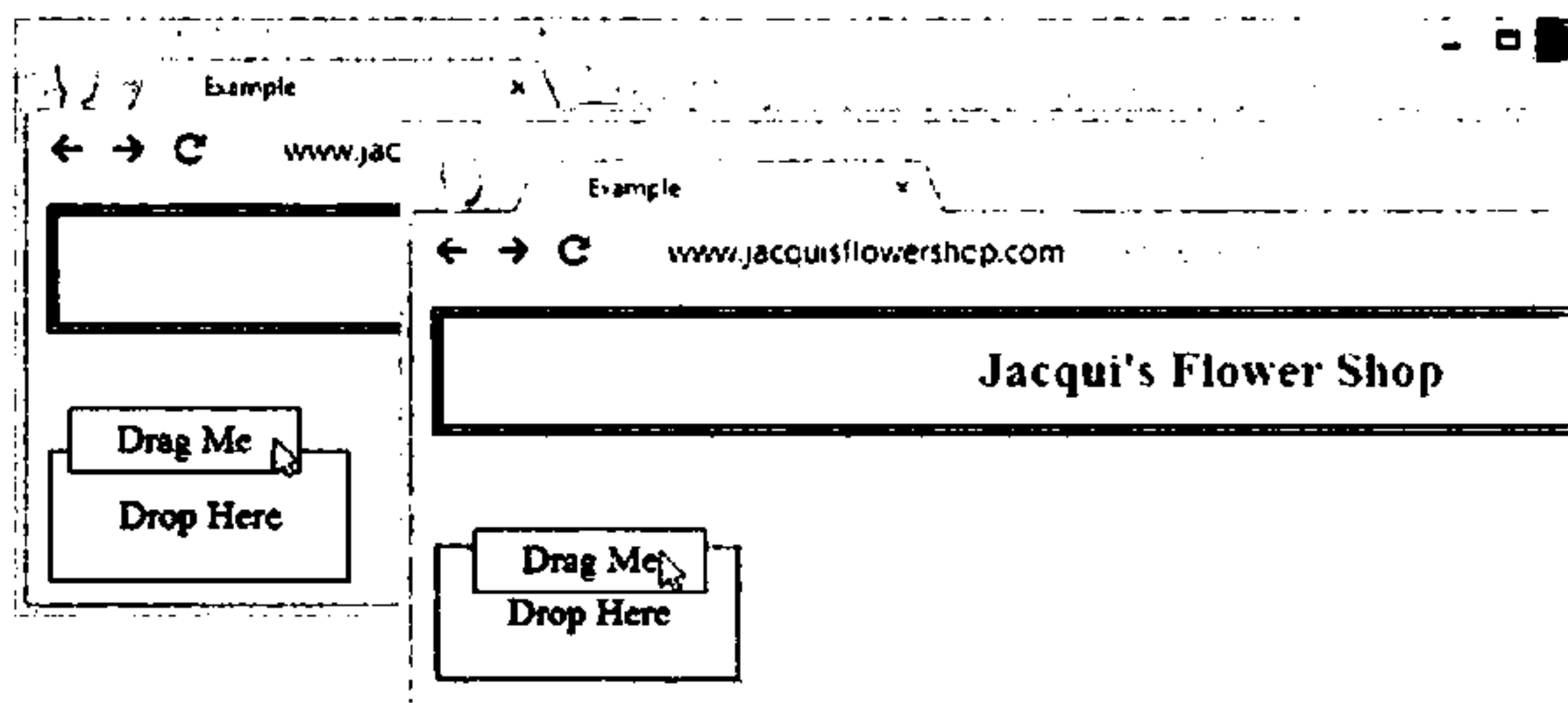


图24-8 响应over和out事件

**提示** 这里提到的50%限制，又称为临界值。我们可以配置接收元素使用不同的临界值，稍后我会在24.2.3节中的“修改遮盖临界值”部分演示这一功能。

### 24.2.3 配置接收组件

由于这些事件对于接收组件非常重要，在本书的这一部分，我决定打破常规。当然，接收组件确实有一些与组件行为息息相关的设置选项（参见表24-7）。

表24-7 接收组件选项

设 置	描 述
disabled	若设置该选项为true，就会禁用组件。默认值是false
accept	指定可被接收组件接收的拖曳元素，默认值是*，表示任何元素都行
activeClass	一个类的名字，activate事件发生时会把这个类添加到接收元素，deactivate事件发生则移除这个类
hoverClass	一个类的名字，over事件发生时会把这个类添加到接收元素，out事件发生则移除这个类
tolerance	触发over事件所需的覆盖比例值

**提示** 在24.3节，我会讲解一些可改变拖曳元素与接收元素之间关系的附加选项。

#### 1. 指定哪些拖曳元素可被接受

通过设置accept选项，我们可以按照需要指定有资格被接收元素接收的元素。我们给accept选项指定一个选择器，这个选择器匹配的拖曳元素就是可接收的元素。代码清单24-10给出了一个例子。

#### 代码清单24-10 指定可被接收的拖曳元素

```
<!DOCTYPE html>
<html>
```

```

<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    .draggable, #droppable {font-size: large; border: thin solid black; padding: 4px;
      width: 100px; text-align: center; background-color: lightgray; margin: 4px;}
    #droppable {padding: 20px; position: absolute; left: 5px; bottom: 5px}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $(".draggable").draggable();

      $("#droppable").droppable({
        drop: function(event, ui) {
          ui.draggable.text("Dropped");
        },
        activate: function() {
          $("#droppable").css({
            border: "medium double green",
            backgroundColor: "lightGreen"
          });
        },
        deactivate : function() {
          $("#droppable").css("border", "").css("background-color", "");
        },
        accept: "#drag1"
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="droppable">
    Drop Here
  </div>
  <div id="drag1" class="draggable">
    Drag 1
  </div>
  <div id="drag2" class="draggable">
    Drag 2
  </div>
</body>
</html>

```

在这个例子中有两个拖曳元素，它们的ID分别是drag1和drag2。我还生成了一个接收元素，并指定只接收drag1元素。拖动drag1元素时，我们会看到与上一个例子完全相同的效果。activate、deactivate、over和out事件分别在适当的时机触发。然而，拖动drag2元素时，虽然我仍然能四处拖动这个元素，但却无法将它置于接收元素之上。图24-9演示了这个效果。

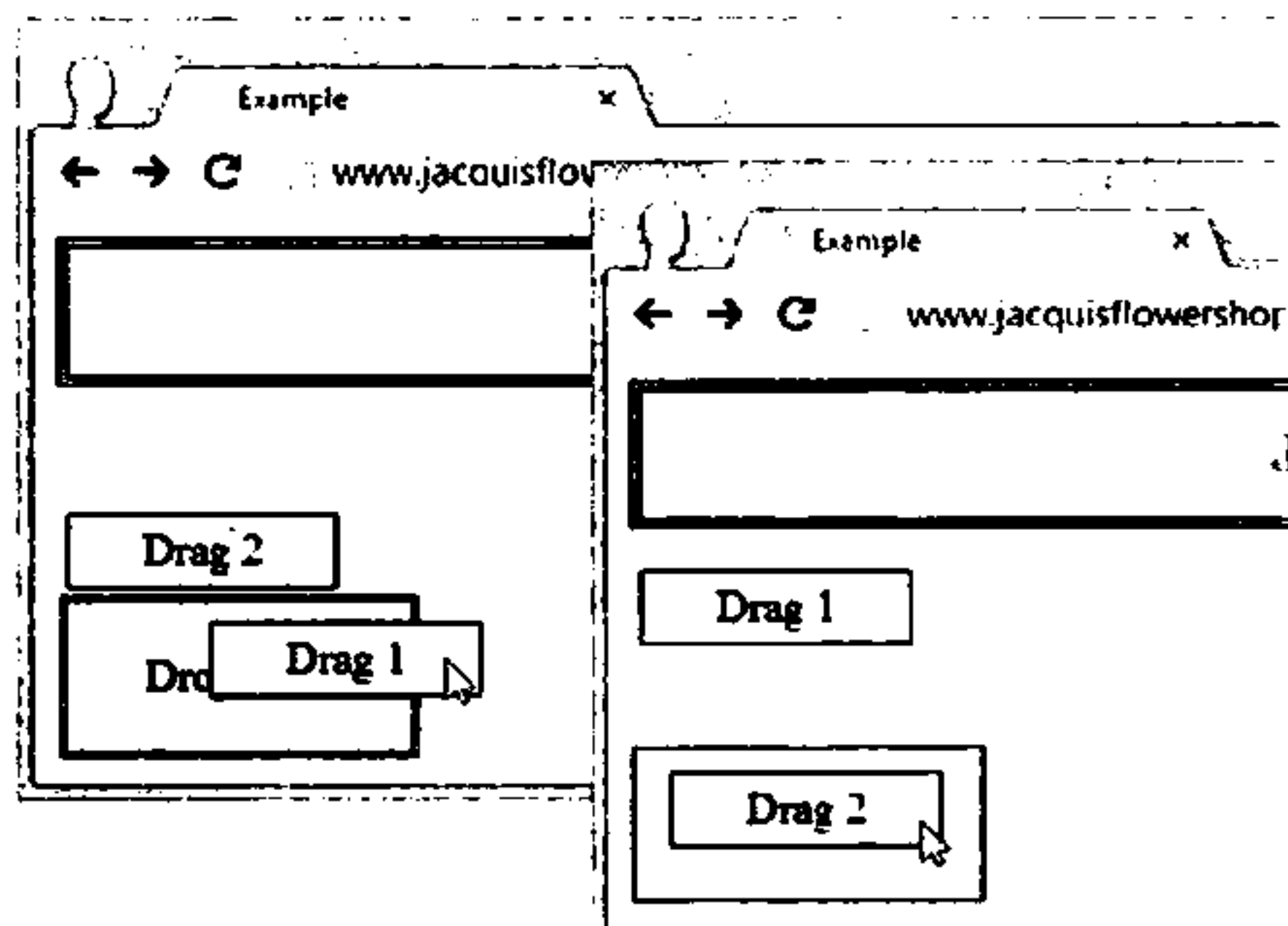


图24-9 使用accept选项

注意我换了一种方法选择被拖曳的元素，以便正确调用text方法。当页面上只有一个拖曳元素时，我们可以像下面这样直接使用id属性选择元素：

```
...
$("#draggable").text("Dropped");
...
```

然而，在本例中有多个拖曳元素，不适合使用id属性。由于我们总是修改同一元素的文本，并不关心放下的元素是哪个。因此我们改用ui对象，它是jQuery UI提供给事件处理函数的额外参数。ui对象有一个draggable属性，它是一个包含着用户正在拖曳或者已经释放的元素的jQuery对象。我们可以像下面这样利用这个对象：

```
...
ui.draggable.text("Dropped");
...
```

## 2. 使用类实现接收元素高亮

我们也可以不使用activate、deactivate、over和out事件，而是使用activeClass和hoverClass选项修改接收元素的外观（参见代码清单24-11）。

### 代码清单24-11 使用activeClass 和 hoverClass 选项

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    .draggable, #droppable {font-size: large; border: thin solid black; padding: 4px;
      width: 100px; text-align: center; background-color: lightgray; margin: 4px;}
    #droppable {padding: 20px; position: absolute; left: 5px; bottom: 5px}
    #droppable.active {border: thick solid green}
```

```

    #droppable.hover {background-color: lightgreen}
</style>
<script type="text/javascript">
    $(document).ready(function() {
        $(".draggable").draggable();

        $("#droppable").droppable({
            drop: function(event, ui) {
                ui.draggable.text("Dropped");
            },
            activeClass: "active",
            hoverClass: "hover"
        });
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <div id="droppable">
        Drop Here
    </div>
    <div class="draggable">
        Drag Me
    </div>
</body>
</html>

```

上面代码清单里的粗体部分是我定义的两个新样式。这里增加了一个针对接收元素的类（**#draggable.active**），以便让新样式优先。如果需要详细了解样式规则的优先级，请参考第3章。

定义完这些样式之后，我把它们分别设置为**activeClass**和**hoverClass**选项的值。这样接收组件就能根据发生的事件为接收元素自动添加类或者删除类。图24-10展示了这个脚本的效果。

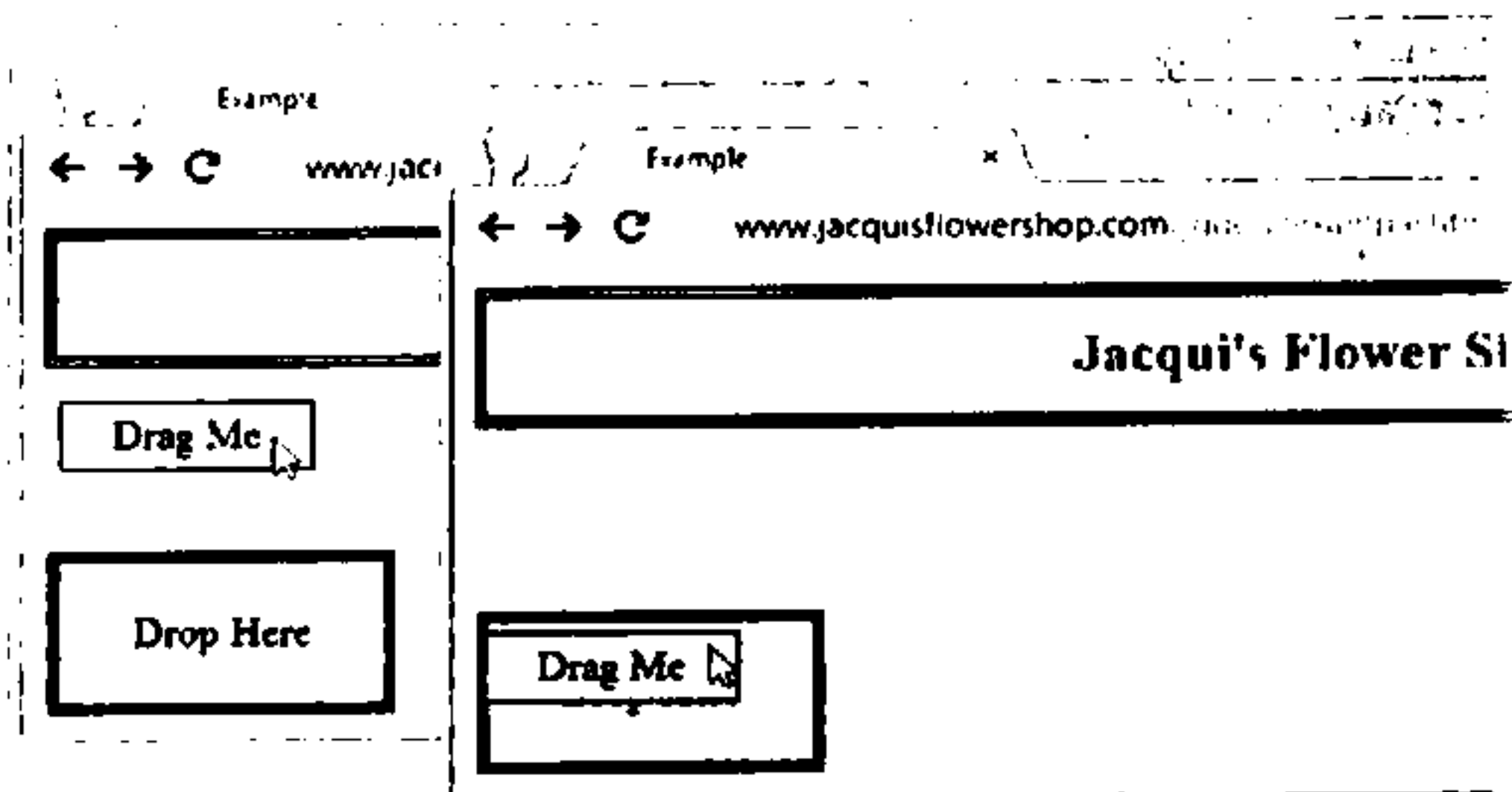


图24-10 activeClass和hoverClass选项的应用

### 3. 修改遮盖临界值

默认情况下，若超过50%的拖曳元素遮盖住接收元素就会发生over事件。我们可以使用**tolerance**选项修改这个比例，**tolerance**选项支持的设置值见表24-8。

表24-8 tolerance选项的取值

值	描 述
fit	拖曳元素必须完全遮盖接收元素
intersect	至少拖曳元素的50%遮盖住接收元素，这是默认设置
pointer	不管用户拖曳元素的什么地方，鼠标指针都必须位于接收元素之上
touch	只要拖曳元素与接收元素发生遮盖就可以

我最喜欢使用fit或者touch，因为这两个选项对于用户来说最容易理解。需要记录拖曳元素的放下位置时，我使用fit选项，而当放下的元素需要回到原始位置时，我使用touch选项（稍后演示这种用法）。代码清单24-12演示了选项值fit和touch的用法。

代码清单24-12 修改拖曳元素的tolerance选项

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    .draggable, .droppable {font-size: large; border: thin solid black; padding: 4px;
      width: 100px; text-align: center; background-color: lightgray;}
    .droppable {margin-bottom: 10px; margin-right: 5px; height: 50px; width: 120px}
    #dropContainer {position: absolute; right: 5px;}
    div span {position: relative; top: 25%}
    .droppable.active {border: thick solid green}
    .droppable.hover {background-color: lightgreen}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      $(".draggable").draggable();

      $(".div.droppable").droppable({
        drop: function(event, ui) {
          ui.draggable.text("Dropped");
        },
        activeClass: "active",
        hoverClass: "hover",
        tolerance: "fit"
      });

      $("#touchDrop").droppable("option", "tolerance", "touch");
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="dropContainer">
```



```
<div id="fitDrop" class="droppable">
  <span>Fit</span>
</div>
<div id="touchDrop" class="droppable">
  <span>Touch</span>
</div>
</div>
<div class="draggable">
  <span>Drag Me</span>
</div>
</body>
</html>
```

在这个例子里有两个接收元素。一个tolerance选项被设置为fit，一个则设置为touch。例子中只有一个拖曳元素，图24-11展示了这两个选项值对应的效果。两张屏幕截图都截自over事件触发的瞬间。请留意以下接收元素的边框，为方便判断遮盖情况，我设置了实体的边框包住接收元素。

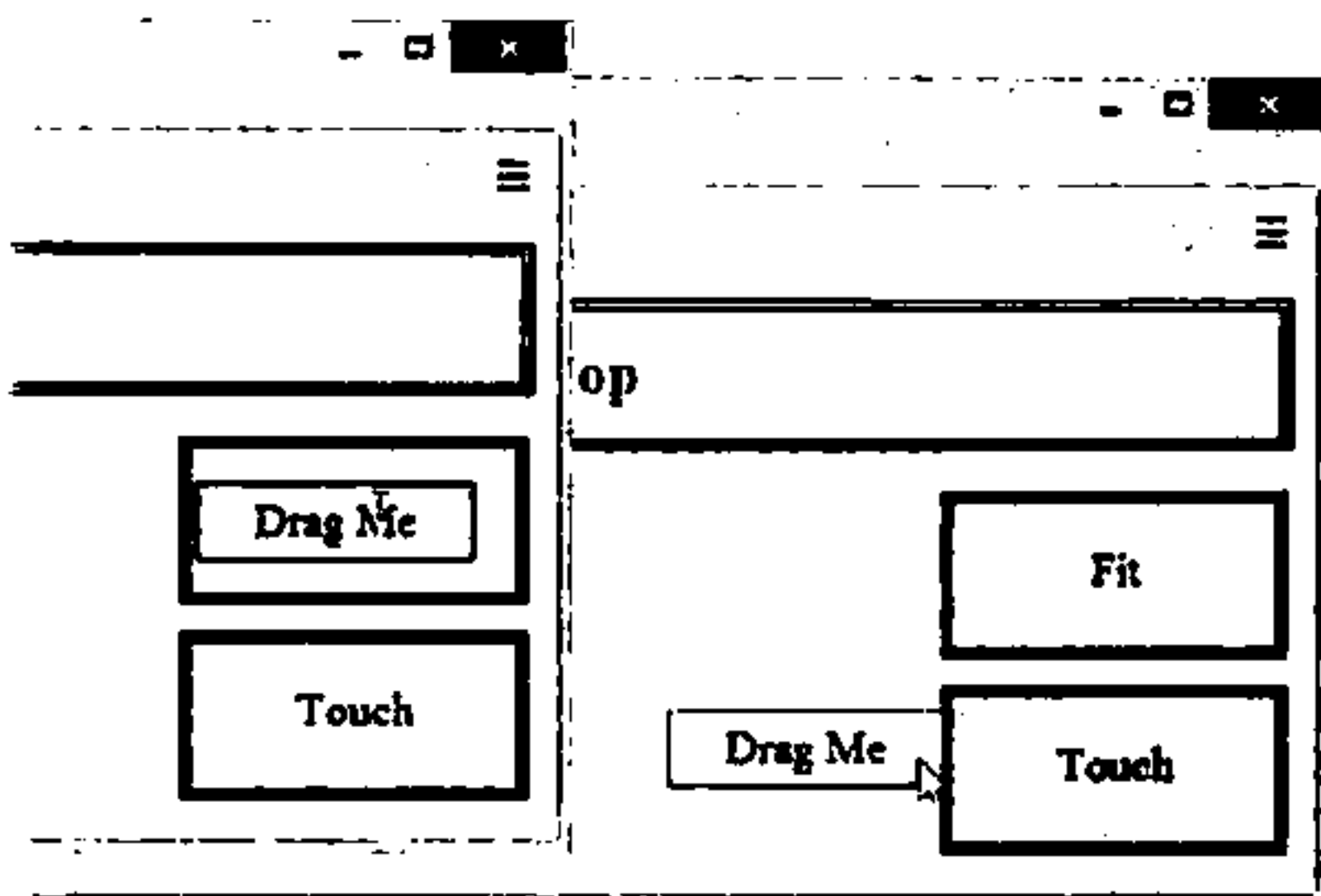


图24-11 tolerance选项fit和touch值的应用

24.2.4 使用droppable方法

我们已经看到，接收组件只有一个核心方法droppable，没有专门用于交互的方法。表24-9列出了droppable方法的用法。

表24-9 droppable方法

方 法	描 述
droppable("destroy")	删除组件功能
droppable("disable")	禁用接收组件
droppable("enable")	启用接收组件
droppable("option")	修改设置

24.3 优化拖放功能

我们可以利用一些选项微调jQuery UI拖放功能，下面就将讲解并演示这些选项的用法。

### 24.3.1 scope选项

在本章前面，我展示过使用droppable方法的accept选项过滤接收元素。对于简单项目来说，只使用选择器就足够了。不过，对于那些需要管理许多拖曳元素的复杂项目，选择器可能会过于复杂，从而容易出错。

同时在拖曳元素和接收元素中使用scope选项是一种解决之道。只有scope设置相同，拖曳元素才能激活接收元素。代码清单24-13展示了scope选项的用法。

---

**提示** 在设置接收元素时，scope选项可以与accept选项一起使用。当且仅当拖曳元素与接收元素的scope设置相同并匹配accept选项指定的选择器时，脚本才会激活接收元素。

---

代码清单24-13 使用scope设置

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    .draggable, .droppable {font-size: large; border: medium solid black;
      padding: 4px; width: 100px; text-align: center;
      background-color: lightgray; margin-bottom: 10px;}
    .droppable {margin-right: 5px; height: 50px; width: 120px}
    #dropContainer {position: absolute; right: 5px;}
    div span {position: relative; top: 25%}
    .droppable.active {border: medium solid green}
    .droppable.hover {background-color: lightgreen}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      $("#apple").draggable({
        scope: "fruit"
      });
      $("#orchid").draggable({
        scope: "flower"
      });

      $("#flowerDrop").droppable({
        activeClass: "active",
        hoverClass: "hover",
        scope: "flower"
      });

      $("#fruitDrop").droppable({
        activeClass: "active",
```

```

        hoverClass: "hover",
        scope: "fruit"
    });
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <div id="dropContainer">
        <div id="flowerDrop" class="droppable">
            <span>Flowers
        </div>
        <div id="fruitDrop" class="droppable">
            <span>Fruit</span>
        </div>
    </div>
    <div id="orchid" class="draggable">
        <span>Orchid</span>
    </div>
    <div id="apple" class="draggable">
        <span>Apple</span>
    </div>
</body>
</html>

```

在这个例子中，我创建了两个拖曳元素和两个接收元素。在设置拖曳和接收功能时，我为它们分别设置了scope值：fruit和flower。这意味着每个拖曳元素只能激活有着同样scope设置的接收元素（见图24-12）。

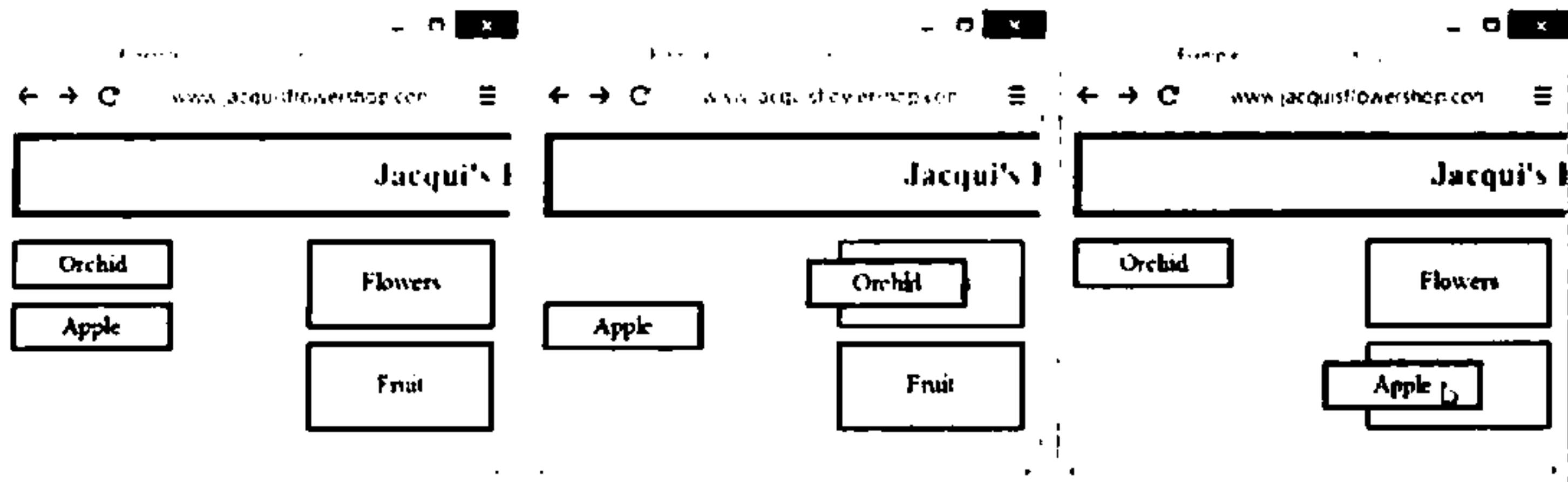


图24-12 使用scope设置分组拖曳元素和接收元素

**注意** 我是在初始化draggable和droppable方法时定义的scope设置，而不是使用option参数设置。这是因为截至我写本书的时候，jQuery UI中有一个bug尚未修复：在交互组件初始化之后设置的scope参数不起作用。

### 24.3.2 使用辅助元素

我们可以利用helper选项指定一个辅助元素，这样当拖动拖曳元素时，拖曳元素保持在原来的位

置，真正被拖动的是这个辅助元素。与前面几个例子中拖曳元素被拖离原始位置相比，这是一个完全不同的效果。代码清单24-14展示了一个使用辅助元素的例子。

代码清单24-14 使用大的拖曳元素

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    .draggable, .droppable {font-size: large; border: medium solid black;
      padding: 4px; width: 150px; text-align: center;
      background-color: lightgray; margin-bottom: 10px;}
    .droppable {margin-right: 5px; height: 50px; width: 120px}
    #dropContainer {position: absolute; right: 5px;}
    div span {position: relative; top: 25%}
    .droppable.active {border: medium solid green}
    .droppable.hover {background-color: lightgreen}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      $("div.draggable").draggable({
        helper: "clone"
      });

      $("#basket").droppable({
        activeClass: "active",
        hoverClass: "hover"
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="dropContainer">
    <div id="basket" class="droppable">
      <span>Basket</span>
    </div>
  </div>
  <div class="draggable">
    <label for="lily">Lily</label>
  </div>
</body>
</html>
```

helper参数的值 clone 告诉jQuery UI复制一份拖曳元素作为辅助元素。上面脚本的效果见图24-13。当我们放下辅助元素时，辅助元素会立刻被删除，留下拖曳元素和接收元素各自呆在它们原本的位置上。

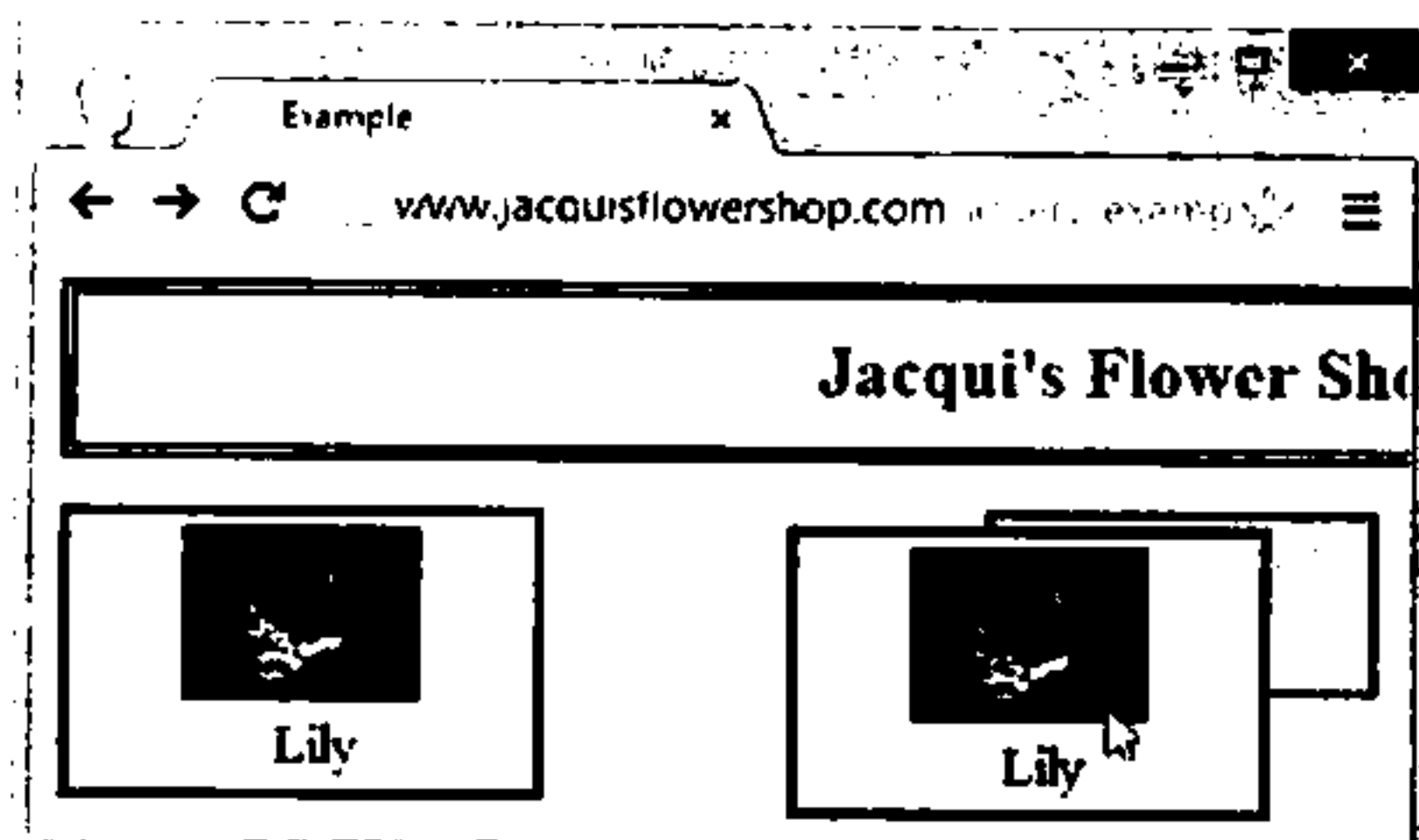


图24-13 大的拖曳元素

如图24-13所示，原始拖曳元素不动，辅助元素随着用户鼠标的移动在屏幕上移动。在上面这个例子中，被拖动的元素如此之大，以至于我们难以看到页面中被遮住的元素，包括接收元素的位置。指定一个函数作为helper选项的值能够解决此问题（参见代码清单24-15）。

#### 代码清单24-15 helper选项的用法

```
...  
<script type="text/javascript">  
    $(document).ready(function() {  
  
        $("div.draggable").draggable({  
            helper: function() {  
                return $("<img src=lily.png />")  
            }  
        });  
  
        $("#basket").droppable({  
            activeClass: "active",  
            hoverClass: "hover"  
        });  
    });  
</script>  
...
```

当用户开始拖动元素时，jQuery UI调用helper选项对应的函数，并将它的返回元素作为实际拖动的元素。在这个例子中，我使用jQuery创建了一个img元素。图24-14展示了实际的效果。

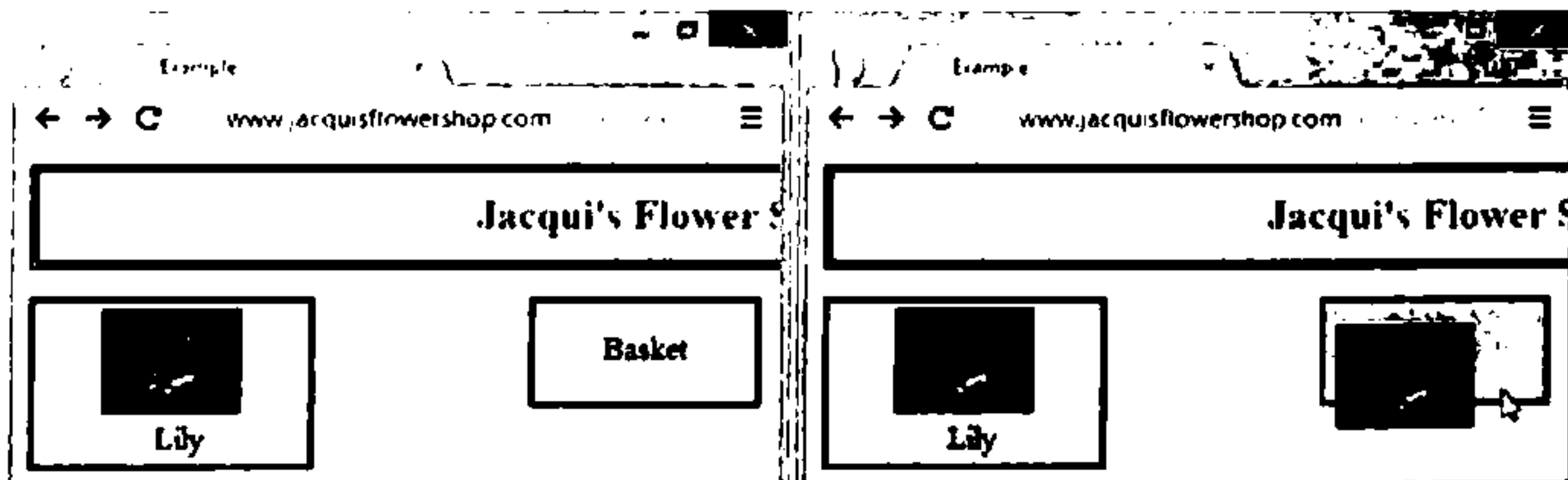


图24-14 使用辅助元素

使用这个小得多的图片作为拖曳元素的替身，在拖曳过程中就能相对容易的看清页面中余下的部分  
处理辅助元素

jQuery UI传递给droppable事件的ui参数拥有一个helper属性，我们可以利用这个属性在拖曳过程中修改辅助元素。代码清单24-16通过over和out事件演示了这个属性的用法。

代码清单24-16 使用ui.helper属性

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("div.draggable").draggable({
            helper: function() {
                return $("<img src=lily.png />")
            }
        });

        $("#basket").droppable({
            over: function(event, ui) {
                ui.helper.css("border", "thick solid green")
            },
            out: function(event, ui) {
                ui.helper.css("border", "")
            }
        });
    });
</script>
...
```

利用over和out事件，还有ui.helper属性，我们成功地实现了：当遮盖住接收元素时让辅助元素显示边框。图24-15展示了上面脚本产生的效果。

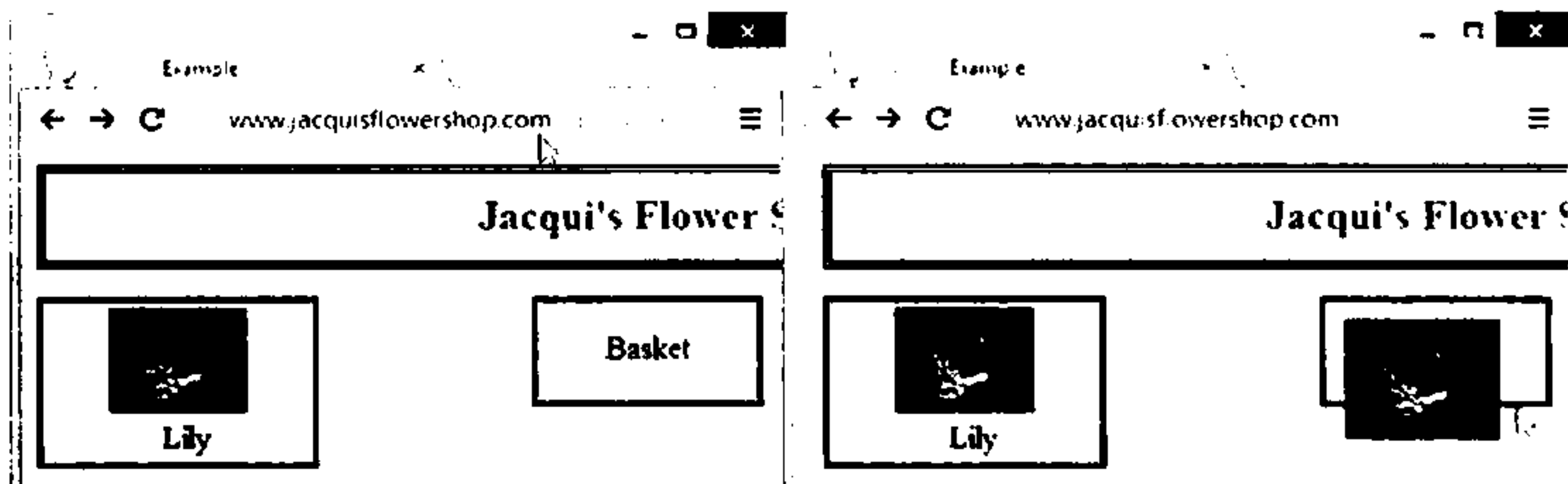


图24-15 处理辅助元素

### 24.3.3 “吸附”到元素边界

使用snap选项（一个选择器）可以让拖曳元素“吸附”到它经过的某些元素上。拖曳元素将被“吸附”到任意一个匹配snap选择器的元素的最近边框上。代码清单24-17演示了snap选项的用法。

## 代码清单24-17 使用snap选项

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    #snapper, .draggable, .droppable {font-size: large; border: medium solid black;
      padding: 4px; width: 150px; text-align: center;
      background-color: lightgray; margin-bottom: 10px;}
    .droppable {margin-right: 5px; height: 50px; width: 120px}
    #dropContainer {position: absolute; right: 5px;}
    div span {position: relative; top: 25%}
    .droppable.active {border: medium solid green}
    .droppable.hover {background-color: lightgreen}
    #snapper {position: absolute; left: 35%; border: medium solid black;
      width: 180px; height: 50px}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      $("div.draggable").draggable({
        snap: "#snapper, .droppable",
        snapMode: "both",
        snapTolerance: 50
      });
      $("#basket").droppable({
        activeClass: "active",
        hoverClass: "hover"
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="dropContainer">
    <div id="basket" class="droppable">
      <span>Basket
    </div>
  </div>
  <div id="snapper"><span>Snap Here</span></div>
  <div class="draggable">
    <span>Drag Me</span>
  </div>
</body>
</html>

```

当拖曳元素接近某个匹配元素时，它就像遇到磁铁那样被“吸引”到匹配元素的最接近的两个边界上。我们可以设置任意的“吸附”元素，吸附元素可以不是接收元素。在上例中，我添加了一个div

元素，并设置`snap`值指向它和页面中的接收元素。不过，使用屏幕截图来展示“吸附”效果实在是一个不可能完成的任务。我鼓励大家在浏览器中自己动手体验一下这个例子的“吸附”效果。

还有两个选项可用来微调“吸附”行为。第一个选项是`snapMode`，可用于指定拖曳元素被吸附到哪个边界。这个选项可选择的值有`inner`（内边框）、`outer`（外边框）和`both`（默认值，表示吸附到内边框和外边框）。

第二个选项是`sanpTolerance`，可用于指定拖曳元素与目标元素之间的吸附距离（隔多远才能吸上）。默认值是20，表示20 px。在上面这个例子中，我指定的值是50，所以即使隔得比较远也能“吸”上。正确设置这个参数非常重要：如果距离太小，用户就会感觉不到“吸附”效果；如果设置得太大，拖曳元素就很容易被意外地“吸附”到相距很远的元素上。

## 24.4 小结

本章讲解了两个非常重要并且有用的jQuery UI组件：如拖曳组件和接收组件。为了能在Web应用程序中精细控制拖放体验，我不但演示了如何配置和使用这些组件，还演示了如何响应组件事件，以及如何优化组件之间的协同工作。下一章将介绍其他jQuery UI组件。



我将在本章讲解剩余的三个jQuery UI组件：sortable、selectable和resizable。与第24章讲到的拖曳组件和接收组件相比，这些组件较少使用，也不是那么有用。本章讲到的这几个组件各有各的用途，只不过其应用场景相对较少。鉴于此，相对于其他组件来说，这些组件是最好的补充，是更传统的解决方案。表25-1列出了本章概要。

表25-1 本章概要

问 题	解决方法	代码清单
如何使用sortable组件	选中容器元素，然后调用sortable方法	1
如何得到用户使用sortable组件生成的顺序	调用toArray或serialize方法	2、3
如何设置允许把一个sortable项拖曳到另一个sortable项	使用connectWith选项	4
如何在一个拖曳元素和一个sortable项之间建立连接	在拖曳元素上使用connectToSortable选项	5
如何指定哪些元素可以排序	使用items选项	6
在一个sortable项正被拖曳时，如何设置空出位置的样式	使用placeholder选项	7
如何取消某次顺序变化	使用cancel方法	8
如何刷新一个sortable元素内的元素	使用refresh方法	9
如何在排序过程中得到实时信息	使用ui对象提供的事件处理函数	10
如何使用selectable组件	选中容器元素，然后调用selectable方法	11、12
如何阻止某个元素被选中	使用cancel方法	13
如何使用resizable组件	使用resizable方法	14
如何一次改变多个元素的大小	使用alsoResize选项	15、16
如何限制一个resizable元素的尺寸	使用maxHeight、maxWidth、minHeight和minWidth选项	17
如何设置一个resizable元素可以拖曳的边和角	使用handles选项	18

## 25.1 使用 sortable 组件

我们可以利用sortable组件实现对一组元素的拖曳排序。你只需选中包含被拖曳元素的父元素（容器元素）并调用sortable方法，这样就完成了所有设置工作，参见代码清单25-1。

## 代码清单25-1 使用sortable组件

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    div.sortable { width: 100px; background-color: lightgrey; font-size: large;
      float: left; margin: 4px; text-align: center; border: medium solid black;
      padding: 4px;}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#sortContainer").sortable();
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="sortContainer">
    <div id="item1" class="sortable">Item 1</div>
    <div id="item2" class="sortable">Item 2</div>
    <div id="item3" class="sortable">Item 3</div>
  </div>
</body>
</html>

```

在这个例子中，我添加了一些div元素，并给它们加上了sortable类。sortable组件并不使用这个类，我只是利用它设置div元素的样式。这里先选中这些div元素的父元素div#sortContainer，然后调用sortable方法，这样就设置完了sortable组件。于是我就能够拖曳这三个div元素中的任意一个到新的位置。这个例子的效果见图25-1。（不过，如果能亲自在浏览器中运行这个例子，以及本章所有的例子，就能更好地理解本章的内容。）

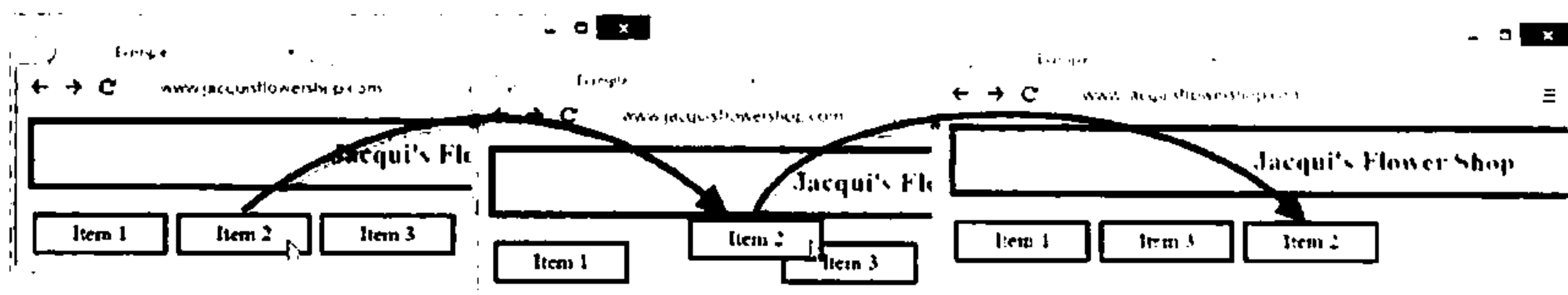


图25-1 拖曳排序

我把标签为Item 2的元素往浏览器窗口右边拖，以演示sortable组件的使用。一旦拖过标签为Item 3的元素，jQuery UI就会重排这些元素，从而更新这些元素的顺序。我仅仅把一个元素拖到了一个位置，但其实可以把任意元素一次拖到几个不同的位置。

### 25.1.1 获取排序之后的新顺序

有时，我们需要获取用户拖曳排序之后的新顺序。调用组件的toArray方法，这会返回以排序元素id值为数组内容的JavaScript数组，这样我们就得到了需要的排序信息。代码清单25-2展示的是在例子中额外使用一个按钮把排序信息输出到控制台。

代码清单25-2 得到排序元素的顺序

```
...
<script type="text/javascript">
  $(document).ready(function() {
    $("#sortContainer").sortable();

    $("<div id=buttonDiv><button>Get Order</button></div>").appendTo("body");
    $("button").button().click(function() {
      var order = $("#sortContainer").sortable("toArray");
      for (var i = 0; i < order.length; i++) {
        console.log("Position: " + i + " ID: " + order[i]);
      }
    });
  });
</script>
...
```

上面例子的效果见图25-2。按下这个按钮时，脚本就会调用toArray方法并遍历结果数组order，把数组内容输出到控制台。

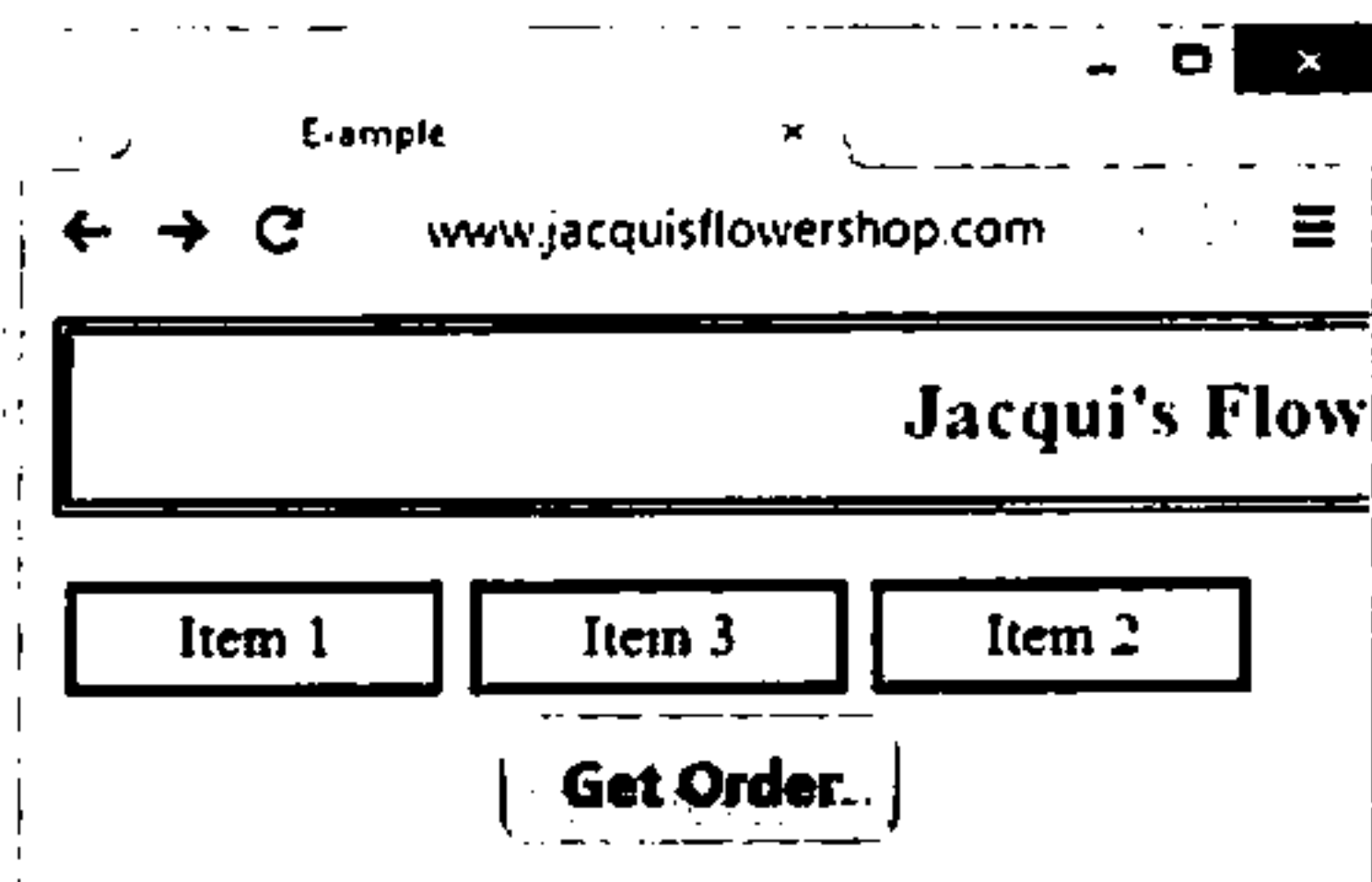


图25-2 添加按钮以输出排序结果

对于图25-2中的顺序来说，按下按钮，你会看到以下输出：

```
Position: 0 ID: item2
Position: 1 ID: item3
Position: 2 ID: item1
```

你也可以使用serialize方法生成一个更适合在表单中使用的字符串，代码清单25-3演示了这种用法。

## 代码清单25-3 使用serialize方法

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    div.sortable { width: 100px; background-color: lightgrey; font-size: large;
      float: left; margin: 4px; text-align: center; border: medium solid black;
      padding: 4px;}
    #buttonDiv {clear: both}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#sortContainer").sortable();
      $("<div id=buttonDiv><button>Get Order</button></div>").appendTo("body");
      $("button").button().click(function() {
        var formstring = $("#sortContainer").sortable("serialize");
        console.log(formstring);
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="sortContainer">
    <div id="item_1" class="sortable">Item 1</div>
    <div id="item_2" class="sortable">Item 2</div>
    <div id="item_3" class="sortable">Item 3</div>
  </div>
</body>
</html>

```

注意，在这个例子里我修改了排序元素的id值。serialize方法在生成字符串时会查找符合<key>\_<index>格式的元素id。同样是图25-2中的顺序，这一次产生的输出是：

---

```
item[]=2&item[]=3&item[]=1
```

---

## 25.1.2 配置sortable组件

显然，sortable组件相当依赖我们在第24章中讲到的拖曳组件。这也意味着前面讲过的拖曳组件选项（如axis和tolerance）也能用于配置sortable组件。在这一章，我不再重复那些选项，而是把只适用于sortable组件并且最有用的一些选项列于表25-2之中。接下来的各节将详细讲述这些设置选项。

表25-2 Sortable设置选项

设置选项	描 述
connectWith	指定要连接的目标sortable组件选择器（从而支持把项从一个组件拖到另一个组件），默认值是false，表示无连接
dropOnEmpty	若为false，表示不允许把元素项拖到空的（一个子元素也没有的）目标sortable组件。默认值是true，表示允许
items	是一个选择器，指定可以排序的项。默认值是>*, 表示排序组件的任意后代元素
placeholder	当把一个元素拖到新位置时，为自动生成的占位元素指定的类

### 1. 连接Sortable组件

我最喜欢这样一个sortable组件特性：可以把两个sortable组件连接在一起，从而允许把一个sortable组件的项拖到另一个sortable组件。只需使用connectWith选项指定一个匹配目标sortable组件的选择器，我们就可实现这种效果。如代码清单25-4所示，在两个sortable组件上都使用connectWith选项，就能达到支持双向拖曳的效果。

代码清单25-4 把sortable组件连接在一起

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></bidirectionalscript>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    div.sortable { width: 100px; background-color: lightgrey; font-size: large;
      margin: 4px; text-align: center; border: medium solid black; padding: 4px;}
    #fruitContainer {position: absolute; right:50px}
    #flowerContainer {position: absolute; left:50px}
    div.flower {background-color: lightgreen}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#fruitContainer").sortable({
        connectWith: "#flowerContainer"
      });
      $("#flowerContainer").sortable({
        connectWith: "#fruitContainer"
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="fruitContainer" class="sortContainer">
    <div id="fruit_1" class="sortable fruit">Apple</div>
    <div id="fruit_2" class="sortable fruit">Orange</div>
    <div id="fruit_3" class="sortable fruit">Banana</div>
    <div id="fruit_4" class="sortable fruit">Pear</div>
  </div>
  <div id="flowerContainer" class="sortContainer">
    <div id="flower_1" class="sortable flower">Flower</div>
    <div id="flower_2" class="sortable flower">Flower</div>
    <div id="flower_3" class="sortable flower">Flower</div>
    <div id="flower_4" class="sortable flower">Flower</div>
  </div>
</body>
</html>
```

```

</div>
<div id="flowerContainer" class="sortableContainer">
  <div id="flower_1" class="sortable flower">Aster</div>
  <div id="flower_2" class="sortable flower">Peony</div>
  <div id="flower_3" class="sortable flower">Lily</div>
  <div id="flower_4" class="sortable flower">Orchid</div>
</div>
</body>
</html>

```

在这个例子中，我创建了两组sortable元素，并分别在它们的父元素（容器元素）上调用了sortable方法。我使用connectWith选项让这两组元素互相连接，得到图25-3所示的结果。

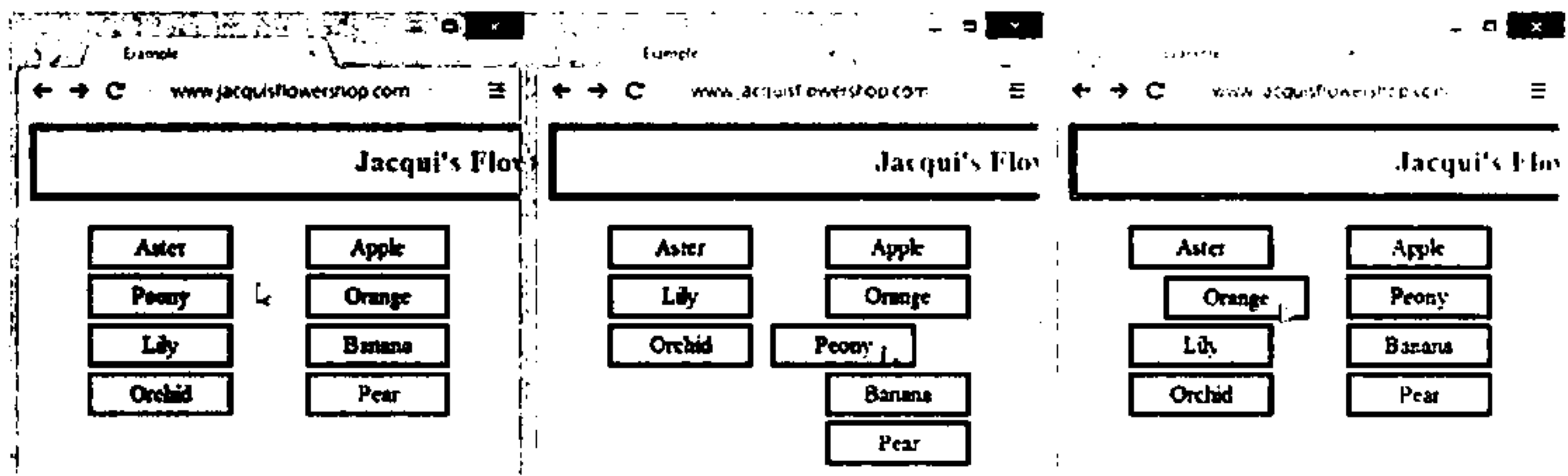


图25-3 在互相连接的两组sortable元素之间拖曳元素

## 2. 连接拖曳元素和sortable元素

我们也可以把一个支持拖曳的元素连接到一组sortable元素。在拖曳元素上调用draggable方法时，使用connectToSortable选项要连接的sortable元素（一个选择器）就可以达到这一目的。代码清单25-5演示了这种用法。

### 代码清单25-5 连接拖曳元素和sortable元素

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    div.sortable { width: 100px; background-color: lightgrey; font-size: large;
      margin: 4px; text-align: center; border: medium solid black; padding: 4px;}
    #fruitContainer {position: absolute; right:50px}
    #flowerContainer {position: absolute; left:50px}
    div.flower {background-color: lightgreen}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#fruit_1").draggable({
        connectToSortable: "#flowerContainer",

```

```

        helper: "clone"
    });
    $("#flowerContainer").sortable();
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <div id="fruitContainer" class="sortContainer">
        <div id="fruit_1" class="sortable fruit">Apple</div>
    </div>
    <div id="flowerContainer" class="sortContainer">
        <div id="flower_1" class="sortable flower">Aster</div>
        <div id="flower_2" class="sortable flower">Peony</div>
        <div id="flower_3" class="sortable flower">Lily</div>
        <div id="flower_4" class="sortable flower">Orchid</div>
    </div>
</body>
</html>

```

在这个例子里，我把水果项减少到只剩一个，让它支持拖曳并连接到支持排序的花卉列表。结果如何？如图25-4所示，支持拖曳的水果能够添加到sortable列表中。当draggable方法的helper选项被置为clone时，connectToSortable选项工作得最好。如果helper选项的值并非clone，connectToSortable选项仍能工作，但是会报告一个错误。

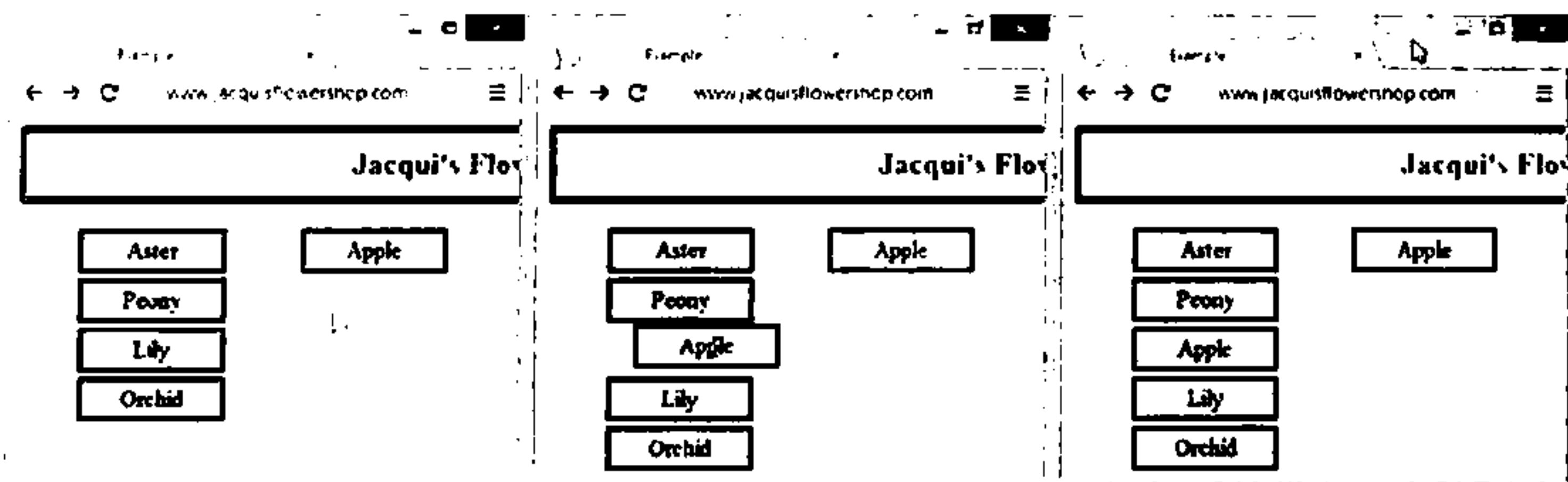


图25-4 连接拖曳元素和sortable元素

### 3. 只允许特定项支持排序

我们可以使用items选项指定容器元素内的哪些元素支持排序，这个选项的值是一个匹配允许排序元素的选择器。所有不匹配该选择器的元素都不支持重新排序。具体的例子见代码清单25-6。

#### 代码清单25-6 仅让特定的元素支持排序

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>

```



```

<style type="text/css">
  div.sortable { width: 100px; background-color: lightgrey; font-size: large;
    margin: 4px; text-align: center; border: medium solid black; padding: 4px;}
  #fruitContainer {position: absolute; right:50px}
  #flowerContainer {position: absolute; left:50px}
</style>
<script type="text/javascript">
  $(document).ready(function() {
    $(".flower:even").css("background-color", "lightgreen");
    $("#flowerContainer").sortable({
      items: ".flower:even"
    });
  });
</script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="flowerContainer" class="sortContainer">
    <div id="flower_1" class="sortable flower">Aster</div>
    <div id="flower_2" class="sortable flower">Peony</div>
    <div id="flower_3" class="sortable flower">Lily</div>
    <div id="flower_4" class="sortable flower">Orchid</div>
  </div>
</body>
</html>

```

在这个例子里，我使用items选项指定容器元素内只有偶数元素才可以排序。在图25-5中，Aster和Lily元素可以拖动排序，而Peony和Orchid元素则无法拖动，始终呆在原地。

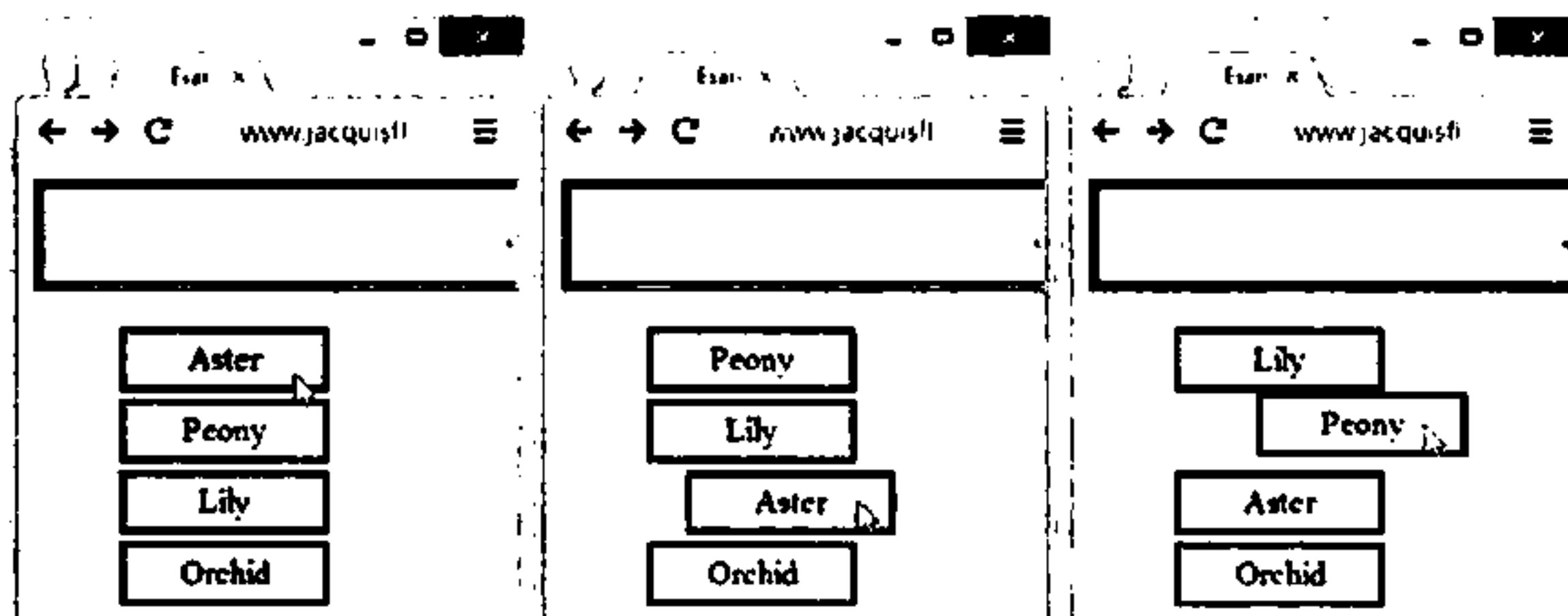


图25-5 指定允许拖动排序的元素

使用items选项时一不小心就会造成混乱，在图25-5中的最后一幅图上我已经指出了这一点。不匹配items选择器的元素不允许拖放，没错。然而，其他元素的变化会使本来不匹配items选择器的元素变得匹配，从而变得支持拖放排序。在图25-5中，我把Aster元素拖到了新的位置，从而导致Peony元素的位置发生了变化，而Peony元素位置的变化导致它匹配items选择器，从而变得支持排序。

#### 4. 设置空位样式

当我们把一个元素拖到新的位置，它原来所在的位置就空出来了。我们可以使用placeholder选项为这个占位元素添加一个类，从而有效地强调空余位置可以放置元素。代码清单25-7演示了placeholder选项的用法。



## 代码清单25-7 设置placeholder选项

```

<!DOCTYPE html>
<html>
<head>
  <title>Example
  <script src="jquery-1.7.js" type="text/javascript"></script>
  <script src="jquery-ui-1.8.16.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.8.16.custom.css"/>
  <style type="text/css">
    div.sortable {width: 100px; background-color: lightgrey; font-size: large;
      margin: 4px; text-align: center; border: medium solid black; padding: 4px;}
    #flowerContainer {position: absolute; left:25%}
    .emptySpace {border: medium dotted red; height: 25px; margin: 4px}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#flowerContainer").sortable({
        placeholder: "emptySpace"
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="flowerContainer" class="sortContainer">
    <div id="flower_1" class="sortable">Aster</div>
    <div id="flower_2" class="sortable">Peony</div>
    <div id="flower_3" class="sortable">Lily</div>
    <div id="flower_4" class="sortable">Orchid</div>
  </div>
</body>
</html>

```

在这个例子中，我定义了一个名为emptySpace的CSS类，它定义了height和margin属性，还定义了红色虚线的边框线。我把这个类用于placeholder选项，如图25-6所示，当拖曳一个元素进行排序时，空出来的位置被自动应用了emptySpace类。

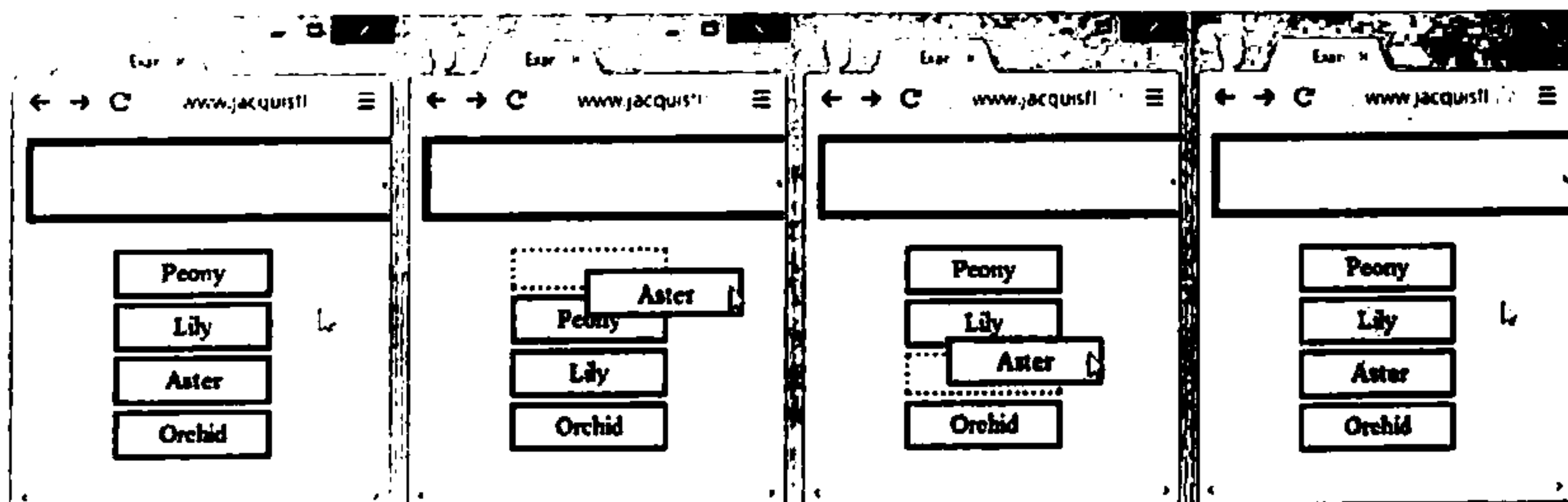


图25-6 设置placeholder选项

### 25.1.3 使用sortable方法

sortable组件定义有所有标准的jQuery UI方法，而且还定义了一些专用方法。表25-3列出了这些方法。

表25-3 sortable方法

方 法	描 述
sortable("destroy")	从一个元素上移除sortable组件
sortable("disable")	禁用sortable组件功能
sortable("enable")	启用sortable组件功能
sortable("option")	修改sortable组件的一个或多个选项
sortable("toArray")	返回一个以排序元素id值为数组内容的数组（25.1.1节有一个该方法的应用示例）
sortable("refresh")	刷新sortable组件
sortable("cancel")	取消排序操作

#### 1. 取消排序操作

我们可以使用cancel方法取消特定的排序。由于它阻止了用户期望的行为，这种做法比较罕见。如果我们确实需要取消某个排序，一定要告诉用户取消排序的原因。代码清单25-8是一个结合update事件使用cancel方法的例子。当用户拖曳一个元素，并在生成新顺序之后释放鼠标，update事件发生不要着急，我会在25.1.4节详细讲解sortable事件。

代码清单25-8 使用cancel方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    div.sortable {width: 100px; background-color: lightgrey; font-size: large;
      margin: 4px; text-align: center; border: medium solid black; padding: 4px;}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#error").dialog({autoOpen: false, modal: true});

      $("#flowerContainer").sortable({
        update: function() {
          var sortedItems = $("#flowerContainer").sortable("toArray");
          if (sortedItems[0] != "item_1") {
            $("#error").dialog("open");
            $("#flowerContainer").sortable("cancel");
          }
        }
      });
    });
  </script>
</head>
<body>
  <div id="flowerContainer">
    <div>item_1</div>
    <div>item_2</div>
    <div>item_3</div>
    <div>item_4</div>
    <div>item_5</div>
  </div>
</body>
</html>
```

```

    });
  </script>
</head>
<body>
  <div id="error">The King must be first</div>
  <h1>Jacqui's Flower Shop</h1>
  <div id="flowerContainer" class="sortable">
    <div id="item_1" class="sortable">King</div>
    <div id="item_2" class="sortable">Queen</div>
    <div id="item_3" class="sortable">Jack</div>
    <div id="item_4" class="sortable">10</div>
  </div>
</body>
</html>

```

在这个例子中，如果用户拖曳排序的结果造成King元素不再是第一个元素，我就会调用cancel方法取消这次排序。这里使用了第22章讲过的对话框组件告诉用户为什么会取消这次排序。只要不影响King元素的位置，任何其他拖曳排序都会正常进行。

## 2. 刷新sortable元素

refresh方法用来刷新sortable容器中元素的缓存。代码清单25-9演示了如何利用这一功能在容器中添加新的sortable元素。

代码清单25-9 添加新的sortable元素

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    div.sortable {width: 100px; background-color: lightgrey; font-size: large;
      margin: 4px; text-align: center; border: medium solid black; padding: 4px;}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#flowerContainer").sortable();

      var itemCount = 2;

      $("#button").click(function() {
        $("<div id=flower_" + (itemCount++) + " class=sortable>Item " +
          itemCount + "</div>").appendTo("#flowerContainer");
        $("#flowerContainer").sortable("refresh");
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <button>Add Sortable Item</button>

```

```

    <div id="flowerContainer" class="sortContainer">
      <div id="flower_1" class="sortable">Aster</div>
      <div id="flower_2" class="sortable">Peony</div>
    </div>
  </body>
</html>

```

在这个例子中，我在页面中添加了一个按钮，它负责往sortable容器中添加新的排序项并调用refresh方法，以确保新项能够排序。

### 25.1.4 sortable事件

我在第24章讲过的由拖曳组件定义的所有事件，sortable组件全部支持。表25-4列出了sortable组件定义的一些专用事件。

表25-4 sortable事件

事 件 名	描 述
change	被排序元素的位置发生变化时触发
receive	当一个元素从连接的sortable组件拖放到当前组件时触发
remove	当一个元素从当前sortable组件拖放到连接的另一sortable组件时触发
sort	排序期间每次鼠标移动均触发
update	用户完成拖曳，元素项顺序发生改变时触发

当以上事件发生时，jQuery UI还通过额外的ui对象参数提供更多信息。表25-5列出了ui对象参数的属性。

表25-5 sortable 组件ui对象的属性

属 性 名	描 述
helper	返回辅助元素
position	返回辅助元素的当前位置（一个具有top和left属性的对象）
item	返回包含着当前拖曳元素的jQuery对象
placeholder	返回包含着占位元素的jQuery对象
sender	返回包含sortable组件连接发起方元素的jQuery对象（如果连接不存在，这个属性的值就是null）

代码清单25-10展示的是在sort和change事件中ui对象的使用法。

#### 代码清单25-10 change和sort事件

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">

```

```

    div.sortable {width: 100px; background-color: lightgrey; font-size: large;
        margin: 4px; text-align: center; border: medium solid black; padding: 4px;}
    #flowerContainer {position: absolute; left: 10px}
    #info {position: absolute; right: 10px; border: medium solid black; padding: 4px}
</style>
<script type="text/javascript">
    $(document).ready(function() {
        $("#flowerContainer").sortable({
            sort: function(event, ui) {
                $("#itemId").text(ui.item.attr("id"))
            },
            change: function(event, ui) {
                $("#pos").text($("#flowerContainer *").index(ui.placeholder));
            }
        });
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <div id="flowerContainer" class="sortContainer">
        <div id="flower_1" class="sortable">Aster</div>
        <div id="flower_2" class="sortable">Peony</div>
        <div id="flower_3" class="sortable">Lily</div>
        <div id="flower_4" class="sortable">Orchid</div>
    </div>
    <div id="info" class="ui-widget">
        <div>Item ID: None</span></div>
        <div>Pos: <span id="pos">None</span></div>
    </div>
</body>
</html>

```

我使用这两个事件显示与排序操作有关的信息。在sort事件发生时，我读取ui.item属性并得到被拖曳元素的id属性。在change事件发生时，我利用ui.placeholder属性并使用index方法得到该排序元素所在的位置。

## 25.2 selectable 组件

selectable组件帮助我们在页面上选中一个或多个元素，它既支持通过拖放鼠标选择一堆元素，也支持利用鼠标单击选择具体某个元素。如代码清单25-11所示，我们通过selectable方法设置selectable组件。

代码清单25-11 设置selectable组件

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>

```

```

<link rel="stylesheet" type="text/css" href="styles.css"/>
<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<style type="text/css">
    div.flower {width: 200px; background-color: lightgrey; font-size: large;
        margin: 4px; text-align: center; border: medium solid black; padding: 4px;}
    #flowerContainer {position: absolute; left:10px}
    div.ui-selected {border: medium solid green; background-color: lightgreen}
    div.ui-selecting {border: medium solid green}
</style>
<script type="text/javascript">
    $(document).ready(function() {
        $("#flowerContainer").selectable();
    });
</script>
</head>
<body>
    h1>Jacqui's Flower Shop</h1>
    <div id="flowerContainer">
        <div id="flower_1" class="flower">Aster</div>
        <div id="flower_2" class="flower">Peony</div>
        <div id="flower_3" class="flower">Lily</div>
        <div id="flower_4" class="flower">Orchid</div>
    </div>
</body>
</html>

```

在包含着选择目标元素的容器元素上调用selectable方法，这样就完成了selectable组件的设置。在本例中，我重复利用了本章前面sortable组件用到的那一组div元素。如下面的代码所示，我选中容器元素，然后调用selectable方法：

```

...
$("#flowerContainer").selectable();
...

```

虽然已经在容器元素上调用了selectable方法，完成了组件设置工作，我还需要定义两个类（表示“选中”和“正在选”的状态），以便给用户直观的视觉反馈。下面是这两个类的定义：

```

...
div.ui-selected {border: medium solid green; background-color: lightgreen}
div.ui-selecting {border: medium solid green}
...

```

selectable组件会自动使用这些类表现元素的当前状态。当用户拖动鼠标选择元素时，ui.selecting类会自动应用到鼠标选中区域的元素上，而ui-selected类则在元素被选中之后自动应用于选中元素（不管是拖动鼠标选中，还是鼠标单击选中）。我使用了很简单的样式：绿边框和浅绿色背景。图25-7展示的是鼠标拖动选择元素的效果。

用户拖动鼠标选择元素的起点一定要在容器元素之内。在图25-7的中间那幅图上，我们能够看到选择区域的轮廓（又叫选取框），jQuery UI自动为选择区域内的元素应用了ui-selecting类。如图25-7中最后那幅图所示，当我们释放鼠标按键时，选取框所覆盖的元素会被自动应用ui-selected类。

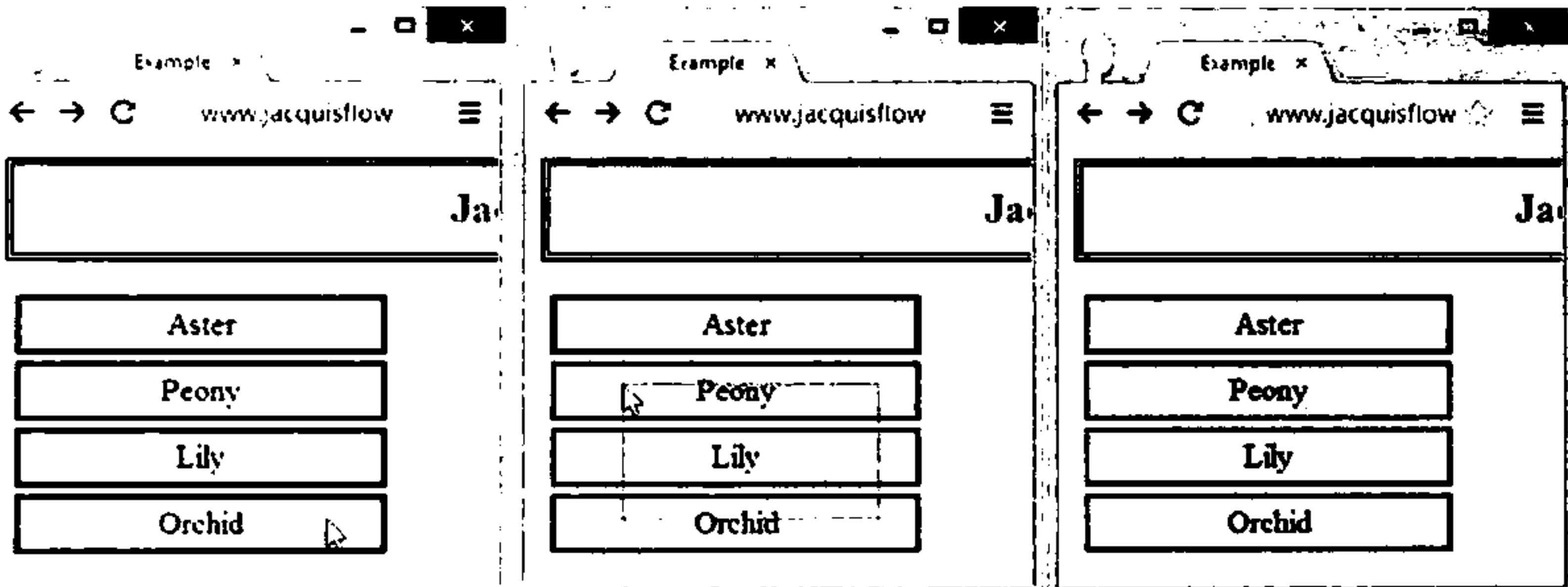


图25-7 拖动鼠标选择元素

用户也可以通过单击元素来选取（反选）元素。我们可以一次选择多个元素，如果在鼠标单击的同时按下Ctrl/Meta键，还能支持不连续的选择。如果你发现鼠标单击无法实现非连续选择，就需要代码清单25-12中的解决方案。

代码清单25-12 让selectable组件允许多重选择

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#flowerContainer")
            .bind("mousedown", function(e) {e.metaKey = true;})
            .selectable();
    });
</script>
...
```

当用户按下鼠标按键时，jQuery UI会自动在选取元素上应用ui-selecting类；当用户释放鼠标按键时，则自动在选中元素上应用ui-selected类。

### 25.2.1 配置selectable组件

表25-6列出了selectable组件的设置选项。

表25-6 selectable组件设置选项

选 项	描 述
disabled	若为true，则初始化时即禁用交互组件，默认值是false
autoRefresh	若为true，组件在每次选取操作开始时刷新每个selectable元素的尺寸和位置，默认值是true
cancel	一个选择器字符串，匹配该选择器的元素无法被选中
delay	参阅第24章拖曳组件的delay选项
distance	参阅第24章拖曳组件的distance选项
filter	一个选择器字符串，匹配容器元素内允许选中的元素，默认值是*，表示匹配所有后代元素



selectable组件的绝大多数设置选项都是可望名知义的，或者与其他组件含义相同。值得注意的是cancel选项，这个选项可用于禁止用户选中某些元素。代码清单25-13提供了一个应用cancel选项的例子

代码清单25-13 应用cancel选项

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#flowerContainer")
            .bind("mousedown", function(e) {e.metaKey = true;})
            .selectable({
                cancel: "#flower_3"
            });
    });
</script>
...
```

在这段脚本中我设置了一个选择器，禁止用户选中id为flower\_3的元素。当用户使用单击元素的方法选择元素时，这个选项工作得很好。然而，它无法阻止用户使用拖曳的方式选取元素。这个理由足够充分，因此最好不要使用这个选项。

## 25.2.2 使用selectable方法

如表25-7所示，这个组件仅定义了一个专用方法（refresh）。其他的方法都是所有小部件与组件都有定义的通用方法。

表25-7 selectable方法

方 法	描 述
selectable("destroy")	删除组件设置
selectable("disable")	禁用组件功能
selectable("enable")	启用组件功能
selectable("option")	修改设置
selectable("refresh")	刷新组件。在autoRefresh选项设置为false时，该方法可以用来替代该选项，完成刷新任务

## 25.2.3 selectable组件事件

selectable组件定义的事件见表25-8。

表25-8 selectable组件定义的事件

事 件	描 述
create	在某个元素上设置组件时触发
selected	当一个元素被选中时触发。若一次选中多个元素，则选中多少个元素就会触发多少次此事件
selecting	当用户开始选择元素时触发（按下鼠标按键，或者正在拖曳鼠标时）
unselected	当反选一个元素时触发，若一次反选多个元素，则取消多少个元素，就会触发多少次此事件
unselecting	当用户按下鼠标按键执行反选操作时触发



jQuery UI通过ui对象参数为绝大多数事件提供额外的信息。当发生selected或selecting事件时，ui对象有一个selected属性，它对应已经选中（或者将要选中）元素的HTMLElement对象。基于同样的理由，当发生unselected和unselecting事件时，ui对象具有一个unselected属性。

## 25.3 resizable 组件

resizable组件为元素添加拖曳的“把手”，以方便用户改变它的大小。有些浏览器会自动为文本输入框添加改变大小的把手，而resizable组件允许我们为页面中任意的元素添加把手。代码清单25-14展示了resizable组件的应用（通过调用resizable方法实施）。

代码清单25-14 设置resizable组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    #aster, #lily {text-align: center; width: 150px; border: thin solid black;
      padding: 5px; float: left; margin: 20px}
    #aster img, #lily img {display: block; margin: auto}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("#aster").resizable({
        alsoResize: "#aster img"
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <div id="aster" class="ui-widget">
    
    Aster
  </div>
  <div id="lily" class="ui-widget">
    
    Lilly
  </div>
</body>
</html>
```

在这个例子中，我创建了两个div元素，它们的内容都是一幅图和一些文本。在脚本中我选中其中的一个，并调用resizable方法（使用了稍后将详细讲解的alsoResize选项）。如图25-8所示，jQuery UI自动给选中元素添加了适合拖曳的把手，允许在垂直和水平方向上改变元素的大小。在图25-8中，我加大了元素的高度，并减小了元素的宽度。

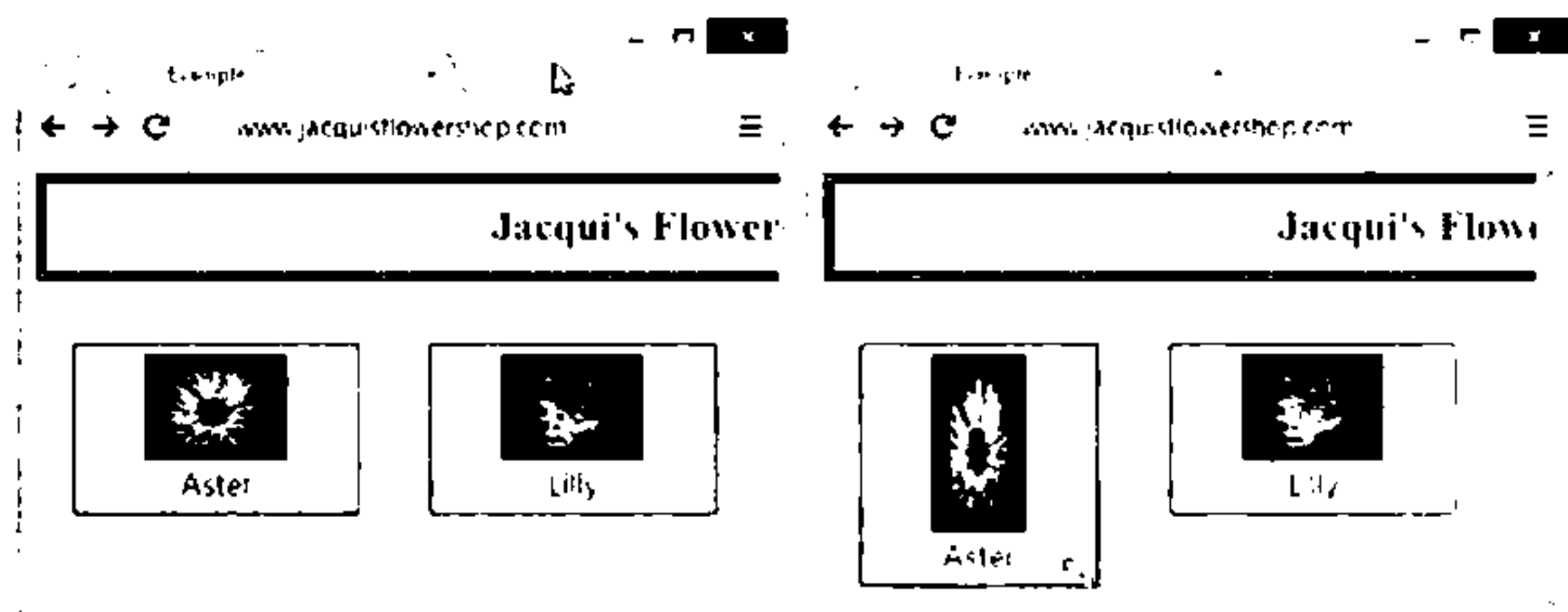


图25-8 使用拖曳把手改变resizable元素的大小

## 配置resizable组件

我们可以使用表25-9中列出的选项配置resizable组件。resizable组件依赖于第24章讲过的拖曳组件,也就是说,除了表25-9中列出的这些选项,我们也可以使用拖曳组件的选项(包括delay、distance、grid和containment)配置resizable组件。

表25-9 resizable组件选项

选 项	描 述
alsoResize	一个选择器,匹配改变元素大小时也需要同步改变大小的关联元素。默认值是false,表示没有关联元素
aspectRatio	若设置为true,元素在改变大小时保持宽高比不变,默认值是false
autoHide	若设置为true,则拖曳把手仅当鼠标悬停在resizable元素上时才显示,默认值是false
ghost	若设置为true,则会有一个透明的辅助元素帮助用户预览元素的新大小。默认值是true
handles	指定把手出现在resizable元素上的位置,本章后面会列出该选项的所有可取值
maxHeight	指定resizable元素的最大高度限制,默认值是null,表示没有限制
maxWidth	指定resizable元素的最大宽度限制,默认值是null,表示没有限制
minHeight	指定resizable元素的最小高度限制,默认值是10px
minWidth	指定resizable元素的最小宽度限制,默认值是10px

### 1. 改变关联元素的大小

在我看来,alsoResize选项是resizable组件最有用的选项。它允许我们指定一个关联元素,该元素的尺寸会随着主元素的尺寸变化而变化。我主要使用这个选项来保证父元素尺寸变化的同时,它们的内容元素尺寸也同步跟着变化,就像前面的例子中让div元素尺寸变化的同时,也让它的内容元素(img)尺寸同步变化一样。我们现在看看改变一个有内容的元素的尺寸(而不使用alsoResize选项)会发生什么。代码清单25-15演示了这一场景。

代码清单25-15 在不使用alsoResize选项情况下改变元素内容的尺寸

```
...
<script type="text/javascript">
  $(document).ready(function() {
    $("#aster").resizable();
```

```

    });
</script>
...

```

如图25-9所示，缺少alsoResize选项的情况下，只有div元素改变了大小，内容元素仍呆在原处。

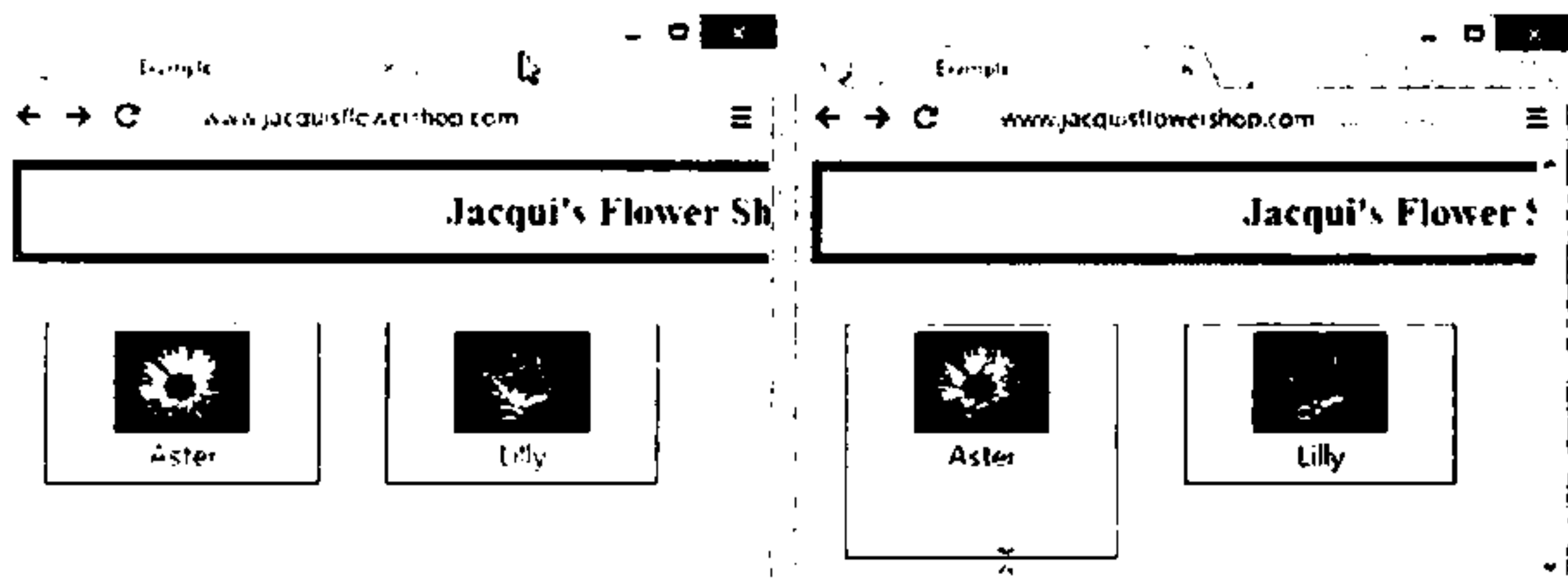


图25-9 改变一个元素的尺寸，而不改变其内容的尺寸

有时候这么做很有用，不过我在使用resizable组件时，几乎每一次都会使用alsoResize选项。在我看来，alsoResize选项最大的优点在于它匹配的元素并不受主元素的限制。如代码清单25-16所示，我们可以指定任意元素为主元素的关联元素。

#### 代码清单25-16 使用alsoResize属性同步改变关联元素的尺寸

```

...
<script type="text/javascript">
    $(document).ready(function() {
        $("#aster").resizable({
            alsoResize: "#aster img, #lily, #lily img"
        });
    });
</script>
...

```

在这个脚本中，我扩充了alsoResize选择器的选择范围，让它包含了页面中的其他div和img元素。这意味着，当我改变resizable的div元素时，jQuery UI会同步改变四个元素的大小。图25-10演示了这一效果。

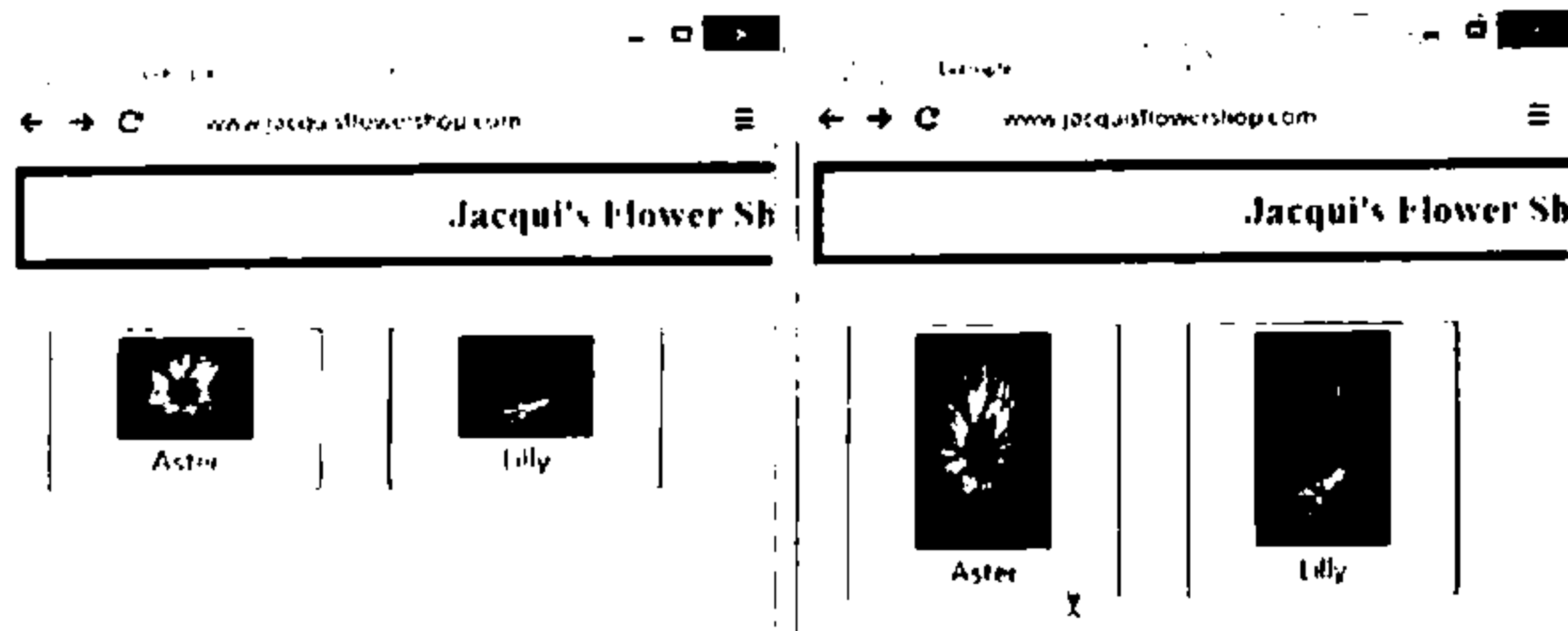


图25-10 一次改变多个元素的尺寸

## 2. 限制resizable元素的尺寸

我们还可以使用maxHeight、maxWidth、minHeight、minWidth选项限制resizable元素的尺寸。这四个选项的可取值都是像素数或者null值（表示没有限制）。代码清单25-17演示了这些选项的用法。

---

**提示** minWidth和minHeight选项的默认值是10 px。如果设置更小的值，jQuery UI将无法显示拖曳的“把手”，这意味着用户将失去增大元素尺寸的能力。因此，如果你打算使用更小的值，请一定要加小心。

---

代码清单25-17 限制resizable元素的尺寸

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#aster").resizable({
            alsoResize: "#aster img",
            maxWidth: 200,
            maxHeight: 150
        });
    });
</script>
...
```

---

**提示** 你也可以使用我在第24章讲过的拖曳组件的containment选项，那个选项允许把resizable元素的最大尺寸设置为另一个元素的尺寸。

---

## 3. 拖曳把手的出现位置

我们可以使用handles选项指定哪个边和哪个角可以拖曳。这个选项的可取值要么是all（表示所有边和角都可以拖曳），要么是一些方位的组合（如n、e、s、w、ne、se、nw、sw，表示指定的边和角）。

我们可以一次指定多个值，值与值之间使用逗号分隔。默认值是e,s,se，表示允许右下角（se，即东南）及右边（e）和下边（s）的拖曳。只有指定se作为handles值的一部分时，jQuery UI才会在右下角画出拖曳的把手图案。对于所有其他边和角，只有当鼠标悬停其上时，光标才会发生变化，指示允许拖曳。代码清单25-18展示了handles选项的用法。

代码清单25-18 使用handles选项

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("#aster").resizable({
            alsoResize: "#aster img"
        });

        $("#lily").resizable({
            alsoResize: "#lilyimg",
```

```

        handles: "n, s, e, w"</b>
    });
});
</script>
...

```

在这个脚本中，我让两个div元素都允许改变大小，并为其中的一个div元素定制了拖曳把手。在图25-11中我们可以看到，jQuery UI画出了可见的拖曳把手，而且光标发生了变化。

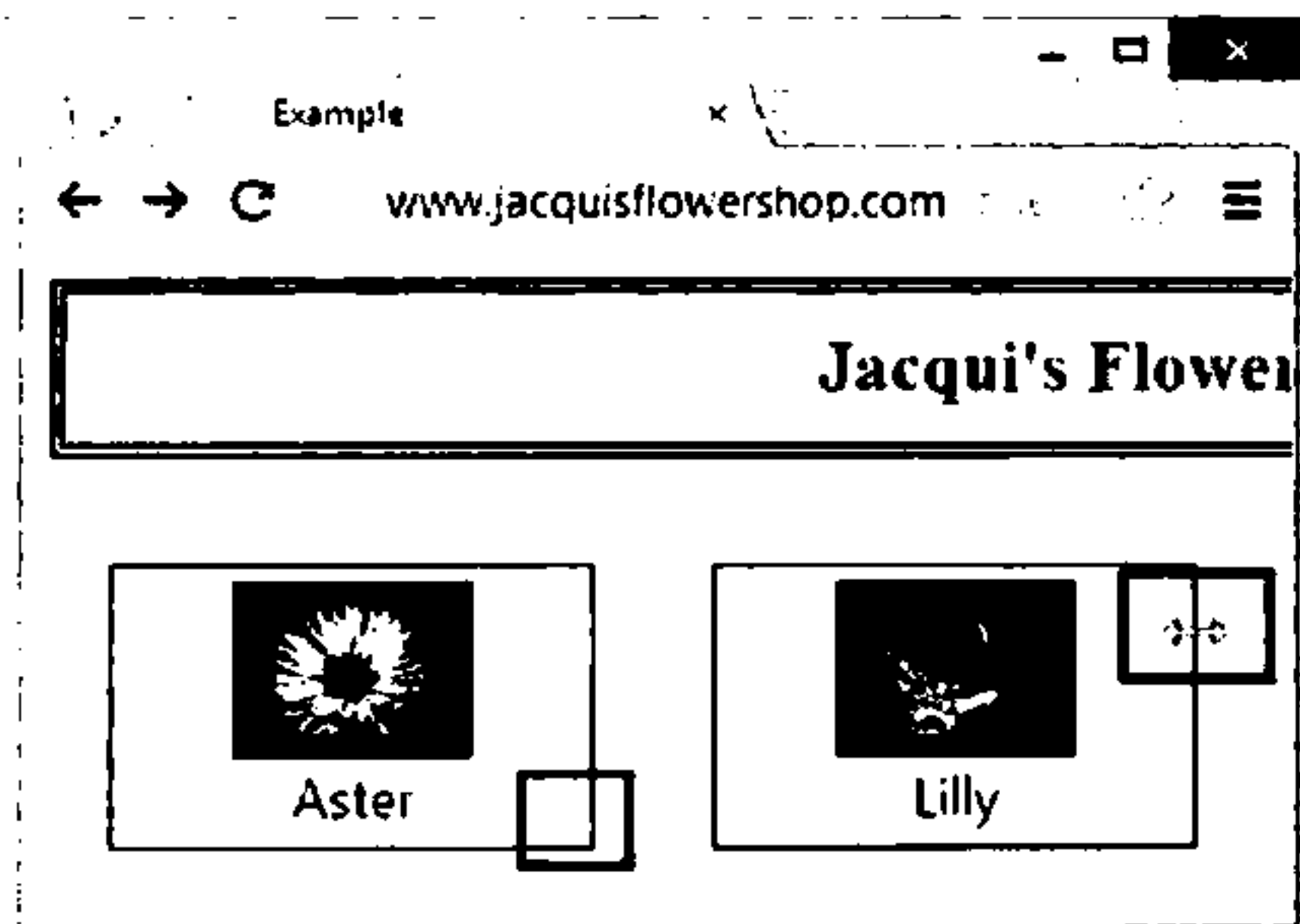


图25-11 使用handles选项

## 25.4 小结

本章讲解并演示了三个jQuery UI组件：sortable、selectable和resizable。与我在第24章讲到的拖曳和接收组件相比，这些组件较少使用，然而只要应用得当，这些组件很有用。与所有jQuery UI组件一样，最大的挑战在于当Web应用程序缺少标准的视觉提示时，如何使用户留意到能够拖曳、选择、排序或者改变元素的尺寸。因此，我们应将这些组件作为用户与应用程序或者网页交互的其他机制的补充（而非主要机制）。这样，高级用户就会发现这些高级交互功能，而一般用户则继续依赖那些更明显和约定俗成的技术。

在本书这一部分，我介绍了jQuery UI的种种小部件和交互组件。我们不仅可以利用它们创造出界面丰富的Web应用程序，还能让这些程序具有一致的风格，并且可以拥有近乎无限的定制优化能力，足以满足现实中的种种需要。本章，我会在示例中添加一些jQuery UI功能，并演示这些功能如何协同工作。

## 26.1 回顾重构示例

上一次重构示例页面时，我们一直在纠结是否使用jQuery核心功能重造几个jQuery UI已经实现的功能。图26-1是我们最终得到的效果。

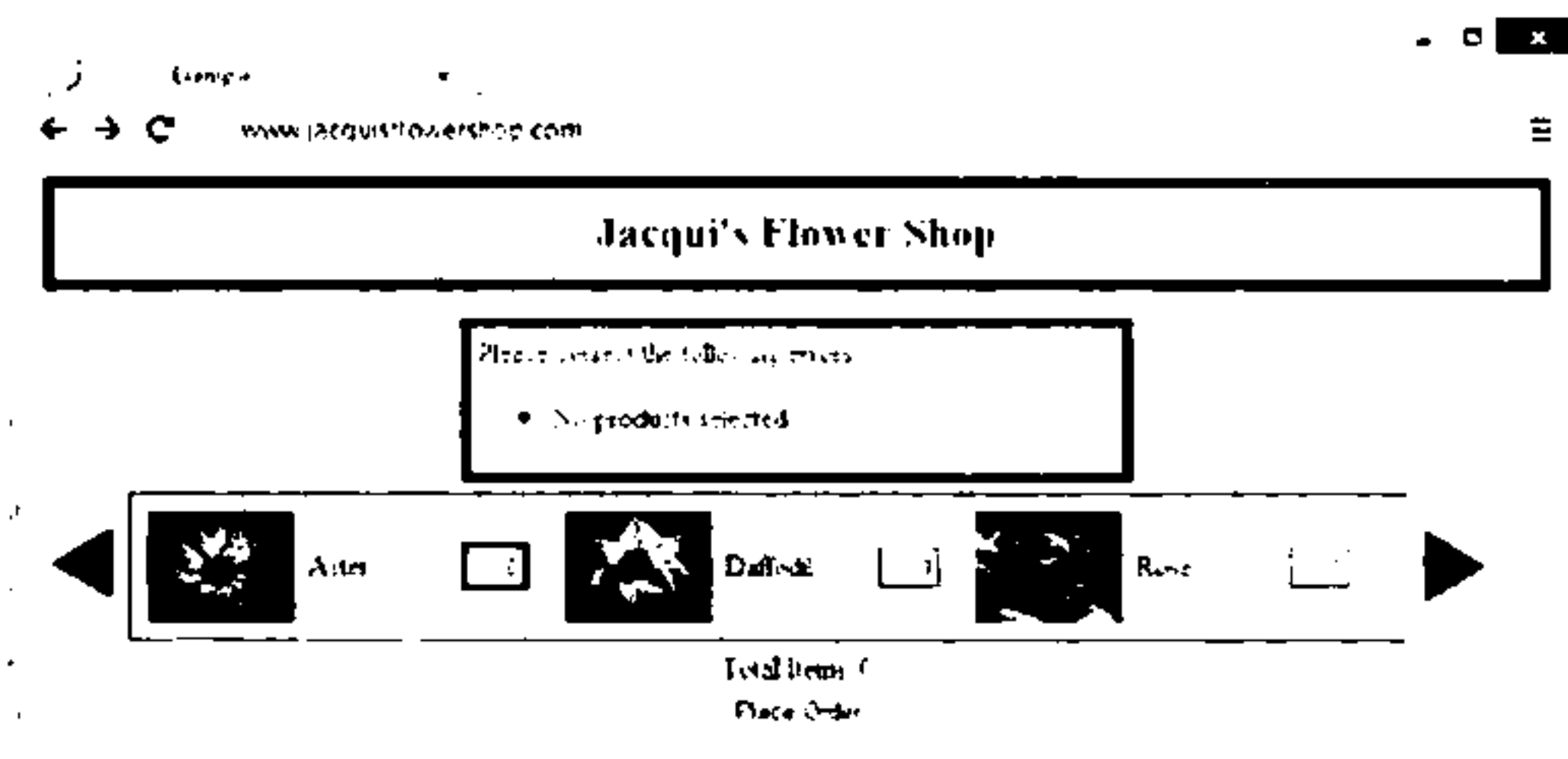


图26-1 上次重构结束之后的示例页面

本书的上一部分主要讲解了数据模板、表单验证和Ajax，我们还实现了一个简单的翻页效果，以支持在一行空间里显示出所有商品的效果。本章我会使用这些功能，不过重点是jQuery UI的应用。代码清单26-1是本章的基础：初始示例页面。

### 代码清单26-1 本章的基础：初始示例页面

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
```

```

<script src="handlebars.js"></script>
<script src="handlebars-jquery.js"></script>
<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<script id="flowerTpl" type="text/x-handlebars-template">
  {{#flowers}}
  <div class="dcell">
    
    <label for="{{product}}">{{name}}:</label>
    <input name="{{product}}" value="0" />
  </div>
  {{/flowers}}
</script>
<script type="text/javascript">
  $(document).ready(function () {
    $.getJSON("mydata.json", function (data) {
      $("#flowerTpl").template({ flowers: data })
        .filter("*").appendTo("#products");
    });
  });
</script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="products"></div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
</html>

```

我使用getJSON方法从一个JSON文件中得到商品的详细数据，并通过数据模板生成需要的HTML。我把这些HTML放到一个div元素内，并把它的id设置成products。这个基础页面的显示效果见图26-2。

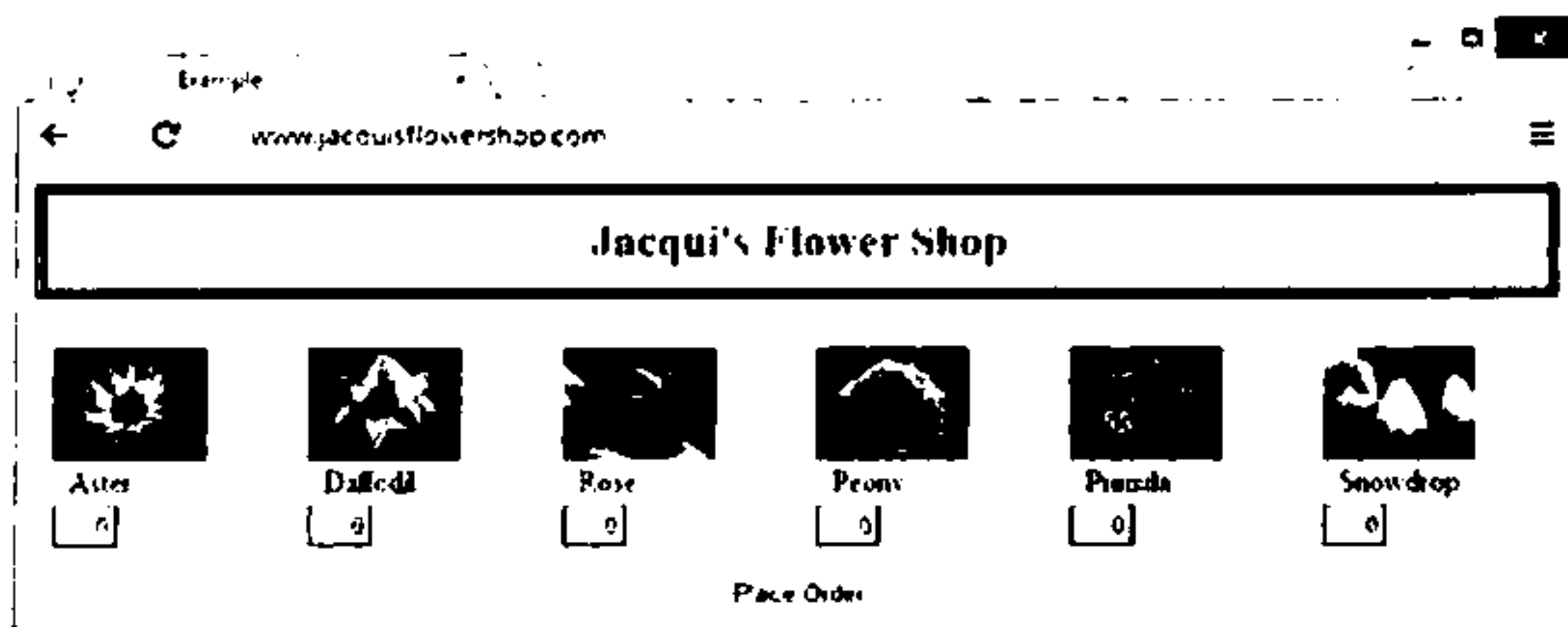


图26-2 本章起点：初始示例页面

## 26.2 展示商品

我准备使用折叠菜单组件显示商品给用户。尽管一共只有六种商品需要处理，我仍会把它们分成两种商品一组（一共三组），并使用jQuery生成折叠菜单组件所需的页面结构。代码清单26-2展示了示

例文档的变更。

代码清单26-2 重排商品顺序，改变商品HTML的结构

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    .dcell img {height: 60px}
  </style>
  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
      <div class="dcell">
        
        <label for="{{product}}">{{name}}:</label>
        <input name="{{product}}" value="0" />
      </div>
    {{/flowers}}
  </script>
  <script type="text/javascript">
    $(document).ready(function () {
      $.getJSON("mydata.json", function (data) {
        var flowers = $("#flowerTpl").template({ flowers: data }).filter("*");

        var rowCount = 1;
        for (var i = 0; i < flowers.length; i += 2) {

          $("<a>").text(data[i].name + " & " + data[i + 1].name)
            .appendTo("<h2>").parent().appendTo("#products");

          $("<div>").attr("id", "row" + (rowCount++))
            .appendTo("#products")
            .append(flowers.slice(i, i + 2))
        }
        $("#products").accordion();
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="products"></div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
</html>
```



我重写了getJSON方法的回调函数，以便生成折叠菜单。在这个函数中我改变了HTML结构，然后调用accordion方法。这次我从JSON数据对象中得到花的名字，将它作为小节的标题，不过仍使用数据模板插件生成需要的HTML，它把生成的HTML切段并放置到一个div包装元素中，以满足折叠菜单组件的需要。在图26-3中可以看到调用accordion方法之前的页面效果，以及它在调用之后的样子。

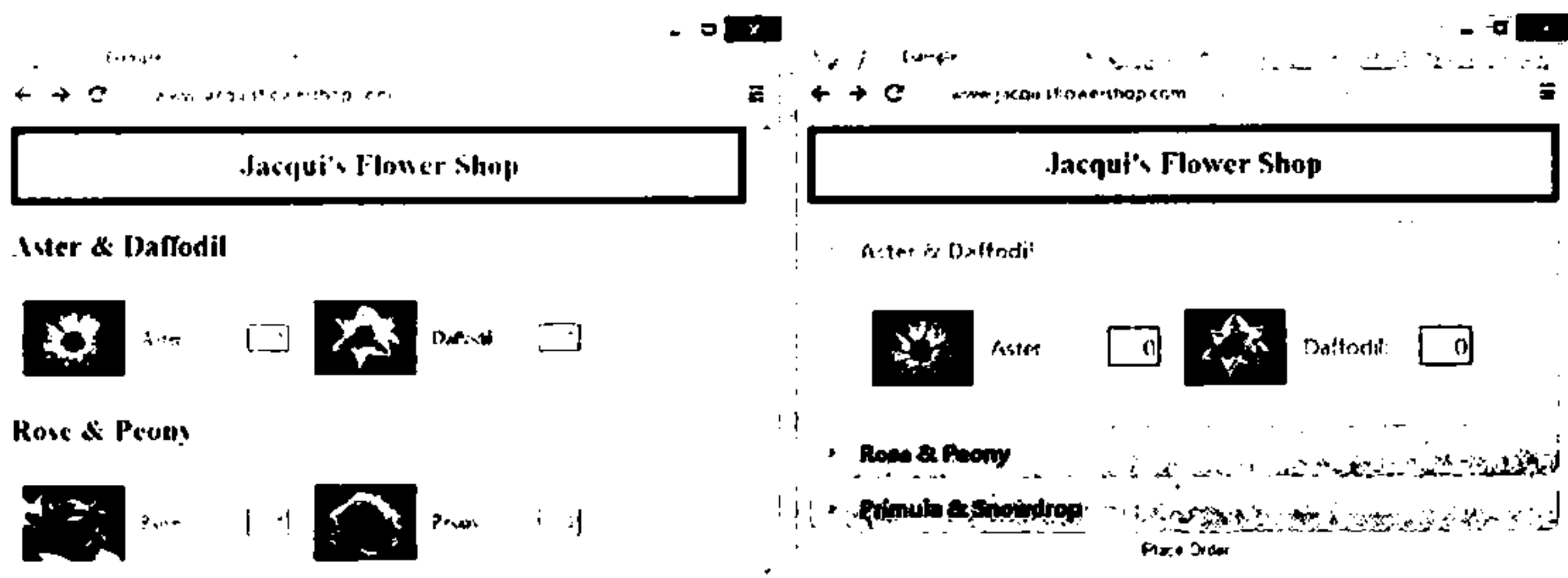


图26-3 生成HTML结构并调用accordion方法

## 26.3 添加购物车

接下来要做的是添加一个简单的购物车，用它显示顾客选中的商品。代码清单26-3展示了示例页面中新加的代码。

### 代码清单26-3 添加购物车

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    .dcell img {height: 60px}
    #basketTable {border: thin solid black; border-collapse: collapse}
    th, td {padding: 4px; width: 50px}
    td:first-child, th:first-child {width: 150px}
    #placeholder {text-align: center}
    #productWrapper {float: left; width: 65%}
    #basket {width: 30%; text-align: left; float: left; margin-left: 10px}
    #buttonDiv {clear: both}
  </style>
  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
```

```

    <div class="dcell">
        
        <label for="{{product}}">{{name}}:</label>
        <input name="{{product}}" value="0" />
    </div>
    {{/flowers}}
</script>
<script id="rowTpl" type="text/x-handlebars-template">
    <tr id="{{name}}"><td>{{product}}</td><td>{{val}}</td>
        <td><a href="#">Remove</a></td>
    </tr>
</script>
<script type="text/javascript">
    $(document).ready(function () {

        $.getJSON("mydata.json", function (data) {

            var flowers = $("#flowerTpl").template({ flowers: data }).filter("*");

            var rowCount = 1;
            for (var i = 0; i < flowers.length; i += 2) {
                $("<a>").text(data[i].name + " & " + data[i + 1].name)
                    .appendTo("<h2>").parent().appendTo("#products");
                $("<div>").attr("id", "row" + (rowCount++))
                    .appendTo("#products")
                    .append(flowers.slice(i, i + 2));
            }
            $("#products").accordion();

            $("input").change(function (event) {
                $("#placeholder").hide();
                var fname = $(this).attr("name");
                var row = $("tr[id=" + fname + "]");

                if (row.length == 0) {
                    $("#rowTpl").template({
                        name: fname,
                        val: $(this).val(),
                        product: $(this).siblings("label").text()
                    }).appendTo("#basketTable").find("a").click(function () {
                        removeTableRow($(this).closest("tr"));
                        var iElem = $("#products").find("input[name=" + fname + "]");
                        $("#products").accordion("option", "active",
                            iElem.closest("div[id^=row]").index("div[id^=row]"));
                        iElem.val(0).select();
                    });
                } else if ($(this).val() != "0") {
                    row.children().eq(1).text($(this).val());
                } else {
                    removeTableRow(row);
                }
            });
        });
    });

```

```

        function removeTableRow(row) {
            row.remove();
            if ($("#basketTable tbody").children(":visible").length == 1) {
                $("#placeholder").show();
            }
        }
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="productWrapper">
            <div id="products"></div>
        </div>
        <div id="basket" class="ui-widget">
            <table border=1 id="basketTable">
                <tr><th>Product</th><th>Quantity</th><th>Remove</th></tr>
                <tr id="placeholder"><td colspan=3>No Products</td></tr>
            </table>
        </div>
        <div id="buttonDiv"><button type="submit">Place Order</button></div>
    </form>
</body>
</html>

```

### 26.3.1 增加一个div元素包住折叠菜单

我想让购物车显示在折叠菜单的旁边。为了达到这种效果，我得像下面这样把调用accordion方法的元素放到另一个div元素之内：

```

...
<div id="productWrapper">
    <div id="products"></div>
</div>
...

```

如果折叠菜单组件的宽度小于父元素的宽度，就会使人困惑。为此，在添加了包装div之后，我使用以下CSS代码设置了它的位置与宽度：

```

...
#productWrapper {float: left; width: 65%}
...

```

这满足了折叠菜单宽度与包装元素相同的要求，同时包装元素的宽度又是其父元素的65%。

### 26.3.2 添加表格

我决定使用table元素显示购物车，这个表格已经作为静态内容添加到了页面中，代码如下。

```

...
<div id="basket" class="ui-widget">
    <table border=1 id="basketTable">
        <tr><th>Product</th><th>Quantity</th><th>Remove</th></tr>

```

```

        <tr id="placeholder"><td colspan=3>No Products</td></tr>
    </table>
</div>
...

```

为了配合折叠菜单，我把table元素也放到了一个包装div元素之内，并使用以下CSS代码设置它的位置与宽度：

```

...
#basket {width: 30%; text-align: left; float: left; margin-left: 10px}
...

```

这个表格只有两行，一行标题，一行为表格内容占个位置。这个表格参见图26-4。

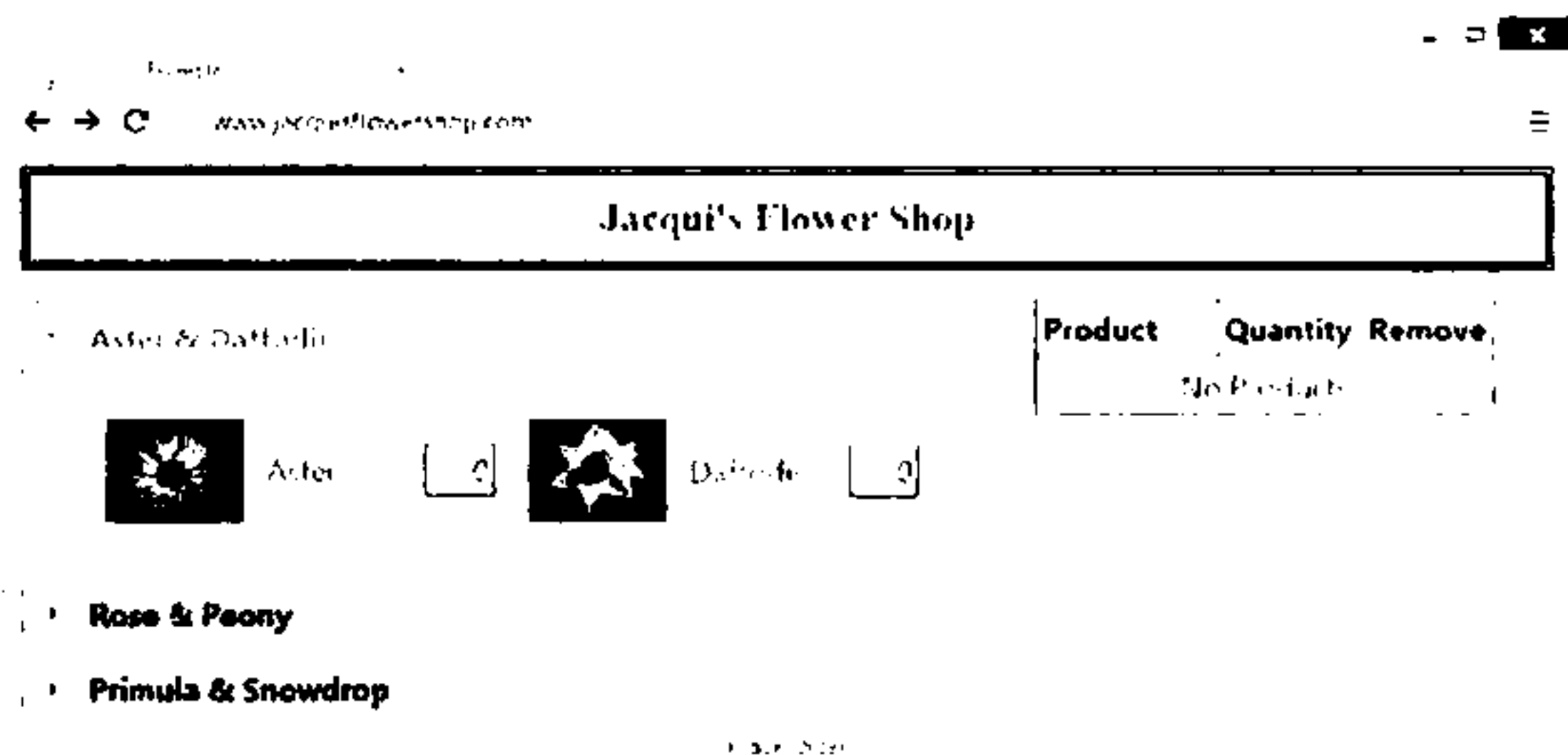


图26-4 在页面中添加购物车表格

### 26.3.3 输入值变更

为了让表格与折叠菜单建立联系，我监听了通过getJSON函数生成的那些文本框的change事件：

```

...
$("input").change(function(event) {
    $("#placeholder").hide();
    var fname = $(this).attr("name");
    var row = $("tr[id=" + fname + "]");

    if (row.length == 0) {
        $("#rowTpl").tmpl({
            name: fname,
            val: $(this).val(),
            product: $(this).siblings("label").text()
        }).appendTo("#basketTable").find("a").click(function () {
            removeTableRow($(this).closest("tr"));
            var iElem = $("#products").find("input[name=" + fname + "]")
            $("#products").accordion("option", "active",
                iElem.closest("div[id^=row]").index("div[id^=row]"));
            iElem.val(0).select();
        });
    } else if ($(this).val() != "0") {

```

```

        row.children().eq(1).text($(this).val());
    } else {
        removeTableRow(row)
    }
});
...

```

这个函数做了许多事。当用户改变一个输入框的值，我就会检查表格中是否已经包含该商品；如果没有，就使用下面的数据模板生成一行：

```

...
<script id="rowTpl" type="text/x-handlebars-template">
    <tr id="{{name}}"><td>{{product}}</td><td>{{val}}</td>
    <td><a href=#>Remove</a></td>
    </tr>
</script>
...

```

为了替模板准备数据，我使用jQuery核心方法拿到触发change事件的input元素的信息。除了这些信息，这里还需要商品的显示名。通过在DOM中访问下一个label元素并读取它的文本，我就得到了商品名：

```

...
$(this).siblings("label").text()
...

```

我把新行追加到表格中。而函数的第一行代码，就把占位置的那一行表格隐藏了：

```

...
$("#placeholder").hide();
...

```

新加到表里的那些行见图26-5。用户在input元素中输入一个值，在输入框焦点发生变化时，脚本就会在购物车中新加一行。

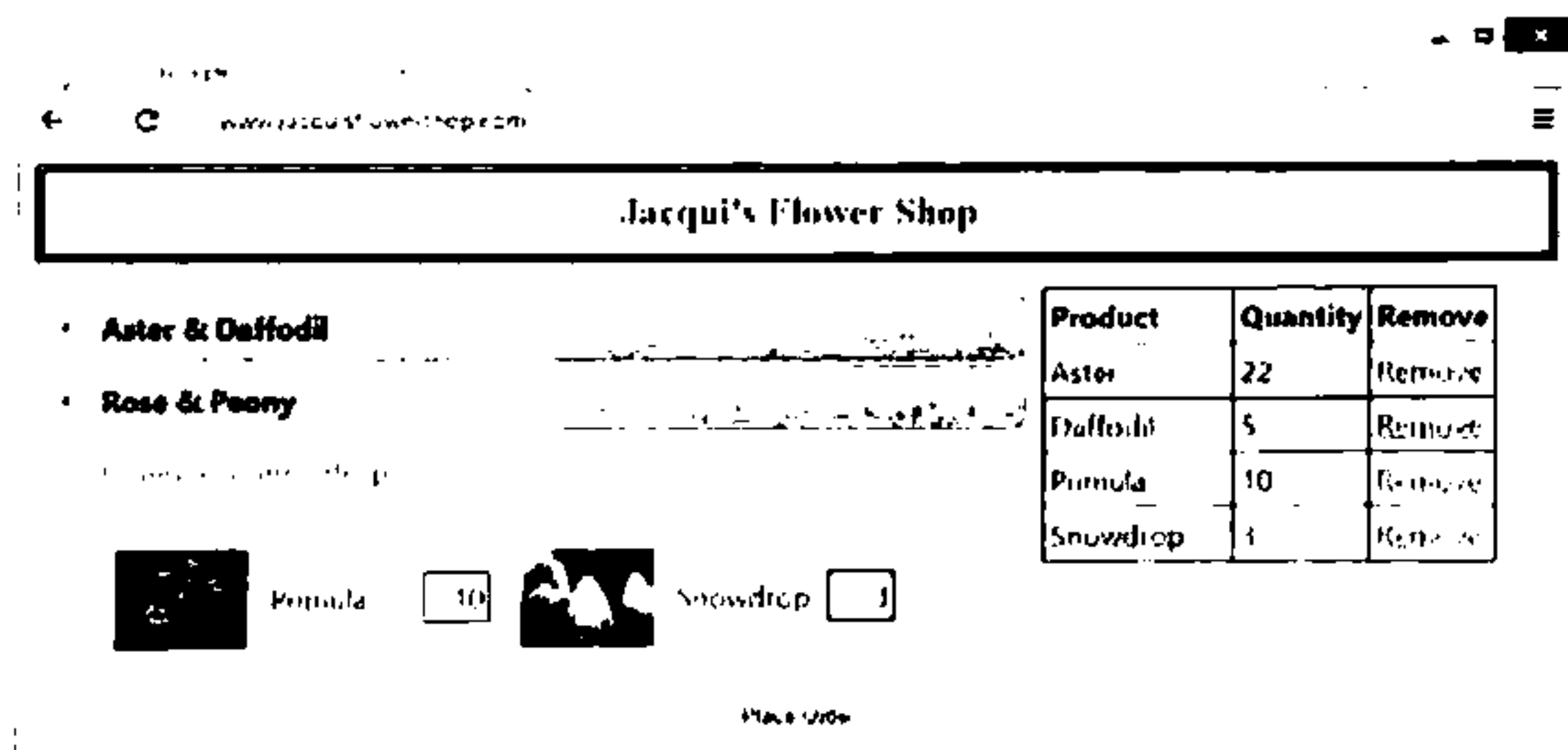


图26-5 在购物车表格中添加几行

### 1. 删除商品

注意，数据模板中表格行的一个单元格中有一个a元素。如以下代码所示，我为数据模板生成行中的这个a元素注册了事件处理函数：

```

...
}).appendTo("#basketTable").find("a").click(function () {
    removeTableRow($(this).closest("tr"));
    var iElem = $("#products").find("input[name=" + fname + "]);
    $("#products").accordion("option", "active",
        iElem.closest("div[id^=row]").index("div[id^=row]"));
    iElem.val(0).select();
});
...

```

在事件处理函数中，第一件事就是调用removeTableRow函数，并把a元素最近的祖先tr元素作为参数传递给它。removeTableRow函数使用remove方法从页面中删除指定元素。如以下代码所示，它也会在表格中再无商品有关行的情况下，恢复占位行的显示：

```

...
function removeTableRow(row) {
    row.remove();
    if ($("#basketTable tbody").children(":visible").length == 1) {
        $("#placeholder").show();
    }
}
...

```

当用户删掉其中一行时，我就到产品行中找出与它关联的input元素，然后遍历DOM找出该input元素的父元素对应的折叠面板，得到它的顺序号，并把它设为折叠组件的active属性的值。这样就打开了包含该商品（即用户刚从购物车中删除的商品）的折叠菜单。最后，把input元素的值设置为0，并调用select方法让它得到焦点，使当前值（即0）呈选中状态。具体的效果见图26-6（这个效果非常值得打开浏览器欣赏一下）。

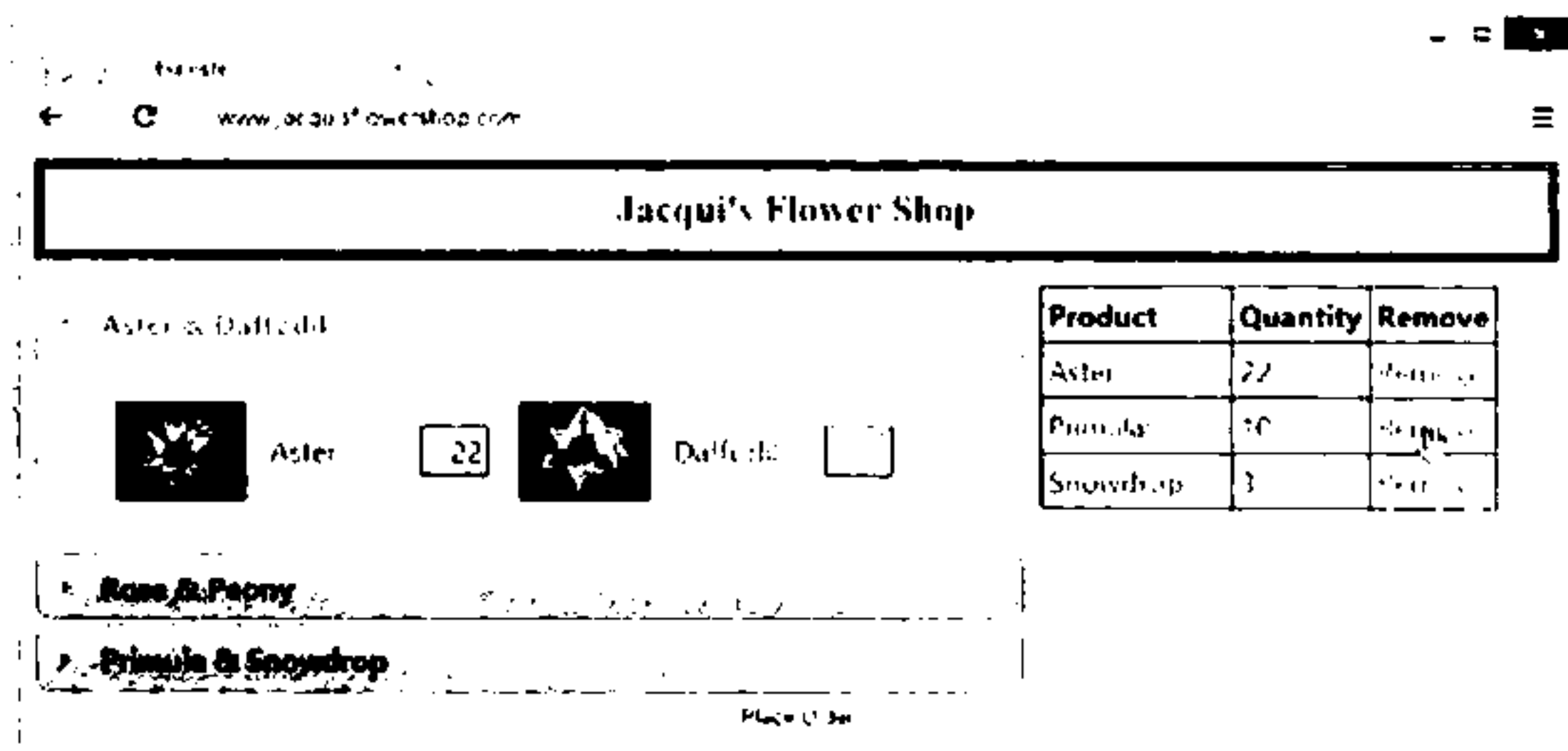


图26-6 从购物车中删除一种商品，让折叠菜单中对应该商品的input元素得到焦点

**提示** 当用户把一个input元素的大于0的值变为0时，我同样会从购物车中删除有关的行。我使用removeTableRow函数完成这一行为，而且在必要的时候还会自动恢复占位行的显示。

## 2. 更新已有商品

如果购物车中已经有了某种商品，那就有修改订货数量的需要。我并不是先删除这一行再用新行

替换，而是找出这一行，然后更新它的内容：

```
...
row.children().eq(1).text($(this).val());
...
```

row变量是一个包含着商品对应tr元素的jQuery对象，我通过位置（使用eq方法）找到保存数量的td元素，然后使用text方法设置它的内容。

## 26.4 装饰：应用主题样式

购物车的功能已经齐备，然而外观非常糟糕。幸亏jQuery UI提供了样式框架，这样我们就能够让元素像jQuery UI组件使用的主题一样，实现相同的视觉表现。代码清单26-4展示了对页面中HTML元素所做的一些修改。

代码清单26-4 让table元素（购物车）应用jQuery UI样式框架的样式

```
...
<body>
  <h1>Jacqui's Flower Shop</h1>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="productWrapper">
      <div id="products"></div>
    </div>
    <div id="basket" class="ui-widget ui-widget-content">
      <table border=0 id="basketTable">
        <tr class="ui-widget-header">
          <th>Product</th><th>Quantity</th><th>Remove</th>
        </tr>
        <tr id="placeholder"><td colspan=3>No Products</td></tr>
      </table>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
...
```

或许你已经注意到，在前面的几章内容中，我们已经使用过ui-widget class。它是jQuery UI基础样式，只需添到需要设置外观的外层容器元素上，就能让内部的元素得到和jQuery UI组件一致的外观。class ui-widget-content用于内容元素，而class ui-widget-header——顾名思义——专门用于标题元素。

---

**提示** 要详细了解jQuery UI样式框架，请阅读本书第35章。

---

除了添加这些class，我还使用下面的代码禁止表格显示边框：

```
...
#basketTable {border: none; border-collapse: collapse}
...
```

具体效果见图26-7。

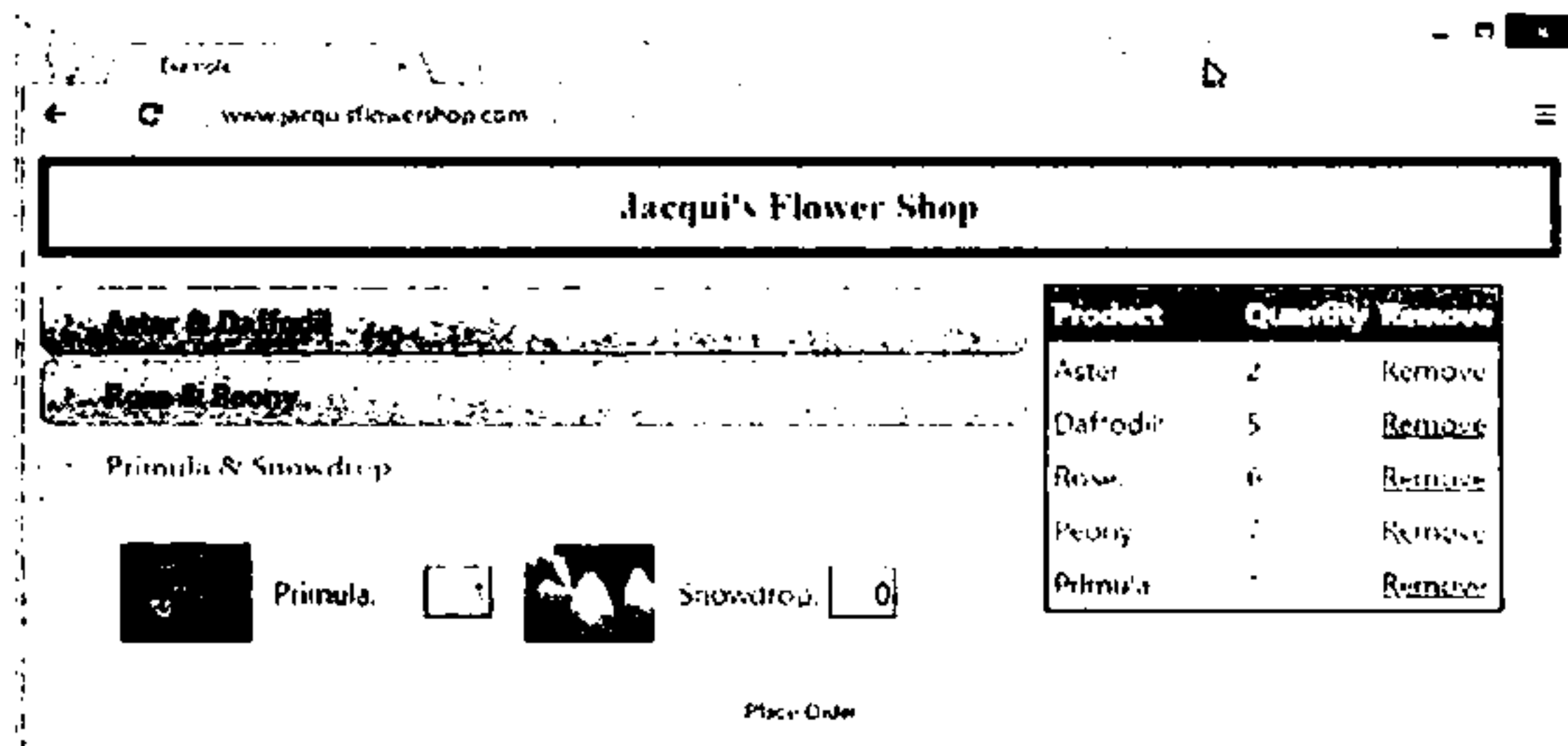


图26-7 让表格应用jQuery UI样式框架的样式

### 26.4.1 为更多元素应用框架样式

我们还可以更进一步，为更多的元素使用这些样式。代码清单26-5展示了页面代码的一些有价值的变更。

代码清单26-5 为更多元素应用框架样式

```
...
<body>
  <div id="logoWrapper" class="ui-widget ui-widget-content ui-corner-all">
    <h1 id="logo">Jacqui's Flower Shop</h1>
  </div>
  <form method="post" action="http://node.jacquisflowershop.com/order">
    <div id="productWrapper">
      <div id="products"></div>
    </div>
    <div id="basket" class="ui-widget ui-widget-content">
      <table border=0 id="basketTable">
        <tr class="ui-widget-header">
          <th>Product</th><th>Quantity</th><th>Remove</th>
        </tr>
        <tr id="placeholder"><td colspan=3>No Products</td></tr>
      </table>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
...
```

我把h1元素放到一个div元素内，并为它添加了一些框架样式，其中包括ui-corner-all。如图26-8所示，这个class会生成圆角效果。为了达到需要的效果，我还给页面添加了一些新的样式，用这些样式覆盖了从第3章起一直使用的styles.css文件中的样式：

```
...
<style type="text/css">
  .dcell img {height: 60px}
```



```

#basketTable {border: none; border-collapse: collapse}
th, td {padding: 4px; width: 50px}
td:first-child, th:first-child {width: 150px}
#placeholder {text-align: center}
#productWrapper {float: left; width: 65%}
#basket {width: 33%; text-align: left; float: left; margin-left: 10px;
    position: absolute; right: 10px}
#buttonDiv {clear: both}
#logo {font-size: 1.5em; background-size: contain; margin: 1px;
    border: none; color: inherit}
#logoWrapper {margin-bottom: 5px}
</style>
...

```

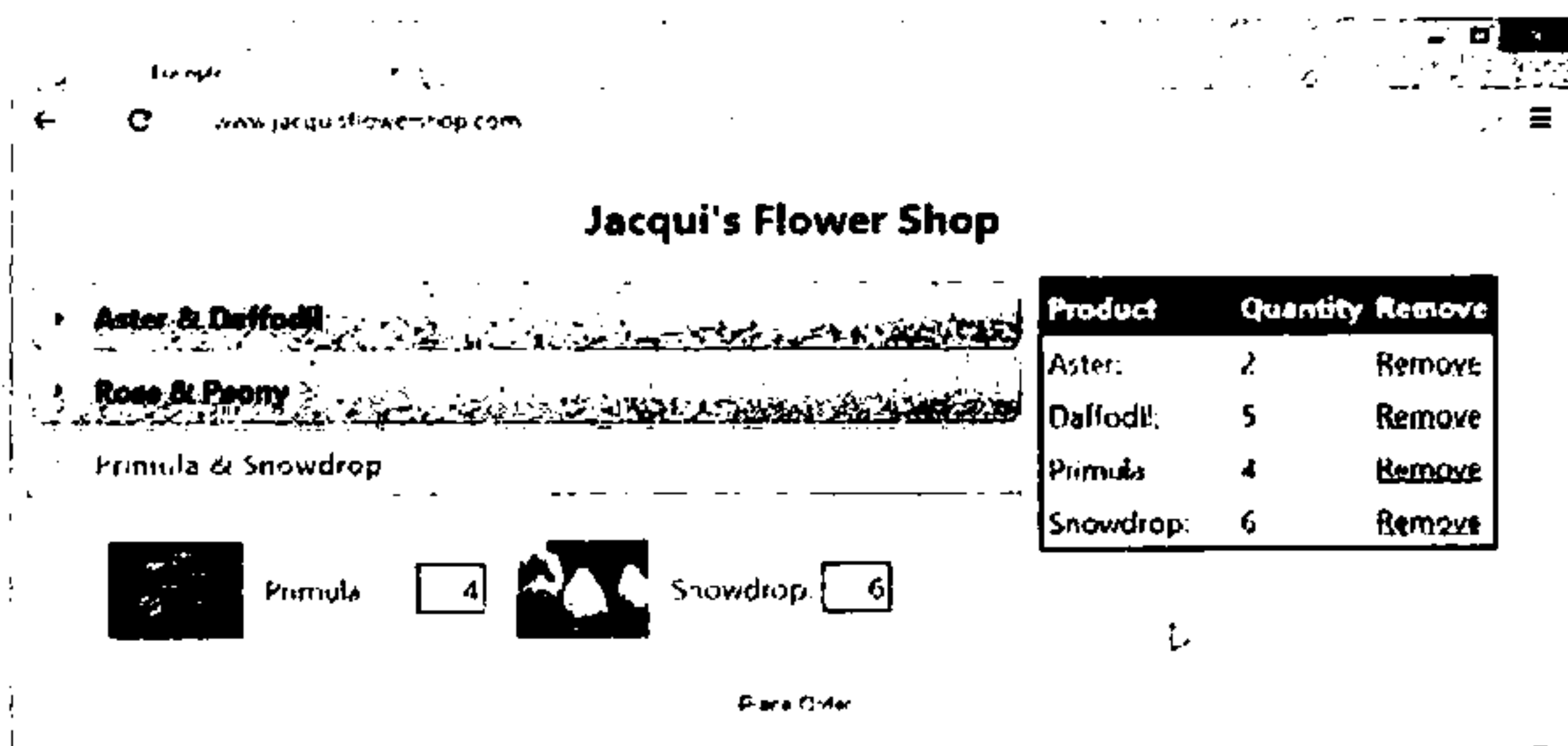


图26-8 给页面标题应用框架样式

### 26.4.2 为表格增加圆角效果

如图26-9所示，给table元素添加ui-corner-all class产生了一些问题。我们发现圆角效果对table元素无效。这是因为jQuery UI样式框架处理表格的方式与绝大多数浏览器处理表格的方式有冲突。

Product	Quantity	Remove
No Products		

图26-9 表格上的圆角效果

为了解决这个问题，我们需要修改table元素，略微改变施加到表格上的框架样式class，再添加一点自定义样式。首先，如代码清单26-6所示，我们需要修改table代码。

#### 代码清单26-6 修改table元素以支持圆角效果

```

...
<form method="post" action="http://node.jacquisflowershop.com/order">
  <div id="productWrapper">
    <div id="products"></div>
  </div>
  <div id="basket" class="ui-widget ui-widget-content ui-corner-all">

```

```

        <table border=0 id="basketTable">
            <thead id="thead" class="ui-widget-header">
                <tr>
                    <th class="ui-corner-tl">Product</th>
                    <th>Quantity</th>
                    <th class="ui-corner-tr">Remove</th>
                </tr>
            </thead>
            <tr id="placeholder"><td colspan=3>No Products</td></tr>
        </table>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
</form>
...

```

我在表格中添加了一个thead元素，这样就分开了表格标题和表格内容。给thead元素设置id属性并且添加ui-widget-header class很重要。由于标题现在成了ui-widget-header class（即thead元素）的一部分，我们可以从tr元素上删除ui-widget-header class。

接下来，我们把ui-corner-tl和ui-corner-tr class添加到标题行最外层的单元格上。这些class为元素生成左上角和右上角的圆角效果。（关于jQuery UI样式框架，详见本书第35章。）

然后，我们使用刚才给thead元素设置的id，在style元素内禁用它和table元素的边框。代码如下：

```

...
<style type="text/css">
    .dcell img {height: 60px}
    #basketTable {border: none; border-collapse: collapse}
    th, td {padding: 4px; width: 50px}
    td:first-child, th:first-child {width: 150px}
    #placeholder {text-align: center}
    #productWrapper {float: left; width: 65%}
    #basket {width: 33%; text-align: left; float: left; margin-left: 10px;
        position: absolute; right: 10px}
    #buttonDiv {clear: both}
    #logo {font-size: 1.5em; background-size: contain; margin: 1px;
        border: none; color: inherit}
    #logoWrapper {margin-bottom: 5px}
    #thead {border: none}
</style>
...

```

最后，我们要对removeTableRow函数做一点细微的调整。现在，我们把标题行挪到了thead元素内，因而tbody元素内少了一行。代码变更部分如下：

```

...
function removeTableRow(row) {
    row.remove();
    if ($("#basketTable tbody").children(":visible").length == 0) {
        $("#placeholder").show();
    }
}
...

```

**提示** 浏览器在解析table元素时会自动生成tbody元素。这是HTML的怪异点之一，因此我们不必手写这一元素。（如果你愿意，当然也可以写进代码里。）

做完这些修改之后，如图26-10所示，我们得到了一个与页面其他元素风格一致的圆角表格。

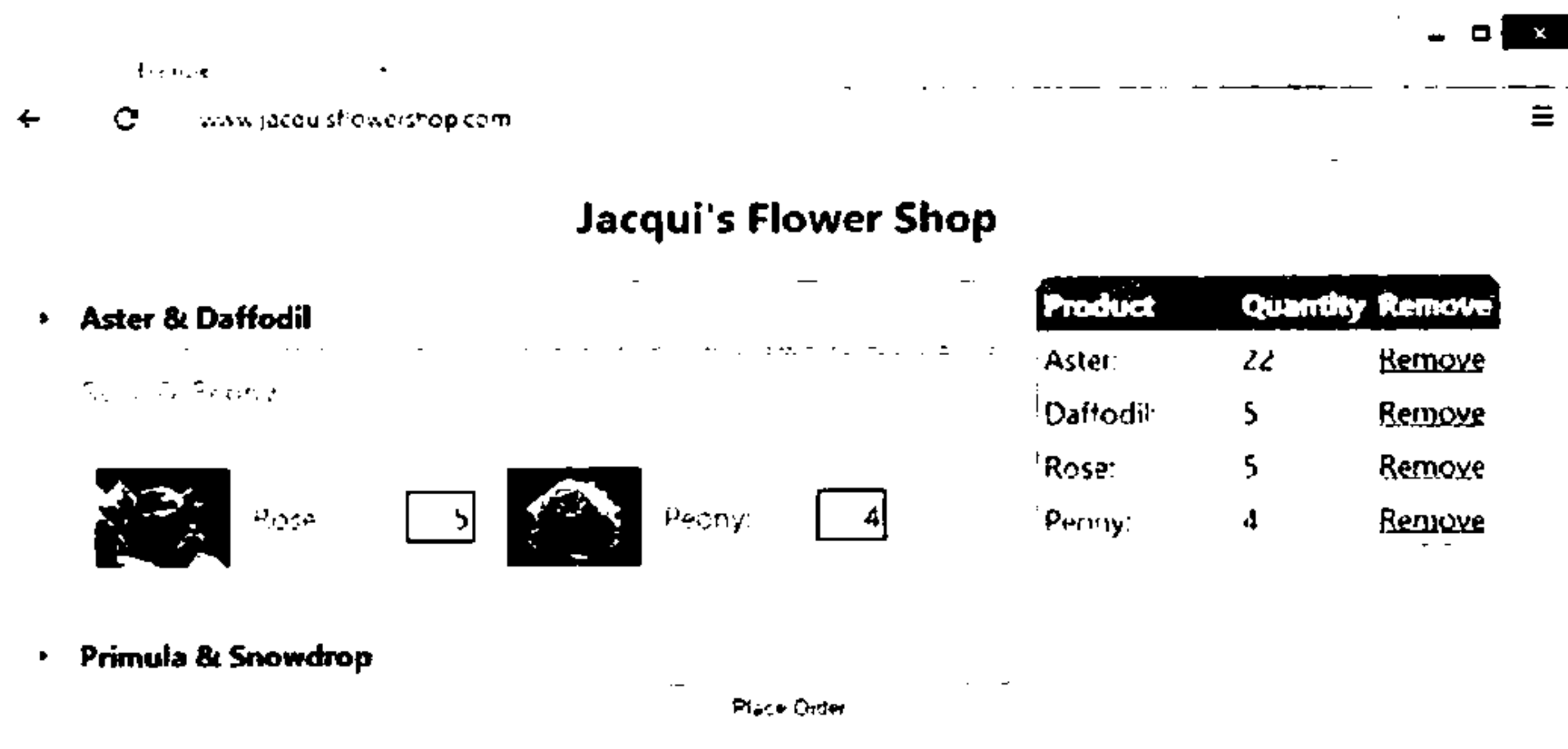


图26-10 圆角表格

## 26.5 生成 jQuery UI 按钮

接下来是找出页面中的按钮，并把它转换为jQuery UI按钮组件。代码清单26-7列出了示例页面变更的代码。

**代码清单26-7** 找出页面中的按钮并把它们转换为按钮组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    .dcell img {height: 60px}
    #basketTable {border: none; border-collapse: collapse}
    th, td {padding: 4px; width: 50px}
    td:first-child, th:first-child {width: 150px}
    #placeholder {text-align: center}
    #productWrapper {float: left; width: 65%}
    #basket {text-align: left;}
    #buttonDiv {clear: both; margin: 5px}
```

```

    #logo {font-size: 1.5em; background-size: contain; margin: 1px;
        border: none; color: inherit}
    #logoWrapper {margin-bottom: 5px}
    #thead {border: none}
</style>
<script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
        <div class="dcell">
            
            <label for="{{product}}">{{name}}:</label>
            <input name="{{product}}" value="0" />
        </div>
    {{/flowers}}
</script>
<script id="rowTpl" type="text/x-handlebars-template">
    <tr id="{{name}}"><td>{{product}}</td><td>{{val}}</td>
        <td><a href="#">Remove</a></td>
    </tr>
</script>
<script type="text/javascript">
    $(document).ready(function () {

        $.getJSON("mydata.json", function (data) {

            var flowers = $("#flowerTpl").template({ flowers: data }).filter("*");
            var rowCount = 1;
            for (var i = 0; i < flowers.length; i += 2) {
                $("<a>").text(data[i].name + " & " + data[i + 1].name)
                    .appendTo("<h2>").parent().appendTo("#products");
                $("<div>").attr("id", "row" + (rowCount++))
                    .appendTo("#products")
                    .append(flowers.slice(i, i + 2));
            }
            $("#products").accordion();

            $("input").change(function (event) {
                $("#placeholder").hide();
                var fname = $(this).attr("name");
                var row = $("tr[id=" + fname + "]");

                if (row.length == 0) {
                    $("#rowTpl").template({
                        name: fname,
                        val: $(this).val(),
                        product: $(this).siblings("label").text()
                    }).appendTo("#basketTable").find("a").click(function () {
                        removeTableRow($(this).closest("tr"));
                        var iElem = $("#products").find("input[name=" + fname + "]");
                        $("#products").accordion("option", "active",
                            iElem.closest("div[id^=row]").index("div[id^=row]"));
                        iElem.val(0).select();
                    });
                } else if ($(this).val() != "0") {
                    row.children().eq(1).text($(this).val());
                }
            });
        });
    });

```

```

        } else {
            removeTableRow(row);
        }
    });
});

$("#buttonDiv, #basket").wrapAll("<div />").parent().css({
    float: "left",
    marginLeft: "2px"
});

$("#button").button();

function removeTableRow(row) {
    row.remove();
    if ($("#basketTable tbody").children(":visible").length == 0) {
        $("#placeholder").show();
    }
}
});
</script>
</head>
<body>
    <div id="logoWrapper" class="ui-widget ui-widget-content ui-corner-all">
        <h1 id="logo">Jacqui's Flower Shop</h1>
    </div>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="productWrapper">
            <div id="products"></div>
        </div>
        <div id="basket" class="ui-widget ui-widget-content ui-corner-all">
            <table border=0 id="basketTable">
                <thead id="thead" class="ui-widget-header">
                    <tr>
                        <th class="ui-corner-tl">Product</th>
                        <th>Quantity</th>
                        <th class="ui-corner-tr">Remove</th>
                    </tr>
                </thead>
                <tr id="placeholder"><td colspan=3>No Products</td></tr>
            </table>
        </div>
        <div id="buttonDiv"><button type="submit">Place Order</button></div>
    </form>
</body>
</html>

```

我已经把buttonDiv和购物车元素放到一个新的div元素中，并修改了样式以调整这些元素的位置。而且，如图26-11所示，这里通过调用button方法创建了jQuery UI按钮。

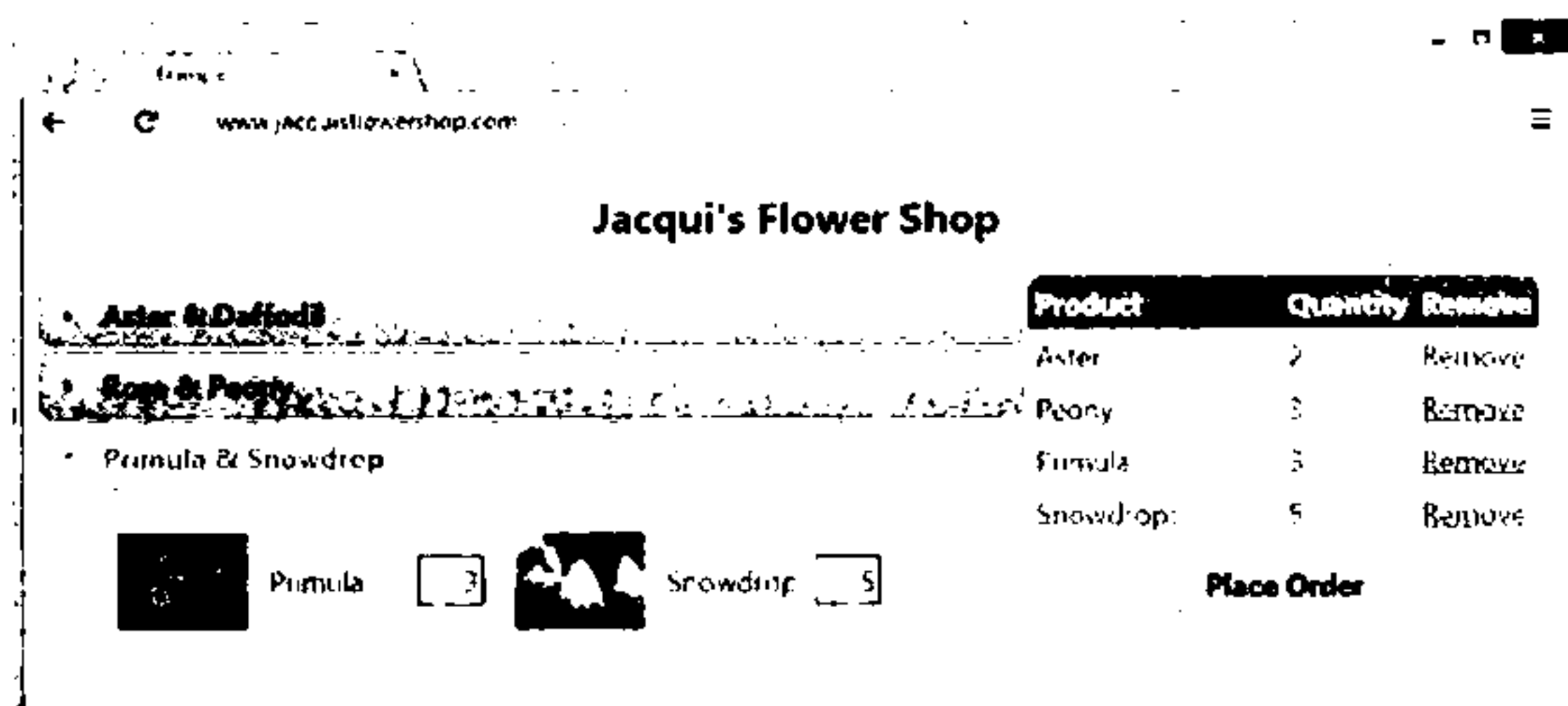


图26-11 重新摆放按钮，并把button元素转换为jQuery UI按钮

## 26.6 添加完成购买对话框

当用户单击Place Order（下单）按钮时，我希望从用户那里额外收集一些信息。在第20章时我已经演示过应该如何使用标签（tabs）组件显示由多个部分组成的表单，为了不那么单调，这次我们改用对话框组件。为了对话框组件而做的代码变更见代码清单26-8。

### 代码清单26-8 增加对话框组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <style type="text/css">
    .dcell img {height: 60px}
    #basketTable {border: none; border-collapse: collapse}
    th, td {padding: 4px; width: 50px}
    td:first-child, th:first-child {width: 150px}
    #placeholder {text-align: center}
    #productWrapper {float: left; width: 65%}
    #basket {text-align: left;}
    #buttonDiv {clear: both; margin: 5px}
    #logo {font-size: 1.5em; background-size: contain; margin: 1px;
      border: none; color: inherit}
    #logoWrapper {margin-bottom: 5px}
    #thead {border: none}
    #completeDialog input {width: 150px; margin-left: 5px; text-align: left}
    #completeDialog label {width: 60px; text-align: right}
  </style>
  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#flowers}}
      <div class="dcell">
```

```

        
        <label for="{{product}}">{{name}}:</label>
        <input name="{{product}}" value="0" />
    </div>
    {{/flowers}}
</script>
<script id="rowTpl" type="text/x-handlebars-template">
    <tr id="{{name}}"><td>{{product}}</td><td>{{val}}</td>
        <td><a href="#">Remove</a></td>
    </tr>
</script>
<script type="text/javascript">
    $(document).ready(function () {

        $.getJSON("mydata.json", function (data) {

            var flowers = $("#flowerTpl").template({ flowers: data }).filter("*");

            var rowCount = 1;
            for (var i = 0; i < flowers.length; i += 2) {
                $("<a>").text(data[i].name + " & " + data[i + 1].name)
                    .appendTo("<h2>").parent().appendTo("#products");
                $("<div>").attr("id", "row" + (rowCount++))
                    .appendTo("#products")
                    .append(flowers.slice(i, i + 2));
            }
            $("#products").accordion();

            $("#products input").change(function (event) {
                $("#placeholder").hide();
                var fname = $(this).attr("name");
                var row = $("tr[id=" + fname + "]");

                if (row.length == 0) {
                    $("#rowTpl").template({
                        name: fname,
                        val: $(this).val(),
                        product: $(this).siblings("label").text()
                    }).appendTo("#basketTable").find("a").click(function () {
                        removeTableRow($(this).closest("tr"));
                        var iElem = $("#products").find("input[name=" + fname + "]");
                        $("#products").accordion("option", "active",
                            iElem.closest("div[id^=row]").index("div[id^=row]"));
                        iElem.val(0).select();
                    });
                } else if ($(this).val() != "0") {
                    row.children().eq(1).text($(this).val());
                } else {
                    removeTableRow(row);
                }
            });
        });

        $("#buttonDiv, #basket").wrapAll("<div />").parent().css({

```

```

        float: "left",
        marginLeft: "2px"
    });

    $("button").button();

    $("#completeDialog").dialog({
        modal: true,
        buttons: [{ text: "OK", click: sendOrder },
            {
                text: "Cancel", click: function () {
                    $("#completeDialog").dialog("close");
                }
            }
        ]
    });

    function sendOrder() {

    }

    function removeTableRow(row) {
        row.remove();
        if ($("#basketTable tbody").children(":visible").length == 0) {
            $("#placeholder").show();
        }
    }
    });
</script>
</head>
<body>
    <div id="logoWrapper" class="ui-widget ui-widget-content ui-corner-all">
        <h1 id="logo">Jacqui's Flower Shop</h1>
    </div>
    <form method="post" action="http://node.jacquisflowershop.com/order">
        <div id="productWrapper">
            <div id="products"></div>
        </div>
        <div id="basket" class="ui-widget ui-widget-content ui-corner-all">
            <table border=0 id="basketTable">
                <thead id="thead" class="ui-widget-header">
                    <tr>
                        <th class="ui-corner-tl">Product</th>
                        <th>Quantity</th>
                        <th class="ui-corner-tr">Remove</th>
                    </tr>
                </thead>
                <tr id="placeholder"><td colspan=3>No Products</td></tr>
            </table>
        </div>
        <div id="buttonDiv"><button type="submit">Place Order</button></div>
    </form>
    <div id="completeDialog" title="Complete Purchase">
        <div><label for="name">Name: </label><input name="first" /></div>
        <div><label for="email">Email: </label><input name="email" /></div>
    </div>

```



```

        <div><label for="city">City: </label><input name="city" /></div>
    </div>
</body>
</html>

```

我在body元素中新加了一个div元素，里面是计划显示给用户看的内容，我还（在style元素中）新加了一些样式，以覆盖通过页面link元素导入的样式表的一些样式。下面是为生成对话框组件而调用dialog方法的代码：

```

...
$("#completeDialog").dialog({
    modal: true,
    buttons: [{ text: "OK", click: sendOrder },
        {
            text: "Cancel", click: function () {
                $("#completeDialog").dialog("close");
            }
        }
    ]
});
...

```

我创建了一个有着两个按钮的模态（独占式）对话框。单击Cancel（取消）按钮会关闭对话框，单击OK按钮则调用sendOrder函数。现在这个函数还是空的，什么也不干。

你应该还记得，在第22章我们提到过对话框组件默认是允许显示状态，也就是说一旦创建完成，它就立刻显示在用户面前。图26-12展示了这个对话框的显示效果。

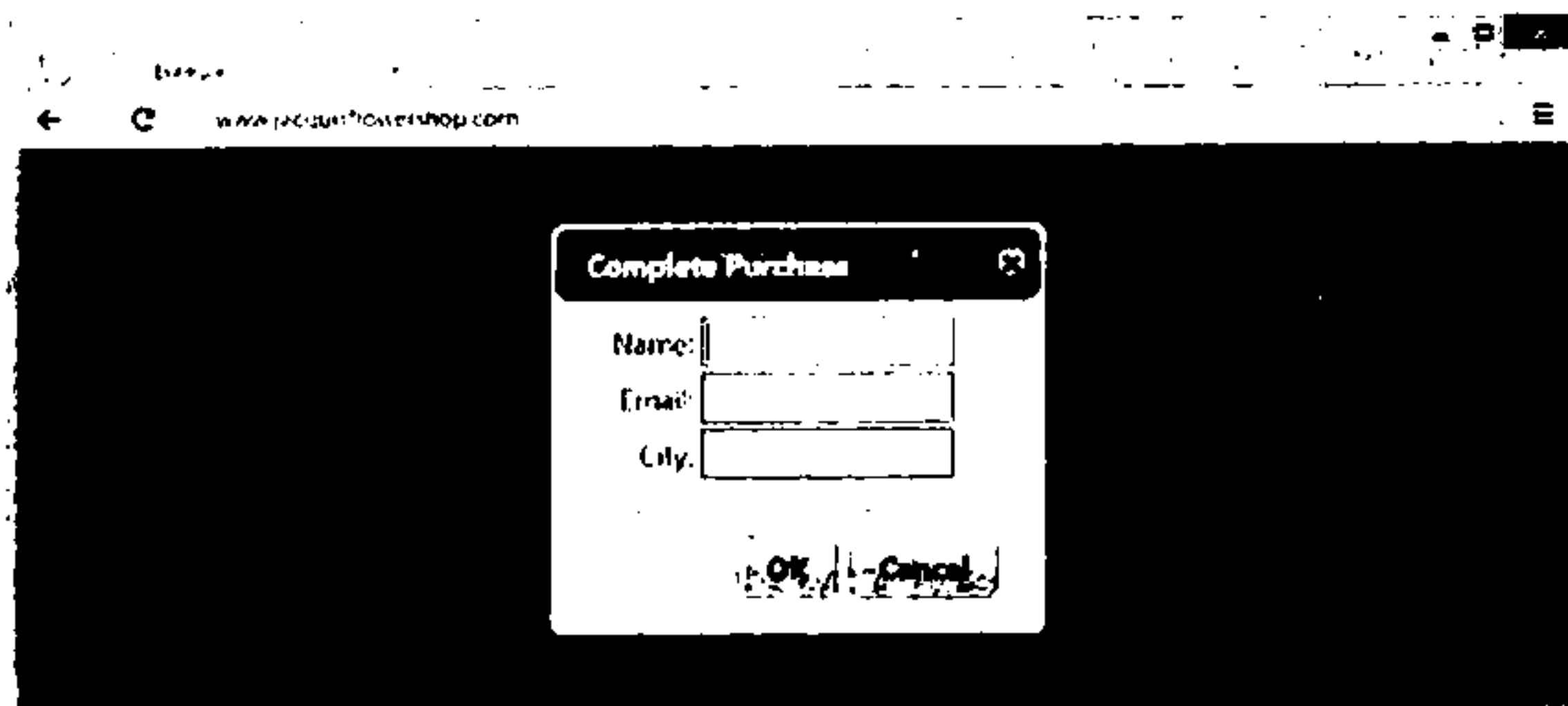


图26-12 完成购买对话框

**提示** 当我设置input元素的change事件时，缩小了结果集的选择范围。我让选择结果排除对话框中的那些input元素。如果不这么做，那么在完成购买对话框中输入数据就会（错误地）在购物车中添加新行。

## 26.7 处理下单按钮的单击事件

除非用户点击了下单（Place Order）按钮，否则我并不希望用户看到完成购买对话框。如代码清

单26-9所示，我利用对话框组件的autoOpen选项使它隐藏，直到用户单击下单按钮才让它显示。

#### 代码清单26-9 隐藏对话框并处理按钮单击事件

```

...
<script type="text/javascript">
    $(document).ready(function () {

        $.getJSON("mydata.json", function (data) {

            var flowers = $("#flowerImpl").template({ flowers: data }).filter("*");

            var rowCount = 1;
            for (var i = 0; i < flowers.length; i += 2) {
                $("<a>").text(data[i].name + " & " + data[i + 1].name)
                    .appendTo("<h2>").parent().appendTo("#products");
                $("<div>").attr("id", "row" + (rowCount++))
                    .appendTo("#products")
                    .append(flowers.slice(i, i + 2));
            }
            $("#products").accordion();

            $("#products input").change(function (event) {
                $("#placeholder").hide();
                var fname = $(this).attr("name");
                var row = $("tr[id=" + fname + "]");

                if (row.length == 0) {
                    $("#rowImpl").template({
                        name: fname,
                        val: $(this).val(),
                        product: $(this).siblings("label").text()
                    }).appendTo("#basketTable").find("a").click(function () {
                        removeTableRow($(this).closest("tr"));
                        var iElem = $("#products").find("input[name=" + fname + "]");
                        $("#products").accordion("option", "active",
                            iElem.closest("div[id^=row]").index("div[id^=row]"));
                        iElem.val(0).select();
                    });
                } else if ($(this).val() != "0") {
                    row.children().eq(1).text($(this).val());
                } else {
                    removeTableRow(row);
                }
            });

            $("#buttonDiv, #basket").wrapAll("<div />").parent().css({
                float: "left",
                marginLeft: "2px"
            });

            $("button").button().click(function (e) {
                e.preventDefault();
            });
        });
    });

```

```

    if ($("#placeholder:visible").length) {

        $("#<div>Please select some products</div>").dialog({
            modal: true,
            buttons: [{
                text: "OK",
                click: function () { $(this).dialog("close") }
            }]
        })
    } else {
        $("#completeDialog").dialog("open");
    }
});

$("#completeDialog").dialog({
    modal: true,
    autoOpen: false,
    buttons: [{ text: "OK", click: sendOrder },
        {
            text: "Cancel", click: function () {
                $("#completeDialog").dialog("close");
            }
        }
    ]
});

function sendOrder() {

}

function removeTableRow(row) {
    row.remove();
    if ($("#basketTable tbody").children(":visible").length == 0) {
        $("#placeholder").show();
    }
}

});
</script>
...

```

当用户单击下单按钮时，我先用jQuery检查购物车中的占位元素是否可见。如果占位元素可见（说明用户没有购买任何商品），我就使用一个选择器生成一个jQuery对象，里面存放一些需要给用户看的元素。

在这里我实际上把占位元素当成了用户是否选中一些商品的标志。如果占位元素隐藏，说明用户购买了一些东西，如果占位元素可见，就说明他们什么也没买。

---

**提示** 这是一个在页面中使用对话框的好例子，然而这也意味着检查用户是否购买商品的逻辑依赖于购物车，一旦我修改购物车的实现方式，也就需要修改检查用户是否购买商品的逻辑。

---

如果用户单击了下单按钮，但没有购买任何商品，我就动态生成一个对话框告诉他。图26-13展示了这个对话框的效果。如果用户选择了一些商品，就会看到完成购买对话框。利用这个对话框，我能

够从用户那里得到发货需要的信息。



图26-13 在没有购买任何商品时单击下单按钮，就会显示这个对话框

## 26.8 完成订单

现在只剩下实现sendOrder函数了。我们已经知道有许多种方法可以使用Ajax发送数据给服务器，因此简单起见，在这个函数中我只是把从各个input元素那里收集来的数据汇总为一个JSON对象，这个对象可以发送给服务器以进行后续处理（而没有真正发送给服务器）。代码清单26-10展示了添加到sendOrder函数中的代码。

代码清单26-10 处理完成订单

```
...
function sendOrder() {
    var data = new Object();
    $("input").each(function(index, elem) {
        var jqElem = $(elem);
        data[jqElem.attr("name")] = jqElem.val();
    })
    console.log(JSON.stringify(data));
    $("#completeDialog").dialog("close");
    $("#products input").val("0");
    $("#products").accordion("option", "active", 0)
    $("#basketTable tbody").children(":visible").remove();
    $("#placeholder").show();
}
...
```

在这个函数中，我从每个input元素中得到值，并把它们以键值对的方式追加到一个对象，之后又把这个对象转换成JSON字符串输出到控制台。

接下来的内容更实用，我再一次初始化这个页面：关闭对话框，把所有input元素的值恢复为0，并且把折叠菜单的第一个面板打开，清空购物车。图26-14展示的是用户购买了一些商品的页面，我使用这些商品生成需要发送给服务器的JSON字符串。

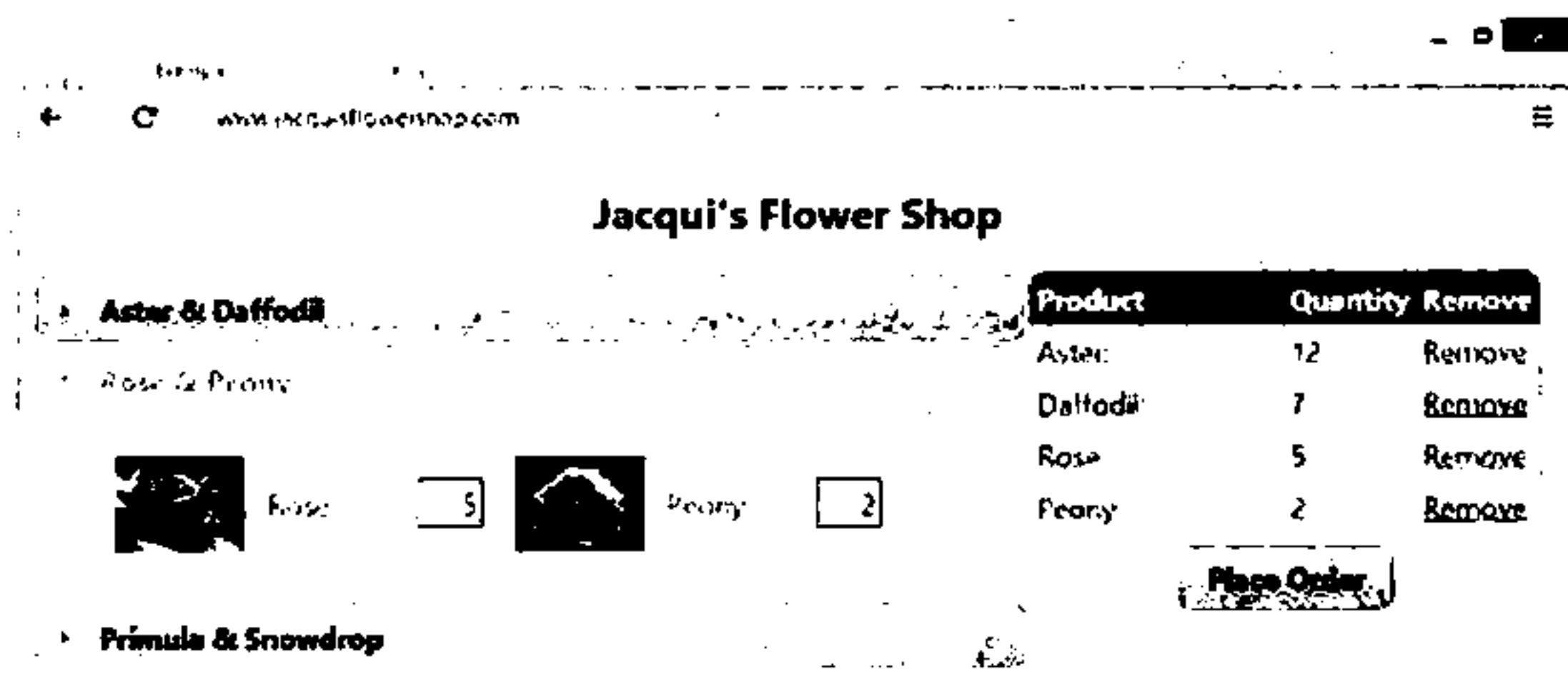


图26-14 在页面中选择一些商品

单击Place Order按钮时，如图26-15所示，脚本立即显示额外收集用户信息的对话框。

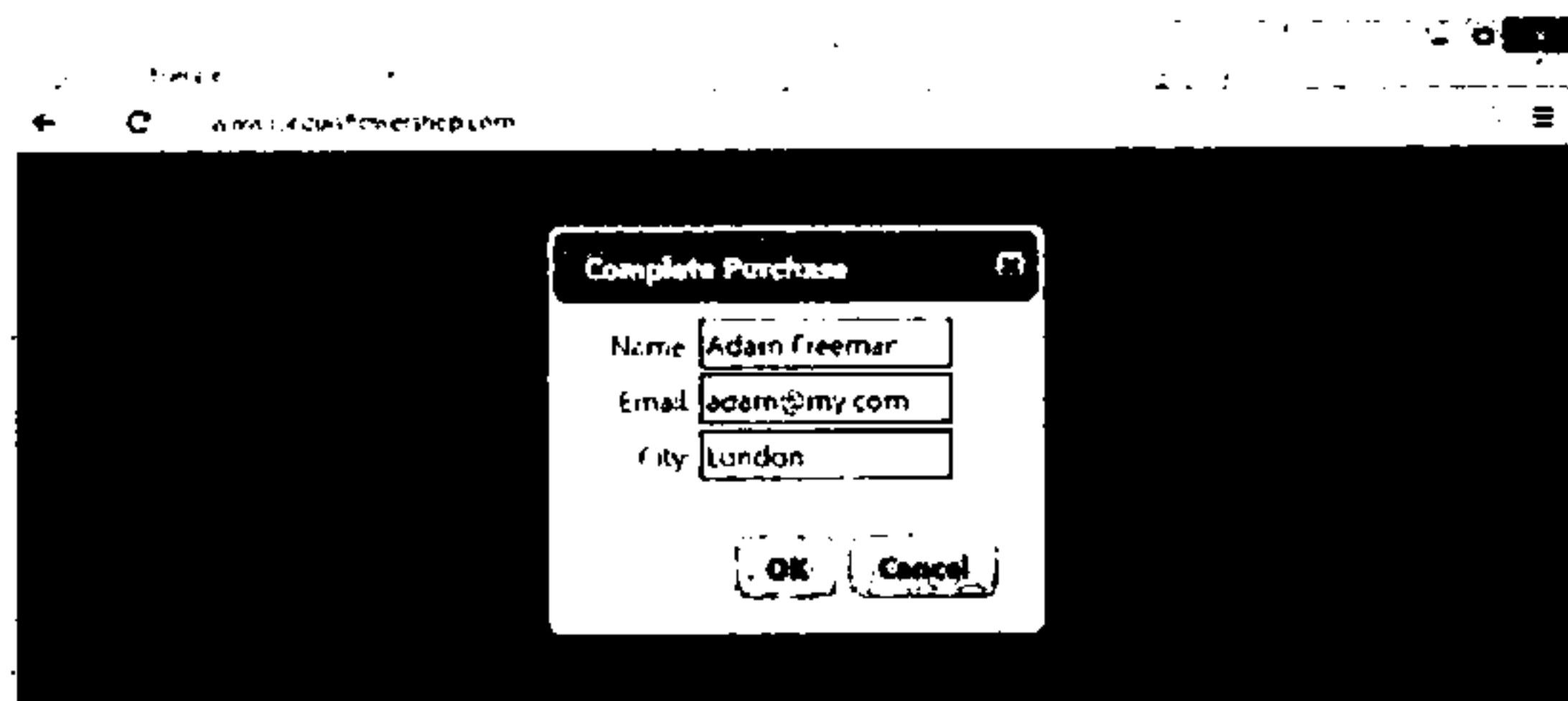


图25-15 显示收集信息对话框完成购买

最后，单击OK按钮，生成JSON字符串并重新初始化页面。最后我们得到的JSON字符串如下：

```
{"aster": "12", "daffodil": "7", "rose": "5", "peony": "2", "primula": "0", "snowdrop": "0",
  "first": "Adam Freeman", "email": "adam@my.com", "city": "London"}
```

如图26-16所示，我们现在又回到了起点，准备接受下一个订单。

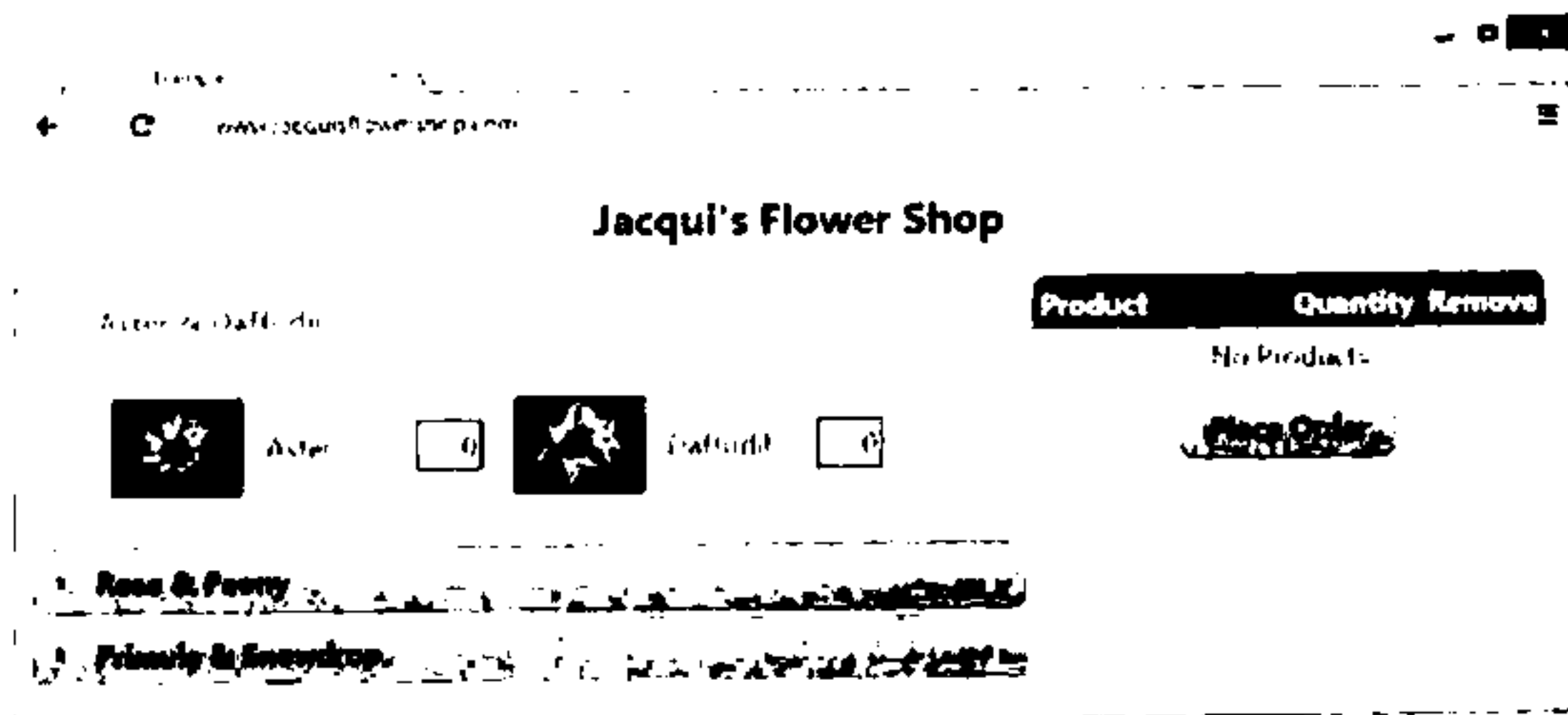


图26-16 重新初始化之后的页面

## 26.9 小结

本章重构了示例页面以整合jQuery UI特性。我新加了一些组件，如折叠菜单、对话框和按钮，也给出了未经jQuery UI装饰的最初效果。在这个效果的基础上，我们添加了jQuery UI样式框架所提供的一些class来改善其他元素的显示效果。我将在第35章详细介绍这些样式class。在第五部分，我们将把注意力转向jQuery Mobile，来创建丰富的移动应用。

本章，我会展示如何获取jQuery Mobile，以及怎么把jQuery Mobile添加到页面。我还会讲解jQuery Mobile别出心裁的组件创建方式，以及必须怎么做才能适应这种方式。对于Web应用开发者来说，唯一的挑战是怎么支持触摸设备。我还会讲解jQuery Mobile的一些核心特性，这些特性能帮助我们简化触摸设备的处理。在这一章，我还会提供一些移动Web应用开发和测试的指南。

表27-1 本章概要

问 题	解决方法	代码清单
如何把jQuery Mobile添加到页面	使用script元素导入jquery及jQuery Mobile库文件，并使用一个link元素导入jQuery Mobile对应的样式文件	1
如何创建一个jQuery Mobile页	使用data-role属性并把它值设置为page	2
如何禁用浏览器虚拟页	配置视口 (viewport)	3
如何延迟自定义脚本的执行，直到jQuery Mobile增强完当前页面	使用pageinit事件	4
如何简化触摸事件的处理	使用jQuery Mobile手势和虚拟鼠标事件	5~7
如何响应设备横屏竖屏的变化	处理orientationchange事件或者使用CSS媒体查询	8、9

**注意** 如我在第一章所说，jQuery Mobile的API在新发布的1.3版本里有许多变化。在后面的章节中，我会标注出这些变化。

## 27.1 安装 jQuery Mobile

首先，我们来了解如何获取并安装jQuery Mobile。jQuery Mobile构建在jQuery及jQuery UI基础上，因此它的一些常用模式与底层库是完全一致的。

### 27.1.1 下载jQuery Mobile

首先我们得搞到jQuery Mobile：浏览<http://jquerymobile.com>的下载页就可以下载到。编写本书这一部分内容之时，jQuery Mobile的版本是1.3.1，我们可以从下载页得到一个zip文件，当时的文件名是jquery.mobile-1.3.1.zip。

---

**提示** 类似于jQuery和jQuery UI, 我们也可以通过内容分发网络(CDN)加载jQuery Mobile。我在第5章讲过内容分发网络, 对于基于因特网的Web应用程序来说, CDN实在是一个极佳的解决方案(不过它很少适用于企业内部应用程序)。在jQuery Mobile的下载页上, 有使用CDN上的jQuery Mobile所需的各个链接。

---

### 1. 创建主题

jQuery Mobile支持主题, 它的主题有点儿像是jQuery UI主题的简化版。jQuery Mobile安装包中包含一个默认主题, 如果你需要创建一个更加个性的主题, 请访问<http://jquerymobile.com/themeroller>。使用ThemeRoller应用程序会生成一个包含着适用于你网站的CSS文件的zip文件。我会在第28章详细介绍如何使用ThemeRoller应用程序, 不过届时我会使用默认的主题, 而不是创建一个新主题, 部分原因在于jQuery Mobile版本的ThemeRoller尚不具备便利主题库。

### 2. 下载jQuery

我们也需要jQuery库。1.3.1版的jQuery Mobile声称只能与jQuery 1.7.0~1.9.1版本配合。jQuery Mobile的发布往往会滞后jQuery, 编写本章时, jQuery Mobile尚不支持jQuery 2.0。不过, 我们可以放心使用1.x版本, 而不会遇到任何问题。因此在这一部分的例子中, 我会使用jQuery 1.10.1版本。

---

**提示** 尽管jQuery Mobile构建在jQuery UI基础之上, 我们并不需要下载和安装jQuery UI库。jQuery Mobile需要的jQuery UI内容都已包含在jQuery Mobile库中了。

---

## 27.1.2 安装jQuery Mobile

我们需要从下载到的jQuery Mobile下载包中复制三个文件和一个目录到Web服务器:

- jquery.mobile-1.3.1.js文件(jQuery Mobile库文件);
- jquery.mobile-1.3.1.css文件(jQuery Mobile CSS样式文件);
- images目录(jQuery Mobile图标目录)。

当然也需要jQuery库。把这些文件复制到位之后, 我们就可以制作jQuery Mobile页面了。和前几章一样, 我把页面命名为example.html, 并保存到刚才复制库文件的目录中。这个文件的内容见代码清单27-1。

代码清单27-1 example.html的内容

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
</head>
<body>
  <div data-role="page">
```



```

    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is Jacqui's Flower Shop
      <p><button>Press Me</button></p>
    </div>
  </div>
</body>
</html>

```

代码中突出显示的代码是jQuery Mobile必需的内容。两个script元素负责导入jQuery和jQuery Mobile库，而link元素则负责导入jQuery Mobile所需要的样式文件。因为我的HTML文件与库文件和样式文件位于同一个目录，所以可以直接用文件名来引用。

**提示** 在这里暂时忽略页面文件的其他内容。稍后会详细解释这些元数据，以及body元素内的那些元素的使用法。

## 27.2 jQuery Mobile 工作原理

尽管jQuery Mobile基于jQuery UI，但是与jQuery UI仍然有不少显著的区别。在我们开始深入研究jQuery Mobile的强大功能之前，需要清楚这些不同之处，以便为后面几章的学习做好铺垫。

27

### jQuery Mobile支持级别

jQuery Mobile一共提供了三种支持级别以应对形形色色的移动浏览器。每种级别对应着一长串设备及浏览器清单。其中最高级别是A，可提供最佳的体验，这一级别实现了本书这一部分提到的所有功能。

与A级别相比，B级别除了不支持Ajax导航（详见第28章），其他的都支持。除了在不同页面之间切换不如A级别平滑之外，B级别提供的功能也是相当丰富的。

C级别则相当基础，它用来支持那些较老的设备。jQuery Mobile仅能为这些设备提供很少的功能。

好在绝大多数现代移动设备都符合A级别对设备的要求。在<http://jquerymobile.com/gbs>页面，你可以查看各级别支持的设备列表。

### 27.2.1 自动增强技术

jQuery Mobile最惊人的特性是组件不必明确地创建。当我们使用jQuery UI时，必要使用jQuery选中一个或多个元素，然后调用某个方法（比如button、tabs等）才能在页面中创建一个jQuery UI组件。看一下代码清单27-1，你会发现我并没有在页面中添加用来生成组件的script元素，仅有的script元素都是用来导入jQuery和jQuery Mobile库的。而且，如图27-1所示，我们的页面格式已经过美化处理。（这张截图里显示的设备是Opera手机仿真器，在本书这一部分我几乎一直使用这个仿真器。稍后我会详细介绍这个好用的工具。）



图27-1 示例页面

**注意** 书中这一部分的截图大部分取自Opera Mobile浏览器仿真器，利用仿真器的横屏分辨率，我得以在每一页中显示更多例子。

当我们使用script标签把jQuery Mobile库引入一个页面时，它会自动增强这个页面。首先jQuery Mobile会寻找那些有data-role属性的元素，根据这个属性的值决定对元素做什么处理。代码清单27-2在示例页面中突出显示了data-role属性。

**提示** data属性是指那些名字以data-开头的属性。data属性最初只是开发者在定义自定义属性时约定俗成的习惯，现在已成为HTML5官方标准的一部分。

#### 代码清单27-2 示例页面中的data-role属性

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
</head>
<body>
  <div data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is Jacqui's Flower Shop
      <p><button>Press Me</button></p>
    </div>
  </div>
</body>
</html>
```

在jQuery Mobile中还有一个与众不同的功能，那就是一个HTML页面文件中可以存放许多“页”（我会在第28章演示这一功能）。这些子页是jQuery Mobile的基本构件。在我们的示例页面里目前只有一个页，即data-role属性为page的div元素。由于这个页面实际上内嵌在HTML页面内，我们必须告诉jQuery Mobile一些额外的信息，以描述这个页的目的。因此，页面中还出现了其他两个data-role属性，它们分别告诉jQuery Mobile哪个元素是标题、哪个元素是内容。表27-2列出了页面中出现的这三种data-role属性及其含义。通过观察图27-1中的页面结构，我们能够轻松辨认出示例页面中三种data-role属性对应的元素。

表27-2 示例页面中data-role属性的值

属 性 值	含 义
page	告诉jQuery Mobile把该元素的内容视为一个页
header	告诉jQuery Mobile把该元素的内容视为当前页的标题
content	告诉jQuery Mobile把该元素的内容视为当前页的内容

**提示** jQuery Mobile会自动插入元素，包住代表页内容的元素。也就是说，页面中没有其他身份的元素都被视为内容元素，因而我们不必一定要明确定义内容元素。

我们不必明确通知jQuery Mobile去页面中寻找具有data-role属性的元素来生成页面，这一过程在页面加载时会自动完成。有些元素，比如button元素，会自动应用样式。（在后面的章节中我还会演示更多data-role属性的用法。）

**提示** 在创建一个移动应用程序的过程中，jQuery Mobile为减少自定义脚本代码不遗余力。事实上，完全不写脚本也可以实现一个简单的应用程序。这不是说我们可以在禁用JavaScript的情况下为浏览器开发jQuery Mobile应用程序，JavaScript Mobile是一个JavaScript库，它需要JavaScript支持才可以实施页面自动增强。

## 27.2.2 视口

有一个元素非常重要，尽管它并不是jQuery Mobile的一部分。代码清单27-3突出显示了这个重要的元素。

代码清单27-3 使用meta元素配置视口

```
...
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
</head>
...
```

在上面的代码中，我高亮显示了name属性为viewport的meta元素。为了更好地兼容那些仅为桌面浏览器设计的网页，许多移动浏览器使用虚拟页显示页面内容。在大多数情况下，这个点子不错，用户能看到完整的页面结构，尽管每一部分都因为太小而看不清。图27-2展示的是jQuery Mobile官方网站首页的初始样子以及为了方便阅读放大之后的样子。

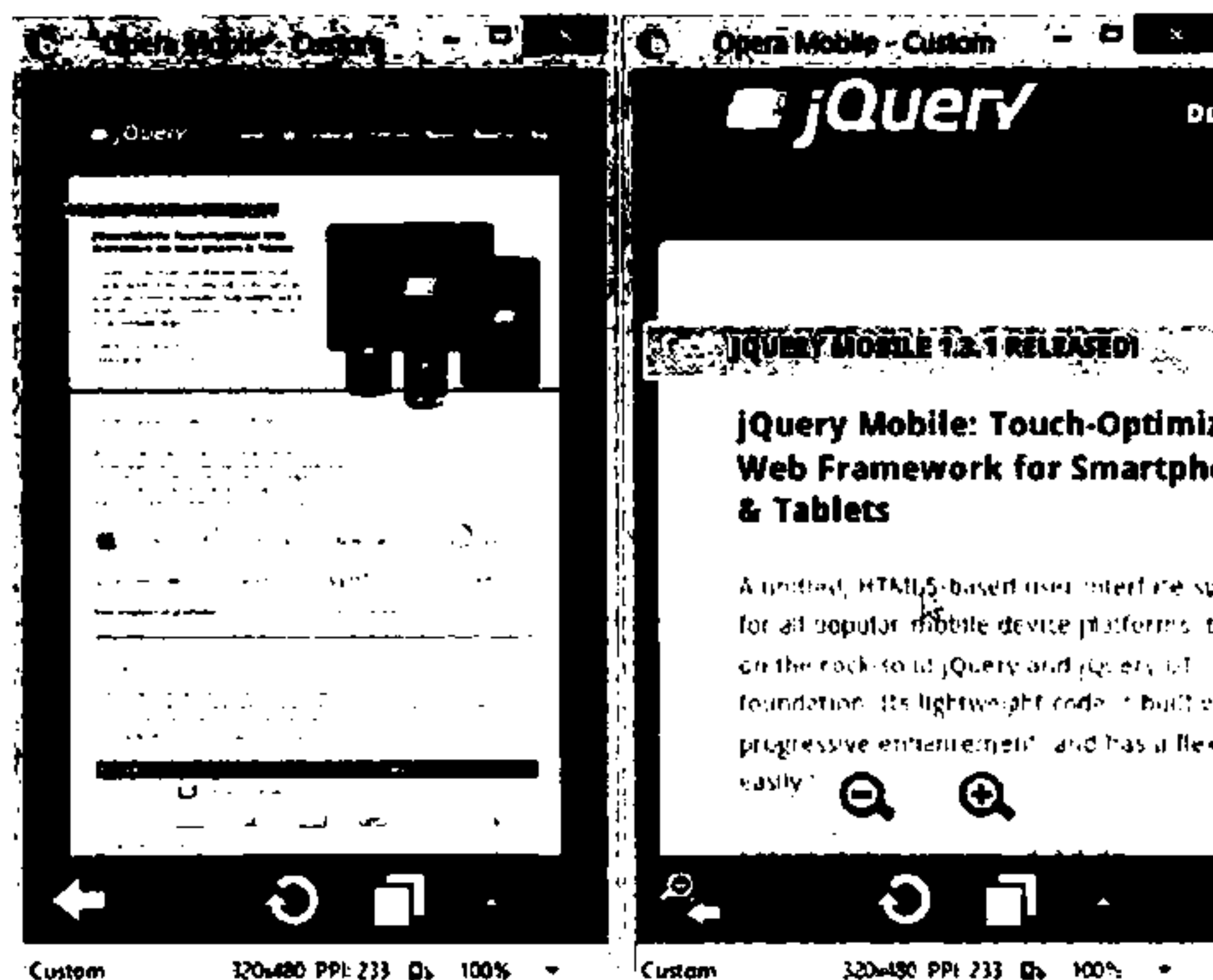


图27-2 移动浏览器的虚拟页

图27-2中的第一幅图显示的是竖屏情况下jQuery Mobile网站的样子（它加剧了看不清的效果）。显然文本太小，以至于无法阅读。不过如第二幅图所示，移动浏览器支持放大页面局部的功能。没错，虚拟页是一个折中的方案，然而在专为移动设备开发的网站相当少的现实情况下，这个方案相当合理。

问题是虚拟页方案不够智能，jQuery Mobile网页在这个方案下有问题。图27-3展示了使用虚拟页显示示例页面的效果。

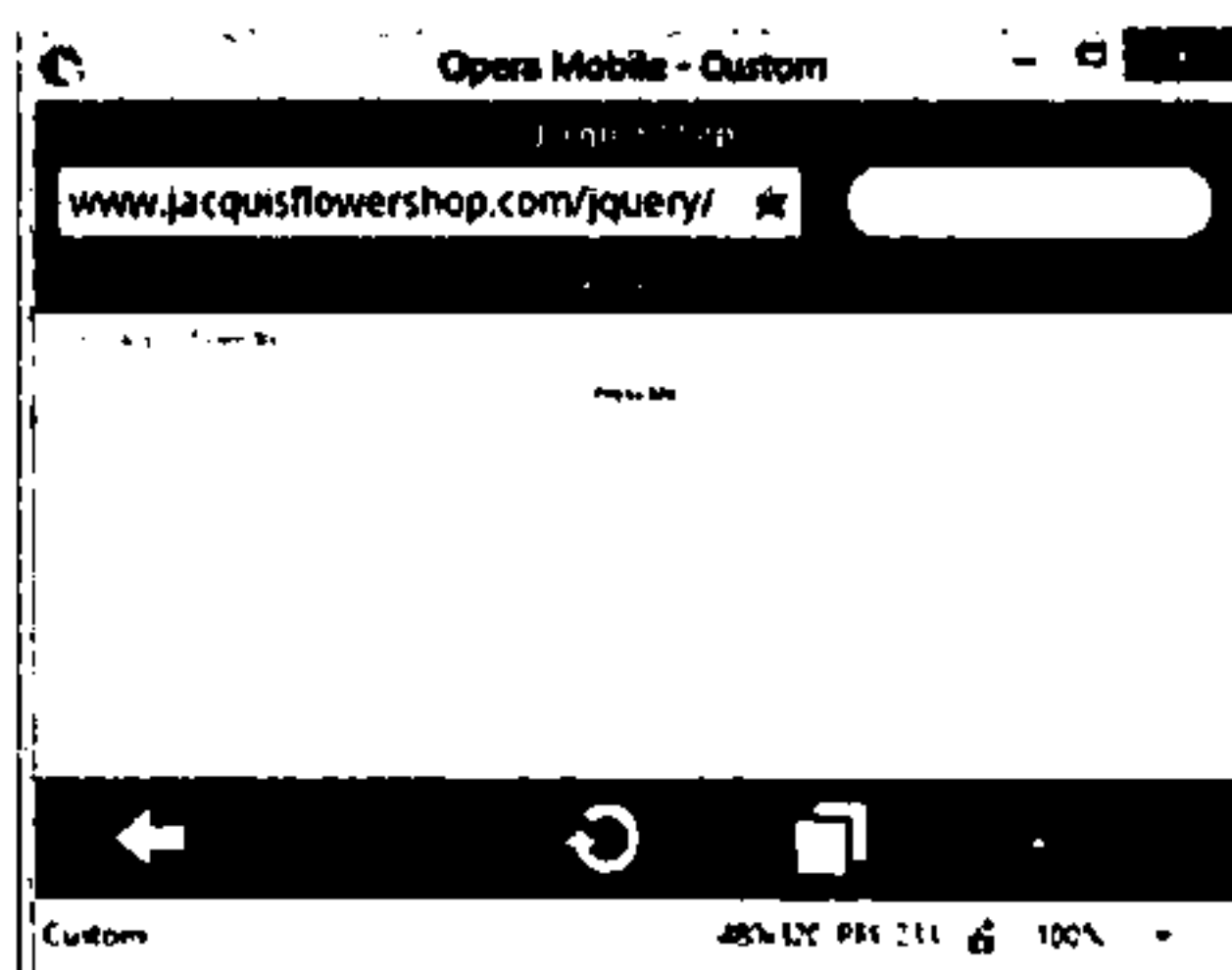


图27-3 示例页面显示在一个宽的虚拟页之中

如图27-3所示，jQuery Mobile元素显示得如此之小，以至于无法使用。在示例页面中，我们使用meta元素告诉浏览器页面宽度应该使用屏幕宽度，这样在移动浏览器上jQuery Mobile页面的显示效果就好得多了。

### 27.2.3 jQuery Mobile事件

说起jQuery Mobile事件，有两个重要部分，下面我们来详细了解一下。

#### 1. jQuery Mobile页事件

在页的生命周期内，jQuery Mobile定义了许多事件。其中最重要的一个就是pageinit事件。在jQuery的ready事件（参见本书前面的内容）发生时，jQuery Mobile会自动增强页面。如果我们希望在页面中加入自定义脚本，就得小心别让自己的代码执行得过早（要等jQuery Mobile处理完页面之后再执行）。这意味着不得不等待pageinit事件，jQuery Mobile处理完页面之后会主动触发这个事件。如代码清单27-4所示，它不像ready事件那样有快捷方法可用，我们只能使用bind方法为pageinit事件绑定处理函数。

代码清单27-4 使用pageinit 事件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript">
    $(document).bind("pageinit", function () {
      $("button").click(function() {
        console.log("Button pressed")
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
</head>
<body>
  <div data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is Jacqui's Flower Shop
      <p><button>Press Me</button></p>
    </div>
  </div>
</body>
</html>
```

bind方法的第一个参数是事件名，第二个参数是事件处理函数。当我们选中的元素发生指定事件时，脚本会自动调用处理函数。

在这个例子中，我使用bind方法为pageinit事件绑定了一个事件处理函数。我在这个函数中放上了jQuery Mobile页面初始化之后我希望执行的代码。这个例子使用jQuery在页面选中button元素，给它绑定了click事件处理函数。这种事儿咱们在这本书里已经做得很多了，可以说是轻车熟路。

**提示** 注意一点，我是在导入jQuery Mobile库的script元素之前新加了一个script元素（存放我写的代码）。对于pageinit事件来说，这并非必要。不过，对于mobileinit事件（我们通常使用它修改jQuery Mobile库的一些设置），这是必需的。（我会在第28章演示mobileinit事件的使用法。）我认为把自定义脚本放在jQuery Mobile库之前导入是个好主意，即使仅仅是为了绑定pageinit事件。

## 2. 触摸事件

在浏览器中针对触摸事件有一个标准，但是因为触摸交互模型如此之多，这个标准相当底层。表27-3列出了这些低级别的触摸事件。

表27-3 标准触摸事件

事 件	描 述
touchstart	当用户触摸屏幕时触发。对于多点触控设备来说，每个手指每触摸屏幕一次都会触发这个事件
touchend	当用户手指离开屏幕时触发
touchmove	当用户手指不离开屏幕并移动时触发
touchcancel	当用户的触摸序列中断时触发。不同的设备对此事件的解释并不一致，不过一个典型的例子是用户手指滑到屏幕边缘时触发此事件

解释这些事件，弄清这些事件含义的任务落到了我们开发者身上。这是一个痛苦的任务，解决过程中错误百出。我建议只要可能就尽量避开这种任务。稍后，我会详细阐述这个问题，这是jQuery Mobile能够大显身手的领域。

**提示** 如果你希望深入细致地了解touch事件，可以在[www.w3.org/TR/touch-events](http://www.w3.org/TR/touch-events)查看事件的标准。标准中有事件的完整描述，以及事件对象的所有属性（用来了解每个touch交互细节）。

绝大多数Web站点在设计时根本没有考虑触摸事件。为了支持各种各样的用户脚本，移动浏览器会自动为一些触摸事件额外生成相应的鼠标事件。也就是说，浏览器先触发触摸事件，然后再伪造一个表示同样行为的鼠标事件，就像真的有一个鼠标在触发事件一样。代码清单27-5中的脚本完整地演示了这一幕的具体细节。

### 代码清单27-5 监控触摸事件以及伪造的鼠标事件

```
<!DOCTYPE html>
<html>
<head>
  <title>Event Test</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <style type="text/css">
    table {border-collapse: collapse; border: medium solid black; padding: 4px}
    #placeholder {text-align: center}
    #countContainer * {display: inline; width:50px}
    th {width: 100px}
```

```

</style>
<script type="text/javascript">
    $(document).bind("pageinit", function() {
        var eventList = [
            "mousedown", "mouseup", "click", "mouseleave",
            "touchstart", "touchend", "touchmove", "touchcancel"]
        for (var i = 0; i < eventList.length; i++) {
            $("#pressme").bind(eventList[i], handleEvent)
        }
    });
    $("#reset").bind("tap", function() {
        $("tbody").children().remove();
        $("#placeholder").show();
        startTime = 0;
    });
    startTime = 0;
    function handleEvent(ev) {
        var timeDiff = startTime == 0 ? 0 : (ev.timeStamp - startTime);
        if (startTime == 0) {
            startTime = ev.timeStamp
        }
        $("#placeholder").hide();
        $("<tr><td>" + ev.type + "</td><td>" + timeDiff + "</td></tr>")
            .appendTo("tbody");
    }
</script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
    <div data-role="page">
        <div data-role="content">
            <div id="tcontainer" class="ui-grid-a">
                <div class="ui-block-a">
                    <button id="pressme">Press Me</button>
                    <button id="reset">Reset</button>
                </div>
                <div class="ui-block-b">
                    <table border=1>
                        <thead>
                            <tr><th>Event</th><th>Time</th></tr>
                            <tr id="placeholder"><td colspan=2>No Events</td></tr>
                        </thead>
                        <tbody></tbody>
                    </table>
                </div>
            </div>
        </div>
    </div>
</body>
</html>

```

在示例页面中有两个按钮和一个表格，Press Me按钮已经就绪。当我们单击这个按钮，一组鼠标事件和触摸事件就会显示在表格中。每一事件会显示事件类型，以及事件发生时间与第一个事件发生



时间的时间差。Reset按钮用来清空表格中的内容并重设时间计数器。这个例子的执行结果见图27-4。

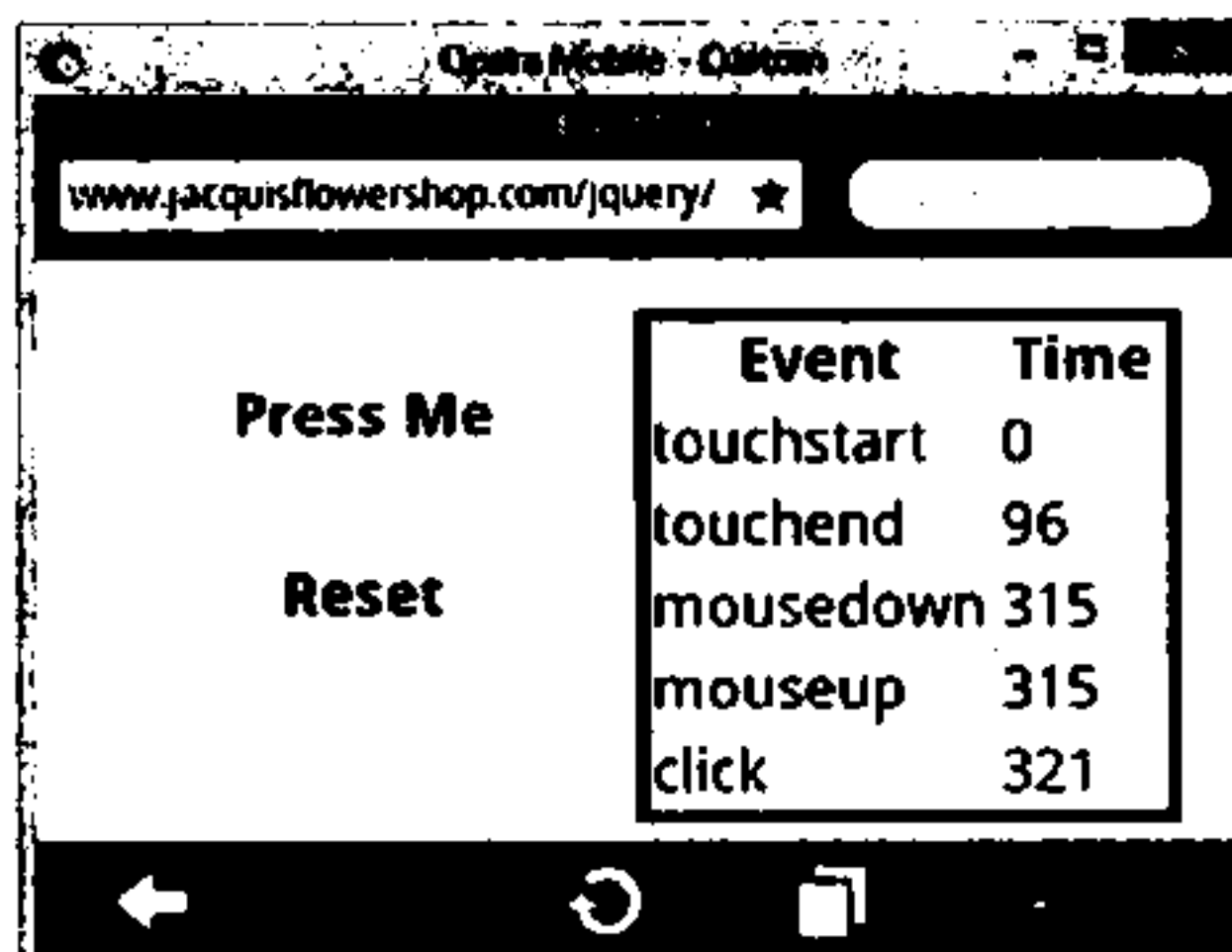


图27-4 观察触摸事件和鼠标事件的发生顺序

表27-4 展示了当我在Opera Mobile浏览器中单击按钮时触发的事件序列及相应的时间差。

表27-4 采自Opera Mobile浏览器的事件序列

事 件	时间差
touchstart	0
touchend	96
mousedown	315
mouseup	315
click	321

我们注意到，首先触发的是touchstart和touchend事件，这两个事件分别发生于触摸和手指移开屏幕时。浏览器还自动地先后生成了mousedown、mouseup和click事件。注意，在touchend和mousedown事件之间有一段相当长的时间延迟，几乎有300 ms。这段延时实在是太长了，对于那些依赖这些伪造事件的Web应用来说，由于页面的反应大大滞后于用户触摸屏幕的时间，这很容易造成问题。不是所有的浏览器都有这个毛病，但这个问题确实常见。为了避免发生这种情况，我建议你测试一下自己开发项目时面对的目标浏览器。

### 3. jQuery Mobile手势方法

为了使触摸事件更容易使用，jQuery Mobile做了两件事。第一件是定义了一系列手势事件，每一个手势对应着一个特定的底层事件序列。也就是说，我们用不着自己分析这些事件序列的含义。表27-5列出了这些手势事件。

表27-5 标准触摸事件

事 件	描 述
tap	当用户在很短的时间内单击屏幕并迅速离开时触发
taphold	用户用手指压住屏幕并保持约1 s，然后离开屏幕时触发
swipe	在1 s内用户水平拖曳至少30 px且垂直方向变动不超过20 px时触发
swipeleft	在1 s内用户由右向左水平拖曳至少30 px且垂直方向变动不超过20 px时触发
swiperight	在1 s内用户由左向右水平拖曳至少30 px且垂直方向变动不超过20 px时触发

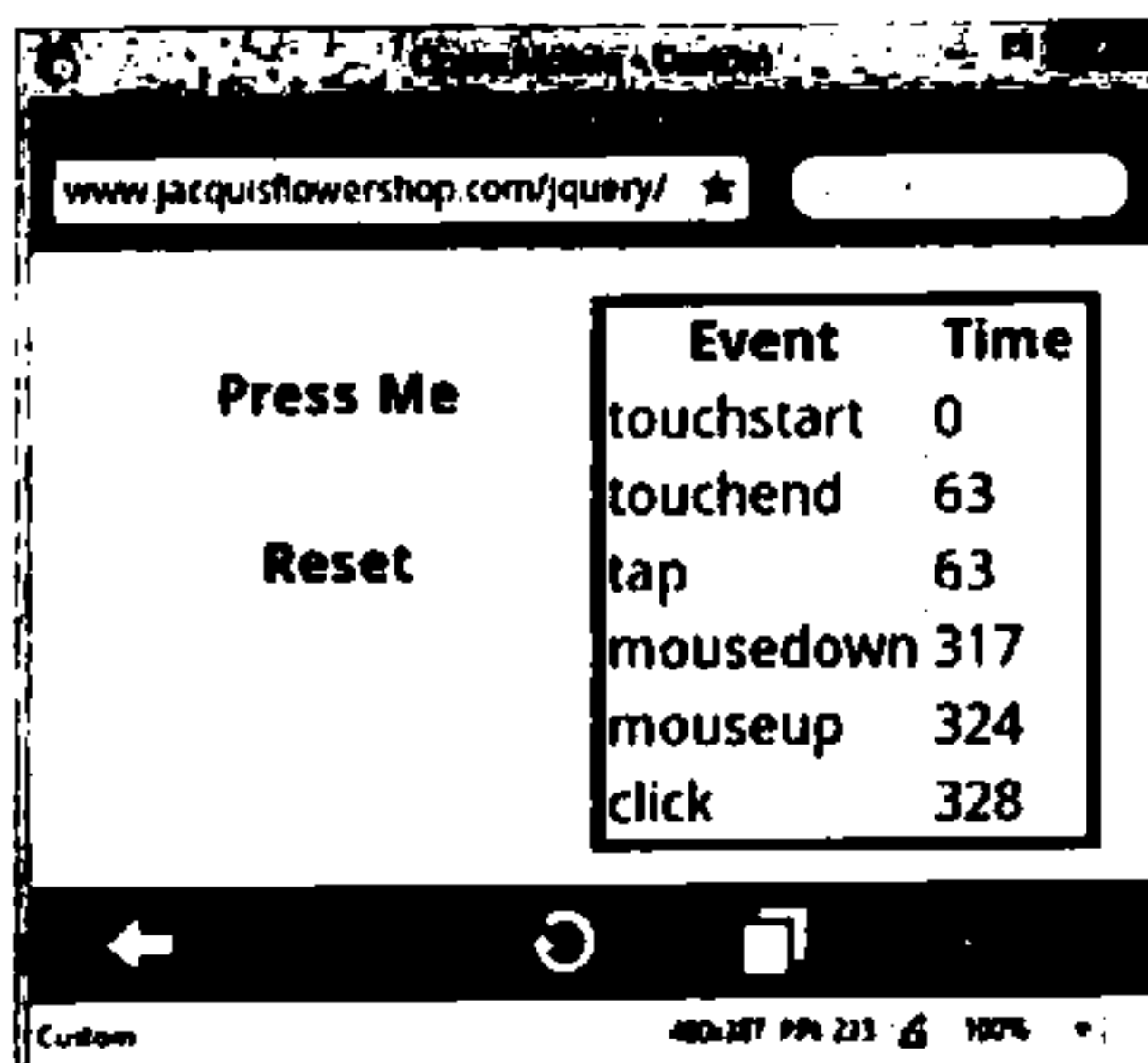


利用这些事件，我们能够轻松处理常见手势。代码清单27-6在事件计时的例子中添加了这些事件。

代码清单27-6 在事件计时的例子中添加jQuery Mobile手势事件

```
...
<script type="text/javascript">
  $(document).bind("pageinit", function() {
    var eventList = [
      "mousedown", "mouseup", "click", "mouseleave",
      "touchstart", "touchend", "touchmove", "touchcancel",
      "tap", "taphold", "swipe", "swipeleft", "swiperight"]
    for (var i = 0; i < eventList.length; i++) {
      $("#pressme").bind(eventList[i], handleEvent)
    }
    $("#reset").bind("tap", function() {
      $("tbody").children().remove();
      $("#placeholder").show();
      startTime = 0;
    });
  });
  startTime = 0;
  function handleEvent(ev) {
    var timeDiff = startTime == 0 ? 0 : (ev.timeStamp - startTime);
    if (startTime == 0) {
      startTime = ev.timeStamp
    }
    $("#placeholder").hide();
    $("<tr><td>" + ev.type + "</td><td>" + timeDiff + "</td></tr>")
      .appendTo("tbody");
  }
</script>
...
```

图27-5展示了单击按钮时发生的事情



Event	Time
touchstart	0
touchend	63
tap	63
mousedown	317
mouseup	324
click	328

图27-5 在事件计时的例子中添加jQuery Mobile手势事件

表27-6以一种更容易阅读的方式展示了事件序列。由于我的动作是单击按钮，因此只触发了tap手势事件。值得注意的是，tap事件反应相当快，我的手指刚刚离开屏幕2 ms，就触发了tap事件。

表27-6 采自Opera Mobile浏览器的事件序列

事 件	时间差
touchstart	0
touchend	63
tap	63
mousedown	317
mouseup	324
click	328

令人称赞的是，在不支持触摸事件的浏览器上或者非触摸设备上，jQuery Mobile同样会触发手势事件。图27-6展示了示例在桌面版Google Chrome浏览器中的运行情况。

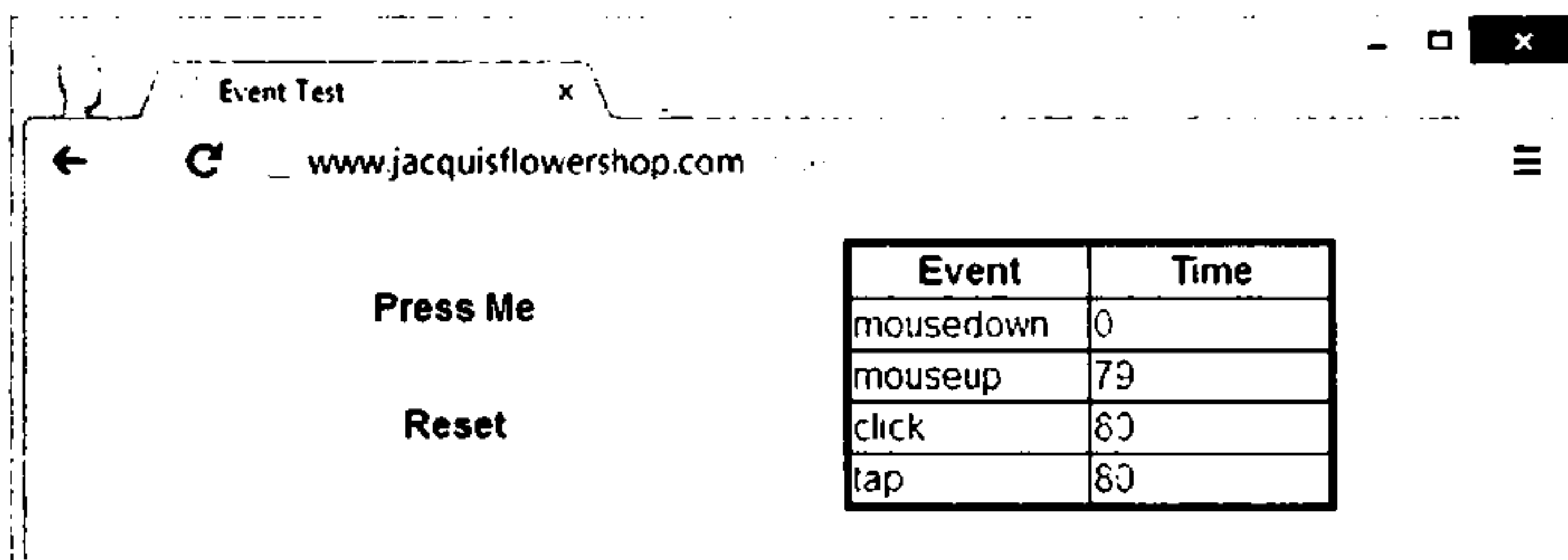


图27-6 桌面浏览器中发生的事件序列

表27-7以更清晰的方式展示了发生的事件及事件发生的相对时间。

表27-7 采自Google Chrome浏览器的事件序列

事 件	时间差
mousedown	0
mouseup	79
click	80
tap	80

也许和你想的差不多，这个事件序列里没有touchstart和touchend方法，而且事件发生的顺序也有所不同（因为桌面浏览器中鼠标单击事件是真实发生而非伪造出来的）。即便如此，tap事件仍会在click事件发生之后立即发生。

**提示** 我在开发移动Web应用程序时不使用click事件，而是使用tap事件，这是因为它既能避免伪造事件延时过长的问题，又支持非触摸平台。

#### 4. jQuery Mobile虚拟鼠标事件

由于浏览器并没有义务伪造鼠标事件，我们的Web应用程序要想同时支持触摸设备和非触摸设备，就必须同时监听鼠标事件和触摸事件。移动浏览器确实会伪造事件，这样每个交互我们都能捕获到触

摸事件和鼠标事件。为了简化这一处理，jQuery Mobile定义了一套虚拟鼠标事件，当绑定这些事件时，jQuery Mobile会小心地删除那些重复发生的事件，并保证那些需要触发的事件一定会发生（不管你的设备是否支持触摸）。这些虚拟事件见表27-8。

表27-8 虚拟鼠标事件

事 件	描 述
vmouseover	mouseover事件发生时触发（由于用户的手指在离开屏幕后无法与屏幕保持通信，这个事件没有对应的触摸事件）
vmousedown	touchstart或mousedown事件发生时触发
vmousemove	touchmove或mousemove事件发生时触发
mouseup	touchend或mouseup事件发生时触发
vclick	click事件发生时触发
vmousecancel	touchcancel或mousecancel事件发生时触发

这种事件生成方法，即使在触摸设备上，也生成类似于鼠标的事件序列。如代码清单27-7所示，为了进一步解释这句话的意思，我在事件计时的例子里添加了一些虚拟鼠标事件。

代码清单27-7 在事件计时的例子中添加jQuery Mobile虚拟事件

```
...
<script type="text/javascript">
    $(document).bind("pageinit", function() {
        var eventList = [
            "mousedown", "mouseup", "click", "mousecancel",
            "touchstart", "touchend", "touchmove", "touchcancel",
            "tap", "taphold", "swipe", "swipeleft", "swiperight",
            "vmouseover", "vmousedown", "vmouseup", "vclick", "vmousecancel"]
        for (var i = 0; i < eventList.length; i++) {
            $("#pressme").bind(eventList[i], handleEvent)
        }
    });
    $("#reset").bind("tap", function() {
        $("tbody").children().remove();
        $("#placeholder").show();
        startTime = 0;
    });
    startTime = 0;
    function handleEvent(ev) {
        var timeDiff = startTime == 0 ? 0 : (ev.timeStamp - startTime);
        if (startTime == 0) {
            startTime = ev.timeStamp
        }
        $("#placeholder").hide();
        $("<tr><td>" + ev.type + "</td><td>" + timeDiff + "</td></tr>")
            .appendTo("tbody");
    }
</script>
...
```

当我触摸屏幕时，jQuery Mobile生成了vmouseover和vmousedown事件。在一个纯触摸设备上，这

么做并没有什么意义。如果我们希望应用程序能够跨平台（也支持非触摸设备），就很可能需要在鼠标悬停在某个元素上时执行一些事件。当真正的touchstart事件发生时，jQuery Mobile伪造出来的vmouseover事件也会触发，这样我们就能够在触摸设备上得到一致的行为。（尽管触摸设备不支持mouseover事件。）图27-7展示了这个例子的结果。

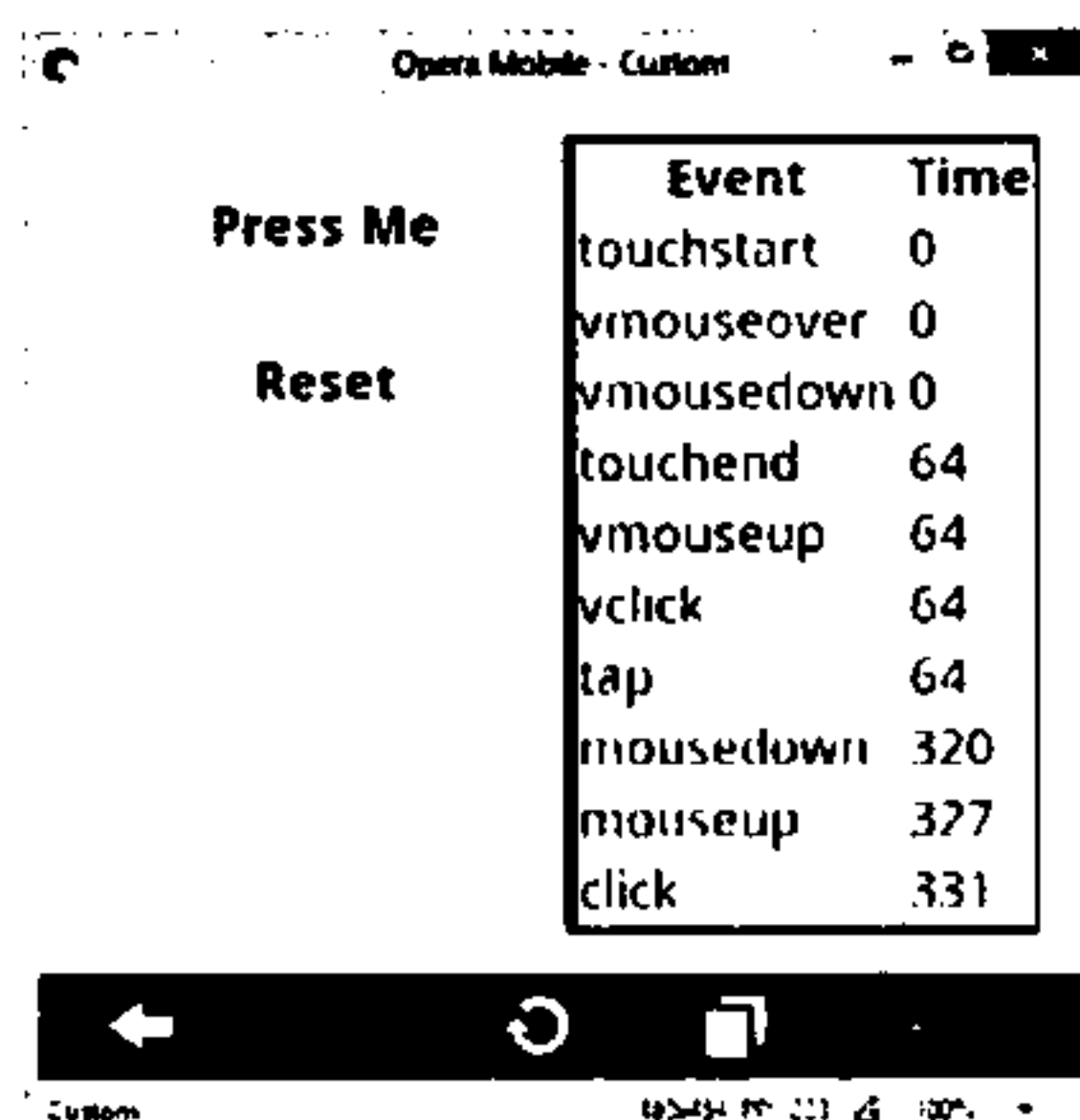


图27-7 在事件计时的例子中添加jQuery Mobile虚拟事件

表27-9以更清晰的方式展示了发生的事件及事件发生的相对时间。尽管从这个表格中看vclick事件的触发时间远早于伪造的click事件，但事实并非总是如此。因此，我并不建议大家使用vclick事件代替click事件以解决事件滞后问题。

表27-9 采自Google Chrome浏览器的事件序列

事 件	相对时间
touchstart	0
vmouseover	0
vmousedown	0
touchend	64
vmouseup	64
vclick	64
tap	64
mousedown	320
mouseup	327
click	331

**警告** 千万不要假定真实事件与虚拟事件总是交替发生。在非触摸设备上，这个事件序列可能发生变化。（在不同的平台上）虚拟事件之间的相对顺序保持不变，而中间插入的真实事件有所不同

### 27.2.4 响应设备手持方向的变化

绝大多数移动浏览器支持orientationchange事件，这个事件在设备手持方向旋转90°时发生。为

为了给开发工作提供方便，jQuery Mobile会在不支持此事件的浏览器上伪造这一事件。这是通过监视窗口大小的变化以及新高度与新宽度的比例来实现的。代码清单27-8展示了这一事件的用法。

代码清单27-8 响应设备手持方向的变化

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript">
    $(document).bind("pageinit", function() {
      $(window).bind("orientationchange", function(e) {
        $("#status").text(e.orientation)
      });
      $("#status").text(jQuery.event.special.orientationchange.orientation());
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <p>Device orientation is: <b><span id=status></span></b></p>
    </div>
  </div>
</body>
</html>
```

为了绑定orientationchange事件处理函数，我们必须选中window对象。在这个例子中，我采用改变span元素文本的方式指示新的方向。利用传递给事件处理函数的Event对象的orientation属性，我们能够获知设备现在是横屏还是竖屏。

jQuery Mobile也提供了一个方法判断当前屏幕的方向，即：

```
...
jQuery.event.special.orientationchange.orientation()
...
```

由于在页面初始化时并不会触发orientationchange事件（这个事件仅在页面初始化之后的设备方向改变时触发），我只好在例子中把span元素的值设置为这个方法的返回值。

如果你在测试这个例子时没有真实设备，可以使用本章后面提到的某个仿真器。对于几乎所有的仿真器，我们都能通过敲击某个键或者单击某个按钮来触发其模拟一次旋转。如图27-8所示，对于我使用的Opera Mobile仿真器来说，按下Ctrl+Alt+R就会触发旋转效果。

对于那些不支持方向变化的浏览器（如桌面浏览器），当改变窗口大小时，我们可以利用jQuery Mobile伪造的事件营造类似的效果。这种情况下，jQuery Mobile通过窗口的宽高比来判断设备方向。

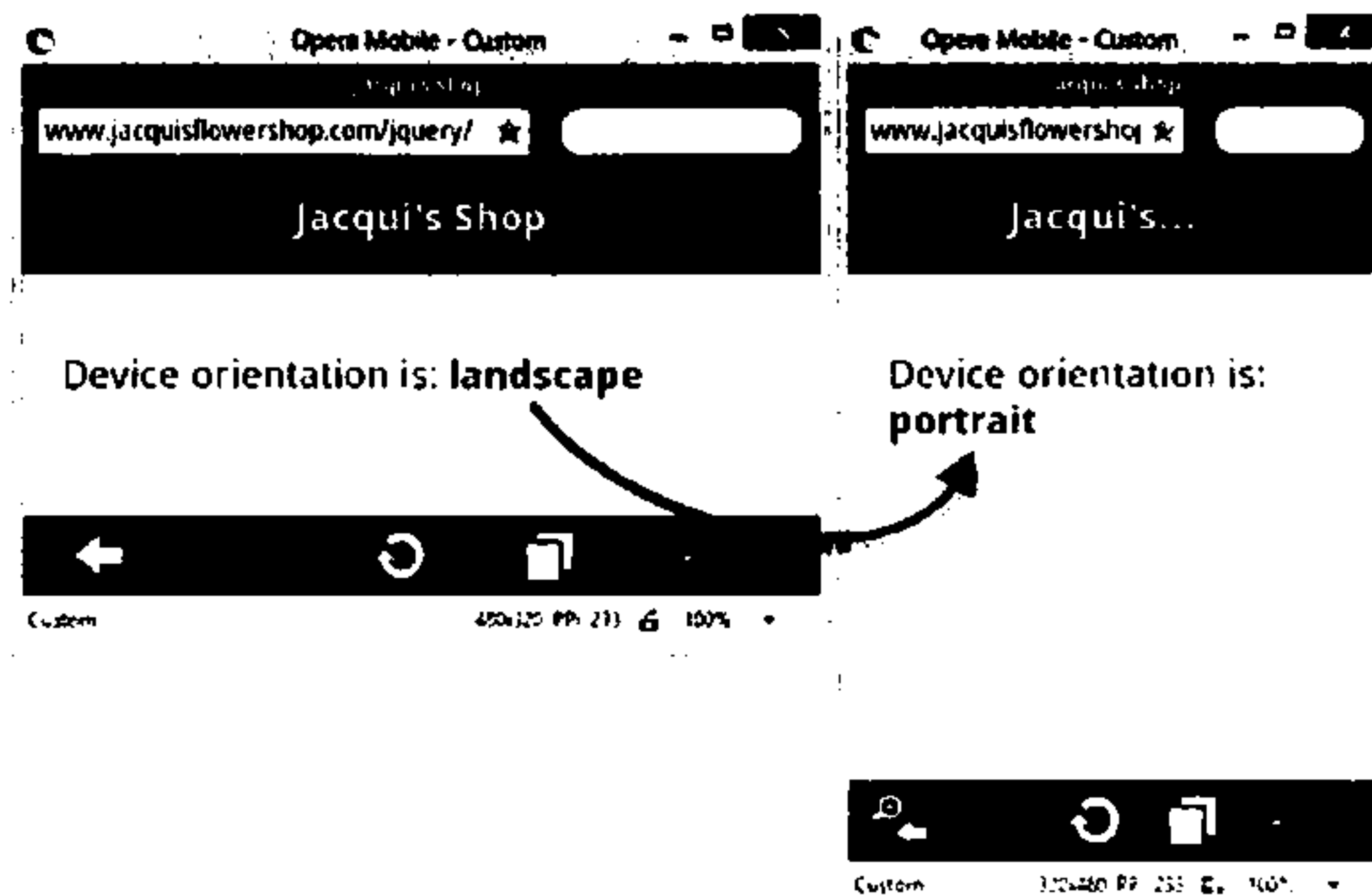


图27-8 响应设备手持方向的变化

### 利用媒体查询管理设备朝向

orientationchange事件允许我们在设备朝向发生变化时，利用JavaScript做一些事情。还有一种方法，即利用CSS的媒体查询功能为设备的不同朝向应用不同的样式。代码清单27-9演示了这种做法。

#### 代码清单27-9 利用CSS媒体查询功能响应设备朝向变化

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    @media screen and (orientation:portrait) {
      #pstatus {display: none}
      #lstatus {display: inline}
    }

    @media screen and (orientation:landscape) {
      #pstatus {display: inline}
      #lstatus {display: none}
    }
  </style>
</head>
<body>
  <div data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <p>Device orientation is:
        <span id=pstatus><b>Portrait</b></span>
      </p>
    </div>
  </div>
</body>
</html>
```

```

        <span id=lstatus><b>Landscape</b></span>
      </p>
    </div>
  </div>
</body>
</html>

```

我们能利用CSS媒体查询功能为特定的环境定义特定的样式,比如本例中设备朝向的横向或竖向。利用CSS display属性来控制元素的显示或隐藏,我们可以达到与前面使用JavaScript的例子一样的效果。针对设备朝向的媒体查询功能,不论是桌面浏览器,还是移动浏览器都工作得非常好。在这种情况下,无需使用任何伪造事件就能达成目标。

## 27.3 处理移动设备

与普通Web开发相比,为移动设备开发应用程序有一些特别之处。在接下来的几节当中,我来做一个“向导”,先介绍移动开发所需的基础知识,再带领大家开始实践移动开发,并重点讲解那些移动开发中经常遇到的关键问题。

### 甄别移动设备

如果我们打算让应用同时支持桌面用户和移动用户,就可能需要为不同的用户群提供不同的界面。一种常见的做法是为桌面浏览器用户提供jQuery UI界面,为移动设备用户提供jQuery Mobile界面。

难点在于识别移动设备上运行着的是何种浏览器。有许多识别移动浏览器的技术,它们都是在服务器端识别浏览器,然后根据浏览器重定向到更合适的页面。由于这已经超出了本书的讲述范围,因此我就不在这里详细讲解这些技术了。如果你是初次接触这类问题,建议访问<http://wurfl.sourceforge.net>,这个网站提供了一个有用的服务器端组件,可以识别出绝大多数移动设备。也可访问<https://github.com/kaimallea/isMobile>,获取客户端解决方案。

如果识别出用户正在使用移动设备,我建议自动强制用户访问应用程序的移动版本。与桌面版本相比,由于应用程序的移动版本经常缺少一些功能,因此有的用户更喜欢使用应用程序的桌面版本,即使是在移动设备上。我建议当用户访问你的站点时,如果检测到用户使用的是移动设备,就让用户做一道简单的选择题。除此之外,即使用户作出了自己的选择,我们也要为用户切换应用程序的版本提供方便。

### 27.3.1 在移动开发时应该避免的两大“陷阱”

在移动开发过程中要避开两个“陷阱”:错误的假设和不切实际的仿真。我会对这两个问题进行解释,并提供一些学习移动开发所需的背景知识。这有助于你在实际开发中避免一些常见的错误。



## 27.3.2 避免错误的假设

移动设备市场极其活跃，远未成熟，而且缺乏标准。在为桌面浏览器开发Web应用程序时，我们常常会预设一些前提条件（一般不会明确声明）。通常我们会假定最小的屏幕分辨率、支持JavaScript、安装了某个插件，以及用户会用鼠标单击，用键盘输入。

其实这些假定未必合理。举例来说，如果我们假设JavaScript未被禁用，那就排除掉了那些禁用了JavaScript的客户。你也许会断定这是一个合理的折中，毕竟绝大多数用户不会禁用JavaScript，因此便放弃了那些不符合“标准”的用户。

由于移动市场如此分散，因此在移动设备市场上境况更坏。桌面环境虽多，不外乎Mac、Windows个人计算机和Linux这几个。移动设备可就复杂得多了，针对屏幕大小、网络连接和输入方法所做的任何假定都会舍弃相当市场份额的移动用户。

### 1. 移动设备不等于iPhone

我见过的最坏的假设（而且经常见到）是只瞄准iPhone市场。iPhone确实取得了很大的成功，但它远不是移动设备市场的全部，而且即使是iPhone也有型号差异。一种常见的屏幕分辨率是320×480，较老的iPhone使用这种分辨率。确实有很多设备使用这种分辨率，不过越来越多的设备开始采用其他分辨率。如果你的移动应用使用固定的分辨率，那你不但抛弃了那些使用更小分辨率设备的用户，而且会使拥有更高分辨率设备的用户恼火。

### 2. 移动设备不等于手机

另一种常见的假设是只瞄准手机市场，从而忽略了广阔的平板电脑市场。平板电脑不仅拥有更大的屏幕分辨率，其用户手持设备的方式与使用习惯也和手机不同。如果你不明白我的意思，那就随便找一个咖啡厅，仔细观察里面的顾客。我的观点（不一定科学但很坚定）是：平板电脑较大的尺寸决定了它们的手持不那么方便，因此更多的是被靠着什么东西放在某个地方。这意味着它们不是那么稳定，当用户手指在屏幕上移动时，会导致平板轻微地晃动（从而导致触控精确度问题），并且会遮住大块的屏幕（用户在操作时手和胳膊会位于平板电脑的上方）。

我认为移动设备的移动使用特性决定了如何使用才合理，以及如何交互才方便、合理。解决这个问题的最好方式是观察人们如何与各种设备交互。如果你的时间和钱都比较富裕，最好设一间可用性实验室。不过即使你时间紧张并有预算压力，去星巴克呆上一个下午，也能收获一堆珍贵的见解。

### 3. 移动设备不等于触控设备

不是所有的移动设备都拥有触控屏幕。一些设备依赖于集成在键盘上的迷你鼠标，还有一些拥有各种各样的输入方法。我有一个测试设备是用一个小笔记本改装而成的平板电脑。它既有触控屏幕，也有全功能的键盘和鼠标。用户总是使用自己能用并且最方便的输入方法，假定用户拥有某种输入设备只会让没有这种设备的用户失望（所以除非为了测试，我很少使用笔记本/平板二合一设备）。

### 4. 移动设备的流量不但有限而且收费

（移动）网络的使用费按周期计算，相对固定，与用户使用的网络及活跃程度相关。现在网络提供商正努力建设，以提供足够的网络能力来满足人们的需要，尤其是那些人口稠密的地区。最终，网络的使用费会降低，用户带宽会增加。但是，现在网络提供商不但对数据访问额外收费，而且限制用户的每月下载总量。

假设用户愿意为我们的应用程序贡献大量的流量是很危险的。用户不像你期望的那样关心你的应



用程序，这是铁律。这听起来确实不太舒服，但几乎总是正确的。你百分百关心自己开发的应用程序，这是理所当然的事，然而用户对这个程序的关心程度可能只是你的几分之一。

在第28章，我会演示：jQuery Mobile可在用户需要某些内容之前预先下载这些内容。这个功能确实很好，不过我们要慎用。因为它假定用户愿意将流量消耗在他们可能永远不需要的数据上。同样要慎用的功能还有自动并且频繁的数据更新。尽量少用，慎重使用，仅当用户明确表示同意时，我们的应用程序才可以大量消耗用户的数据流量。

同样，我们不应假定用户设备的下载速度。请慎重考虑那些体积较大的资源，比如图片和视频。有些用户能够快速下载这些内容，但更多人不能。根据我的经验，如果你提供低流量选项，那么这个选项总是受人欢迎的。

我们还应该时刻准备应对网络不可用的情况。我有一阵子经常乘坐火车，当火车钻进隧道时，网络随时都会中断。一个优秀的Web应用程序会事先考虑这种情况，友好地告诉用户发生了什么情况，并且能在用户网络恢复之后优雅地恢复工作。遗憾的是，优秀的Web应用程序太少了。

**提示** 我曾为一家全球集装箱运输公司工作过一段时间，严苛的要求促使他们创造出一些无比健壮且适应能力极其强大的应用程序。几乎在世界上的每个港口都有他们的运输代理商，在一些偏远地区，代理商办公室只不过是码头尽头的一个窝棚。不过在这些地方，他们都配备有现代个人计算机（这不是问题，因为他们是运输公司），但网络只能通过很慢的拨号连接，而且由于电力经常中断，一天只能正常工作三两个小时，这意味着两次成功的连接之间，可能会间隔好几天。每个应用都不得不允许当地运输办公室在没有网络连接的情况下继续工作，一旦网络连接建立起来，就立即把本地数据同步到数据中心。这套系统经过了周密的计划和测试，最终是IT基础设施帮助他们从全球集装箱运输业中脱颖而出。在设计移动应用时，我常常想起这些要求。现代设备常常期望更好的运转条件，而最好的应用程序往往会抱最好的希望，做最坏的打算，并代表用户处理棘手的问题。

### 27.3.3 避免不切实际的仿真和测试

移动设备如此之多，而我们只能彻底地测试。在开发的早期阶段使用真实设备测试令人沮丧。由于网络请求要走蜂窝网络（手机网络），因此开发机器只能部署到公开环境。有些移动设备提供了开发模式，不过开发模式也有开发模式的问题。

一句话，我们想要一个仿真环境，以便不必把开发环境公开到外网，又能方便地开发和测试。幸运的是，有许多仿真器程序可以为我们的开发工作提供便利。我会在本章末尾介绍一些仿真器，这些仿真器可以划分为两大类。

第一类仿真器是真实移动浏览器的另一平台版本。这种仿真器浏览器的所有东西都几乎与真实机器上的浏览器一模一样。另一大类仿真器基于这样一个事实：移动及桌面浏览器中的绝大多数都使用某个通用的渲染引擎。举例来说，如果我们想大概地知道iPhone浏览器如何处理一个页面，就可以使用苹果公司的Safari浏览器，因为它和iPhone浏览器使用同样的底层。这类仿真器不过是对桌面渲染引擎做了屏幕尺寸限制和一点点视觉封装而已。

这两类仿真器都很有用，值得试用。在移动产品开发的早期，我会频繁地使用仿真器。不过一旦基础功能开发到位，我就把测试工作扩展到真实设备上。当产品接近完成时，我就只使用真机测试，彻底不再使用仿真器了。

这是因为仿真器有两大不足。首先，它们做不到100%的仿真效果，即使最好的仿真器也做不到总和真实设备一模一样。其次，在我看来也是最重要的缺点，就是它们没法仿真触控输入。

我们不支持触控的桌面电脑上使用鼠标在触控浏览器上工作。而鼠标无论如何也不能造就手指触控的体验。有三样东西是桌面仿真器无论如何也仿真不出来的：触感、遮挡和不精确。

### 1. 没有触感

没有触感意味着我们不能全面了解WEB应用的使用体验。用手指在玻璃显示器上点点滑滑的感觉很别扭。（在真实设备上）应用程序恰当响应触控动作的体验既优雅又有趣。应用程序缺少触感、响应滞后的体验是很糟糕的。鼠标根本没有能力给你触控的美好感觉。。

### 2. 不会遮挡

前面我已经提到过缺少遮挡带来的问题。当我们使用触控设备时，即使是很小的设备，手指和手也会遮住部分屏幕。我们在为触控设备设计WEB应用程序时应该考虑这一点。我们需要小心的安排控制元素，以确保用户在触控屏幕时能够看到程序的反应。我们也应该记住几乎每十个人就有一个左撇子，这些用户操控时的遮挡部位不同于我们。只有亲自动手去触控按钮和链接，才能真正搞清楚应用程序是不是容易使用。

---

**提示** 如果你去咖啡厅做用户观察，你会看到有些用户使用触控设备的方式很特别。他们触摸一下屏幕，然后迅速的把手挪开，然后再进行下一个触控手势操作。这常常预示着这个应用程序把它的反馈元素放到了被用户的手遮挡住的区域。用户不得不挪开自己的手才能看清程序的反应，然后执行后面的动作。真是既累又令人沮丧的用户体验。

---

### 3. 过于精准

不精确的问题一直很烦人。利用鼠标，用户能够精确的击中目标区域。稍加练习任何一只鼠标都能达到像素级的精确度。人类的手指就完全是另一种状况了，我们所能达到的精确度只能是“差不多是这个区域”。这意味着我们必须选择容易触控的组件，并且在布局时就要考虑这种不精确。在一个仿真器上，我们没法体会到产品是不是容易操控。用鼠标代替操控不够好。我们需要测试更多大大小小的屏幕和分辨率，才能理解用户面对的状况。而这些信息为我们的页面布局及安排组件的密度提供了必要的基础。

## 一个与不精准有关的故事

说起我个人的沮丧体验，还和说说我那段经常乘火车旅行的经历。我住在英国，那是一个火车难得准时的国度。在夏天，我不是很在意火车晚点，因为我可以晒太阳打发时间。冬天可就不一样了，我每隔几分钟就想查查这辆车到底会晚点多长时间。只要有在线查询软件，这是可行的。

设想一下：太阳还没有升起，寒风刺骨，地面上满是积雪。虽然我裹得紧紧的，可我从汽车里带出来的那点热量很快就跑光了。我希望打开应用程序，找到我所在的车站，才能看看我还需要等多久。（如果需要等待的时间比较长，我就回到车上，考虑要不要驾车去办公室。）

一脱下手套，我的手指立刻感觉到了寒冷。几分钟之后，我的手指就不能自如的弯曲，我的手开始抖。更恼人的是，需要触控的组件实在太小，那只是一个普通的链接，字体还特别小。我没有一次很容易地点中目标。我错误的点中过别的链接，不得不等待着不需要的信息载入，然后回来再试一次。整个过程，我的手越来越冷，更难点中那个链接。我越来越憎恨这种假定用户能够精确点中的WEB应用，特别是那个应用程序。

## 27.3.4 移动浏览器仿真器

尽管仿真器有着这样那样的局限性，移动浏览器仿真器的作用却不可小觑。在本书第1版中，我介绍了一些仿真器及其优缺点。后来我改变了自己的开发习惯，只使用两个工具：Opera Mobile模拟器和BrowserStack。

### 使用桌面浏览器测试移动应用程序

桌面浏览器的移动版与桌面版虽有区别，但它们的底层是一样的。在把应用程序部署到真实机器之前，完全可以使用桌面浏览器对应用程序做一些粗放的测试。当应用程序的主体结构已经完工，需要对功能区修饰润色时，我经常会使用桌面浏览器。使用桌面浏览器最大的好处在于它们拥有出色的开发工具，包括JavaScript调试器。最重要的是，桌面浏览器提供的出色的开发工具，特别适合移动项目的早期阶段，或者跟踪代码或HTML标签问题的场合。

27

#### 1. Opera移动仿真器

在项目的早期使用Opera移动仿真器，往往是用它做测试。它允许我仿真不同屏幕大小的设备，包括平板和横屏设备。

Opera移动仿真器所仿真的Opera移动浏览器应用十分广泛，而且它在精确呈现内容方面做的确实不错（虽然还称不上完美）。有些jQuery Mobile特性尚未得到支持，比如导航切换时的过场动画（详见第28章）。在本章前面的部分，它是最常使用的截图工具。

这种测试的最大优点在于它非常快，非常适合我比较喜欢的边写边测的开发风格。其中有一个特别棒的特性，就是可以使用Opera浏览器桌面版的调试器来调试Opera移动仿真器。为此，需要做一些稍显枯燥的设置工作，不过这个功能真的很有用。可以从[www.opera.com/developer/mobile-emulator](http://www.opera.com/developer/mobile-emulator)免费下载Opera移动仿真器。

#### 2. BrowserStack

BrowserStack是一项商业服务，它提供运行着各种常见操作系统的虚拟机，在这些虚拟机里又运行着各式各样的浏览器。与维护我自己的测试环境相比，它更加简单，所以赢得了我的青睐。它并不是一种理想的解决方案，因为移动浏览器运行在仿真硬件而不是真实的硬件上。但它们的速度非常快，而且易于使用，支持的浏览器品种也相当全。访问<http://browserstack.com>可获得试用帐号，在提供类似功能的公司当中，有些服务相当有竞争力。

---

**注意** 我只是BrowserStack的一个普通用户，与BrowserStack没有任何其他关系。我为自己的帐号付钱，不享受半点特殊待遇或者折扣。

---

## 27.4 小结

在这一章中，我介绍了如何得到jQuery Mobile并把它安装到页面，jQuery Mobile自动增强页面的原理，以及把页面分成多个“页”的技术。我还讲解了jQuery Mobile为了方便我们开发支持触摸设备的应用程序所提供的各种自定义事件。在最后一部分，我讲了一些移动开发和测试的基础知识。

# 页、主题及布局

本章的主题是jQuery Mobile应用程序中的关键功能：页。关于页，第27章有所提及，在本章中，我们将详细介绍如何定义页、如何配置页以及如何在各页之间导航。此外，还将展示两个对页中样式及内容结构化有用的jQuery Mobile功能：主题和栅格布局。表28-1列出了本章概要。

表28-1 本章概要

问 题	解决方法	代码清单
如何定义页	把一个元素的data-role属性设置为"page"	1
如何定义页头和页尾	把页头元素的data-role设置为"header"，页尾元素的data-role属性设置为"footer"	2
如何在一个页面中定义多个页	在一个页面中定义多个data-role属性为"page"的元素	3
如何在各页之间导航	使用href值为页面id的a元素	4
如何为a元素设置转换效果	设置data-transition属性	5
如何设置全局转换效果	为defaultPageTransition设置一个值	6
如何把一个页链接到其他页面文件	使用href值为目标页面URL的a元素	7、8
如何禁用某个链接的Ajax行为	把它的data-ajax属性设置为false	9
如何合站禁用Ajax	把ajaxEnable事件设置为false值	10
如何设置页面预读	设置data-prefetch属性	11、12
如何改变当前页	使用changePage方法	13
如何控制切换动画的方向	使用changePage方法的reverse选项	14
如何设置loading（加载中）对话框的延时	使用loadMsgDelay选项	15
如何禁用loading对话框	使用showLoadMsg选项	16
如何判断当前页是哪页	使用activePage属性	17
如何后台载入页	使用loadPage方法	18
如何响应页面加载行为	使用页loading事件	19
如何响应页面切换	使用页transition事件	20
如何为页或元素设置主题色系	把data-theme属性的值设置为应使用的色系值	21、22
如何在栅格中摆放元素	使用jQuery Mobile布局类	23

## 28.1 什么是 jQuery Mobile 页

在上一章，我讲过如何使用data-role属性在一个HTML文件中定义jQuery Mobile页。作为温习，

我们来看看代码清单28-1。

**代码清单28-1** HTML文件中的一个简单的jQuery Mobile页

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div data-role="page">
    <div data-role="content">
      This is Jacqui's Flower Shop
    </div>
  </div>
</body>
</html>
```

这是一个最小化的页，它只包含两个关键元素，每一个元素都有data-role属性。一个元素的data-role属性为page表明它包含的HTML内容就是一个jQuery Mobile“页”。我在第27章已经说过，jQuery Mobile的一个关键特征就是用户看到的页的内容仅仅是包含这个页的HTML文件的部分内容。

data-role属性为content的元素是另一个重要的元素。这表明它包含的HTML内容是这个页的内容。稍后我会展示一个例子，它会清楚地表明一个页可以包含一些小节，但data-role值为content的元素只能有一个。图28-1展示了一个jQuery Mobile页在浏览器中显示的样子。

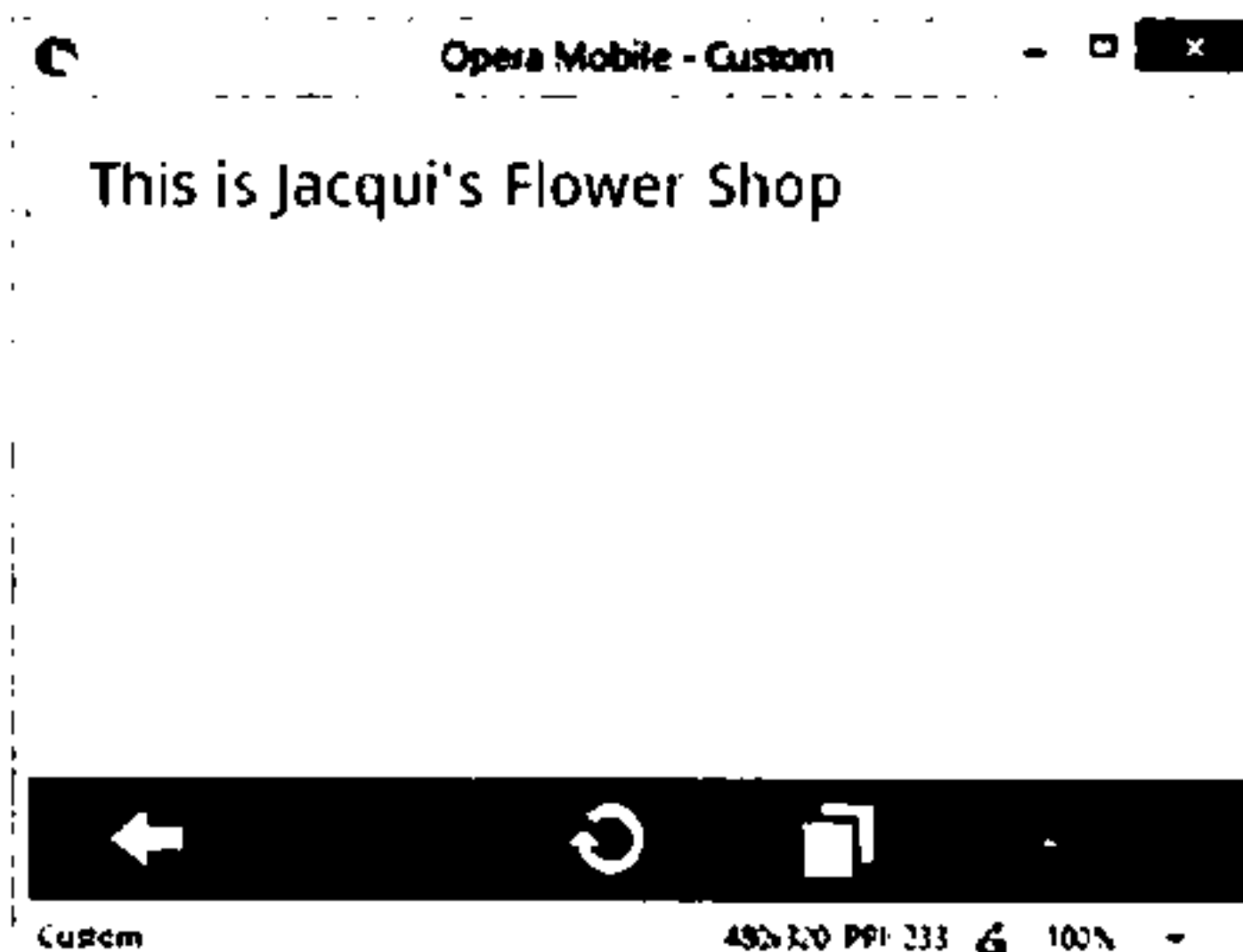


图28-1 在浏览器中显示这个最小化的jQuery Mobile页

### 28.1.1 为一个页添加页头和页脚

作为对content节的补充，jQuery Mobile页可以包含页头和页脚。它们由data-role属性分别为header和footer的元素定义。代码清单28-2中的例子页已经添加了这两节内容。



## 代码清单28-2 为例子页添加页头和页脚

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is Jacqui's Flower Shop
    </div>
    <div data-role="footer">
      <h1>Home Page</h1>
    </div>
  </div>
</body>
</html>

```

添加这些内容之后，页的新样子见图28-2。

**警告** 如图所示，页头和页脚占据了小小屏幕的大量空间。



图28-2 在页中添加页头和页尾

**提示** 注意，页脚显示在内容的末尾而不是页面的底部。把data-position属性设置为fixed，就可以修正页头及页尾的位置问题，这样一来，页头与页尾就能显示在正确的位置，同时内容区还可以自由滚动。若使用此功能需要彻底测试计划支持的浏览器：不是所有的浏览器都支持这一CSS特性，让页首页尾显示在固定位置。

## 28.1.2 在页面文件中添加页

可以在一个页面文件中定义许多页。对简单的Web应用来说，这很有用，因为我们可以把所有的东西都放到一个页面文件中，从而减少向服务器发出的请求次数和总的数据传输量（因为有些元素在包含多个页的页面文件中只需传输一次，比如head区域的元素）。代码清单28-3展示的就是一个包含多个页的页面文件。

**代码清单28-3** 在一个HTML文件中定义多个页

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is Jacqui's Flower Shop
    </div>
  </div>
  <div id="page2" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is page 2
    </div>
  </div>
</body>
</html>
```

这个文件中定义了两个页。我使用id属性为每个页赋予唯一身份，而且id属性的值也是在各页面中导航的基础。在页面加载完成之后，只有第一个页处于显示状态。如代码清单28-4所示，为了便于用户访问各个页面，我们增加一个a元素，把它的href属性设置为目标页的id属性。

**代码清单28-4** 在页之间导航

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
```



```

</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is Jacqui's Flower Shop
      <p>Go to page 2</a></p>
    </div>
  </div>
  <div id="page2" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is page 2
      <p><a href="#page1">Go to page 1</a></p>
    </div>
  </div>
</body>
</html>

```

在这个例子中,我在页面上添加了去另一页的链接。如图28-3所示,当用户点击这个链接时,jQuery Mobile会小心处理这个页面文件,让适当的页显示在正确的位置。

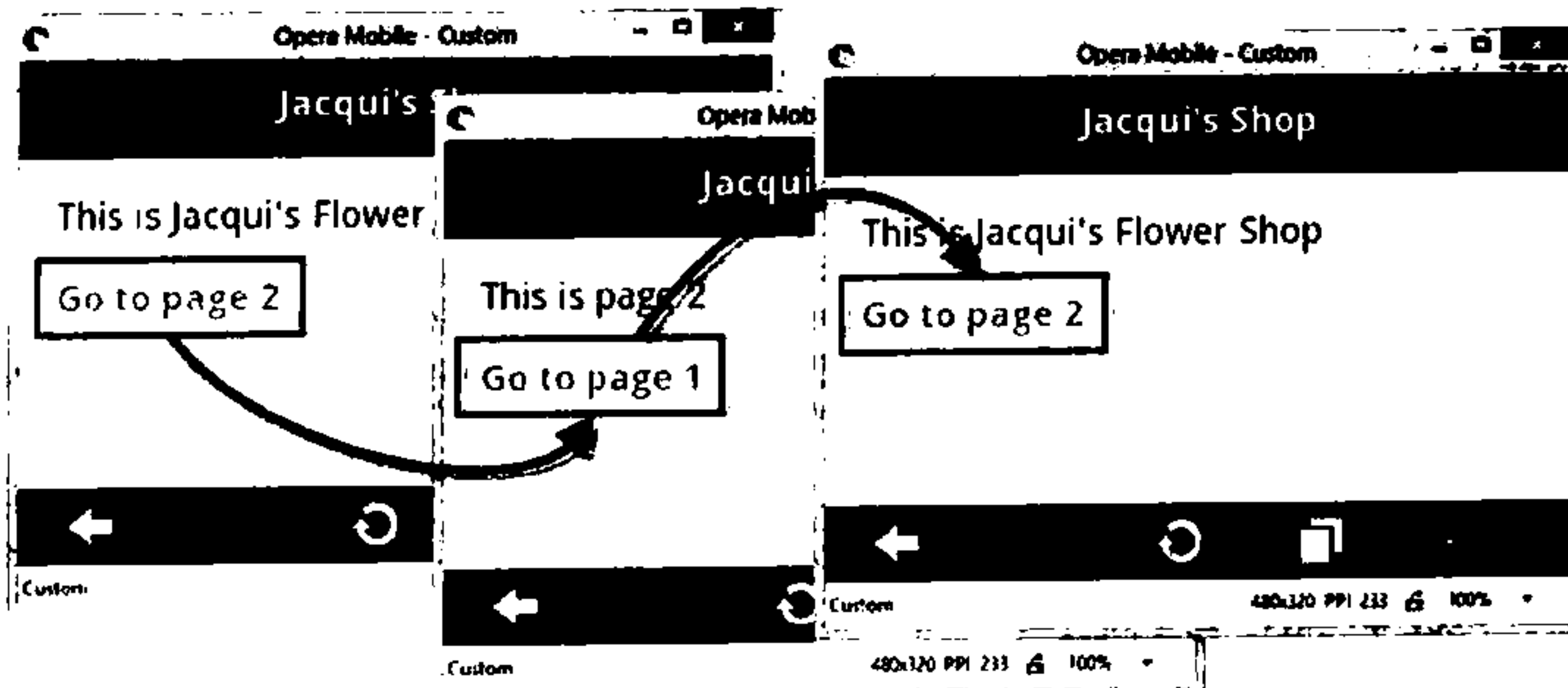


图28-3 在一个页面文件中的多个页之间导航

### 配置页切换效果

jQuery Mobile为用户切换页面的动作添加了动画特效。默认使用的是滑动特效,旧页会向左滑出,而新页会从右边滑入。jQuery Mobile定义了一些特效,即:

- slide (滑动)
- pop (弹出)
- slideup (向上卷起)
- slidedown (向下卷起)

- ☐ slidefade (滑动并淡入淡出)
- ☐ fade (淡入淡出)
- ☐ flip (反转)
- ☐ turn (转换)
- ☐ flow (流动)
- ☐ none (亦作null, 即没有特效)

不是所有的移动设备都能正确支持所有的特效, 有时候你会遇到闪烁或者卡顿。jQuery Mobile每发布一个新版本, 都会增加支持所有特效的设备数目。不过一定要彻底测试你希望支持的所有设备, 确保在那些设备上没有任何问题。如果没有把握, 建议使用fade或slide特效, 我发现几乎所有的设备都支持这两种特效。

---

**提示** 移动浏览器仿真器不能很好地处理切换特效, 常常是把特效直接忽略掉。不过这些特效在真实移动设备上却能运行无误。如果你想在桌面上看到切换效果, 那就使用Google Chrome或者Apple Safari, 这两个浏览器都能很好地处理特效。

---

我们可以使用data-transition属性为单独某个页设置切换效果, 只要把这个属性的值设置为想要的特效名字就好。具体的例子见代码清单28-5。

#### 代码清单28-5 data-transition属性

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is Jacqui's Flower Shop
      <p><a href="#page2" data-transition="pop">Go to page 2</a></p>
    </div>
  </div>
  <div id="page2" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is page 2
      <p><a href="#page1">Go to page 1</a></p>
    </div>
  </div>
</body>
</html>
```

```

    </div>
</body>
</html>

```

**警告** 我很难使用截图表现特效的动画效果。因此这个例子需要在浏览器中才能体验到动画效果。如果你懒得敲写这些代码，可以到出版这本书的Apress.com免费下载到本书的所有源代码。

当用户点击上面代码中突出显示的链接时，在显示目标页时你会看到turn效果。只有这一个链接具有特效（它定义了data-transition属性），页中的其他链接或者同一文件其他页中的链接都是默认效果。如果你不想要动画，就把data-transition属性设置为none。

**提示** 把data-direction属性设置为"reverse"，就能改变a元素的动画方向。在28.2.1节中，我给出了一个反向动画的例子，它将证明这个选项多么有用。

如果希望改变所有导航的切换动画，我们需要设置全局选项。jQuery Mobile定义了defaultPageTransition选项，我们可以在mobileinit事件触发时设置这个选项，具体例子见代码清单28-6。

#### 代码清单28-6 改变默认的页切换效果

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js">
  <script type="text/javascript">
    $(document).bind("mobileinit", function() {
      $.mobile.defaultPageTransition = "fade"
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is Jacqui's Flower Shop
      <p><a href="#page2">Go to page 2</a></p>
    </div>
  </div>
  <div id="page2" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">

```

```

        This is page 2
        <p><a href="#page1">Go to page 1</a></p>
    </div>
</div>
</body>
</html>

```

mobileinit事件没有快捷绑定的方法，因此我们只能选中document对象，使用bind方法绑定这个事件的处理函数。bind方法的第一个参数是事件的名字，第二个参数是事件发生时要调用的处理函数。

---

**警告** 一旦加载jQuery Mobile库，就会触发mobileinit事件。这意味着我们必须在jQuery Mobile库加载之前（我们的代码必须放到加载jQuery Mobile库的script元素之前）注册mobileinit事件的处理函数，才能改变jQuery Mobile的全局设置。可以参见代码清单28-6中的做法。如果不是在加载jQuery Mobile库的script标签之前执行注册事件处理函数的代码，则事件处理函数永远没有机会执行。

---

要改变一个全局设置，直接给\$.mobile对象相应的属性赋值就行。比如我打算改变页面默认的切换特效，就可以像下面这样为\$.mobile.defaultPageTransition属性赋一个新值：

```

...
$.mobile.defaultPageTransition = "fade";
...

```

这行语句把页面的默认切换动画变更为淡入淡出。我仍然可以通过data-transition属性覆盖这个设置。

### 28.1.3 链接其他页面文件

我们不必非要在一个HTML文件里包含所有的页。我们完全可以像标准HTML那样使用标准的链接。为了演示这种做法，我准备了一个新文件document2.html，内容请见代码清单28-7。

代码清单28-7 document2.html文件内容

```

<!DOCTYPE html>
<html>
<head>
    <title>Document 2</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.10.1.js"></script>
    <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div data-role="content">
            This is page 1 in document2.html

```

```

        <p><a href="#page2">Go to page 2 in this document</a></p>
        <p><a href="example.html">Return to example.html</a></p>
    </div>
</div>
<div id="page2" data-role="page">
    <div data-role="header">
        <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
        This is page 2 in document2.html
        <p><a href="#page1">Go to page 1</a></p>
    </div>
</div>
</body>
</html>

```

这个页面文件里也有两个页，和上一个示例文件的结构相同。添加链接到其他页面文件的链接非常简单。如代码清单28-8所示，我们定义一个a元素，然后把href属性设置为目标文件的URL就行了。

代码清单28-8 链接到另一个页面文件的某个页

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.10.1.js"></script>
    <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div data-role="content">
            This is Jacqui's Flower Shop
            <p><a href="#page2">Go to page 2</a></p>
            <p><a href="document2.html">Go to document2.html</a></p>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div data-role="content">
            This is page 2
            <p><a href="#page1">Go to page 1</a></p>
        </div>
    </div>
</body>
</html>

```

jQuery Mobile会使用Ajax载入指定的页面文件，并自动显示它的第一个页，如果指定了切换特效

的话，则会使用切换特效。具体结果见图28-4。

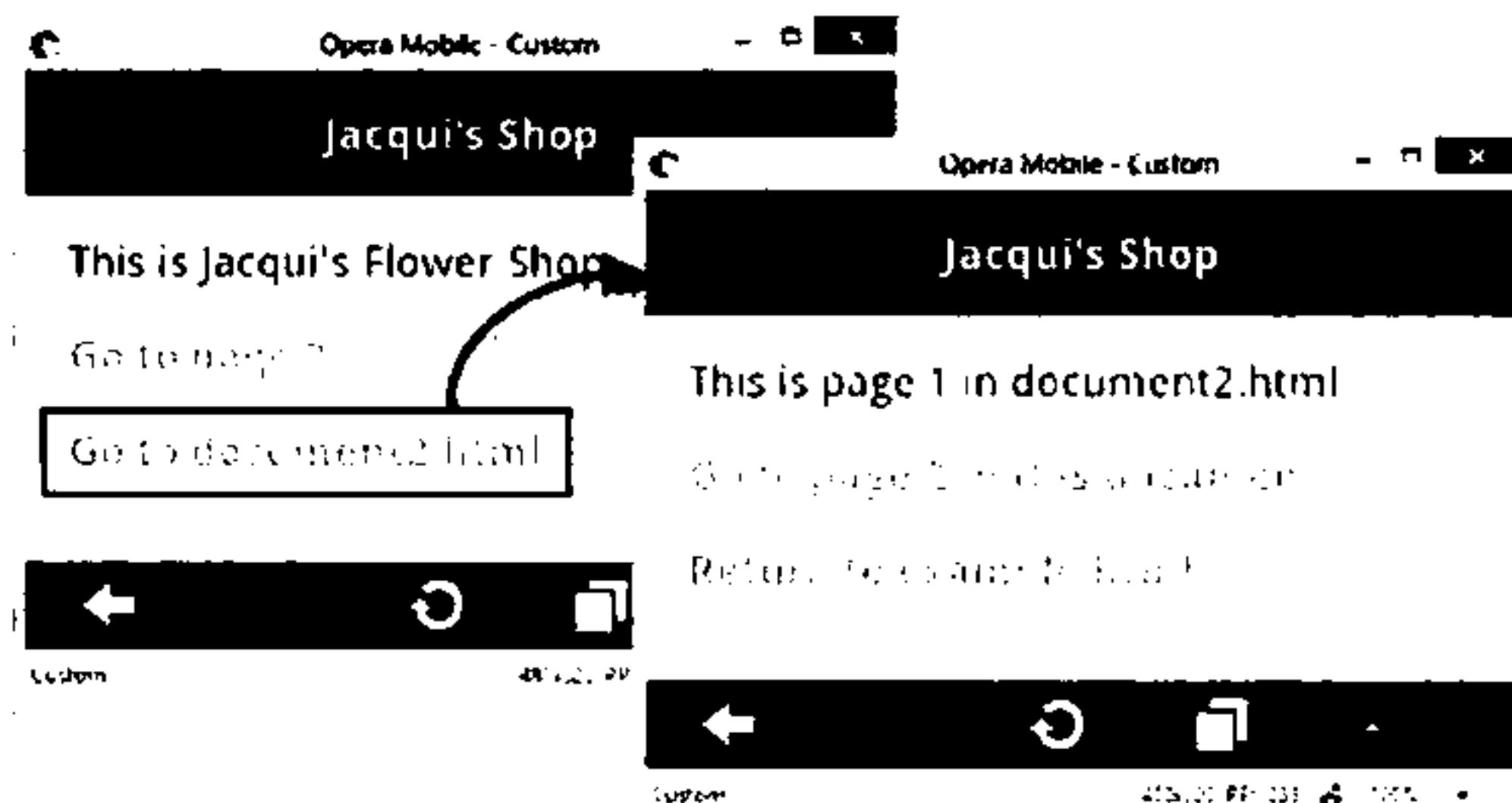


图28-4 访问另一页面文件中的某个页

**提示** jQuery Mobile会自动应用服务器端文件的样式并自动增强这个通过Ajax载入的文件。也就是说我们不必在document2.html等示例文件中包括jQuery和jQuery Mobile库。即便如此，我仍然建议你在页面文件中包含这些库的引用。因为在载入这类文件的时候，jQuery Mobile是允许我们禁止它使用Ajax方式载入文件的。如果我们禁用了Ajax功能，并且页面文件中没有包含这些库的引用，麻烦就来了（这个HTML文件不会自动增强，在这个文件里面的自定义脚本也可能执行出错）。

### 1. 处理Ajax/Page ID问题

当我们链接其他页面文件时，不会总是一帆风顺。由于大家都依赖元素的id属性，管理Ajax载入内容的方式与管理原有页面内容的方式会发生冲突。图28-5展示的就是这个问题。

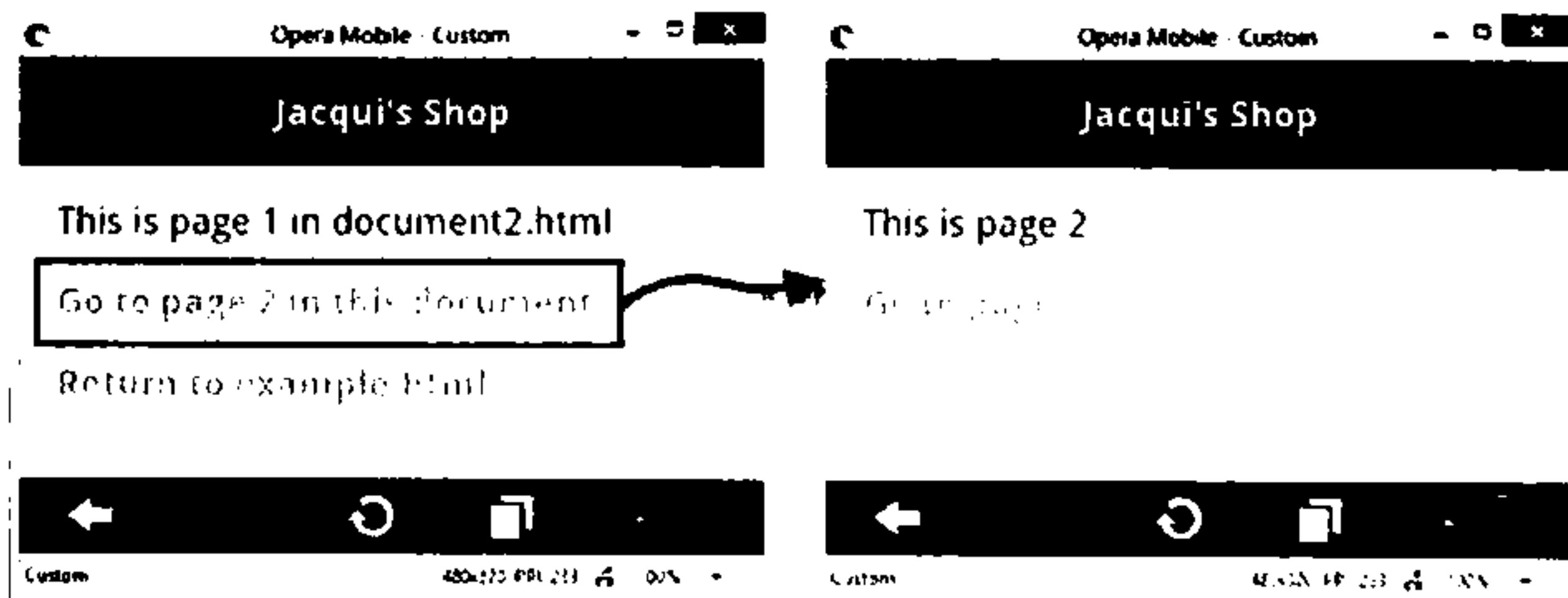


图28-5 多页Ajax问题

点击图中的链接，它应该显示document2.html中的page2元素，然而我得到的却是example.html中的page2元素。一团糟。

有两个办法解决这个问题。第一个是在每个HTML文件中只定义一个jQuery Mobile页。这也是jQuery Mobile开发团队建议的做法。

第二个就是在加载多页面文件时禁用Ajax。这确实能解决问题，不过这也意味着在显示新页面的时候，jQuery Mobile无法使用特效了。如代码清单28-9所示，把a元素中的data-ajax属性设置为false，就可以在载入这个链接URL时禁用Ajax。

代码清单28-9 禁用单个链接的Ajax载入

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is Jacqui's Flower Shop
      <p><a href="#page2">Go to page 2</a></p>
      <p><a href="document2.html" data-ajax="false">Go to document2.html</a></p>
    </div>
  </div>
  <div id="page2" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is page 2
      <p><a href="#page1">Go to page 1</a></p>
    </div>
  </div>
</body>
</html>
```

在这个例子中，我禁止使用Ajax方式载入document2.html的链接。如图28-6所示，这样确实没有问题了。

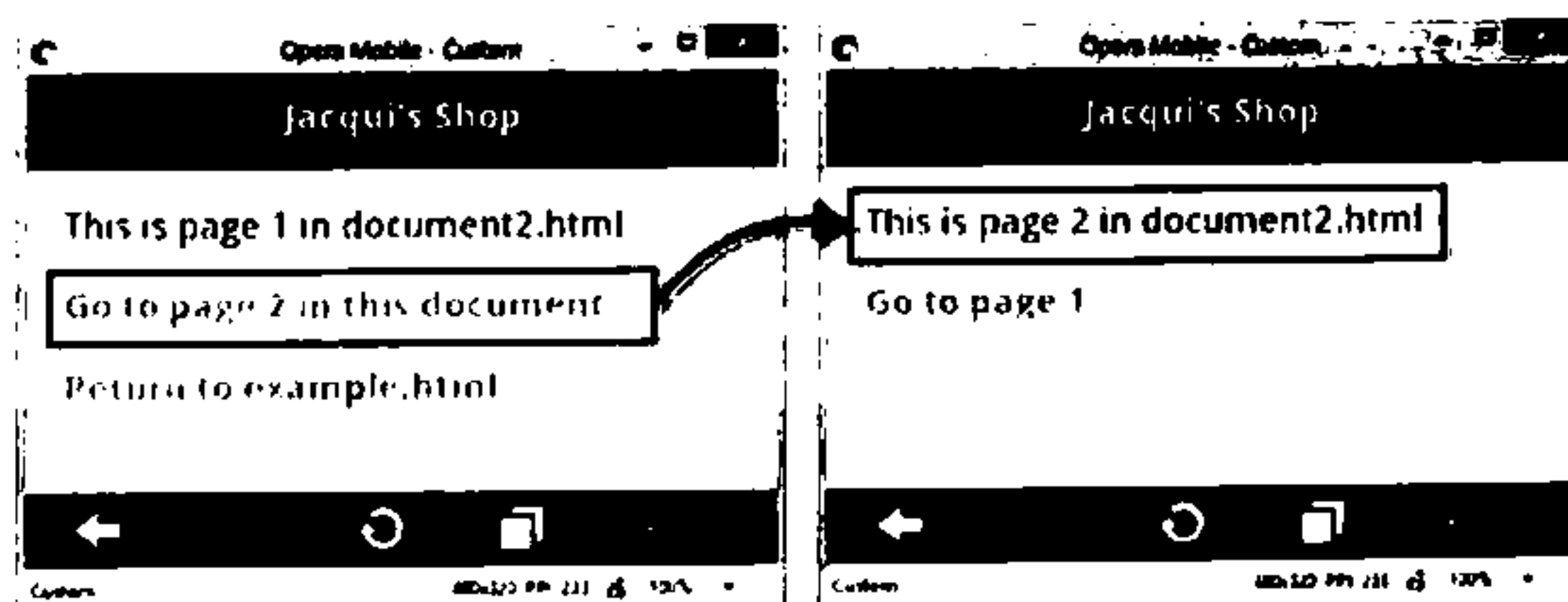


图28-6 禁用Ajax以避免元素id冲突

如代码清单28-10所示，我们也可以使用ajaxEnabled全局选项关掉Ajax支持。当把这个选项设置为false时，所有链接默认都不使用Ajax载入，除非你把某个a元素的数据-ajax属性设置为true。

**代码清单28-10 使用全局选项禁用Ajax**

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript">
    $(document).bind("mobileinit", function() {
      $.mobile.ajaxEnable = false
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is Jacqui's Flower Shop
      <p><a href="#page2">Go to page 2</a></p>
      <p><a href="document2.html">Go to document2.html</a></p>
    </div>
  </div>
  <div id="page2" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is page 2
      <p><a href="#page1">Go to page 1</a></p>
    </div>
  </div>
</body>
</html>
```

## 2. 预加载页面

我们可以让jQuery Mobile悄悄加载用户还没有点击的链接，这样当用户点击一个链接时，这个链接的内容就会马上展现。这么做的好处是，用户确实感到我们的应用很快，缺点是会下载一些用户可能不需要的内容。为了演示这一功能，我新建了一个文件singlepage.html，内容见代码清单28-11。

**代码清单28-11 singlepage.html文件内容**

```
<!DOCTYPE html>
<html>
<head>
  <title>Single Page</title>
```



```

<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
<script type="text/javascript" src="jquery-1.10.1.js"></script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is the only page in this document
      <p><a href="example.html">Return to example.html</a></p>
    </div>
  </div>
</body>
</html>

```

### 决定是否需要预加载内容

是否要预加载内容？有时难以抉择。从应用程序的角度来说，预加载是一个好主意，因为它能在用户切换页面时（尽可能地）做到立即响应。尤其是在无线连接比较慢，而且时好时坏的情况下。没有人喜欢等待，当连接迟迟不能恢复且内容又不可用的时候，应用程序也就失去了价值。

从另一个角度说，我们猜测用户下一步要访问的内容并主动下载这些内容，这是一场冒险，因为用户可能不会访问这些内容。对超出流量就会被额外收费的每月固定流量用户来说，这一定是不受欢迎的。预加载内容表明你在做这样一个假定，你的应用非常重要，以至用户愿意用流量（和费用）换取性能。这个假定很可能是错的。令人沮丧的是，尽管你在过去的一年里都沉迷在自己的项目中，但在用户看来，这可能只是一个能带来些许方便的小玩意。

我的建议是不要预加载页。我们可以提供预加载功能选项，这样那些认为应用程序相当重要的用户，就可以主动启用这个功能。

把这个a元素的data-prefetch属性设置为true，就会启用预加载特性。在代码清单28-12中，我们在example.html页面上展示了data-prefetch属性的用法。

#### 代码清单28-12 预加载内容

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">

```

```

        <h1>Jacqui's Shop</h1>
      </div>
      <div data-role="content">
        This is Jacqui's Flower Shop
        <p><a href="#page2">Go to page 2</a></p>
        <p>
          <a href="singlepage.html" data-prefetch>Go to singlepage.html</a>
        </p>
      </div>
    </div>
    <div id="page2" data-role="page">
      <div data-role="header">
        <h1>Jacqui's Shop</h1>
      </div>
      <div data-role="content">
        This is page 2
        <p><a href="#page1">Go to page 1</a></p>
      </div>
    </div>
  </body>
</html>

```

在这个例子中，我要求jQuery Mobile预加载页面。当我点击这个链接时，jQuery Mobile无需任何延迟，立刻就把我带到了目的地。

---

**提示** 当我们在一个快速可靠的网络上开发应用时，很难确定预加载功能是否正常工作。要测试这一功能，我喜欢使用一个用来调试的HTTP代理，它能显示浏览器发出的所有请求。如果你使用Windows，我建议你使用Fiddler，这是一个相当出色的工具，选项极多，支持各种定制。这个工具可以从它的官网[www.fiddler2.com](http://www.fiddler2.com)下载到。

---

## 28.2 使用脚本控制 jQuery Mobile 页

有时我们需要主动而不是依赖用户点击来切换页。jQuery Mobile提供了一些方法和选项，支持我们使用JavaScript控制页导航。在下面的内容中，我将和大家一起利用这些方法，实现对jQuery Mobile应用中的导航的精细管理。

### 28.2.1 改变当前页

changePage方法用来改变当前显示的页。代码清单28-13演示了这个方法的基本用法，当用户点击一个按钮时，使用它改变当前页。

**代码清单28-13** 改变jQuery Mobile应用程序中的当前页

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>

```

```
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
<script type="text/javascript" src="jquery-1.10.1.js"></script>
<script type="text/javascript">
    $(document).bind("pageinit", function() {
        $("button").bind("tap", function(e) {
            var target = this.id == "local" ? "#page2" : "document2.html";
            $.mobile.changePage(target)
        });
    });
</script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div data-role="content">
            <fieldset class="ui-grid-a">
                <div class="ui-block-a"><button id="local">Local</button></div>
                <div class="ui-block-b"><button id="remote">Remote</button></div>
            </fieldset>
        </div>
    </div>
    <div id="page2" data-role="page">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div data-role="content">
            This is page 2
            <p><a href="#page1">Go to page 1</a></p>
        </div>
    </div>
</body>
</html>
```

在这个例子中，我添加了两个按钮。点击这些按钮会调用changePage方法。我使用bind方法为一个名为tap的事件绑定了处理函数，它是jQuery Mobile定义的很有用的几个自定义事件中的一个。当用户轻触屏幕（或者使用鼠标在一个非触控设备上点击）时，这个事件就会发生。第27章详细讲解了这个事件，以及其他事件。

这些按钮都是标准HTML元素，jQuery Mobile自动把它们转换成了按钮组件。我会在第30章介绍jQuery Mobile按钮的配置选项。最后，你可能注意到我给一些元素添加了ui-grid-a、ui-block-a以及ui-block-b等class。jQuery Mobile使用这些class帮助我们创建页面布局，稍后我会详细介绍这些class。如图28-7所示，这个例子的显示结果相当简单。当用户点击其中一个按钮，就会调用changePage方法，显示当前页面文件中的指定id的页，或者从一个URL中载入要显示的页。内容顺利加载，页面切换特效也照常显示，效果与我们使用标准的链接一模一样。

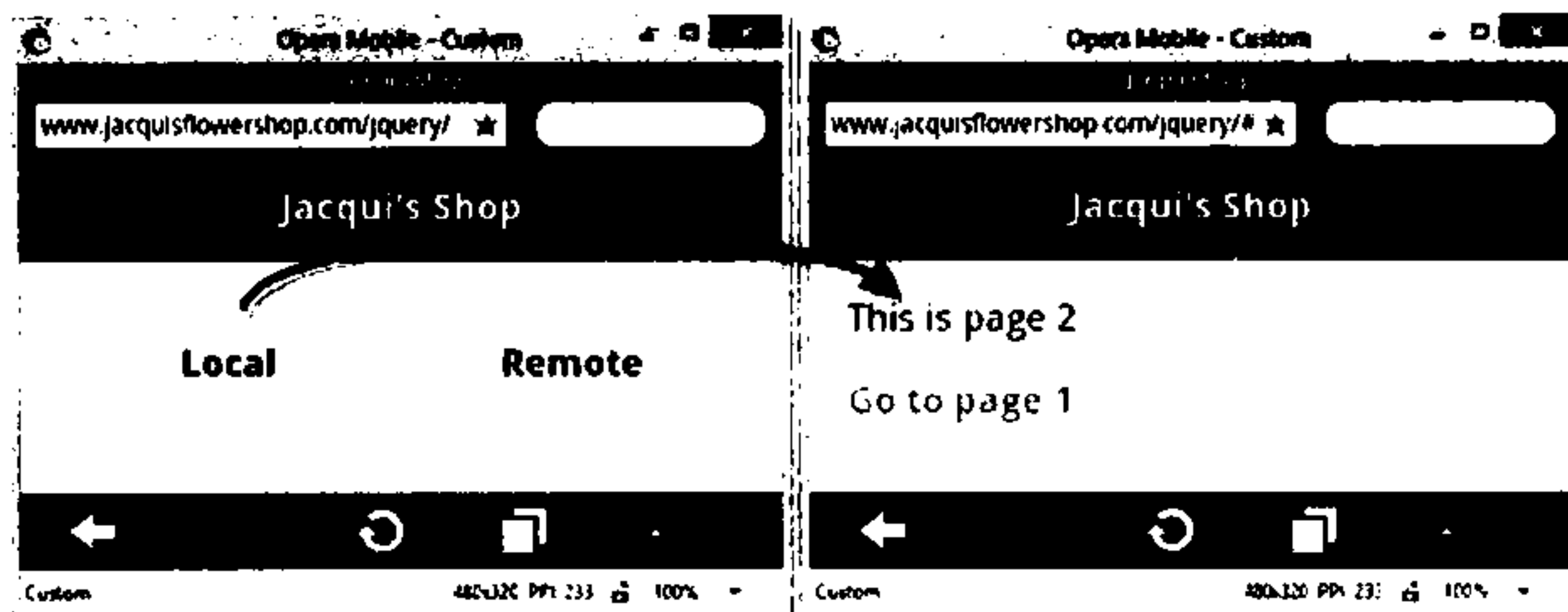


图28-7 使用changePage方法切换页面

以上是changePage方法的基本用法，它还有一些可用的配置选项。要配置页面切换效果，可以传入一个选项对象，作为changePage方法的第二个参数。利用这个对象，我们可以一次配置一个或多个选项。表28-2列出了所有可用的选项。绝大多数选项都不必调整，默认设置就是最好的。不过在接下来的内容中，我会介绍两个需要频繁修改才更有用的选项。

表28-2 changePage方法的配置选项

选 项	描 述
allowSamePageTransition	默认值 false，忽略对当前页的请求。如果设置为true，则允许再次请求当前页，这可能会造成页面切换动画出错
changeHash	默认值true。表示切换页面时会更新URL中的hash
data	指定发起Ajax请求时载入页面时，需要发送给服务器的数据
dataUrl	指定用来更新浏览器URL栏的URL。默认值为空字符串，表示数据来自元素的id或者远程文件的URL
loadMsgDelay	默认值50（单位毫秒）指定loading信息显示时间
pageContainer	指定包含新页的容器元素
reloadPage	默认值false。如果设置为true，即使请求内容已经有本地缓存，jQuery Mobile仍会再次向服务器请求内容
reverse	默认值false。如果设置为true，切换动画将反向播放
role	为新内容设置data-role属性（详见第29章）
showLoadMsg	默认值true，表示在载入远端文件时显示加载中对话框
transition	指定切换新页时要使用的动画效果
type	默认值为get。指定请求服务器端页面时使用的HTTP方法，可选值为get和post

**提示** loadPage方法也使用这些选项，后文会就此做介绍。

### 1. 控制页面切换动画的播放方向

我最常使用的就是reverse选项。jQuery Mobile总是使用同样的方式播放页面切换动画。这在用户完成一个动作就立即返回上一页面，或者正在响应swiperight事件时，就不太适用了。代码清单28-14演示了这一问题。

代码清单28-14 切换动画的方向与用户导航意愿相反

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript">
    $(document).bind("pageinit", function() {
      $.mobile.defaultPageTransition = "slide";
      $("button").bind("tap", function(e) {
        var target = this.id == "forward" ? "#page2" : "#page1";
        $.mobile.changePage(target, {
          reverse: (target == "#page1")
        });
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is page 1
      <button id="forward">Go to Page 2</button>
    </div>
  </div>
  <div id="page2" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is page 2
      <button id="back">Back to Page 1</button>
    </div>
  </div>
</body>
</html>

```

页面中有两个页，每个页中有一个按钮，可用来跳转到另一个页。第二个页上的按钮标识着Back to Page 1（回到page 1），当我们点击这个按钮时，我把reverse选项设置为true，从而改变了切换动画的播放方向。我无法通过一个静止的页面演示这个效果，然而这样修改之后，切换动画就更加自然了。切换动画有意无意地提示用户正在返回上一页，而不再傻乎乎的总是标识向前走（跳到新页）。如果你能在浏览器中查看这个例子，就一定能明白我的意思。

## 2. 控制“加载中”动画

当jQuery Mobile使用Ajax加载一个服务器端页面超过50毫秒时会显示“加载中”动画图片。如果

我们使用的是移动浏览器仿真器或者网速很快，jQuery Mobile能够快速载入内容，那么这个对话框就不会显示出来。但如果我们使用真实的移动网络，或者像我在例子中那样，主动引入一些延迟，则这个对话框就会在屏幕上显示足够长的时间（如图28-8所示）。

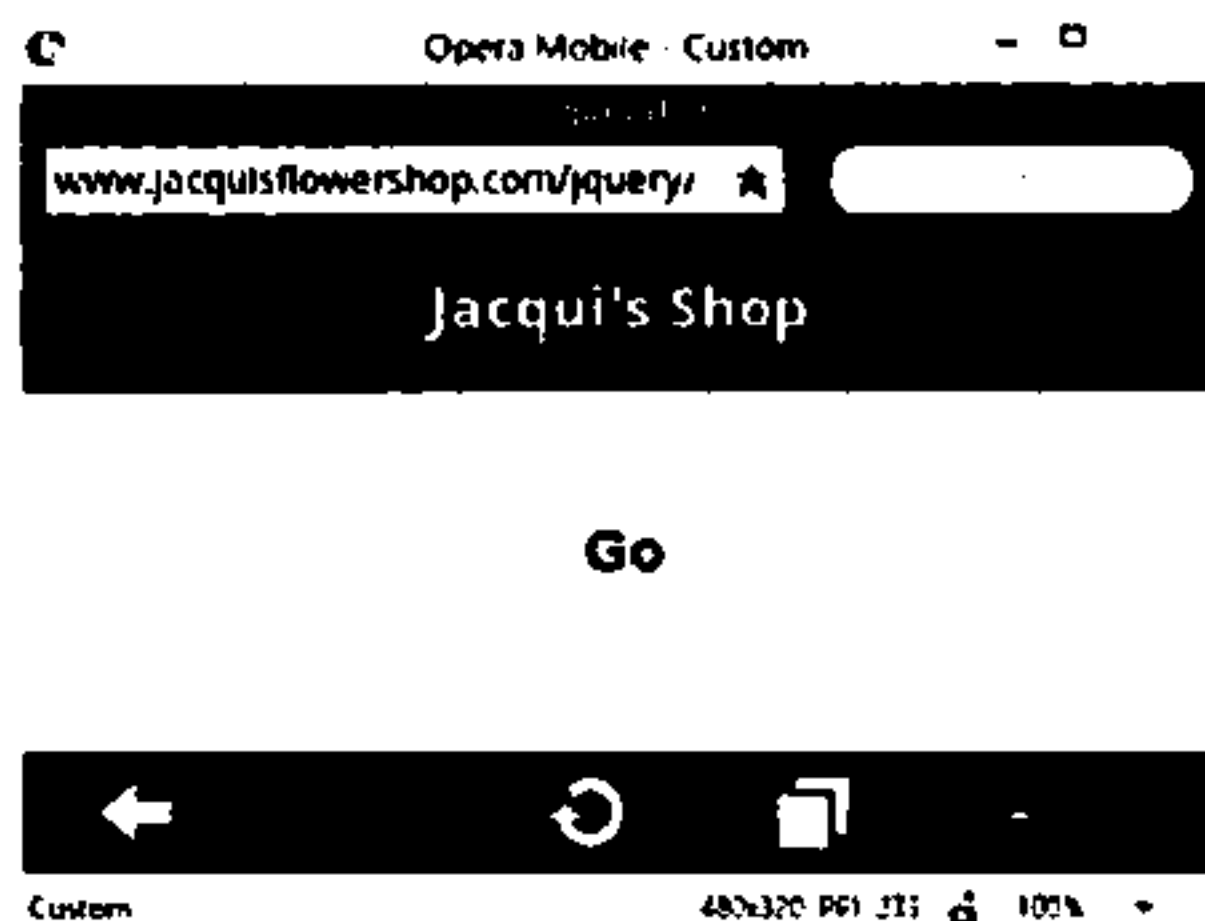


图28-8 jQuery Mobile “加载中”对话框

通过指定loadMsgDelay选项的值，我们可以改变对话框显示前需要延迟的时间，如代码清单28-15所示。

#### 代码清单28-15 改变载入中对话框显示之前的延迟时间

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript">

    $(document).bind("mobileinit", function() {
      $.mobile.loadingMessage = "Loading Data..."
    });

    $(document).bind("pageinit", function() {
      $('button').bind("tap", function(e) {
        $.mobile.changePage("document2.html",{
          loadMsgDelay: 1000
        });
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <button id="forward">Go</button>
    </div>
  </div>
```

```

    </div>
</body>
</html>

```

在这个例子中，我要求jQuery Mobile等待1秒钟才显示“载入中”对话框。

**提示** 像上面这个例子中所做的那样，我们可以修改全局属性loadingMessage的值，从而实现修改loading对话框中显示的文本。和所有其他jQuery Mobile全局属性一样，这个属性也必须在mobileinit事件触发之前设定。

我们也可以在调用changePage方法时把showLoadMsg选项设置为false，这样loading对话框就不会显示。但我不建议你这样做，因为及时提供给用户响应才是正确的做法。不过代码清单28-16还是演示了这一用法。

#### 代码清单28-16 禁用loading对话框

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript">
    $(document).bind("pageinit", function() {
      $("button").bind("tap", function(e) {
        $.mobile.changePage("document2.html", {
          showLoadMsg: false
        });
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <button id="forward">Go</button>
    </div>
  </div>
</body>
</html>

```

### 28.2.2 当前页是哪个

利用\$.mobile.activePage属性能够知道jQuery Mobile正在显示哪个页。代码清单28-17演示了activePage属性的用法。

## 代码清单28-17 activatePage属性

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript">

    var eventHandlerCreated = false;
    $(document).bind("pageinit", function () {
      if (!eventHandlerCreated) {
        $("button").bind("tap", function (e) {
          var nextPages = {
            page1: "#page2",
            page2: "#page3",
            page3: "#page1"
          }
          var currentPageId = $.mobile.activePage.attr("id");
          $.mobile.changePage(nextPages[currentPageId]);
        })
        eventHandlerCreated = true;
      }
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is page 1
      <button id="forward">Go</button>
    </div>
  </div>
  <div id="page2" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is page 2
      <button id="forward">Go</button>
    </div>
  </div>
  <div id="page3" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is page 3
    </div>
  </div>
</body>
</html>

```



```

        <button id="forward">Go</button>
    </div>
</div>
</body>
</html>

```

在这个例子里有三个页，每个页里有一个按钮。点击按钮时，我就读取activePage以获取当前页。activePage属性返回包含着当前页元素的jQuery对象，因此我可以用attr方法得到它的id属性。

我的脚本里有一个简单的映射对象，它会告诉我页面文件中每个页的下一页是哪个。我把activePage属性所对应的页的id属性作为参数传递给changePage方法，从而保证了这些页的显示顺序与映射对象中的定义相符。

---

**提示** 注意，我使用变量eventHandlerCreated确保只为tap事件绑定一个事件处理函数。changePage方法触发pageinit事件，会导致绑定多个同样的事件处理函数——这是jQuery Mobile应用程序的一个常见问题，往往导致一次按钮点击发生多次页面切换。避免此问题最可靠的方式就是像本例一样使用一个变量。

---

### 28.2.3 后台载入页

利用loadPage方法，我们可以载入远端页面，但并不将它显示出来。本章前面演示过的预加载功能，使用的就是这个方法。loadPage方法支持两个参数，第一个是要加载页面的URL，第二个参数是可选的选项对象。loadPage方法支持changePage方法的配置选项（参见表28-2）。代码清单28-18演示了loadPage方法的用法。

#### 代码清单28-18 loadPage方法

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.10.1.js"></script>
    <script type="text/javascript">

        var loadedPages = false;

        $(document).bind("pageinit", function () {
            if (!loadedPages) {
                $.mobile.loadPage("document2.html", {}).done(function () {
                    $("#gobutton").button("enable").bind("tap", function () {
                        $.mobile.changePage("document2.html")
                        loadedPages = true;
                    });
                });
            }
        });
    </script>

```

```
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <button id="gobutton" disabled>Go</button>
    </div>
  </div>
</body>
</html>
```

**提示** 注意，我使用一个空的对象作为loadPage方法的第二个参数，这是因为jQuery Mobile 1.3.1中的一个bug：即使不改变任何设置，也要求提供对象参数。

在这个例子中我使用loadPage方法预加载了document2.html文件。loadPage方法返回一个延迟对象，这样当页面加载完成之后，我们就可以通过这个对象收到通知。我会在第35章详细介绍延迟对象，不过在这里，我们只需要知道：在loadPage方法返回的延迟对象上调用done方法，指定一个函数作为它的参数，则当loadPage完成时，这个函数就会自动执行。

在这个例子中，我在一个jQuery UI按钮组件上使用enable方法让这个按钮在页面中可用，并为tap事件注册了处理函数。若点击这个按钮，就会调用changePage方法切换到预先加载好的页面。（我将在第30章介绍jQuery Mobile中的按钮组件。）

注意我定义了一个全局变量loadedPages。使用changePage方法（内部调用loadPage）时我遇到了同样的问题——不论在什么时候，只要jQuery Mobile初始化一个页，就会触发pageinit事件。也就是说例子页面加载成功，会触发pageinit事件，当loadPage方法使用Ajax载入document2.html，pageinit事件会再次触发。我利用loadedPages全局变量来保证同一内容只尝试预加载一次。如果没有loadPages变量，就有可能调用loadPage方法两次（或者更多次），但是只要我不把reload选项的值设置成true，这未必是件坏事。如果把reload选项设置为true就会导致document.html载入两次，因为这个选项要求jQuery Mobile忽略缓存中的内容。我会在接下来的内容中介绍与jQuery Mobile页有关的事件。

## 28.3 与页相关的事件

jQuery Mobile定义了一些事件，方便我们处理页面有关的变化。表28-3中列出了这些事件，在接下来的几节当中，我会从这些事件中挑出几个特别有用的，详细介绍它们的用法。

表28-3 jQuery Mobile中与页有关的事件

事 件	描 述
pagebeforecreate	页被初始化时触发此事件
pagecreate	若页被创建后多数自动加强还未执行就会触发此事件
pageinit	在页被jQuery Mobile初始化之后触发此事件
pageremove	页被移动前触发此事件

(续)

事 件	描 述
pagebeforeload	在加载页面的Ajax请求发起之前触发此事件
pageload	若Ajax加载页面成功触发此事件
pageloadfailed	若Ajax加载页面失败触发此事件
pagebeforechange	在页面切换之前触发此事件
pagechange	在页面切换开始之后触发此事件
pagechangefailed	当页面切换失败时触发（通常都是因为没有成功加载目标文件）此事件
pagebeforeshow	在页面显示给用户看之前触发此事件
pagebeforehide	在页面即将隐藏之前触发此事件
pageshow	在页面显示出来之后触发此事件
pagehide	在页面隐藏起来之后触发此事件

### 28.3.1 页面初始化事件

在这些页面有关的事件当中，pageinit事件是最有用的一个（参阅第27章）。如果我们需要使用脚本配置页面，就应该把脚本写在这个事件的处理函数里。我不必再次介绍这一事件的使用法，因为我们的每一个例子都用到了这个方法。不过我还是要在这里强调一点，如果使用jQuery Mobile处理页面，jQuery标准库的\$(document).ready()方法并不可靠。

### 28.3.2 页面加载事件

不论是jQuery Mobile自动生成的，还是由changePage和loadPage方法触发的pagebeforeload、pageload和pageloadfailed事件，在监控获取服务器端页面的Ajax请求方面都非常有用。在前面loadPage方法的例子中，我使用了延迟对象来响应page成功加载事件，其实我们使用pageload事件也能实现同样的目的（当然，当页面载入失败，就使用pageloadfailed事件）。代码清单28-19是loadPage例子改用pageload事件的更新版本。

代码清单28-19 pageload事件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript">

    $(document).bind("pageload", function(event, data) {
      if (data.url == "document2.html") {
        $("#gobutton").button("enable").bind("tap", function() {
          $.mobile.changePage("document2.html")
        });
      }
    });
  </script>
</head>
</html>
```

```

        var loadedPages = false;
        $(document).bind("pageinit", function() {
            if (!loadedPages) {
                loadedPages = true;
                $.mobile.loadPage("document2.html")
            }
        });

</script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div data-role="content">
            <button id="gobutton" disabled>Go</button>
        </div>
    </div>
</body>
</html>

```

在这个例子中，我为pageload事件指定了一个回调函数，当pageload事件发生时，这个函数就会执行。jQuery Mobile会自动提供一个数据对象data（里面保存着页面请求的一些信息）作为回调函数的第二个参数。我使用data对象的url属性检查载入的页面是否为我们需要的页面。表28-4列出了data对象支持的各种属性，从这个表格我们可以看出，里面绝大多数属性都与jQuery的Ajax功能（详见本书第14章和第15章）有关。

表28-4 pageload data对象的属性

属 性	描 述
url	返回传递给loadPage方法的url参数（当jQueryMobile请求页面时或者调用changePage方法时会调用loadPage方法）
absUrl	请求URL的绝对地址
options	Ajax请求选项对象，参阅第14章和第15章了解配置Ajax请求的细节
xhr	本次请求对应的jQuery Ajax请求对象，参阅第14章和第15章了解该对象的细节
textStatus	请求状态的文本描述，详情见第14章和第15章

### 28.3.3 响应页面切换

当用户从一个页到另一个页（或者我们的脚本调用changePage方法时），会触发页面切换事件。每次页面切换都会发生这些事件（pagebeforehide、pagehide、pagebeforeshow和pageshow），即使这些页以前已经显示过。代码清单28-20演示了pagehide事件的使用法。

代码清单28-20 响应页面隐藏

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>

```

```

<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
<script type="text/javascript" src="jquery-1.10.1.js"></script>
<script type="text/javascript">

    var registeredHandlers = false;
    $(document).bind("pageinit", function() {
        if (!registeredHandlers) {
            registeredHandlers = true;
            $("#page1").bind("pagehide", function(event, data) {
                $.mobile.changePage($("#page1"));
            });
        }
    });

</script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div data-role="content">
            <a href="document2.html">Go to document2.html</a>
        </div>
    </div>
</body>
</html>

```

在这个例子中，我选中#page1元素，然后调用bind方法为pagehide事件注册了处理函数。这表示当选中页面被隐藏时，这个事件会发生。这是一个相当没劲的例子，这是因为在pagehide事件发生之后，处理函数除了使用changePage方法回到page1，再没干任何别的事。不过它确实演示了pagehide事件的使用。注意例子里我使用了一个变量避免重复注册事件处理函数。如果我不这样做，当document2.html页面载入时，就会在同一个元素上就同一个事件绑定两个处理函数。

## 28.4 使用 jQuery Mobile 主题

jQuery Mobile支持主题，就像jQuery UI提供的主题功能的简化版。我们在第27章下载并安装的jQuery Mobile文件包括默认的主题，我们还可以使用jQuery Mobile ThemeRoller程序生成自定义主题，它是jQuery UI同名应用程序的一个变种。

---

**注意** 我在第27章说过，本书的这一部分并不涉及创建自定义主题。jQuery Mobile ThemeRoller尚不具备一个内建主题库。而且描述创建一个主题库是一件相当困难的事情。我会在本章使用默认主题，想要详细了解jQuery Mobile ThemeRoller，请访问<http://jquerymobile.com/themeroller>。

---

每种jQuery Mobile主题包含一个或多个色系，每个色系都是一套针对不同元素的样式。色系由单个字母标识，从A开始。默认主题有从A到E五个色系。

要在ThemeRoller中查看默认主题，访问<http://jquerymobile.com/themeroller>，点击Import链接，然后点击Import Default Theme链接，将jquery.mobile-1.3.1.css文件的内容导入对话框（我们在第27章下载并安装了这个文件）。点击Import按钮，ThemeRoller应用程序会处理这个CSS文件并显示默认的色系（如图28-9所示）。

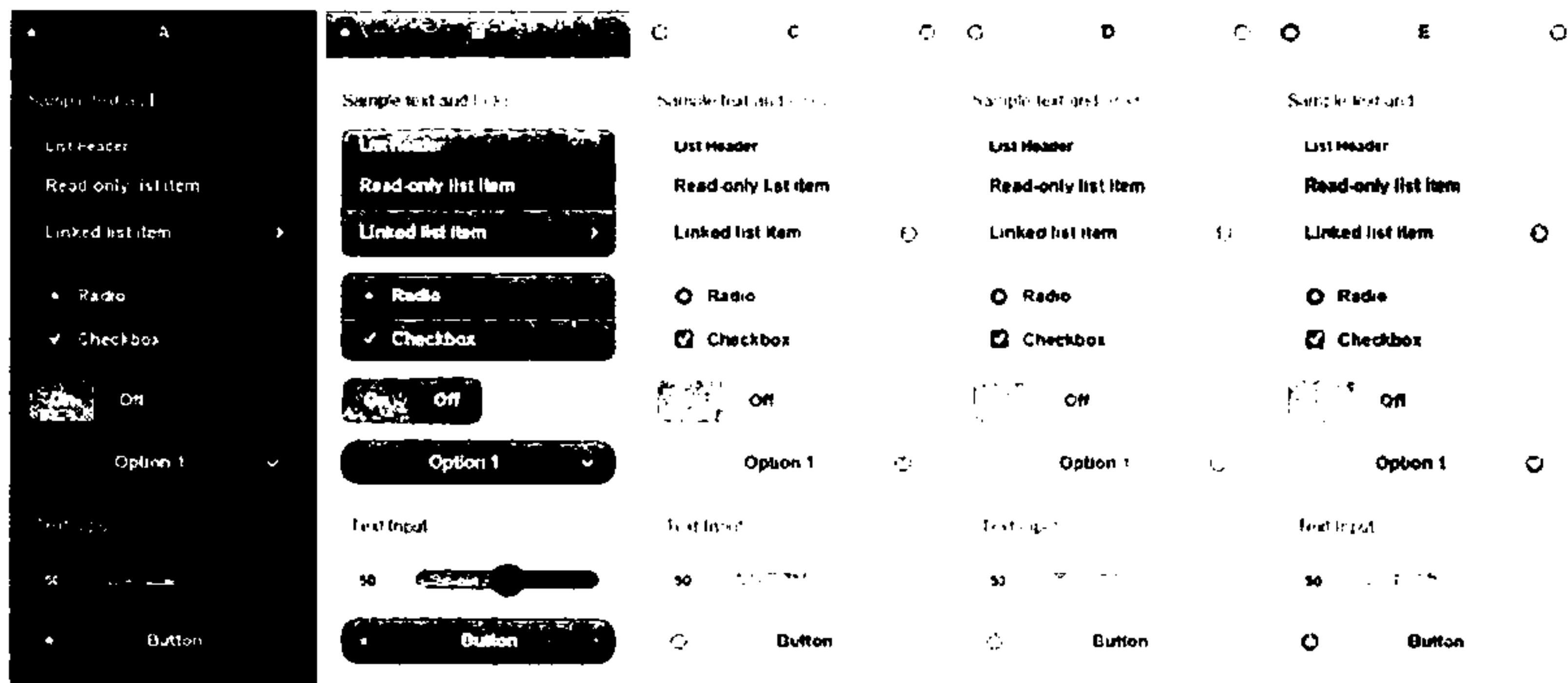


图28-9 默认主题的色系

把data-theme属性设置为希望使用的色系名称，即可设置jQuery Mobile主题。具体例子见代码清单28-21。

#### 代码清单28-21 使用主题色系

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="a">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is Theme A
      <a href="#page2" data-role="button">Switch Theme</a>
    </div>
  </div>
  <div id="page2" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
```

```

    <div data-role="content">
      This is Theme B
      <a href="#page1" data-role="button">Switch Theme</a>
    </div>
  </div>
</body>
</html>

```

我在page元素中设置了data-theme属性，指定的色系会影响当前页中的所有子元素。上面的例子中共有两个页，这样我们就可以对比同样内容使用不同色系的显示效果，并清楚地表明在jQuery Mobile中，一旦页面自动加载工作完成，就没有什么可靠的手段可以改变色系了。只有在页面初始化时，jQuery Mobile才会把data-theme属性转换为添加到元素上的一系列样式类。因此，改变data-theme属性的值并不会改变那些已经添加到元素上的类。图28-10展示了这两个页面在浏览器中的显示效果。

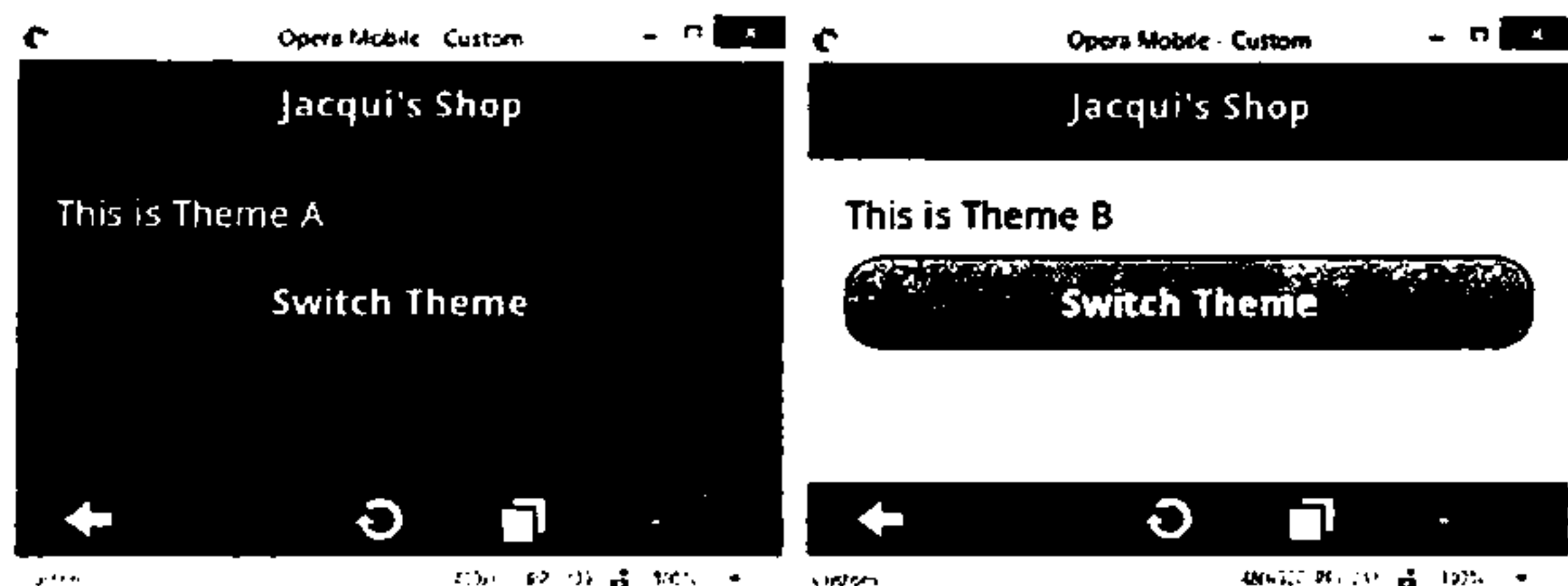


图28-10 同一应用中的两个页面分别使用不同的色系

**提示** 还有一个名为active的色系，它用来突出显示选中的按钮。在需要时jQuery Mobile会自动使用这个色系，当然，我们也可以直接使用。如果我们用到了active色系，请确保用户能够辨别出哪些元素是当前激活的元素（需要突出显示的元素）。

## 为具体某个元素设置色系

上面的例子展示了如何为整个页设置样式，其实具体元素也支持色系设置，混用色系能够匹配出特殊的显示效果。具体例子见代码清单28-22。

### 代码清单28-22 为具体元素设置色系

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>

```



```

<div id="page1" data-role="page" data-theme="a">
  <div data-role="header">
    <h1>Jacqui's Shop</h1>
  </div>
  <div data-role="content">
    <a href="document2.html" data-role="button" data-theme="b">Press Me</a>
    <a href="document2.html" data-role="button" data-theme="c">Press Me</a>
    <a href="document2.html" data-role="button" data-theme="e">Press Me</a>
  </div>
</div>
</body>
</html>

```

在这个例子中，我为page元素和三个button元素分别设置了data-theme属性，每个元素都有着不同的色系（如图28-11所示）。

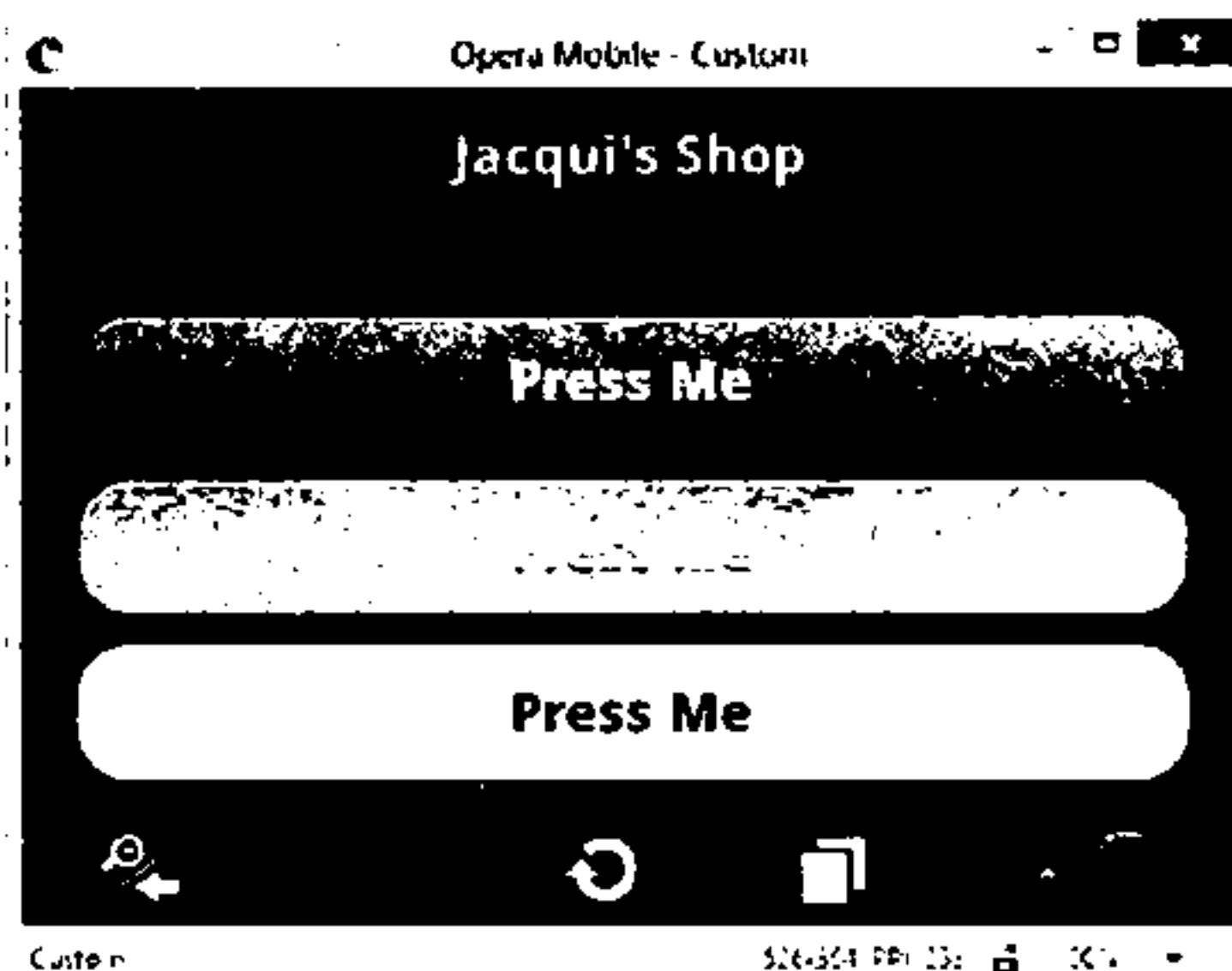


图28-11 为同一页中的元素设置不同的色系

jQuery Mobile的主题支持简单易用。如果我们能够动态改变元素使用的色系就更好了，即便目前缺失这一功能，我们仍然能够使用色系定制移动应用程序的外观。

## 28.5 创建网格布局

jQuery Mobile定义了一些有用的CSS类，我们可以利用它们以网格的形式摆放页面内容。我们完全可以自己实现类似功能，把该功能集成到库里的优点是可以大大减少定制开发的工作量（尤其对一些比较简单的移动应用来说）。jQuery Mobile定义了四个网格类，每个类可以包含不同数目的列，参见表28-5。

表28-5 jQuery Mobile网格布局

CSS类	列数
ui-grid-a	2
ui-grid-b	3
ui-grid-c	4
ui-grid-d	5



我们给容器元素添加一个网格类, 然后为它的每列内容添加相应的类(如ui-block-a、ui-block-b等等)。代码清单28-23使用这些类生成了一个简单的网格。

代码清单28-23 使用jQuery Mobile网格布局类生成一个网格

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <div class="ui-grid-b">
        <div class="ui-block-a"><button>Press Me</button></div>
        <div class="ui-block-b"><button>Press Me</button></div>
        <div class="ui-block-c"><button>Press Me</button></div>
      </div>
      <div class="ui-grid-a">
        <div class="ui-block-a"><button>Press Me</button></div>
        <div class="ui-block-b"><button>Press Me</button></div>
      </div>
      <div><button>Press Me</button></div>
    </div>
  </div>
</body>
</html>
```

在这个例子中, 我创建了两个网格: 一个三列, 一个两列。每列包含一个button元素。我还在网格之外添加了一个button元素, 以强调各元素的默认行为是平分屏幕。这个例子在浏览器中的显示见图28-12。

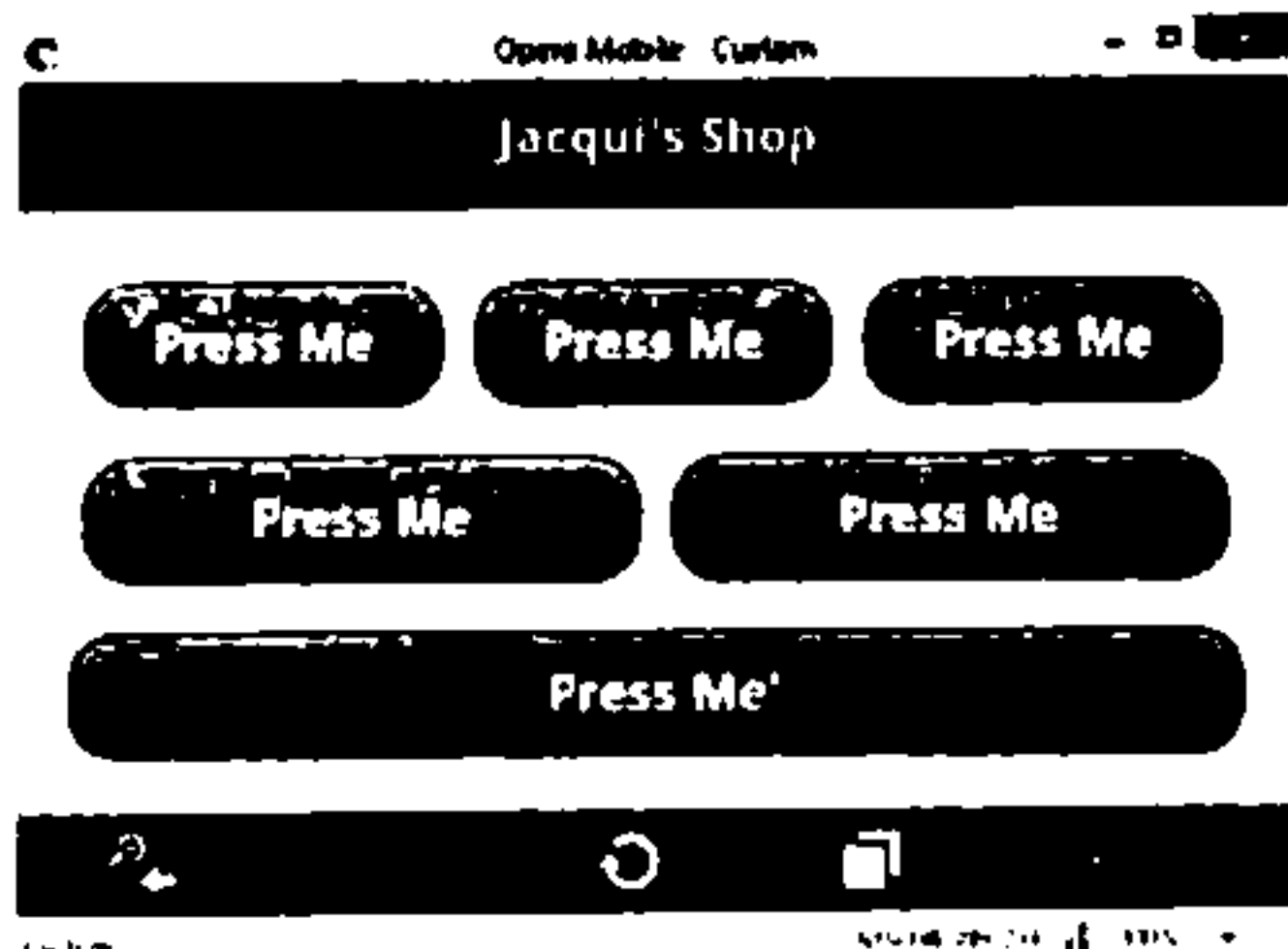


图28-12 在网格中排列元素

## 28.6 小结

本章介绍了jQuery Mobile中的页，它是jQuery Mobile中相当关键的一部分。还介绍了jQuery Mobile对主题及网格布局的支持。这些功能构成了jQuery Mobile功能组件的基础。在下一章，我将介绍对话框和弹窗组件。

# 对话框与弹窗组件

本章的主题是两个jQuery Mobile组件：对话框与弹窗。jQuery Mobile组件与jQuery UI组件略有不同，但两个库共用同一底层实现。表29-1列出了本章概要。

表29-1 本章概要

问 题	解决方法	代码清单
如何创建对话框组件	为包含对话框内容的div元素添加data-role属性，并把属性值设置为dialog，或者给一个导航链接添加值同样为dialog的data-rel属性，点击这个链接就会打开对话框	1、2
如何使用脚本生成一个对话框	使用changePage方法	3
如何在对话框中添加按钮	在对话框元素内添加a元素	4、5
如何配置对话框组件	使用配置data属性，或者调用dialog方法配置	6、7
如何关闭对话框	调用close方法	8
如何创建弹出菜单	在目标jQuery Mobile页（即需要此弹出菜单组件的页）内为div元素添加值为popup的data-role属性	9
如何配置弹出菜单组件	为打开弹出菜单的a元素添加配置data属性，或者为包含弹出菜单的div元素添加配置data属性，或者调用popup方法	10-12
如何使用脚本控制弹出菜单	使用open、close和reposition方法	13
如何响应弹出菜单内发生的变化	处理popup事件	14

## 29.1 jQuery Mobile 对话框组件

顾名思义，对话框组件负责向用户显示对话框。作为我们介绍的第一个jQuery Mobile组件，在本章中我将展示创建组件的多种方法，为本章后面的内容及后续章节打好基础。

### 29.1.1 创建对话框组件

如代码清单29-1所示，只要jQuery Mobile遇到data-rel属性为dialog的元素，就会自动创建对话框组件。

代码清单29-1 使用data-role属性声明创建对话框组件

```
<!DOCTYPE html>
<html>
<head>
```

```

<title>Example</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
<script type="text/javascript" src="jquery-1.10.1.js"></script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <a href="#dialog1">Show the dialog</a>
    </div>
  </div>
  <div id="dialog1" data-role="dialog">
    <div data-role="header">
      <h1>You clicked the link!</h1>
    </div>
    <div data-role="content">
      This is the content area of the dialog
    </div>
  </div>
</body>
</html>

```

这个例子页中有一个a元素，点击这个元素，会导航到名为dialog1的元素。而dialog1元素的data-role属性的值是dialog，即jQuery Mobile会以对话框的形式显示这个元素及其内容（图29-1）。

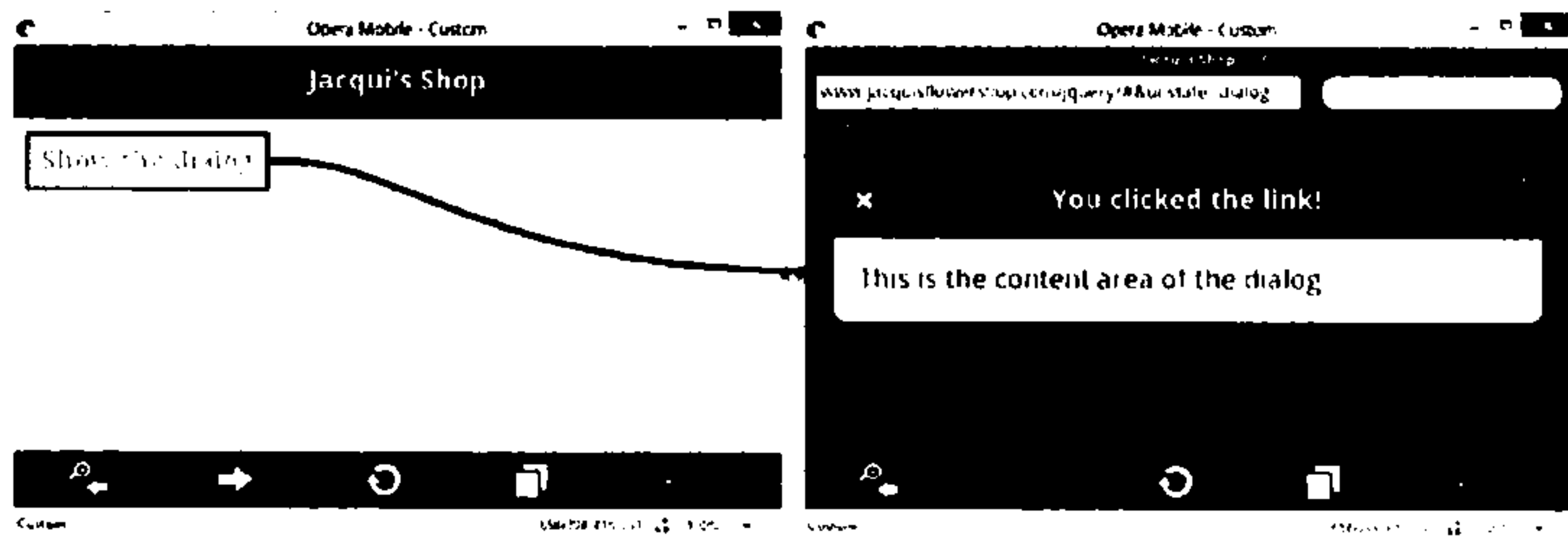


图29-1 以对话框形式显示一个页

为a元素添加data-rel属性并把它值设置为dialog，这样做也可以创建对话框。在代码清单29-2中，我就把一个a元素的data-rel属性的值设置为dialog，从而把一个data-role属性值为page的jQuery Mobile页转换成为对话框组件。

#### 代码清单29-2 使用data-rel属性把jQuery Mobile页转换为对话框

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>

```

```

<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
<script type="text/javascript" src="jquery-1.10.1.js"></script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <a href="#page2" data-rel="dialog">Show the dialog</a>
    </div>
  </div>
  <div id="page2" data-role="page">
    <div data-role="header">
      <h1>You clicked the link!</h1>
    </div>
    <div data-role="content">
      This is the content area of the dialog
    </div>
  </div>
</body>
</html>

```

在这个例子中有两个jQuery Mobile页。第一个页中有一个链接，它非常类似我们在第28章中用于导航的链接，只有一处例外：它有一个data-rel属性，属性的值是dialog。

### 1. 使用脚本生成对话框

虽然利用data-role声明生成对话框很有用，但它要求我们必须以固定的形式显示固定的HTML文本。当我们希望能根据情况动态决定是否要把一个页显示成对话框时，就可以通过changePage方法和它支持的角色选项来实现。代码清单29-3展示了如何利用changePage方法生成一个对话框。

#### 代码清单29-3 使用changePage方法生成对话框

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js">
  <script>
    $(document).bind("pageinit", function (event, data) {
      $("#dialogLink").click(function (e) {
        $.mobile.changePage(this.href, {
          role: "dialog"
        });
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>

```

```

<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <p><a id="dialogLink" href="#page2">Show the dialog</a></p>
      <p><a href="#page2">Show the page</a></p>
    </div>
  </div>
  <div id="page2" data-role="page">
    <div data-role="header">
      <h1>You clicked the link!</h1>
    </div>
    <div data-role="content">
      This is page 2
      <a href="#" data-role="button" data-rel="back">Close</a>
    </div>
  </div>
</body>
</html>

```

这一方法告诉我们，可以把同样一份内容派上多种用途。在这个例子中，我定义了两个链接，其中一个链接用常规方式（标准导航）载入一个jQuery Mobile页；另一个链接有些不同，我为它的click事件定义了事件处理函数。在这个事件处理函数中，我调用了changePage方法并把role选项设置为dialog。（见图29-2。）

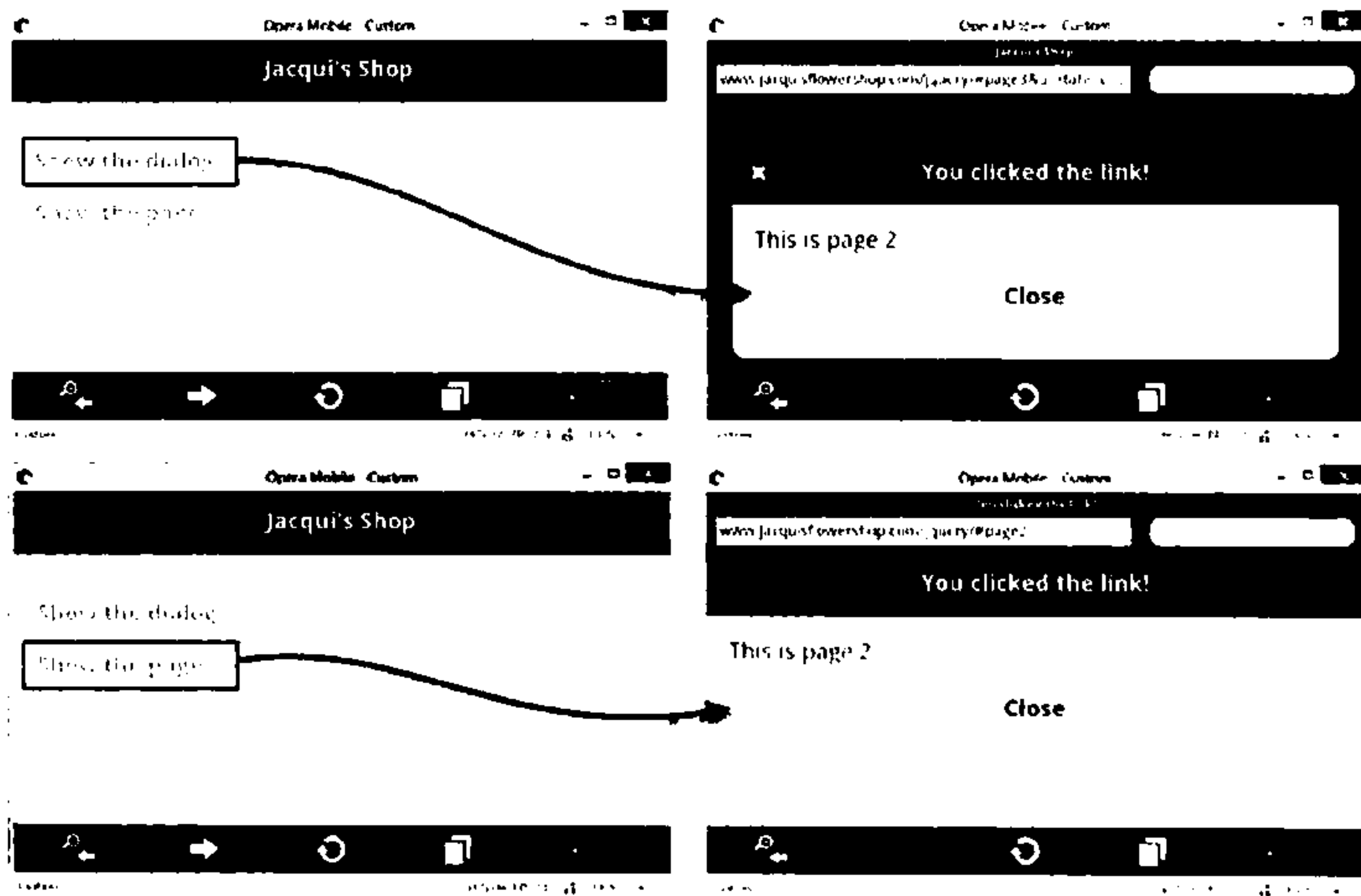


图29-2 使用脚本生成对话框

**注意** 务必慎用这项技术，因为jQuery Mobile会缓存role对应的内容，这就意味着一旦让一个页显示成对话框，它将总是显示成对话框，即便你再次调用changePage方法并把role设置成page（直到整个HTML页面重新加载才恢复正常）。

## 2. 在对话框中添加按钮

可以在对话框内容区添加一个a元素，并为它添加值为button的data-role属性，再添加一个值为back的data-rel属性。我会在下一章详细介绍jQuery Mobile的按钮组件，在代码清单29-4中，你可以了解到如何在对话框内使用按钮。

代码清单29-4 在对话框页中添加关闭按钮

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <a href="#page2" data-rel="dialog">Show the dialog</a>
    </div>
  </div>
  <div id="page2" data-role="page">
    <div data-role="header">
      <h1>You clicked the link!</h1>
    </div>
    <div data-role="content">
      This is the content area of the dialog
      <a href="#" data-role="button" data-rel="back">Close</a>
    </div>
  </div>
</body>
</html>
```

可以不为链接指定链接目的地，在这里把href属性设置为#，让jQuery Mobile自己决定链接到哪里。这非常有用，因为我们很可能在不同的页中显示对话框，并不知道是哪个页调用了这个对话框，因此不知道应该返回哪一个页。在对话框中添加按钮之后的效果见图29-3。

我们可以在对话框中再添加一个a元素，把它的href属性设置为我们希望显示的页。这正是我在代码清单29-5中所做的事情。在这个例子中，我添加了一个链接，并把它链接到同一HTML文档中的另一个jQuery Mobile页。

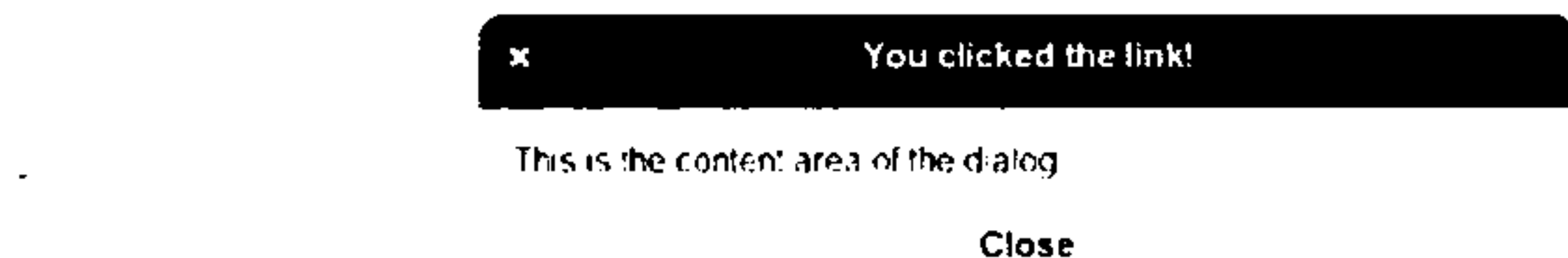


图29-3 在对话框中添加关闭按钮

**提示** 如果在点击按钮时不希望跳转到新页,可以处理a元素的click事件,并调用对话框组件的close方法关闭对话框,29.1.3节中就给出了这样一个例子。

#### 代码清单29-5 在对话框页中添加导航链接

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <a href="#page2" data-rel="dialog">Show the dialog</a>
    </div>
  </div>
  <div id="page2" data-role="page" data-overlay-theme="d">
    <div data-role="header">
      <h1>You clicked the link!</h1>
    </div>
    <div data-role="content">
      This is the content area of the dialog
      <a href="#page3" data-role="button">OK</a>
      <a href="#" data-role="button" data-rel="back">Close</a>
    </div>
  </div>
  <div id="page3" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      This is page 3. You came here via the dialog.
    </div>
  </div>
</body>
</html>
```



在这个例子中，我添加了一个a元素，它会把用户带到我刚刚在文档中添加的新页：page3。点击按钮跳到新页的效果图见图29-4。

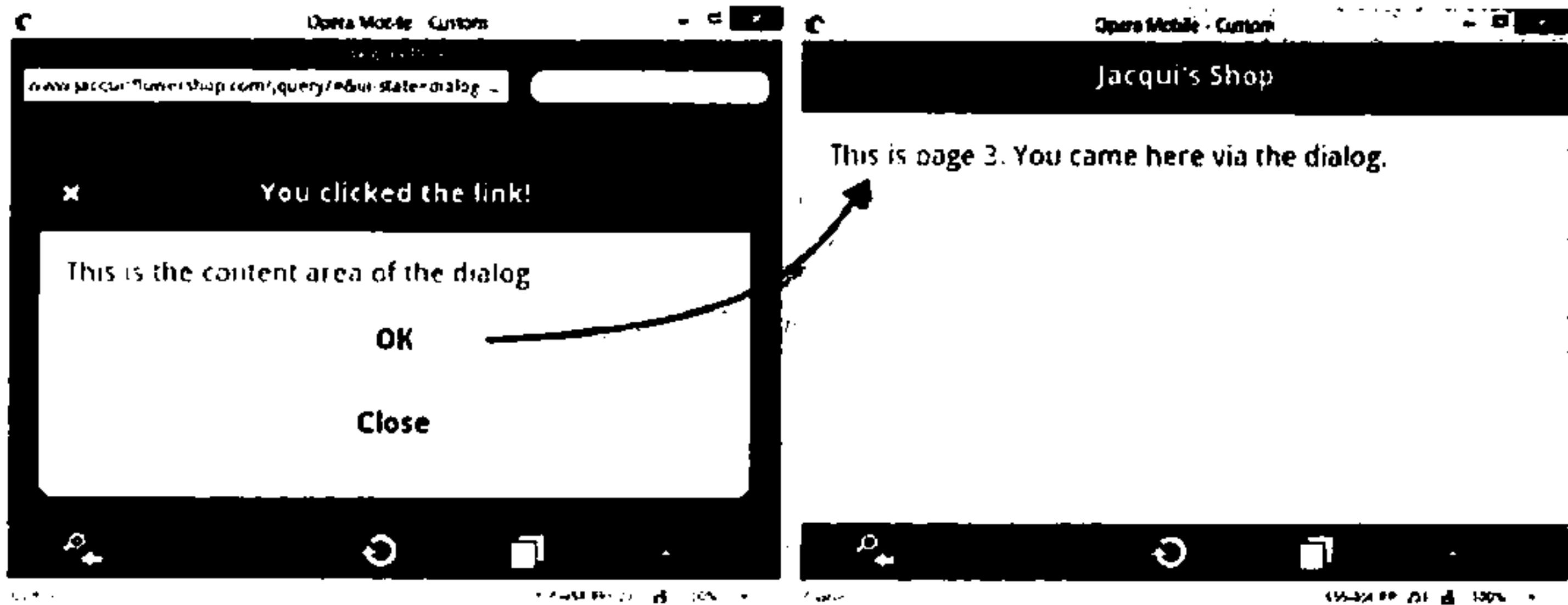


图29-4 在对话框中添加一个导航链接（按钮）

29.1.2 配置对话框组件

对话框组件支持许多选项，这些选项既可以使用data属性声明来配置，也可以通过调用JavaScript方法来配置。所有选项列于表29-2，下面将分别介绍如何使用data属性声明，以及如何使用脚本配置对话框组件。

表29-2 对话框组件选项

data属性	选 项	描 述
data-close-btn	closeBtn	获取或设置关闭按钮在对话框标题中的位置，支持的值有left、right和none
data-close-btn-text	closeBtnText	获取或设置位于对话框标题区域的关闭按钮的文本，这些文本并不显示给用户，但能为视图残障人士使用的阅读器类辅助设备检测到
data-corners	corners	获取或者设置对话框是否显示为圆角，默认值是true
data-overlay-theme	overlayTheme	获取或者设置绘制对话框所使用的主题，这个选项的值区分大小写，且必须使用小写

大多数情况下，使用data属性来配置组件（jQuery Mobile自动增强技术会识别并应用我们的设置）会更简单，也更容易。这是配置jQuery Mobile组件的标准做法。代码清单29-6演示了data-overlay-theme属性的用法。

代码清单29-6 使用data属性配置对话框组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
```

```

</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <a href="#dialog1">Show the dialog</a>
    </div>
  </div>
  <div id="dialog1" data-role="dialog" data-overlay-theme="d">
    <div data-role="header">
      <h1>You clicked the link!</h1>
    </div>
    <div data-role="content">
      This is the content area of the dialog
    </div>
  </div>
</body>
</html>

```

我把对话框的显示背景配置为d色系。具体效果见图29-5：对话框改用浅色背景而不再是默认的深色背景（a色系）。想要详细了解jQuery Mobile主题与色系，请阅读本书第28章。

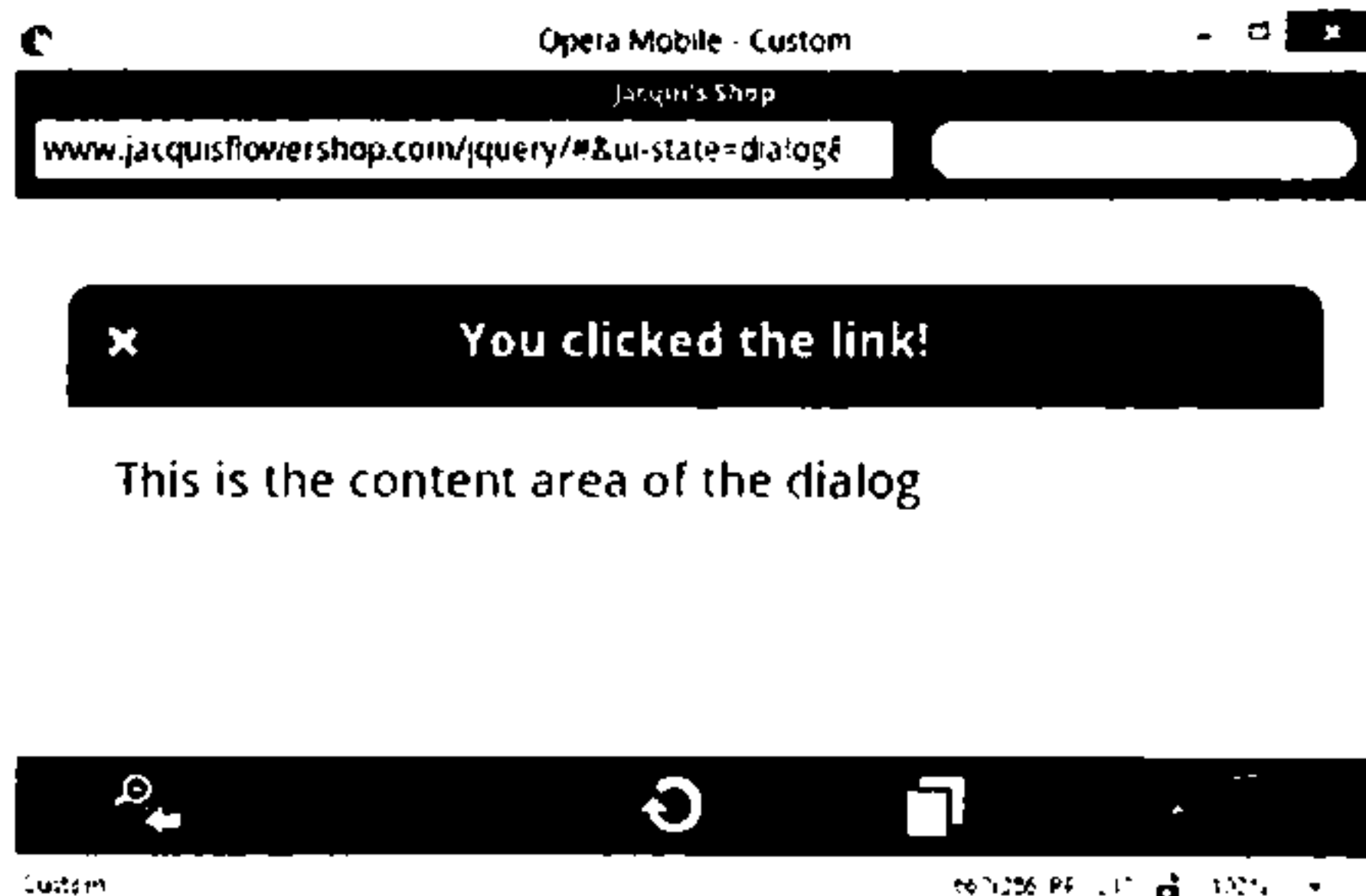


图29-5 使用data属性配置对话框组件

也可以在对话框创建完成之后再使用脚本修改对话框配置。调用jQuery Mobile的dialog方法，并把希望改变的配置选项（详见表29-2）以对象的形式传递给它。在代码清单29-7中，我演示了如何使用dialog方法改变对话框配置。

#### 代码清单29-7 使用脚本修改对话框配置

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />

```

```

<script type="text/javascript" src="jquery-1.10.1.js"></script>
<script>
    $(document).bind("pageinit", function () {
        $("#dialog1").dialog({
            corners: false,
            overlayTheme: "e"
        });
    });
</script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js">
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div data-role="content">
            <a href="#dialog1">Show the dialog</a>
        </div>
    </div>
    <div id="dialog1" data-role="dialog" data-overlay-theme="d">
        <div data-role="header">
            <h1>You clicked the link!</h1>
        </div>
        <div data-role="content">
            This is the content area of the dialog
        </div>
    </div>
</body>
</html>

```

我们可以在pageinit事件处理函数中配置对话框，在这个例子中，我先用jQuery选中对话框的底层元素(#dialog1)，接着调用dialog方法。传递给dialog方法的参数对象的属性都来自表29-2，同时还为corners和overlayTheme选项配置了新的值。传递给dialog方法的选项对象覆盖了data属性声明的值。具体效果见图29-6。

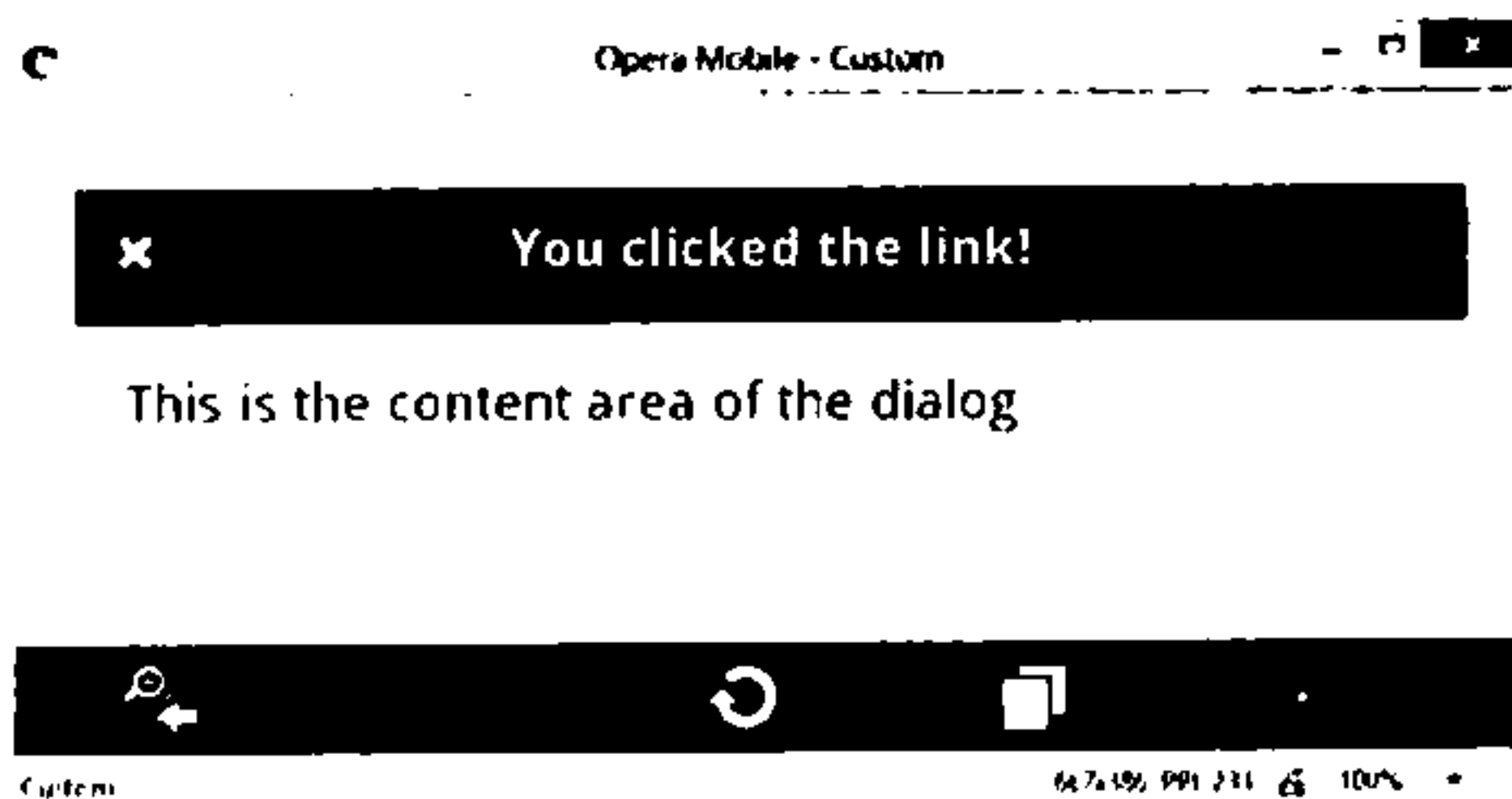


图29-6 使用脚本配置对话框

**警告** jQuery Mobile API文档推荐使用jQuery UI风格的option调用方法来配置dialog组件，如`$('#dialog1').dialog("option", "corners", false)`。然而，如果我们使用这种方法，就会收到出错信息。出错信息指出我们不能在对话框组件初始化之前调用dialog方法。这种配置方法只能在对话框组件显示出来之后调用，但那样就失去了配置对话框的意义。因此，请大家使用上面例子中使用的技术：在pageinit事件处理函数，传递一个配置对象给dialog方法

### 29.1.3 对话框组件方法

对话框组件只定义了一个方法，用来使用脚本关闭对话框。尽管如此，我还是为这个方法做了一个表（表29-3），便于你在需要时能快速找到它。

表29-3 对话框方法

方 法	描 述
<code>dialog("close")</code>	关闭对话框

在代码清单29-8中，我创建了一个对话框，用来显示一个固定的时间。除close方法之外，我还利用data-close-btn属性从菜单标题栏移除了关闭按钮。

代码清单29-8 对话框组件的close方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script>
    $(document).bind("pageinit", function () {
      $("#page1 a").click(function (e) {
        var duration = 15;
        $("#remaining").text(duration);
        var interval = setInterval(function () {
          $("#remaining").text(--duration);
          if (duration == 0) {
            clearInterval(interval);
            $("#dialog1").dialog("close");
          }
        }, 1000);
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
```

```

    </div>
    <div data-role="content">
        <a href="#dialog1">Show the dialog</a>
    </div>
</div>
<div id="dialog1" data-role="dialog" data-close-btn="none" data-overlay-theme="d">
    <div data-role="header">
        <h1>Important</h1>
    </div>
    <div data-role="content">
        This in an important message that will be displayed
        for <span id="remaining"></span> seconds.
    </div>
</div>
</body>
</html>

```

这个例子中script标签内的脚本为click事件定义了处理函数。当点击按钮打开对话框时，就会触发这个函数执行。使用脚本打开对话框的唯一办法是调用changePage方法，因此我只需要保持事件的默认行为（让对话框保持显示），并设置一个计数器，倒计时15秒。每过一秒更新一次对话框中span元素的内容，当倒计时完成时，调用对话框组件的close方法。这个例子的实际效果见图29-7。

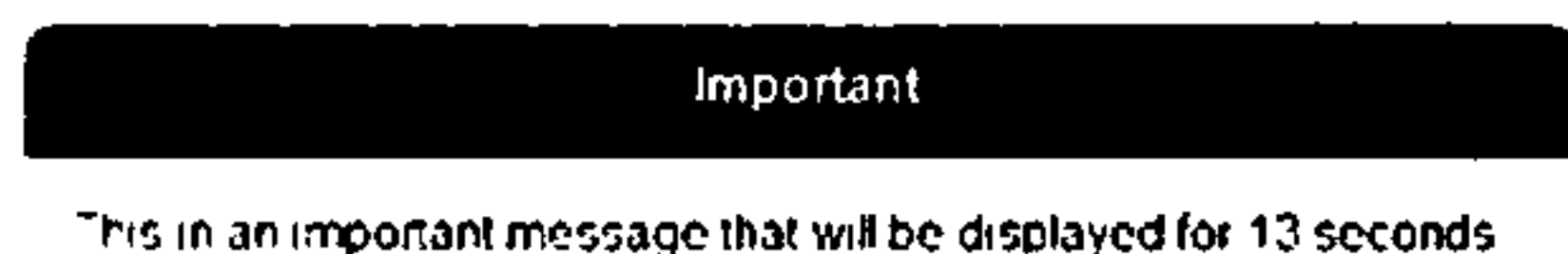


图29-7 会自动关闭的对话框

有两件事值得一提。第一，不必明确刷新对话框的内容以反映span元素内容的变更，对话框组件会自动更新。第二，即便创建对话框时没有提供关闭按钮，用户仍然可以使用浏览器的回退按钮取消对话框。

29

### 29.1.4 对话框组件事件

jQuery Mobile对话框组件只定义了一个事件，见表29-4。我从未用过这一事件，因为我更喜欢使用第28章介绍的那些“页”级别的事件。

表29-4 对话框组件事件

事 件	描 述
create	对话框组件创建完成时触发此事件

## 29.2 jQuery Mobile 弹窗组件

jQuery Mobile弹窗组件以弹窗的形式展示内容。与对话框相比，弹窗组件更轻便，也更可控。

### 29.2.1 创建弹窗

把元素的data-role属性设置成popup，这就创建了一个弹窗组件。弹窗不会直接显示出来，只有

用户点击一个data-rel属性也是popup且链接目标为弹窗元素的a元素时，才会弹窗。如代码清单29-9所示，弹窗组件与打开它的链接必须位于同一个jQuery Mobile页内。

代码清单29-9 创建jQuery Mobile弹窗组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <a href="#popup" data-rel="popup">Show the popup</a>
    </div>
    <div id="popup" data-role="popup">
      <p>This is the popup content</p>
    </div>
  </div>
</body>
</html>
```

如图29-8所示，点击链接会立即弹窗。点击弹窗之外的任何地方，都会取消这个弹窗。

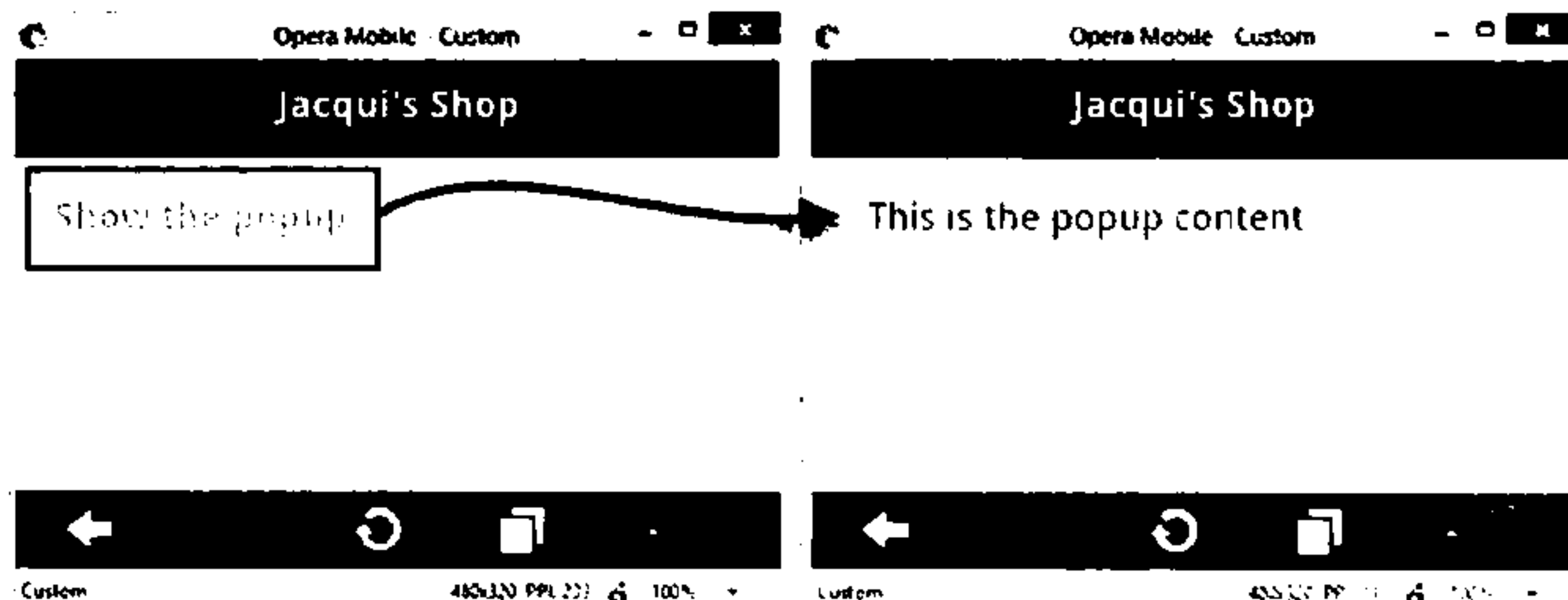


图29-8 生成jQuery Mobile弹窗

### 29.2.2 配置弹窗

从图中可以看出，弹窗盖住了打开它的链接，这通常不是我们期望的结果。有两种方法可以配置弹窗组件：配置打开弹窗的a元素，或者配置弹窗组件本身。

### 1. 配置打开弹窗的链接

配置a元素的优势在于，我们可以多次使用同一个弹窗组件，只需在调用时使用不同的配置即可。这种方法的缺点也很明显，一共只有两个配置选项可用，好在这两个选项都是我们想要的。a元素可用的data配置属性见表29-5。

表29-5 打开弹窗的a元素支持的data配置属性

data配置属性	描 述
data-position-to	指定弹窗相对于触发弹窗的a元素的位置，可取值见表29-6
data-transition	指定显示弹窗所使用的过场动画（参阅第28章了解jQuery Mobile支持的过场动画）

data-position-to属性指定弹窗相对于a元素的位置，这个属性的可取值见表29-6。

表29-6 data-position-to属性的可取值

值	描 述
origin	盖在a元素的正中间
window	位于当前窗口的正中间
selector	位于选择器匹配的第一个元素的正中间，如果该元素不可见，则弹窗位于当前窗口的正中间

在代码清单29-10中，我演示了如何使用data-position-to属性把弹窗的位置定位到参数选择器对应的元素上。

代码清单29-10 配置触发弹窗的a元素

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style>
    #anchor { position: absolute; right: 10px; }
  </style>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <a href="#popup" data-rel="popup"
        data-position-to="#anchor">Show the popup</a>
      <span id="anchor">Anchor</span>
    </div>
    <div id="popup" data-role="popup">
      <p>This is the popup content</p>
    </div>
  </div>
</body>
</html>
```

```
</body>
</html>
```

我使用selector选项，把#anchor指定为data-position-to属性的值。这个选择器匹配页中的一个div元素，我已使用样式把这个div元素定位到窗口的右边。最终效果见图29-9。

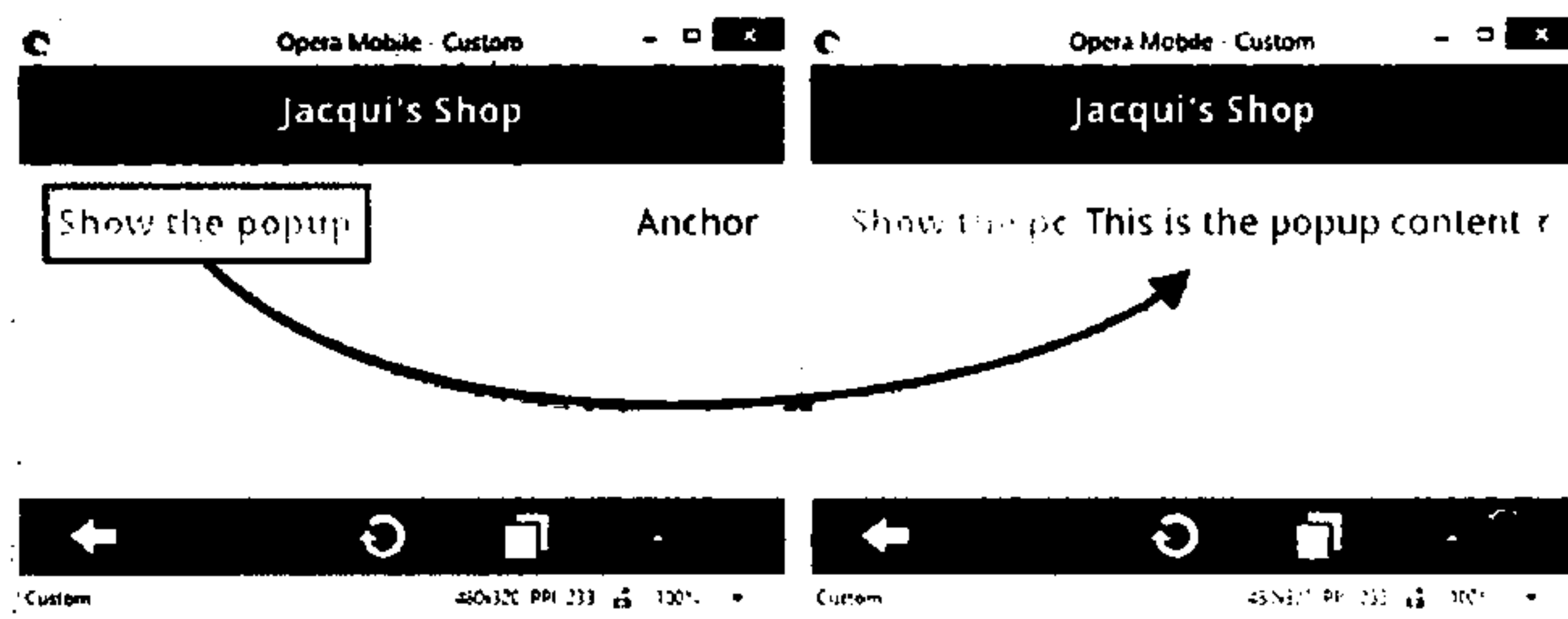


图29-9 使用a元素配置弹窗

## 2. 直接配置弹窗

直接配置弹窗有许多选项。与对话框组件类似，我们既可以使用data属性配置弹窗，也可以使用popup方法配置（详见表29-7）。绝大部分配置选项往往不言自明，或者与对话框组件类似。

表29-7 弹窗组件选项

data配置属性	选 项	描 述
data-corners	corners	配置弹窗的四角是否使用圆角，默认值为true
data-dismissable	dismissable	如果设置为false，当用户点击弹窗四周的空白，弹窗不会消失。默认值为true
data-history	history	决定弹窗前浏览器是否需要为它创建一条浏览记录。默认值是true，意味着用户点击浏览器返回按钮时，弹窗就会消失
data-overlay-theme	overlayTheme	设置弹窗背景所用主题。默认值是null，表示使用透明背景
data-position-to	positionTo	用表29-6中给出的值指定弹窗相对触发元素的位置
data-shadow	shadow	设置弹窗是否需要阴影效果。默认值为true
data-tolerance	tolerance	指定弹窗与窗口边界之间的最小距离。默认值为30, 15, 30, 15
data-transition	transition	指定弹窗显示与隐藏的转场动画

### ● history选项

data-history选项是弹窗组件最令人迷惑的选项，它决定弹窗之前浏览器是否需要为它创建一条浏览记录。这个选项的意义在于当用户点击浏览器返回按钮时，弹窗会消失。然而，由于弹窗的打开和关闭通常都由touch事件触发，这个选项并没有多少实际用处，如代码清单29-11所示。

代码清单29-11 弹窗保存历史选项的作用及鼠标触发弹窗

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
```



```

<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
<script type="text/javascript" src="jquery-1.10.1.js"></script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
<script>
    var mouseHandlerSet = false;
    $(document).bind("pageinit", function () {
        if (!mouseHandlerSet) {
            $("#page1 a").mouseenter(function (e) {
                $("#popup").popup("open", {
                    x: e.pageX, y: e.pageY
                });
            });
            $("#popup").mouseleave(function (e) {
                $(this).popup("close");
            });
            mouseHandlerSet = true;
        }
    });
</script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div data-role="content">
            <p><a href="#popup" data-rel="popup">Popup</a></p>
        </div>
        <div id="popup" data-role="popup" data-history="false">
            <p>This is the popup content</p>
        </div>
    </div>
</body>
</html>

```

这个例子中用到了本章后面将涉及的弹窗组件的open方法和close方法,当用户把鼠标移动到或指向a元素时,弹窗自动显示出来。当鼠标从弹窗移出时,弹窗又自动消失(我使用配置选项为open方法传递位置信息,确保弹窗总是位于鼠标位置之下)。

在这种情况下,由于并非是点击某个链接触发的弹窗,若点击返回按钮,用户可能不希望关闭弹窗。这是一个利用data-history属性创建连贯体验的好机会。

**提示** 建议慎用这个配置选项,因为理解背后的原理,只是关于弹窗组件用法困惑的一部分而已。

另一个问题是应该采纳哪种方法,这应该根据应用的实际情况决定。在这种情形下,很难预知用户会有什么样的体验,唯有用户测试能告诉我们哪种方式可以让Web应用创建更自然的用户体验。千万不要图省事不做测试,或者干脆把它作为一个配置项,因为大多数用户都将不一致的行为归咎于糟糕的实现,他们不会努力去改变什么。

### ● 使用弹窗显示格式内容

弹窗可以用来显示格式内容。如代码清单29-12所示，弹窗组件经常用于点击缩略图显示全尺寸大图の場合。（在Apress.com网站上与本书配套的下载包中包含示例使用的所有图片。）

代码清单29-12 使用弹窗显示格式内容

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style>
    .smallImage { height: 40%; width: 40%; padding: 5px}
  </style>
  <script>
    $(document).bind("pageinit", function () {

      var data = ["beach.png", "clouds.png", "fishing.png", "storms.png"];

      for (var i = 0; i < data.length; i++) {
        $("<img>").addClass("smallImage").attr("src",
          data[i]).appendTo("#contentHolder");
      }

      $("#popup").popup({
        corners: false,
        overlayTheme: "a"
      });

      $("#contentHolder img").bind("tap", function (e) {
        var maxHeight = $(window).height() - 10 + "px";
        $("#LgImage").attr("src", e.target.src).css("max-height", maxHeight);
        $("#popup").popup("open");
      });
    });
  </script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="d">
    <div id="contentHolder" data-role="content"></div>
    <div id="popup" data-role="popup" data-history="true">
      <img id="LgImage" class="zoomImg" src="" />
    </div>
  </div>
</body>
</html>
```

我使用了一个for循环来生成四个img元素，并把它们添加入jQuery Mobile页中，用来显示缩略图。还使用了一些标准样式来保证它们尺寸一致。我为tap事件（详见第27章）编写了处理函数，以选中图片作为弹出窗口的内容，然后调用open方法把窗口弹出来（29.2.3节中会详细介绍open方法）。这个

例子的实际效果见图29-10：用户点击一张缩略图，就立即弹窗显示了该图片的大尺寸版本。

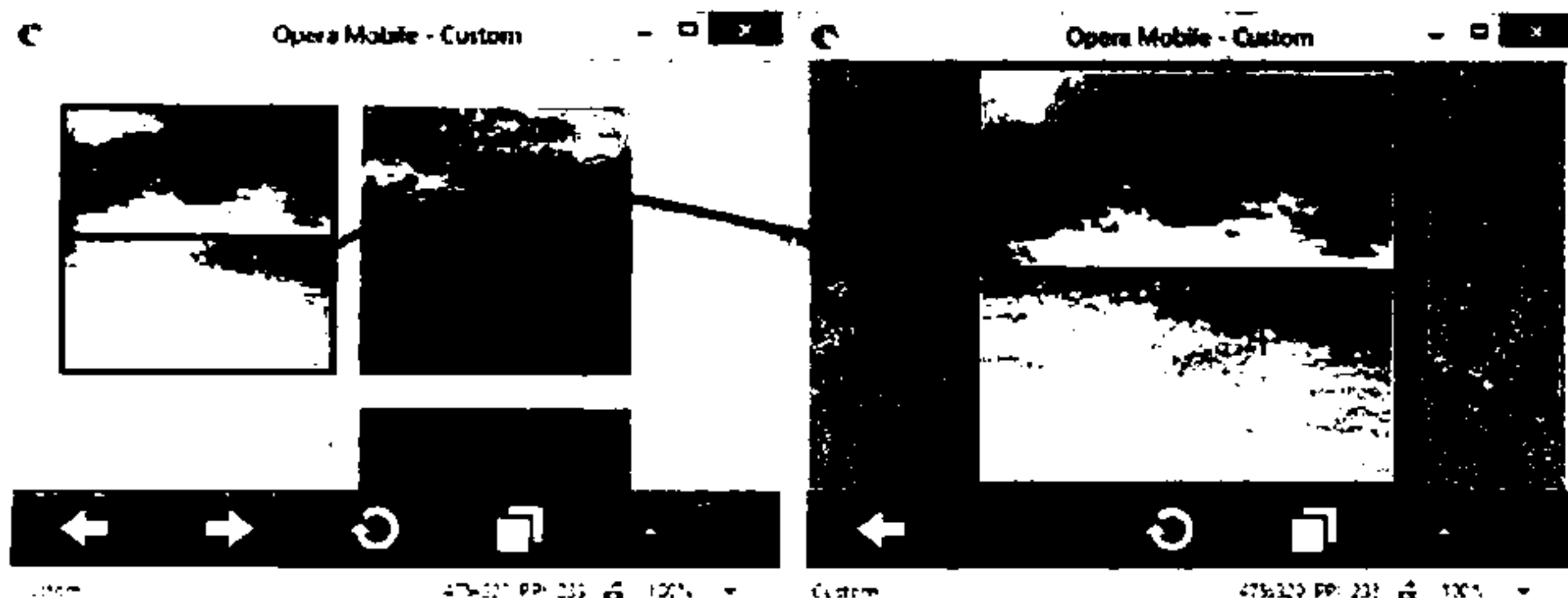


图29-10 用弹窗展示图片

混用data属性和选项以配置弹窗组件是为了演示一下它们的用法。弹窗组件的底层元素div定义了一个data-history属性，如下面的代码所示：

```
...
<div id="popup" data-role="popup" data-history="true">
  <img id="lgImage" class="zoomImg" src="" />
</div>
...
```

其实data-history属性的默认值就是true，但由于该选项很容易让人产生困惑，所以在这里会加以详细说明。在script元素中，我使用popup方法设置了其他选项，代码如下：

```
...
$("#popup").popup({
  corners: false,
  overlayTheme: "a"
});
...
```

使用圆角会削掉大图的部分内容，因此指定corners属性为方角。我希望大图显示出来时，能充分吸引用户的注意力，因此我把弹窗的overlayTheme属性设置为深色背景。

使用弹窗显示的内容大于弹窗本身时，会需要一些额外的工作。这是因为默认情况下，内容是会滚动的（会出现滚动条）。为了避免这种情况发生，我使用下面的代码获取窗口的高度，并使用它作为弹窗内图片的高度：

```
...
var maxHeight = $(window).height() - 10 + "px";
$("#lgImage").attr("src", e.target.src).css("max-height", maxHeight);
...
```

我让图片比窗口小10像素，这样就为图片的边框留出了空间，边框能强化图片的弹出效果。理论上，应该使用tolerance选项对齐边框，然而直到我写到这里时，这个选项仍然相当不稳定。因此，就用最直白的值得到所期望的效果。我制作了beach.png图片，即第一张缩略图，它比其他几张图都要大。在图29-11中，你会看到如果不设置max-height属性的话，会发生什么事：图片的顶和底都将显示不全，而且没有任何可见的线索告诉用户可以通过滚动浏览图片的其余部分。

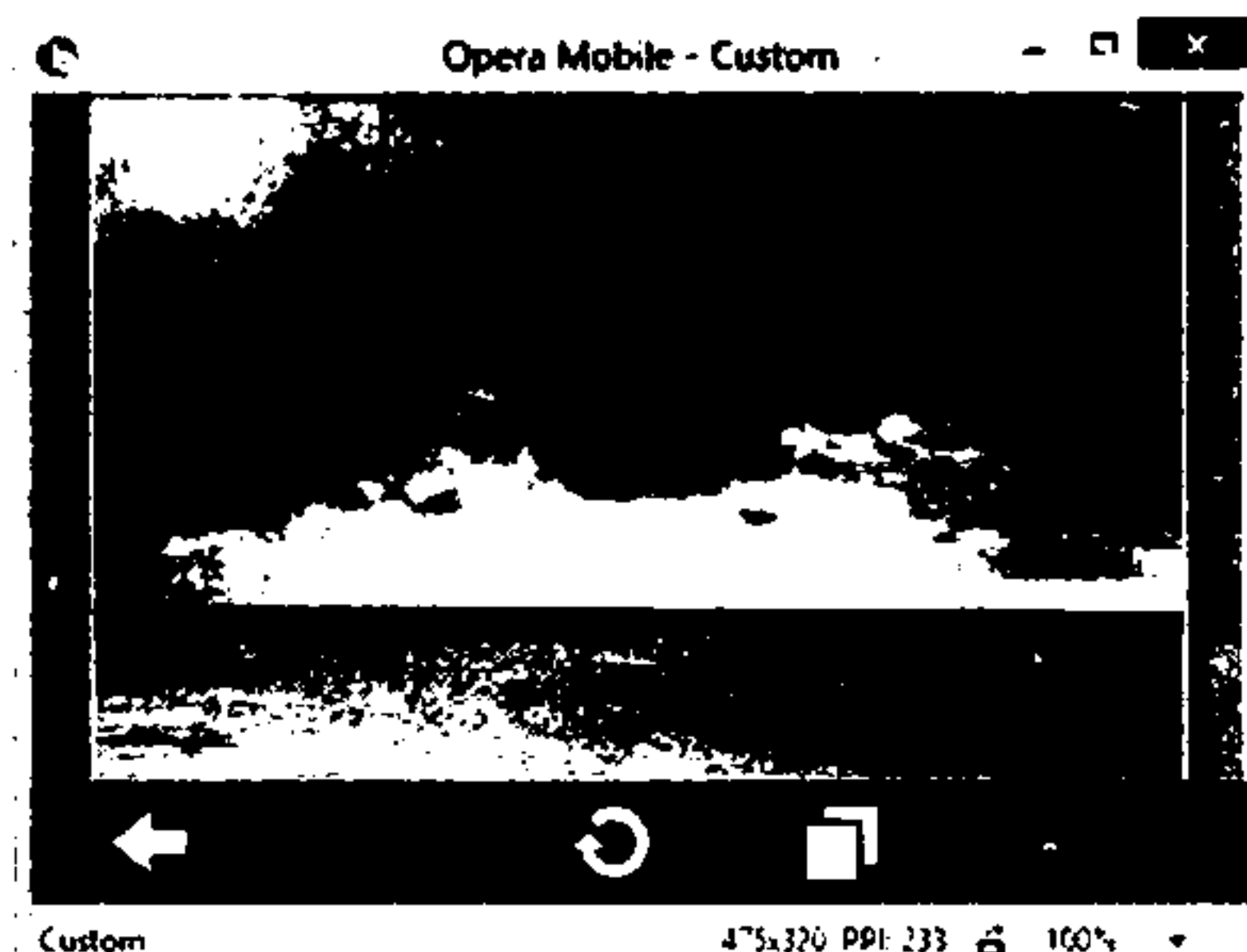


图29-11 显示内容大于弹窗尺寸时的效果

### 29.2.3 弹窗组件的方法

弹窗组件定义的方法见表29-8。

表29-8 弹窗组件方法

方 法	描 述
<code>popup("open")</code>	弹窗
<code>popup("close")</code>	关闭弹窗
<code>popup("reposition")</code>	改变弹窗的位置

在前面的例子中已经演示过`open`方法的基本用法，除此之外，`open`方法还支持一个可选参数，用来控制窗口的打开方式。这个参数的功能与配置`a`元素所用的`data`属性相同。这个可选参数是一个对象，它支持的属性见表29-9。

表29-9 `open`方法可选的参数对象所支持的属性

名 称	描 述
<code>x</code>	指定弹窗位置的x坐标值
<code>y</code>	指定弹窗位置的y坐标值
<code>transition</code>	指定弹窗所用过场动画（详见第28章）
<code>positionTo</code>	指定弹窗的位置，可取值见表29-6

`reposition`方法支持的参数也是一个配置对象，我们可以用`x`、`y`和`positionTo`属性指定弹窗新的显示位置。`close`方法不接受任何参数，它直接让弹窗消失。在代码清单29-13中，我演示了这三个方法的用法。

#### 代码清单29-13 应用弹窗组件的方法

```
<!DOCTYPE html>
<html>
```

```

<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <script>
    $(document).bind("pageinit", function () {
      $("button").bind("vmousedown", function (e) {
        var pop = $("#popup");
        switch (e.target.innerText) {
          case "Open":
            pop.popup("open", {
              x: 10, y: 10, transition: "fade"
            });
            break;
          case "Close":
            pop.popup("close");
            break;
          default:
            pop.popup("reposition", {
              positionTo: e.target.innerText == "Window"
                ? "window" : "#page1 button"
            });
            break;
        }
      });
    });
  </script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="d">
    <div data-role="content">
      <button>Open</button>
    </div>
    <div id="popup" data-role="popup" data-history="true"
      data-dismissible="false">
      <button>Selector</button>
      <button>Window</button>
      <button>Close</button>
    </div>
  </div>
</body>
</html>

```

在这个例子中，jQuery Mobile页中有一个按钮，我在它的vmousedown事件处理函数中，调用open方法以弹出窗口。为open方法传递一个配置对象，让弹窗显示在指定的位置（其x坐标与y坐标均距左上角10像素）以淡入的方式显示出来。

这个例子中的弹窗包含几个按钮，其中Close按钮调用close方法来关闭弹窗，其他的按钮则调用reposition方法为弹窗指定一个新的显示位置。这个例子的实际效果见图29-12。

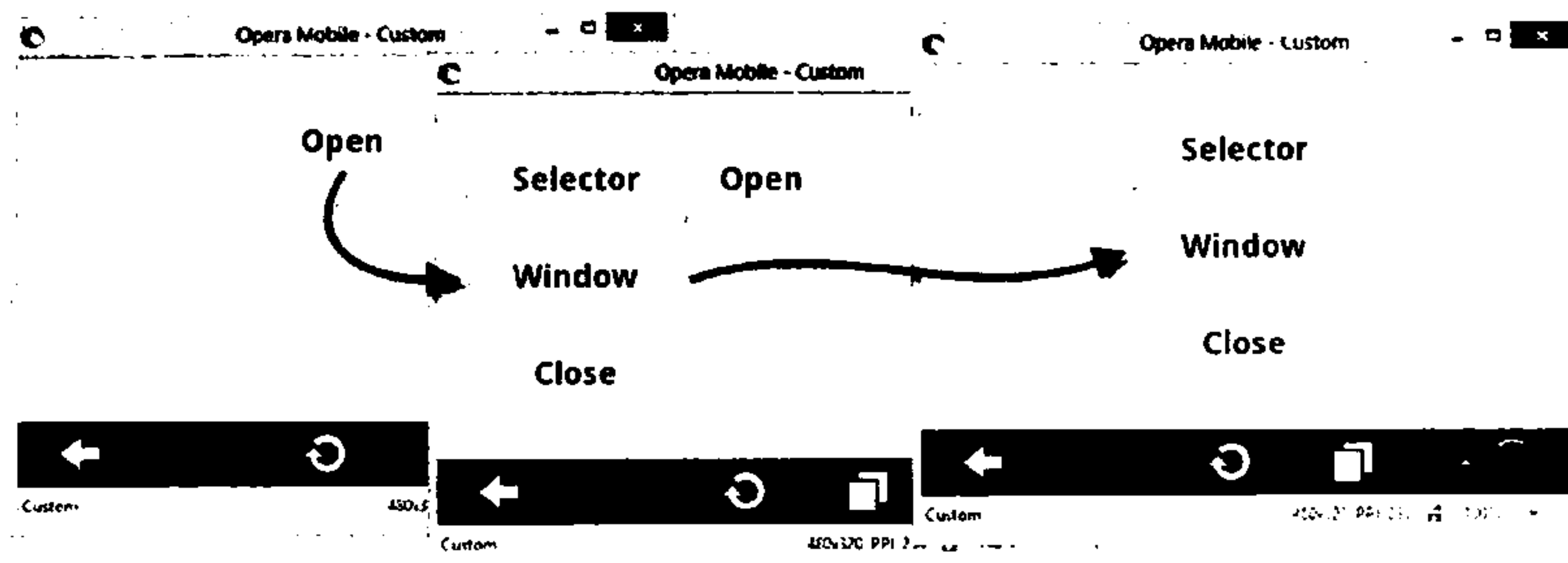


图29-12 应用弹窗组件的方法

**提示** 注意，在这个例子中我把data-dismissible属性设置成了false。如果没有这个选项，绝大多数触屏浏览器所虚拟出来的鼠标事件（详见第27章）都会被解释成弹窗之外的click事件，这会导致弹窗刚一打开就自动关闭。把data-dismissible属性设置成false就能避免这种情况发生：只有点击Close按钮时才调用close方法。

## 29.2.4 弹窗组件事件

jQuery Mobile支持的事件见表29-10。

表29-10 弹窗组件事件

事 件	描 述
create	在弹窗组件创建完成时触发
beforeposition	在弹窗显示位置发生变化时触发
afteropen	当弹窗完整显示之后触发
afterclose	当弹窗完全消失之后触发

这些事件鲜有派上用场之时。我对弹窗组件的使用也仅限于方法和配置选项。有一个例外，如代码清单29-14所示，当我想让一个弹窗在有限的时间内显示时，afteropen事件正当其用。

代码清单29-14 处理弹窗事件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script>
```

```

        $(document).bind("pageinit", function () {
            $("#popup").popup({
                afteropen: function (e) {
                    setTimeout(function () {
                        $("#popup").popup("close");
                    }, 5000);
                }
            });
        });
    </script>
    <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
    <div id="page1" data-role="page">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div data-role="content">
            <a href="#popup" data-rel="popup">Show the popup</a>
        </div>
        <div id="popup" data-role="popup" data-position-to="window">
            <p>This is the popup content</p>
        </div>
    </div>
</body>
</html>

```

**警告** 只有当无法说服顾客改用其他解决方案时，我才会使用这个方法。我更喜欢由用户来控制弹窗，这种显示或者关闭弹窗都不需要用户参与的技术，只会为用户带来困扰。

在这个例子中，我利用afteropen方法调用setTimeout函数，注册了一个5秒之后运行的回调函数，这个回调函数则调用close方法关掉了弹窗。

## 29.3 小结

本章介绍了jQuery Mobile的第一批组件：对话框组件与弹窗组件。jQuery Mobile组件的基本结构与设计理念与jQuery UI组件并无二致，但它们针对移动设备做了许多令人瞩目的优化。在下一章，我将介绍按钮和折叠块组件。

本章将介绍另外两个jQuery Mobile组件：按钮和折叠块。jQuery Mobile的按钮与本书前面部分讲到的jQuery UI按钮工作方式非常相似，唯一的不同在于我们无需手写任何代码，只要正确设置data属性，就能生成简单的按钮。

折叠块组件有点像折叠菜单组件中的一块面板；实际上，我们可以单独使用折叠块，也可以把几个折叠块组合成一个简单的折叠菜单。表30-1列出了本章概要。此外，在本章中我还介绍了同一主题的另外两个组件：一个是导航栏组件，它把几个按钮排成外观一致的一组，用于导航支持；另一个是折叠集组件，我们主要用它生成折叠菜单。

表30-1 本章概要

问 题	解决方法	代码清单
如何自动生成一个按钮	在页面中添加一个button元素或者一个type属性值为submit、reset或button的input元素	1
如何使用其他元素生成按钮	添加data-role属性，并把它值设置为button	2
如何得到一组按钮	添加data-role属性，并把它值设置为controlgroup，使用data-type属性设置排列方向	3、4
如何在按钮上添加图标，并调整图标的位置	使用data-icon和data-iconpos属性	5
如何生成较小的按钮	使用data-mini和data-inline属性	6
如何根据底层元素的变化更新按钮状态	使用refresh方法	7
如何响应按钮事件	响应底层元素的事件	8
如何生成外观一致的导航按钮	使用导航栏（navbar）组件	9
如何设置导航栏图标的位置	使用data-iconpos属性	10
如何生成一个折叠块	添加data-role属性，把它的值设置为collapsible，并保证它的第一个子元素是标题元素	11、12
如何响应折叠块的扩展或者收起事件	处理collapse和expand事件	13
如何生成一个折叠菜单	添加data-role属性，并把它值设置为collapsible -set	14

## 30.1 jQuery Mobile 按钮

在前面的例子里我们已经用过一些按钮，现在是时候回顾并解释它们的工作原理了。



### 30.1.1 生成按钮

作为页面自动增强的一部分，jQuery Mobile自动把button元素和type属性为submit、reset、button或image的input元素增强为按钮。只要元素符合条件，我们什么都不用做，jQuery Mobile会自动做好一切事情。在代码清单30-1中，有一些由jQuery Mobile自动生成的按钮。

代码清单30-1 自动生成的按钮

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <button>Button</button>
      <input type="submit" value="Input (Submit)" />
      <input type="reset" value="Input (Reset)" />
      <input type="button" value="Input (Button)" />
    </div>
  </div>
</body>
</html>
```

在图30-1中，可以看到由jQuery Mobile自动生成的各种按钮。

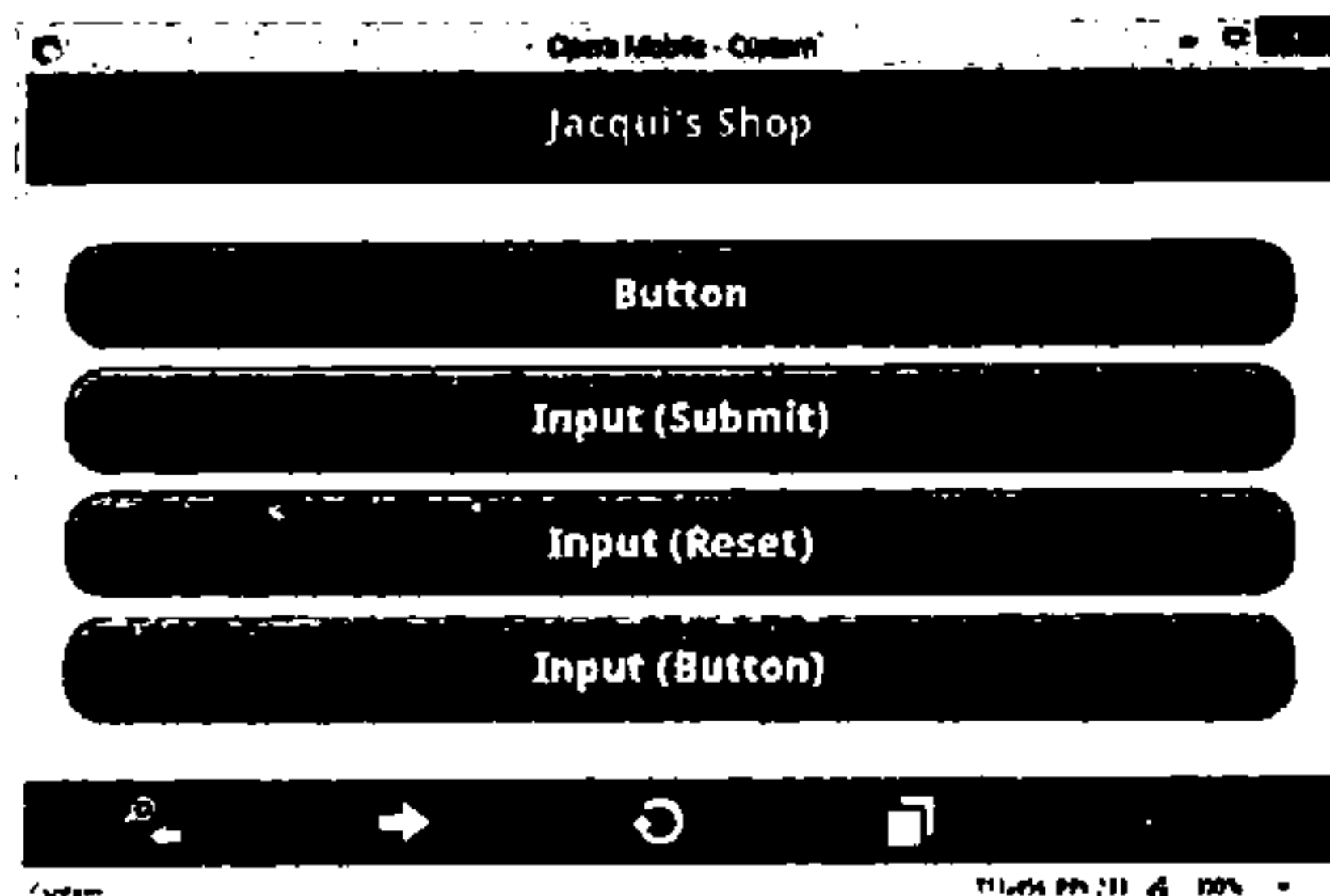


图30-1 jQuery Mobile自动生成的按钮

#### 1. 利用其他元素生成按钮

jQuery Mobile也可以利用其他元素生成按钮。在前面的章节中，我们把a元素的data-role属性设

置为button,从而得到了一个按钮。这一技术也适用于其他的元素,比如div。具体例子见代码清单30-2

代码清单30-2 利用其他元素生成按钮

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <a href="document2.html" data-role="button">A Element</a>
      <div data-role="button">DIV Element</div>
    </div>
  </div>
</body>
</html>
```

经jQuery Mobile处理之后,这个页面中各按钮的显示效果见图30-2。

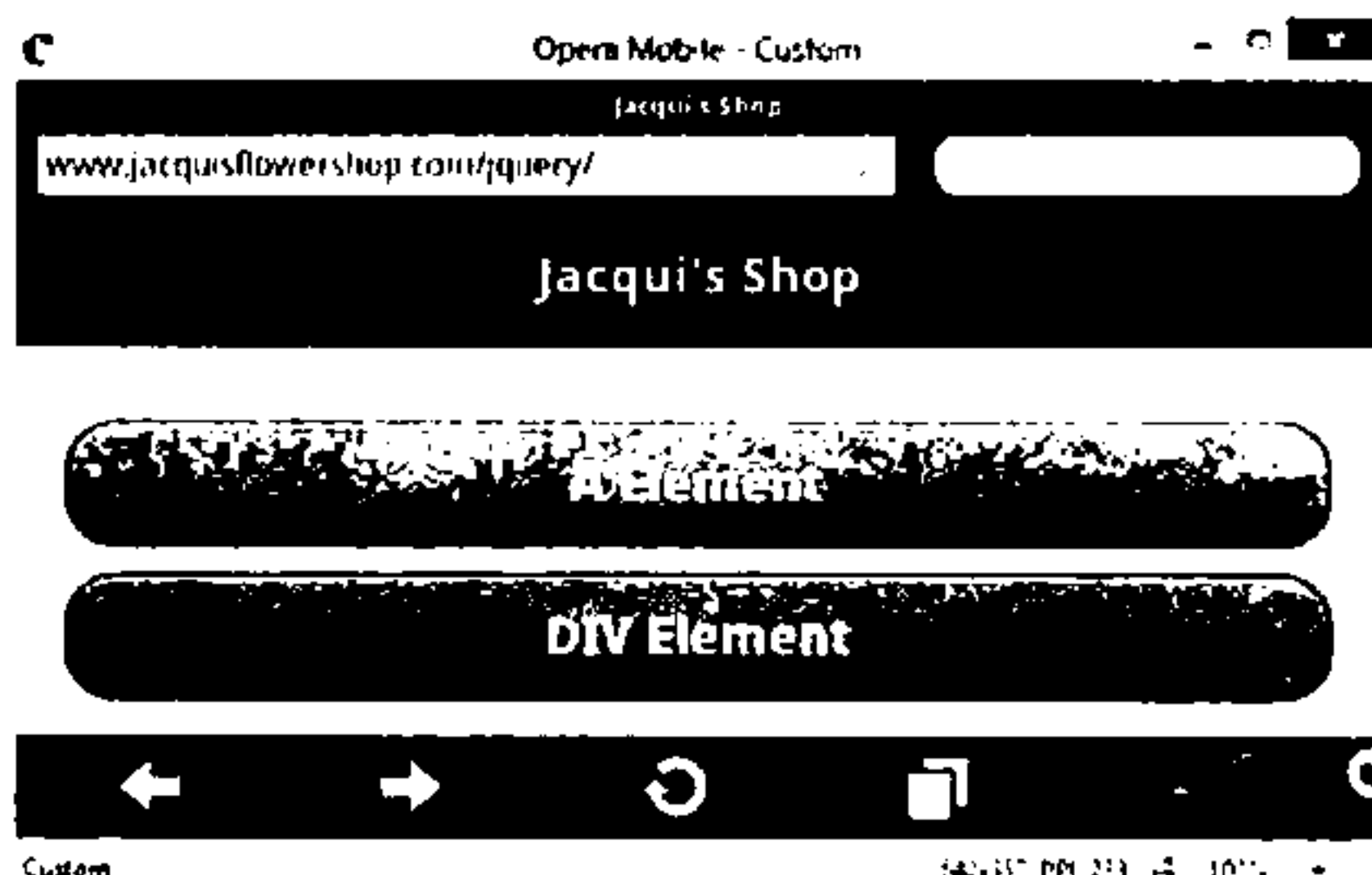


图30-2 使用其他元素生成的按钮

## 2. 生成组合按钮

利用“控制组”,我们能得到一组没有间距的按钮。只需把一组按钮的父元素的data-role属性设置成controlgroup,就能得到这样的一组按钮。具体例子见代码清单30-3。

代码清单30-3 生成一组按钮

```
<!DOCTYPE html>
<html>
<head>
```

```

<title>Example</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
<script type="text/javascript" src="jquery-1.10.1.js"></script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <div data-role="controlgroup">
        <button>Back</button>
        <button>Home</button>
        <button>Next</button>
      </div>
    </div>
  </div>
</body>
</html>

```

在这个例子中一共有三个按钮，它们都是

[data-role=controlgroup]元素的子元素。具体效果见图30-3。注意，仅有最上面的按钮（的上面两个角）和最下面的按钮（下面的两个角）具有圆角效果。

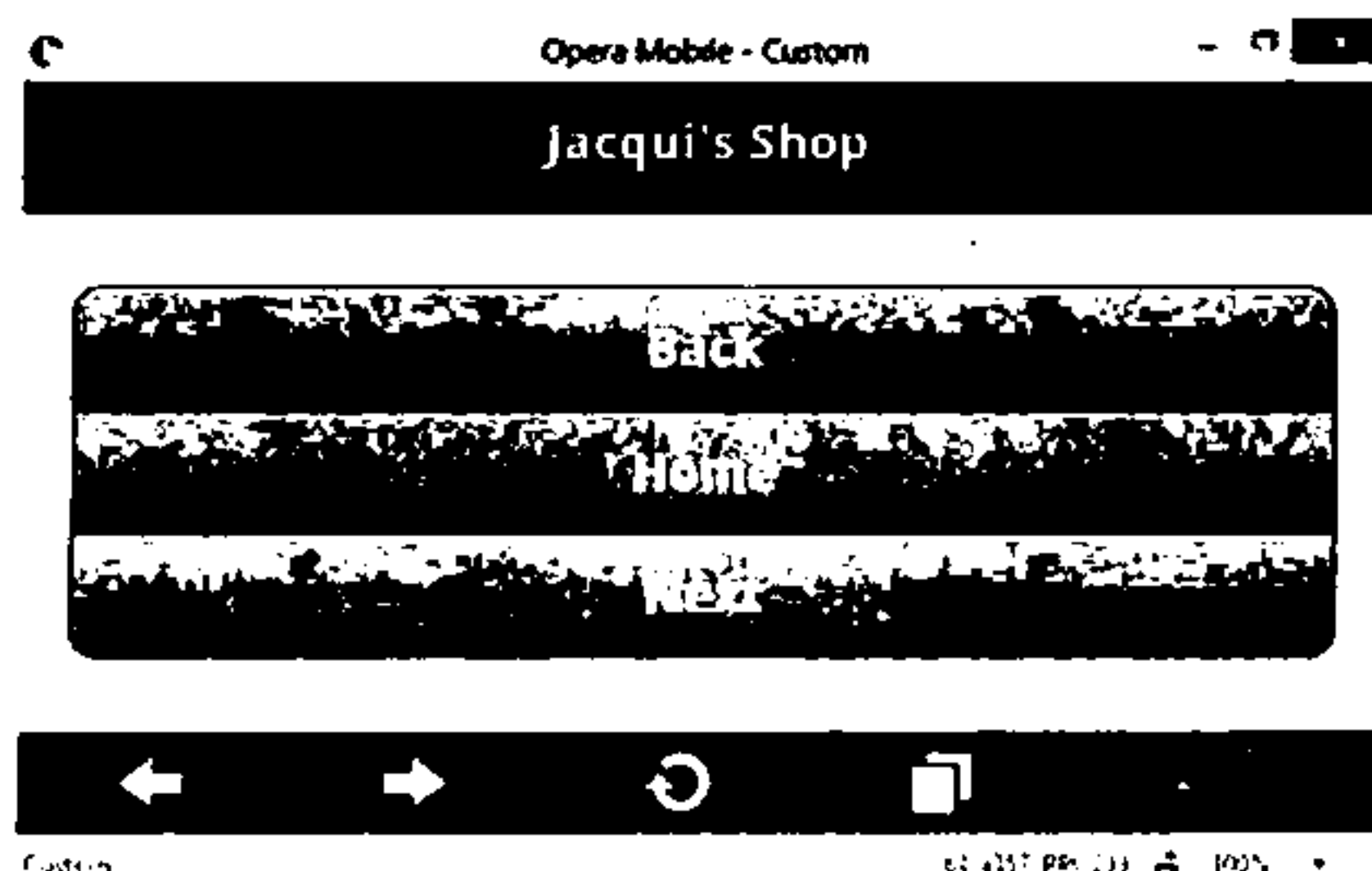


图30-3 一组按钮

如代码清单30-4所示，要让这些按钮由垂直排列变为水平排列，我们只需把这些按钮父元素的data-type属性设置成horizontal。

#### 代码清单30-4 生成一组水平排列的按钮

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>

```

```

</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <div data-role="controlgroup" data-type="horizontal">
        <button>Back</button>
        <button>Home</button>
        <button>Next</button>
      </div>
    </div>
  </div>
</body>
</html>

```

这个例子中按钮的显示效果见图30-4，请注意只有最外侧按钮的外侧两个角具有圆角效果。

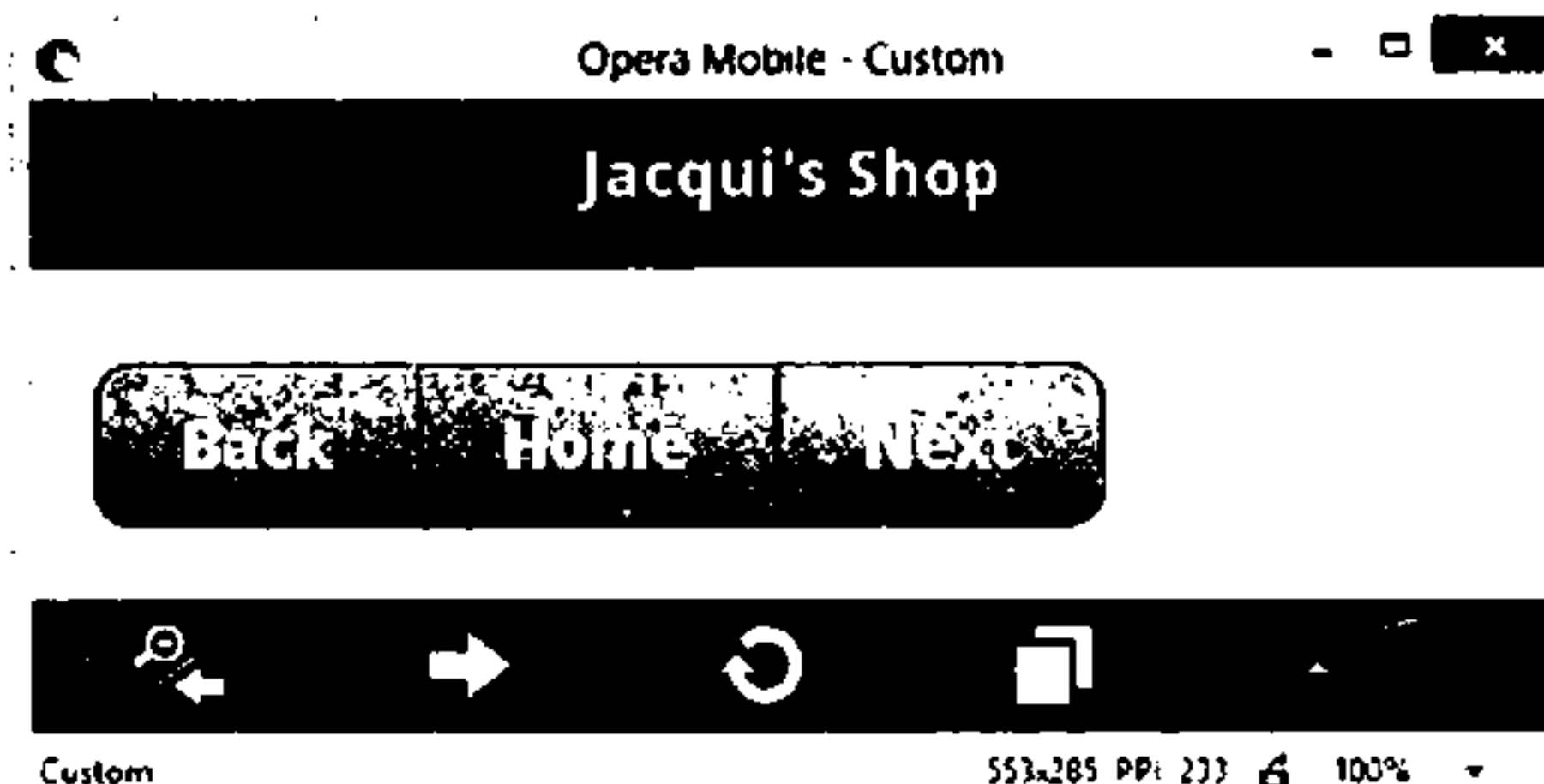


图30-4 生成一组水平排列的按钮

### 30.1.2 配置 jQuery Mobile 按钮组件

jQuery Mobile定义了一些以data为前缀的属性和配置选项，它们既可用来配置按钮，也可用于使用其他元素生成按钮。表30-2列出了这些属性，在接下来的内容中，我会演示与按钮有关的属性的用法。

表30-2 按钮的数据属性及配置选项

数据属性	选项	描述
data-corners	corners	若值为true，jQuery Mobile生成圆角按钮，若值为false，则生成直角按钮。默认值是true
data-icon	icon	指定按钮使用的icon
data-iconpos	iconpos	如果按钮使用了icon，该属性指定按钮icon的位置
data-iconshadow	iconshadow	若设置为true，则为图标添加阴影
data-inline	inline	生成一个根据内容自动设置宽度的按钮（而不是屏幕有多宽，它就多宽）
data-mini	mini	若设置为true，则显示为紧凑型按钮
data-shadow	shadow	若值为true，jQuery Mobile会画阴影，若值为false，则不画阴影。默认值是true

## 1. 为按钮添加图标

jQuery Mobile内置了一套可以用于按钮的图标，这些图标保存在images文件夹（我们在第27章中安装的）内的一个图片文件中。表30-3列了这些图标，并进行了简要介绍。

表30-3 jQuery Mboile内建图标

图 标	描 述
arrow-l, arrow-r, arrow-u, arrow-d	向左、向右、向上、向下的箭头
bars	三条水平线
edit	一只铅笔，提示用户内容可修改
Check, delete	对号和错号
Plus, minus	加号和减号
gear	齿轮
Refresh, forward, back, home, search	浏览器风格的刷新、前一页、后一页、首页及搜索图标（放大镜）
grid	网格
star	五角星
alert	表示警告的感叹号图标
info	表示信息固有格式的字母i

data-icon属性用于为按钮设置图标，把data-icon的值设置成图标的名字，就能得到相应的图标。data-iconpos属性用于指定图标在按钮中的位置。默认情况下图标位于按钮的左侧，我们也可把显示位置指定为上部、下部或者右侧。如果将data-iconpos设置为notext，则只显示图标而不显示按钮文本。代码清单30-5演示了这些属性的用法。

### 代码清单30-5 为按钮添加图标

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <div class="ui-grid-b">
        <div class="ui-block-a">
          <button data-icon="home">Home</button>
        </div>
        <div class="ui-block-b">
          <button data-icon="home" data-iconpos="top">Home</button>
        </div>
      </div>
    </div>
  </div>
```

```

        <div class="ui-block-c">
            <button data-icon="home" data-iconpos="notext"></button>
        </div>
    </div>
</div>
</div>
</div>
</body>
</html>

```

在这个例子中，我添加了三个按钮，每个按钮都用到了home图标。第一个按钮图标位于默认位置，第二个按钮图标位于按钮的上半部分，最后一个按钮使用了notext值，因此仅有图标，没有文本。这些按钮的实际显示效果见图30-5。

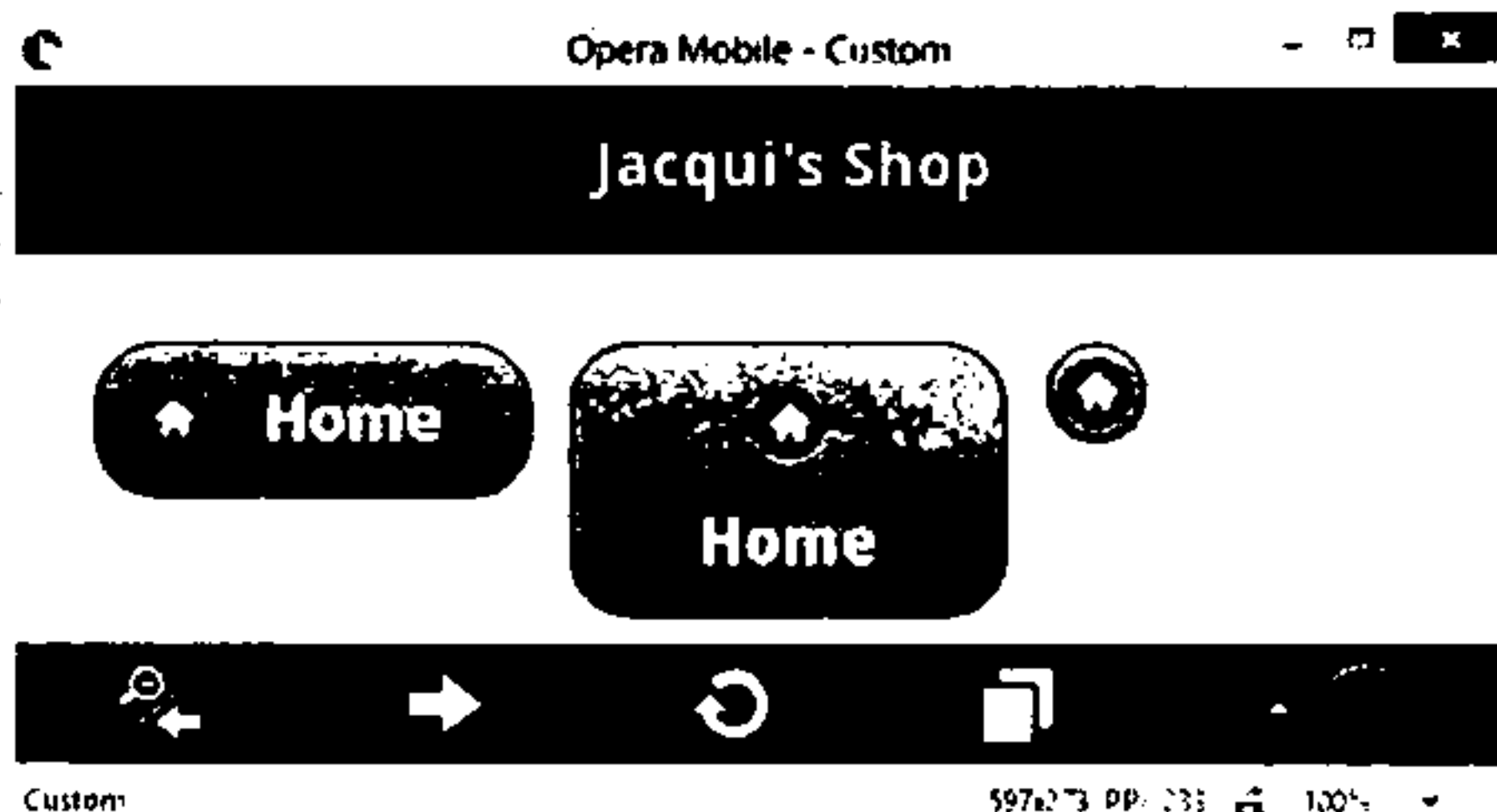


图30-5 生成带有图标的按钮

很显然，每个按钮的样式都是独立的。最引人注目的是最后一个按钮，它没有任何文本，它的视觉效果确实很棒，不过经验告诉我们，用手指点中这种按钮并不容易，也不是所有的用户都能一眼看出这个按钮会把他们带往什么地方。

## 2. 行内按钮与紧凑型按钮

默认情况下，jQuery Mobile按钮会占据屏幕的整个宽度，图30-1已经展示了默认按钮的效果。在后面的例子中我会使用布局栅格生成一些较小的按钮。还有一种办法也能实现类似的效果，这就是行内按钮，它们不大也不小，刚好适应按钮内容的尺寸。只需把data-inline属性设置为true，即可得到行内按钮。

把data-mini属性设置为true，可以得到紧凑型按钮。与标准按钮相比，这种按钮宽度不变，但高度较小。当然，也可以同时使用这两个data属性，从而得到紧凑的行内按钮。与标准按钮相比，它们的宽度与内容匹配，并且占用较小的屏幕高度。代码清单30-6展示了这三种按钮。

### 代码清单30-6 行内按钮与紧凑型按钮

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.10.1.js"></script>

```

```

    <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div data-role="content">
            <button data-icon="home" data-inline="true">Home</button>
            <button data-icon="home" data-mini="true">Home</button>
            <button data-icon="home" data-inline="true" data-mini="true">Home</button>
        </div>
    </div>
</body>
</html>

```

具体显示效果见图30-6。

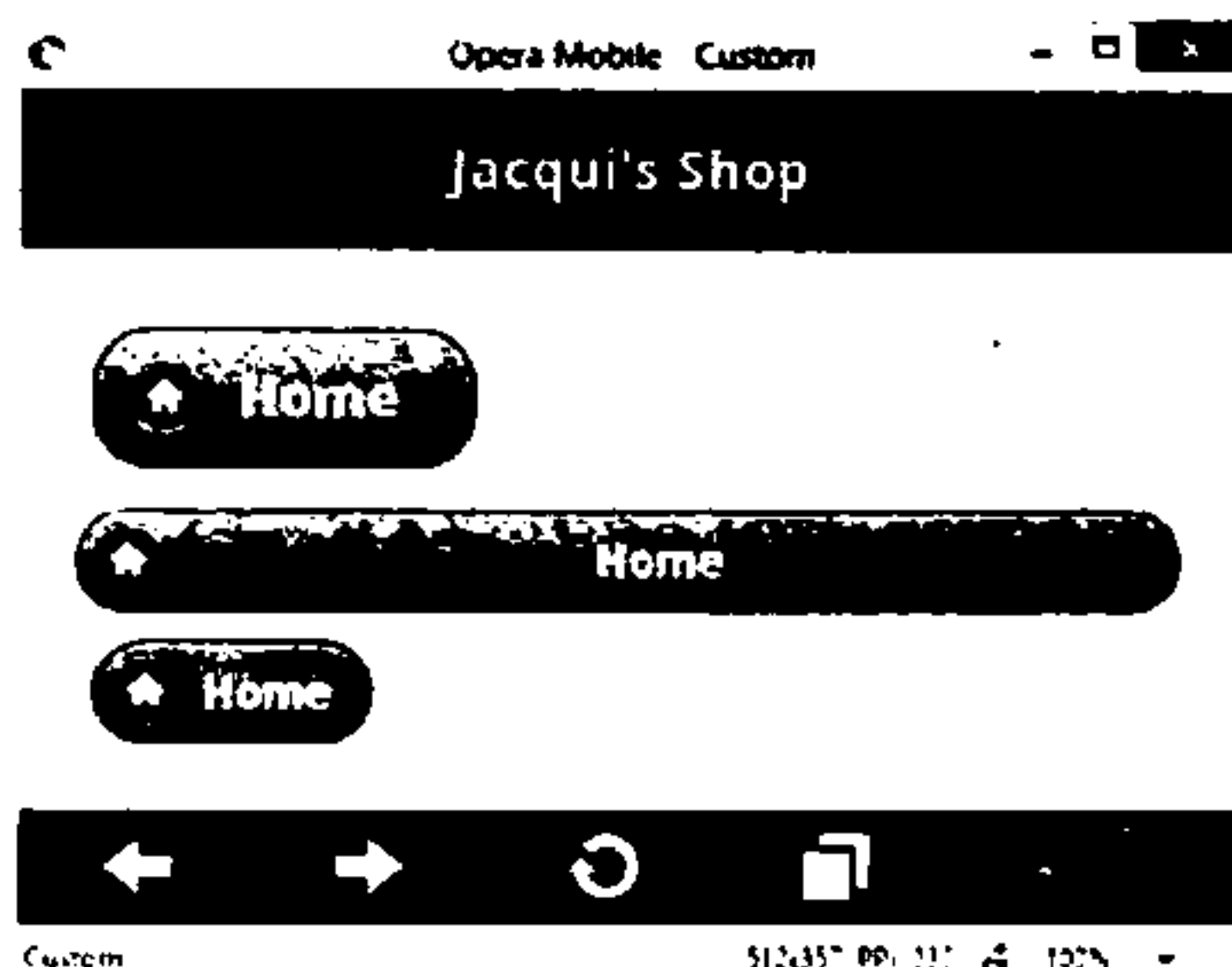


图30-6 行内按钮与紧凑型按钮

### 30.1.3 按钮组件的方法

按钮组件定义了三个方法，见表30-4。

表30-4 按钮组件的方法

方 法	描 述
button("disable")	禁用此按钮，使其不可点击
button("enable")	启用此按钮，使其支持点击
button("refresh")	刷新按钮状态以反映底层元素发生的变化

enable与disable方法的功能无需多说。如代码清单30-7所示，若组件底层的button或input元素内容发生了变化，应该调用refresh方法。

#### 代码清单30-7 更新按钮组件的内容

```
<!DOCTYPE html>
```

```

<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script>
    $(document).bind("pageinit", function () {
      var counter = 0;
      setInterval(function () {
        var msg = "Counter " + counter++;
        $("#buttonElem").text(msg).button("refresh");
        $("#inputElem").val(msg).button("refresh");
        $("#divElem span.ui-btn-text").text(msg);
      }, 1000);
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <button id="buttonElem">Button</button>
      <input id="inputElem" type="button" value="Input" />
      <div id="divElem" data-role="button">Div</div>
    </div>
  </div>
</body>
</html>

```

示例中的三个按钮分别由一个button元素、一个input元素和一个div元素生成。script元素中的脚本在pageinit事件中调用setInterval函数，以每秒一次的间隔定时更新按钮组件底层元素的内容。

我为button和input元素调用了refresh方法，这样按钮上的文本才能随底层元素内容的变化而变化。

```

...
$("#buttonElem").text(msg).button("refresh");
$("#inputElem").val(msg).button("refresh");
...

```

然而，如果按钮底层元素不是button或者input，而是div元素，这个方法就不管用了。若是这样，jQuery Mobile会主动添加内容元素，并为内容元素设置样式以使其显示成按钮的样子，因此改变元素的内容会影响按钮的形状。这时我们必须查看jQuery主动在div元素内添加的元素，以确定文本内容的位置。在这个例子中，jQuery Mobile在div元素里添加了span元素，因此我可以使用.ui-btn-text类定位到这个span元素，并改变它的内容。代码如下：

```

...
$("#divElem span.ui-btn-text").text(msg);
...

```



**提示** 我是在浏览器中按下F12键，利用浏览器开发工具来定位jQuery Mobile在自动增强过程中所添加的元素的。

### 30.1.4 按钮事件

按钮组件定义了准create事件，该事件在按钮组件成功生成之后触发，而按钮组件的底层元素也会触发它们自己的事件，这些事件将是我在第29章介绍的jQuery Mobile事件的有力补充。也就是说，就像代码清单30-8里做的那样，我们可以为tap事件编写处理函数，来接收点击按钮之后的反馈。

**提示** 有些元素的事件具有默认行为，比如点击由input元素生成的按钮组件就会提交按钮所在的表单。如果需要了解jQuery事件的细节，请参阅第28章。

代码清单30-8 处理按钮事件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script>
    $(document).bind("pageinit", function () {
      $("button").tap(function (e) {
        $(this).text("Tapped!").button("refresh");
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js">
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <button>Button</button>
    </div>
  </div>
</body>
</html>
```

在这个例子中，我使用tap方法为按钮的tap事件定义了事件处理函数。当点击这个按钮时，我在事件处理函数中改变了按钮的文本，并调用refresh方法来更新按钮组件的内容（你所看到的按钮文本）。

## 30.2 jQuery Mobile 导航栏组件

导航栏组件是放在页头或页脚的一组按钮。我们使用导航栏在多个页面之间导航。导航栏组件相当简单，但也非常有效，尤其是当你有一套内容关联的页面，并且希望用户能够清晰地知道当前显示的是哪个页面时。如代码清单30-9所示，导航栏组件使用专门的标签结构生成。

代码清单30-9 生成导航栏

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
      <div data-role="navbar">
        <ul>
          <li><a href="#page1"
            class="ui-btn-active ui-state-persist">Page 1</a></li>
          <li><a href="#page2">Page 2</a></li>
          <li><a href="#page3">Page 3</a></li>
        </ul>
      </div>
    </div>
    <div data-role="content">This is page 1</div>
  </div>

  <div id="page2" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
      <div data-role="navbar">
        <ul>
          <li><a href="#page1">Page 1</a></li>
          <li><a href="#page2"
            class="ui-btn-active ui-state-persist">Page 2
          </li>
          <li><a href="#page3">Page 3</a></li>
        </ul>
      </div>
    </div>
    <div data-role="content">This is page 2</div>
  </div>

  <div id="page3" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
      <div data-role="navbar">
```

```

        <ul>
          <li><a href="#page1">Page 1</a></li>
          <li><a href="#page2">Page 2</a></li>
          <li><a href="#page3"
            class="ui-btn-active ui-state-persist">Page 3</a></li>
        </ul>
      </div>
    </div>
    <div data-role="content">This is page 3</div>
  </div>
</body>
</html>

```

一个data-role属性设置为navbar的div元素就定义了一个导航栏。导航栏中的按钮，由ul元素中一个个包在li元素里的a元素构成。在这个例子中，如图30-7所示，我使用的是导航栏组件最常用的表现形式，它会在每个jQuery Mobile页面中出现，带给用户一致的导航体验。

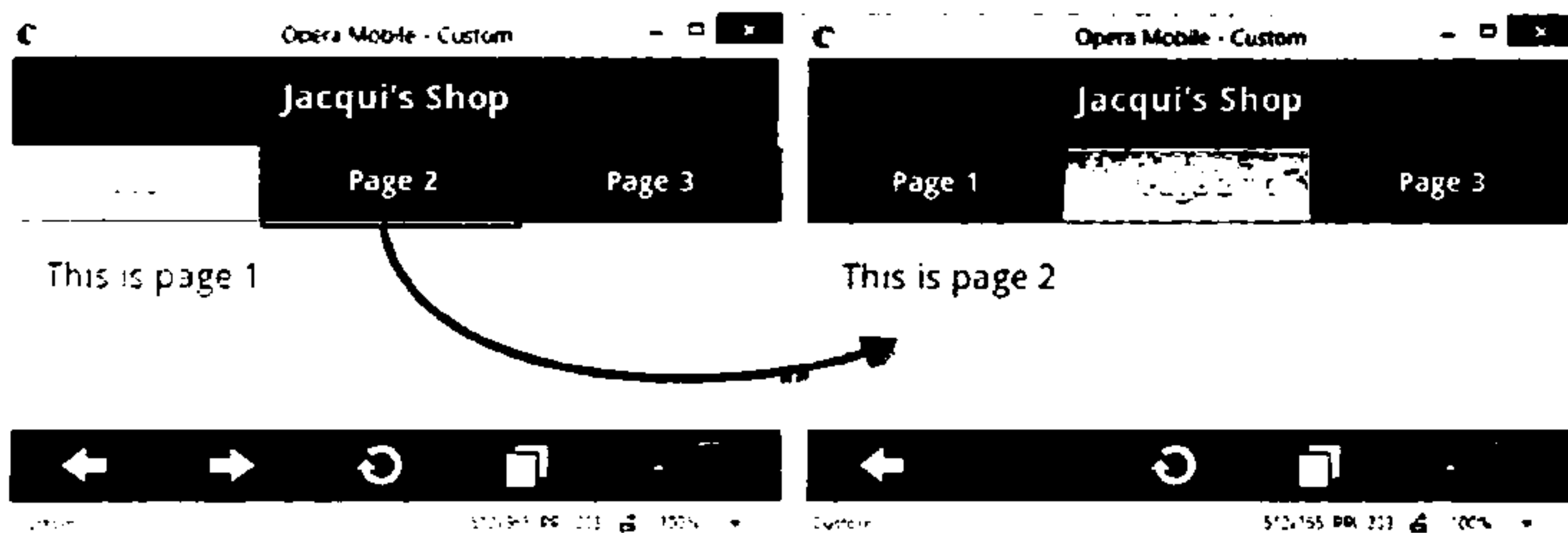


图30-7 导航栏组件

导航栏组件会根据可用空白余量在导航按钮之间均匀分配空白，点击任意一个按钮，则显示由底层a元素的href属性定义的页面。

为指示当前显示的是哪个页面，必须给底层的a元素添加两个CSS类。ui-btn-active类将当前按钮的状态标记为活跃，而ui-state-persist类能保证在用户返回上一页时，按钮的活跃状态不变。

### 30.2.1 配置 jQuery Mobile 导航栏组件

导航栏组件只支持一个data属性，或者说配置选项，参见表30-5。

表30-5 导航栏组件支持的data属性和配置选项

数据属性	选 项	描 述
data-iconpos	iconpos	为导航栏中的按钮指定图标位置。图标位置支持left、right、top（默认值）和bottom，还可以使用notext值，让按钮只显示icon而不显示任何文本

本章前面已经介绍过jQuery Mobile内建了哪些图标，设置a元素的数据-icon属性，即可为每个按钮单独设置图标。在代码清单30-10中，我使用data-iconpos属性为这些图标指定了显示位置。

代码清单30-10 设置导航栏按钮图标位置

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
      <div data-role="navbar" data-iconpos="left">
        <ul>
          <li><a href="#page1" data-icon="alert"
            class="ui-btn-active ui-state-persist">Page 1</a></li>
          <li><a href="#page2" data-icon="info">Page 2</a></li>
          <li><a href="#page3" data-icon="gear">Page 3</a></li>
        </ul>
      </div>
    </div>
    <div data-role="content">This is page 1</div>
  </div>
</body>
</html>

```

图标效果如图30-8所示：

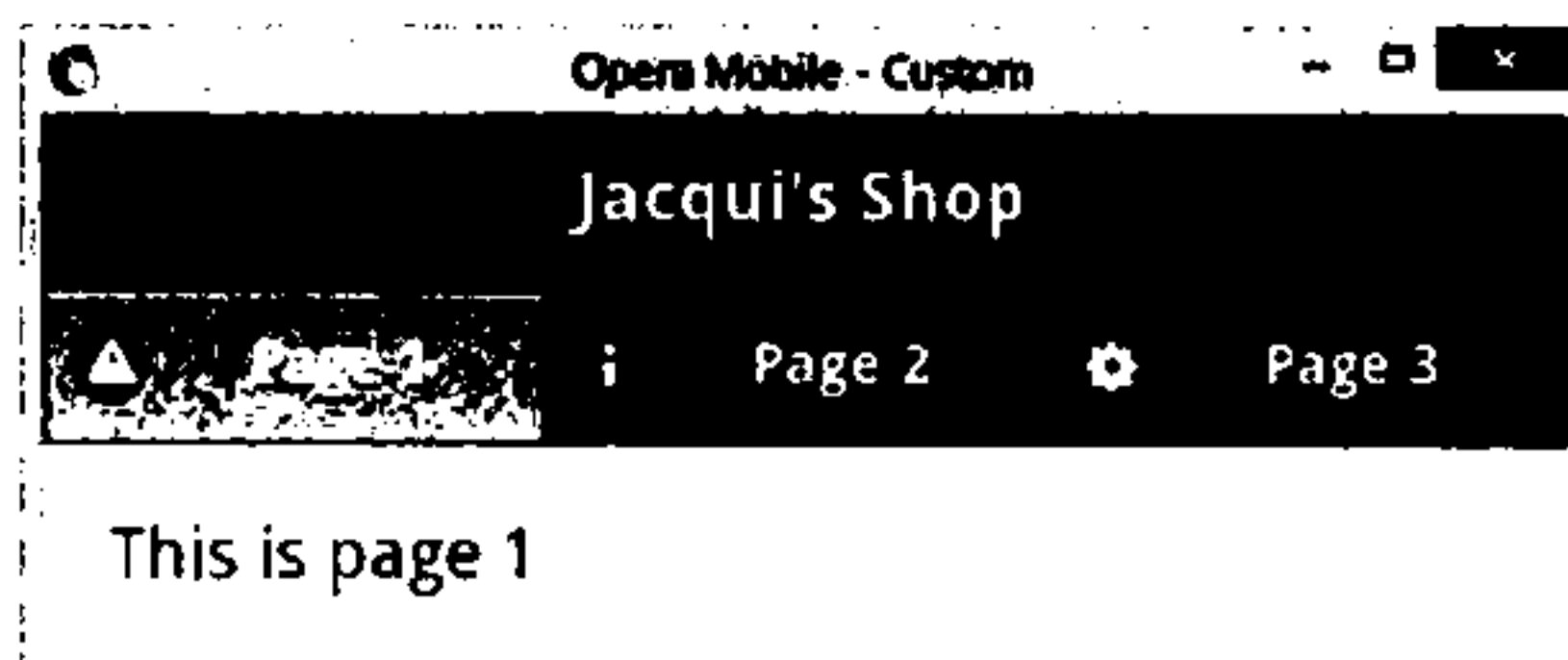


图30-8 在导航栏中添加图标并指定图标的位置

### 30.2.2 导航栏组件支持的方法和事件

导航栏组件没有定义任何方法。在导航栏组件创建完成时会触发create事件，除此之外，不支持任何事件。

**提示** 要响应用户导航请求，请处理a元素的事件。

## 30.3 折叠块组件

jQuery Mobile支持生成折叠内容块，即一组带标题的内容，标题总是可见的，而内容可以打开或者关闭。它和我们在第19章讲到的jQuery UI折叠菜单组件非常类似。

### 30.3.1 生成折叠块

折叠块就是满足jQuery Mobile显示需要的有着专门结构的一组元素。具体的例子见代码清单30-11。

代码清单30-11 生成一个折叠块

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <div data-role="collapsible">
        <h1>New Delivery Service</h1>
        <p>We are pleased to announce that we are starting a home delivery
        service for your flower needs. We will deliver within a 20 mile radius
        of the store for free and $1/mile thereafter.</p>
      </div>
    </div>
  </div>
</body>
</html>
```

我们必须要做的第一件事，是生成一个div元素，并把它的data-role属性设置为collapsible。这个属性告诉jQuery Mobile我们想要一个折叠块，因此它会把这个div元素的第一个子元素当成标题元素。可以使用从h1到h6的任意一个标题元素。在这个例子中我使用了h1元素，不过就这个组件来说，jQuery Mobile并不关心是哪一种标题元素，它会对所有标题元素一视同仁。div元素内剩下的其他元素自动成为折叠块的内容，最终我们得到如图30-9所示的效果。

默认情况下折叠块初始时是收起状态，也就是说我们看不到折叠块的内容，只能看到标题。为了提醒用户这个标题可以展开，jQuery Mobile在标题的左边安排了一个加号（+）图标（这里的样式处理与非行内按钮一致）。点击标题则展开内容并把加号替换为减号（-），表示这个内容还可以再收起来。

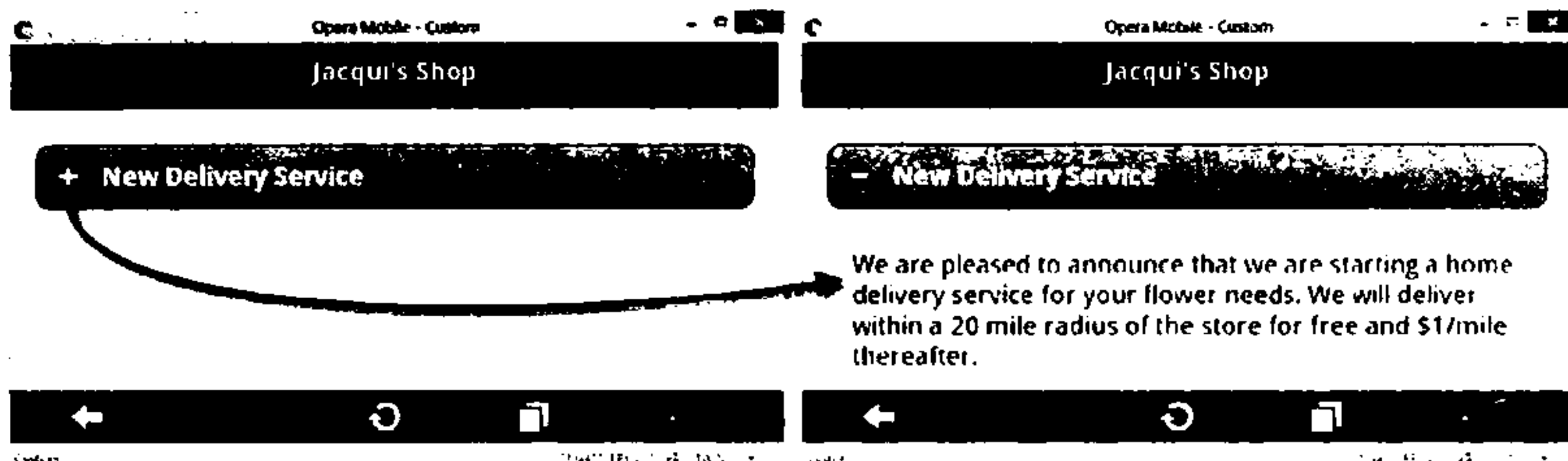


图30-9 展开一个折叠块

### 30.3.2 配置 jQuery Mobile 折叠块组件

jQuery Mobile 折叠块组件支持的 data 属性和配置选项见表 30-6。

表30-6 折叠块组件的 data 属性和配置选项

属 性	选 项	描 述
data-collapsed	collapsed	默认值 true，表示内容块默认隐藏（即只显示标题）；设置为 false 表示折叠块初始为展开状态
data-collapsed-icon	collapsedIcon	指定折叠块处于折叠状态时显示的图标
data-content-theme	contentTheme	指定折叠块内容区的样式主题
data-corners	corners	若设置为 true，折叠块显示为圆角；若设置为 false，则显示为直角
data-expanded-icon	expandedIcon	指定折叠块处于展开状态时显示的图标
data-iconpos	iconPos	指定图标在标题中显示的位置，可选值与按钮组件及导航栏组件相同
data-inset	inset	若设置为 false，标题将紧贴窗口，没有任何留白。默认值是 true
data-mini	mini	若设置为 true，则显示紧凑的标题

在代码清单 30-12 中，我演示了几个 data 属性的用法。

#### 代码清单 30-12 配置折叠块组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
```

```

<div data-role="collapsible" data-collapsed=true data-content-theme="e">
  data-inset="false" data-iconpos="top">
    <h1>New Delivery Service</h1>
    <p> We are pleased to announce that we are starting a home delivery
    service for your flower needs. We will deliver within a 20 mile radius
    of the store for free and $1/mile thereafter.</p>
  </div>
</div>
</div>
</body>
</html>

```

在这个例子中，我改变了data-content-theme、data-inset、data-iconpos和data-collapsed属性的值。这些变更产生的显示效果见图30-10。



图30-10 配置折叠块组件

### 30.3.3 折叠块组件的方法

折叠块组件未定义任何方法。

### 30.3.4 折叠块事件

表30-7列出了折叠块组件支持的事件。

表30-7 折叠块事件

事 件	描 述
create	折叠块组件创建完成时触发此事件
collapse	折叠块内容折叠完成时触发此事件
expand	折叠块内容展开完成时触发此事件

在代码清单30-13中，我分别在折叠及展开事件中报告折叠块的当前状态。由于适合使用这些事件的情况实在太少，这个例子确实不够给力。

## 代码清单30-13 collapse和expand事件的使用法

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript">
    $(document).bind("pageinit", function() {
      $("#colBlock").bind("collapse expand", function(event) {
        $("#status").text(event.type == "expand" ? "Expanded" : "Collapsed");
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      The block is <b><span id="status">Expanded</span></b>

      <div id="colBlock" data-role="collapsible" data-content-theme="e" data-collapsed=false>
        <h1>New Delivery Service</h1>
        <p> We are pleased to announce that we are starting a home
          delivery service for your flower needs. We will deliver within a
          20 mile radius of the store for free and $1/mile thereafter.</p>
      </div>
    </div>
  </div>
</body>
</html>

```

在这个例子中，我使用bind方法来设置expand和collapse事件的处理函数。使用空格分隔的事件名称作为bind方法的第一个参数，能够只调用一次bind方法，即可设置两个事件。无论哪一个事件发生，我都会修改内容区的span元素，以更新折叠块的状态。折叠块状态变化的实际效果见图30-11。

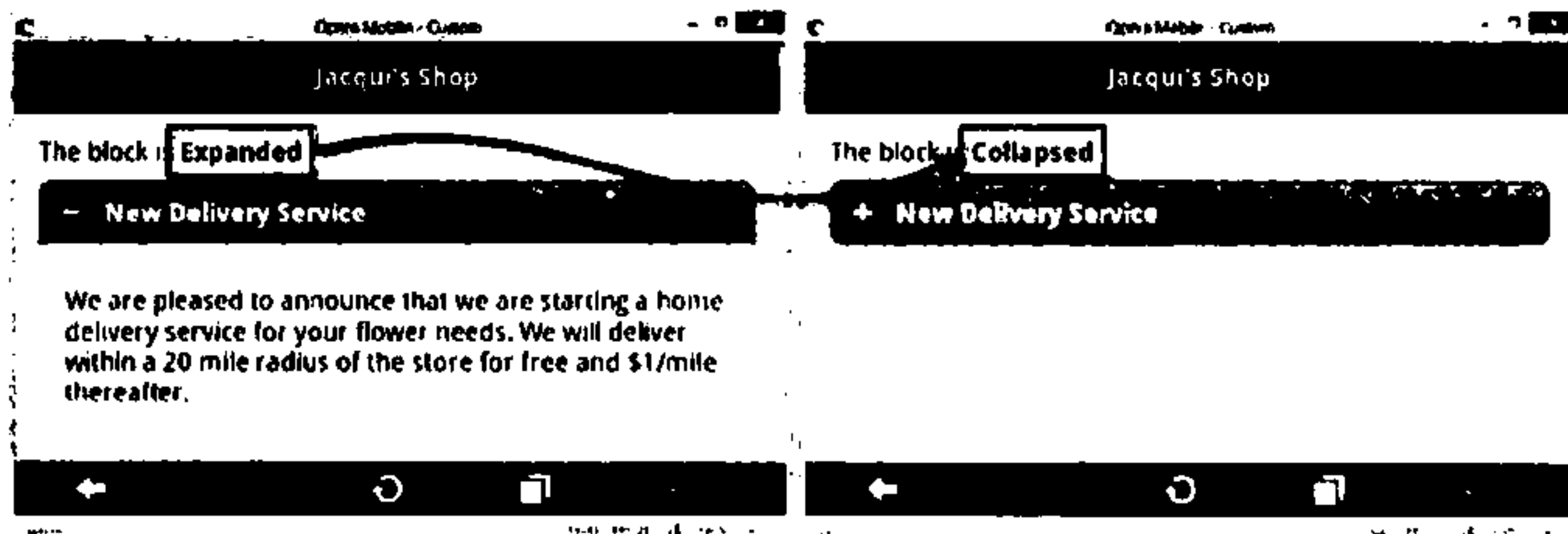


图30-11 响应expand和collapse事件



## 30.4 jQuery Mobile 折叠菜单

折叠菜单由多个折叠块组成。一个包裹着几个折叠块的、其data-role属性设置成collapsible-set的div元素就构成了折叠菜单。折叠菜单的这种结构参见代码清单30-14。

代码清单30-14 jQuery Mobile折叠菜单

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10..js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <div data-role="collapsible-set" data-content-theme="e">
        <div data-role="collapsible">
          <h1>New Delivery Service</h1>
          <p> We are pleased to announce that we are starting a home
            delivery service for your flower needs. We will deliver within a
            20 mile radius of the store for free and $1/mile thereafter.</p>
        </div>
        <div data-role="collapsible" data-collapsed=false>
          <h1>Summer Specials</h1>
          <p> We have a wide range of special summer offers.
            Ask instore for details</p>
        </div>
        <div data-role="collapsible">
          <h1>Bulk Orders</h1>
          <p>We offer discounts for large orders. Ask us for prices</p>
        </div>
      </div>
    </div>
  </div>
</body>
</html>
```

在这个例子中，我定义了一个div元素，并把它的data-role属性设置为collapsible-set，其中包含三个折叠块。

**提示** 注意我在外层容器中使用了data-content-theme属性。这和分别给每个折叠块添加这一属性的效果完全一样（但所需代码更少）。

默认情况下所有的折叠块都是折叠状态。我把一个折叠块的data-collapsed属性设置成了false, 这样当页面第一次打开时, 它会处于展开状态。当用户点击某个标题时, 原来展开的元素就会收起来显示效果见图30-12。

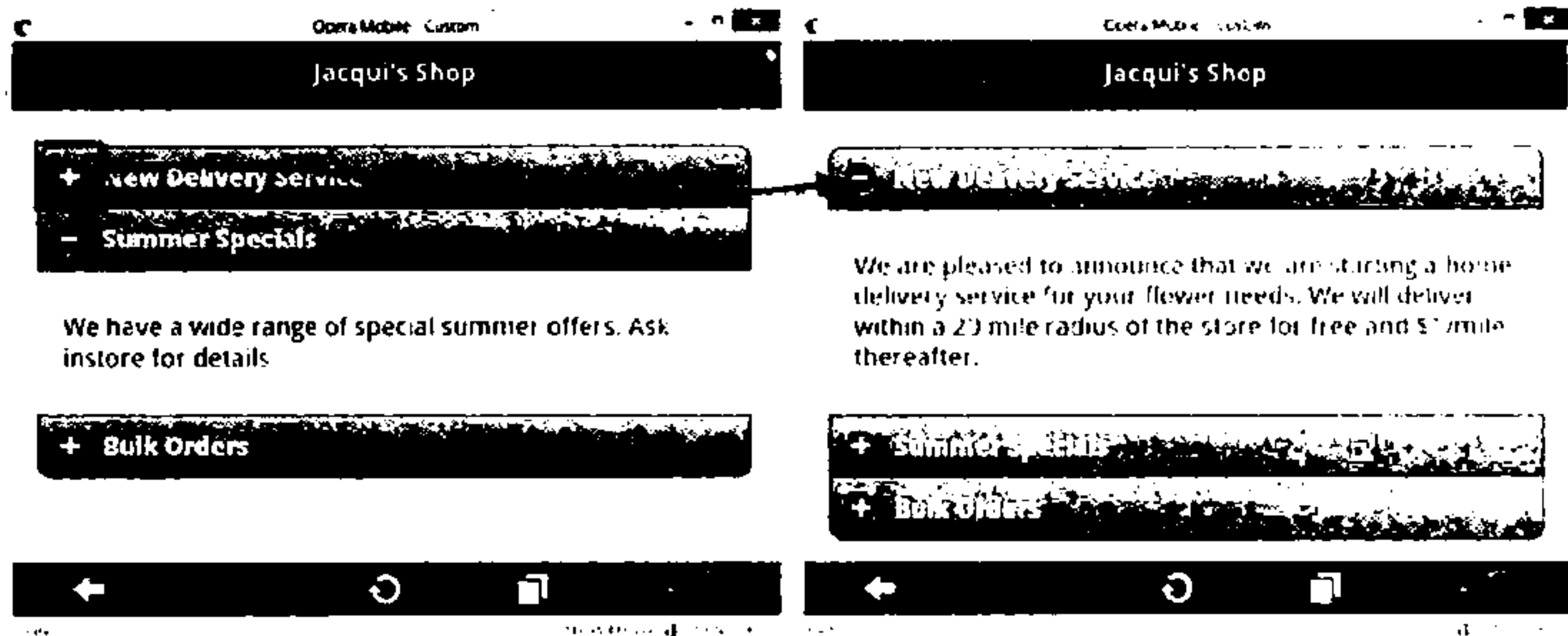


图30-12 在jQuery Mobile折叠菜单中展开一个折叠块

### 30.4.1 配置折叠菜单

折叠菜单支持的数据属性与配置选项, 与折叠块组件完全相同。

### 30.4.2 折叠菜单的方法

折叠菜单只支持一个方法, 见表30-8。

表30-8 折叠菜单的方法

方 法	描 述
<code>collapsibleset("refresh")</code>	刷新组件状态以反映底层HTML元素的变化

### 30.4.3 折叠菜单事件

折叠菜单组件只支持create事件, 它在折叠菜单创建完成时触发。

## 30.5 小结

本章介绍了按钮、折叠块, 以及导航栏和折叠菜单组件。在下一章, 我们将介绍jQuery Mobile表单组件。

在移动设备上呈现表单面临着几个难题。屏幕相当小，我们需要合理安排表单元素的布局，让用户很容易就能触摸操作，还得让页面具备合理的高度，让用户尽量少地滚动屏幕。本章将介绍jQuery Mobile对表单元素的增强，让表单具备与其他组件一致的操作体验，即便是触屏操作也同样容易。

在页面加载时，jQuery Mobile主动做了许多事情。它主动强化了表单元素的功能，并且使表单在提交时默认使用Ajax，这样jQuery Mobile就能平滑过渡到来自服务器的结果页。

为移动设备创建表单时一定要深思熟虑。表单的本意是收集用户输入，而在小屏幕设备上这是一个繁琐的过程，特别是打字。除此之外，有的移动设备不显示滚动条，那样用户就无法滚动。这就意味着用户并不是总能察觉到后面还有表单元素。为了创建最佳用户体验，我们需要遵守以下基本原则。

- 尽可能少地要求用户键入。如果可能，让用户选择（使用复选框或单选按钮）而不是键入。这样做也许得不到全面的用户数据，却能让用户更愿意完成表单。
- 在分页表单之间使用导航。导航能给用户以清晰的指示（表单进度），同时用户无需滚动页面就能知道是否遗漏了什么。
- 去掉任何不必要的表单元素。移动表单应该越精简越好，这意味着与桌面用户相比，我们宁愿从移动用户那里得到更少的数据。

表31-1列出了本章概要。

表31-1 本章概要

问 题	解决方法	代码清单
如何使用form元素生成表单组件	无需专门配置，jQuery Mobile会自动加强表单	1
如何添加一个按钮清除文本输入框的内容，或者改变input元素的显示方式	使用data-clear-btn或者data-mini属性	2
如何启用或禁用input元素	使用enable或disable方法	3
如何创建滑动条组件	定义一个type属性为range的input元素	4
如何创建范围滑动条组件	在一个data-role属性为rangeslider的div元素内定义两个input元素	5
如何配置滑动条的外观	使用 data-highlight、data-mini 和 data-track-theme属性	6、7
如何根据底层input元素的变化更新范围滑动条	使用refresh方法	8
在用户拖动滑块时如何响应位置变化	处理start、stop和normalize事件	9、10

(续)

问 题	解决方法	代码清单
如何使用select元素生成选择组件	无需专门配置	11
如何配置为select元素服务的按钮	使用 data-corners、data-icon、data-iconpos 和 data-mini属性	12
如何配置弹窗以允许用户为select元素选择一个值	使用 data-native-menu、data-overlay-theme 和 data-divider-theme属性	13
如何为选择菜单添加占位栏目	把select元素内某个选项的data-placeholder属性设置为true	14
如何使用脚本控制选择菜单	使用open、close、disable、enable和refresh方法	15
如何创建滑动开关组件	定义一个只有两个选项的select元素，并把它的数据-role属性设置为slider	16
如何生成复选框和单选框组件	无需专门配置	17~20

## 31.1 创建表单元素组件

与前面章节描述的组件一样，在页面加载时，jQuery Mobile也会自动增强表单元素。代码清单31-1展示了包含一个form元素和一些表单相关子元素的jQuery Mobile页。

代码清单31-1 一个简单的jQuery Mobile表单

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    #buttonContainer {text-align: center}
    div[data-role=fieldcontain] { padding: 0 10px; }
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <form method="get">
      <div data-role="fieldcontain">
        <label for="name">Name: </label>
        <input id="name">
      </div>
      <div data-role="fieldcontain">
        <label for="address">Address: </label>
        <textarea id="address"></textarea>
      </div>
    </form>
  </div>
</body>
</html>
```

```

        </div>
        <div id="buttonContainer">
            input type="submit" data-inline="true" value="Submit"/>
        </div>
    </form>
</div>
</body>
</html>

```

尽管这个表单很简单，但对于全面讲解jQuery Mobile处理表单的方法来说已经足够。这个表单中有两个表单元素：一个单行文本输入框，和一个多行文本输入框，每个表单元素都有一个配对的label元素。这个例子的显示效果见图31-1，这里我使用了BrowserStack网站的服务，因为Opera Mobile模拟器没有正确实现jQuery Mobile对表单元素所要求的特性。在下一节中，我会就此问题进行详细说明。

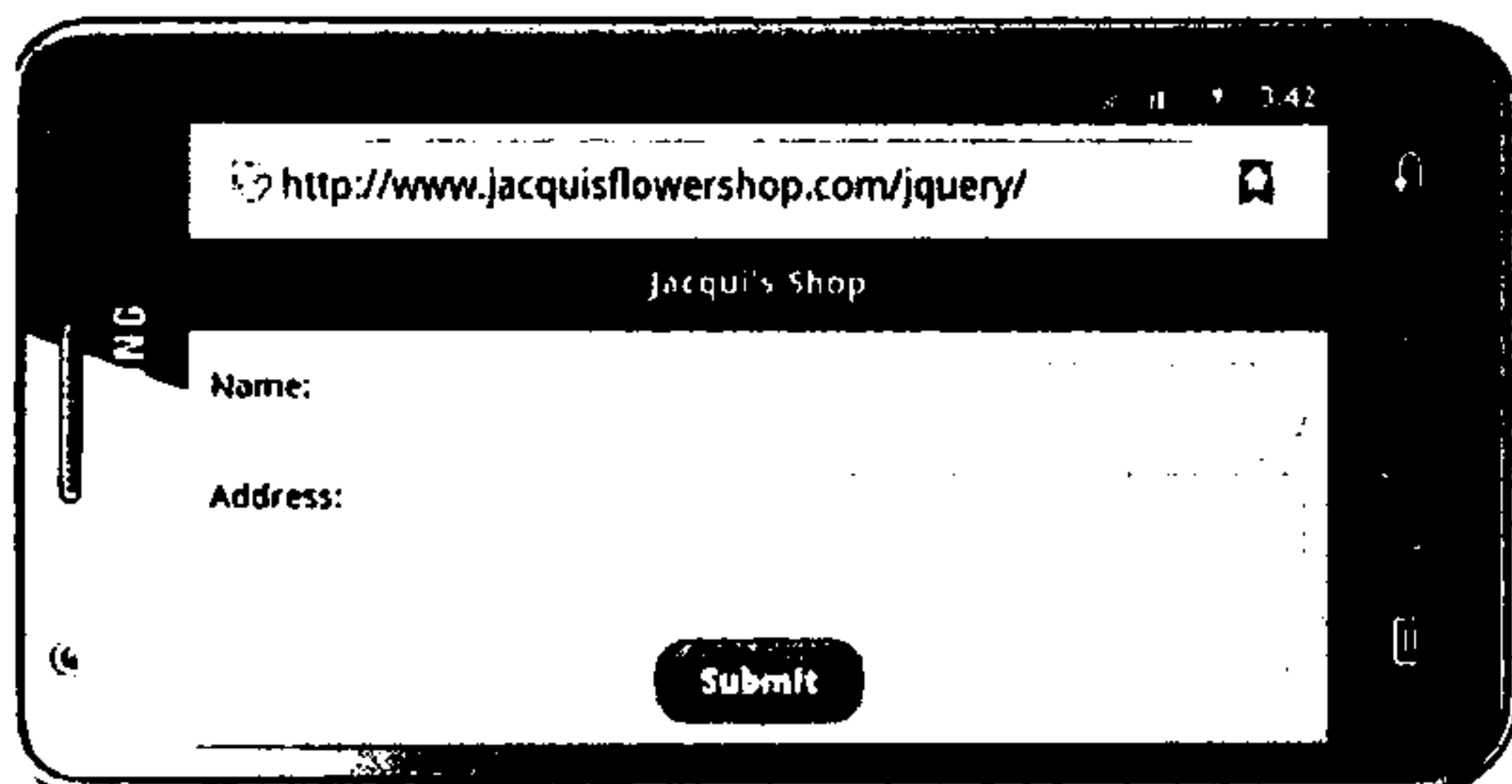


图31-1 一个简单的jQuery Mobile表单

**提示** 注意，我使用样式表为包含form元素的div元素设置了内边距。如果缺少内边距，jQuery Mobile就会让标题和表单元素紧贴窗口的左边和右边。

若form元素内有一个input元素的type属性值为submit，jQuery Mobile会在必要时自动提交表单。表单默认以Ajax方式提交，不过如果把form元素的data-ajax属性设置为false，就能禁用这一行为。

在上一个例子中，每个表单元素和它的标签都被一个div元素包住。这些div元素都有着一个值为fieldcontain的data-role属性，正是这个属性告诉jQuery Mobile，标签元素应该和表单元素显示在一行上（参见图31-1）。

只有当用户屏幕至少450px宽的时候，jQuery Mobile用于排列标签和表单元素的样式才能正常工作。如果屏幕不够宽，就会像图31-2那样，标签与input/textarea元素就会各占一行。但Opera Mobile模拟器总是报告它的屏幕小于450像素，我只好改用另一种浏览器为这些例子截图。（这只是模拟器的问題，真正的Opera Mobile浏览器没有问题，能够正确实现这一特性。）

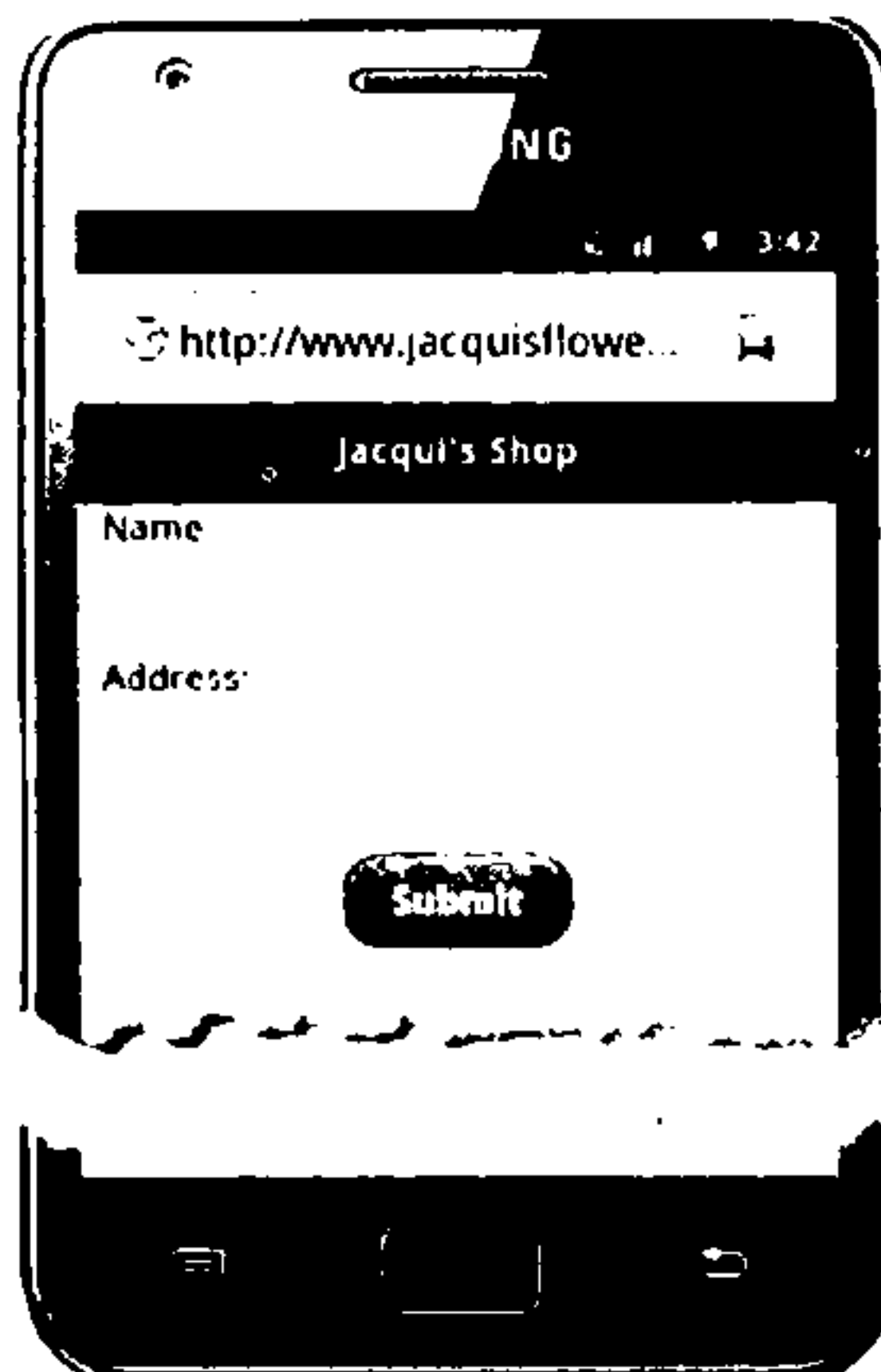


图31-2 垂直排列的表单

## 31.2 文本输入组件

jQuery Mobile支持的每一种表单元素都对应着一种组件，首先要介绍的是文本输入组件，它用于textarea和input元素。前面那个例子中使用的就是文本输入组件。

### 31.2.1 配置文本输入组件

文本输入组件支持的属性与配置选项见表31-2。这些选项通过调用textinput方法发挥作用。

表31-2 文本组件的属性和配置选项

数据属性	选 项	描 述
data-clear-btn	clearBtn	若设置为true，这个组件就会显示成一个按钮，点击按钮则会清除已经输入的内容。默认值为false
data-clearn-btn-text	clearBtnText	为残疾人辅助软件设置清除按钮的示意文本
data-mini	mini	若设置为true，则以紧凑模式显示组件。默认值为false
data-prevent-focus-zoom	preventFocusZoom	若设置为true，则当文本框得到焦点时禁止浏览器对文本框进行放大或者缩小

代码清单31-2演示了如何在一个input元素上使用data-clear-btn和data-mini属性配置文本输入组件。

代码清单31-2 配置文本输入组件

```
...
<form method="get">
  <div data-role="fieldcontain">
    <label for="name">Name: </label>
    <input id="name" data-clear-btn="true" data-mini="true">
  </div>
  <div data-role="fieldcontain">
    <label for="address">Address: </label>
    <textarea id="address"></textarea>
  </div>
  <div id="buttonContainer">
    <input type="submit" data-inline="true" value="Submit"/>
  </div>
</form>
...
```

这个例子的结果见图31-3。在用户开始输入内容之前，清除按钮不会显示。点击这个按钮，会删除任何已键入的内容。如果使用了清除按钮功能，就应保持一致，确保所有输入元素都有清除按钮。

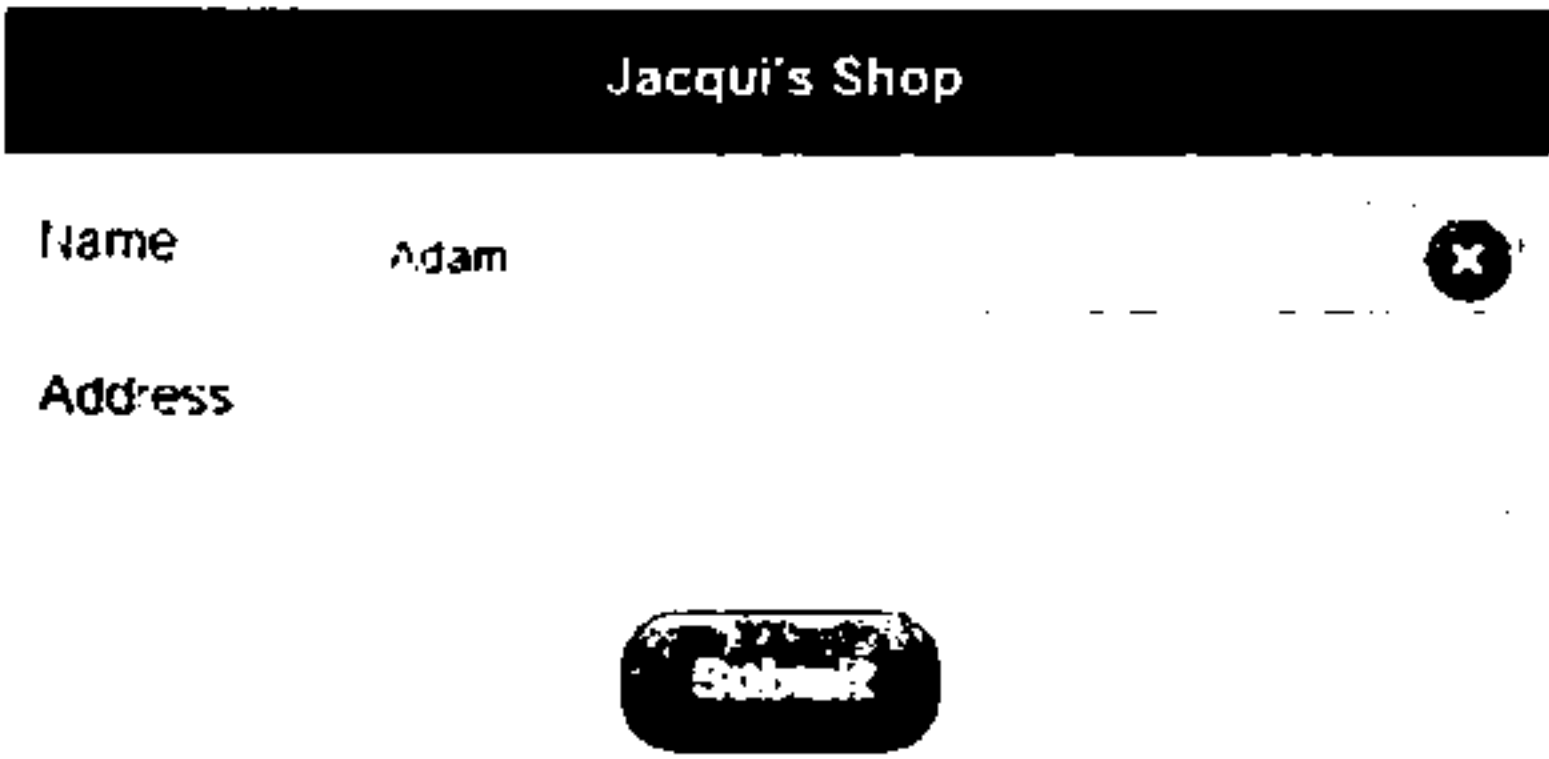


图31-3 配置文本输入组件

31.2.2 文本输入组件方法

文本输入组件定义了两个方法，见表31-3。

表31-3 textinput方法

方 法	描 述
textinput("disable")	禁用组件，阻止用户输入内容或者修改内容
textinput("enable")	启用组件，允许用户输入内容或者修改内容

在代码清单31-3中，通过这两个方法修改底层input元素的可用状态。

代码清单31-3 应用textinput方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
```

```

<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
<script type="text/javascript" src="jquery-1.10.1.js"></script>
<script>
    $(document).bind("pageinit", function () {
        $("button").tap(function (e) {
            $("#name").textinput(e.target.id);
            e.preventDefault();
        });
    });
</script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
<style type="text/css">
    #buttonContainer {text-align: center}
    div[data-role=fieldcontain] { padding: 0 10px; }
</style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <form method="get">
            <div data-role="fieldcontain">
                <label for="name">Name: </label>
                <input id="name" data-clear-btn="true" data-mini="true">
            </div>
            <div id="buttonContainer">
                <button id="enable">Enable</button>
                <button id="disable">Disable</button>
            </div>
        </form>
    </div>
</body>
</html>

```

我添了一些按钮，用它们改变input元素的状态。在图31-4中，可以看到被禁用的组件如何显示。

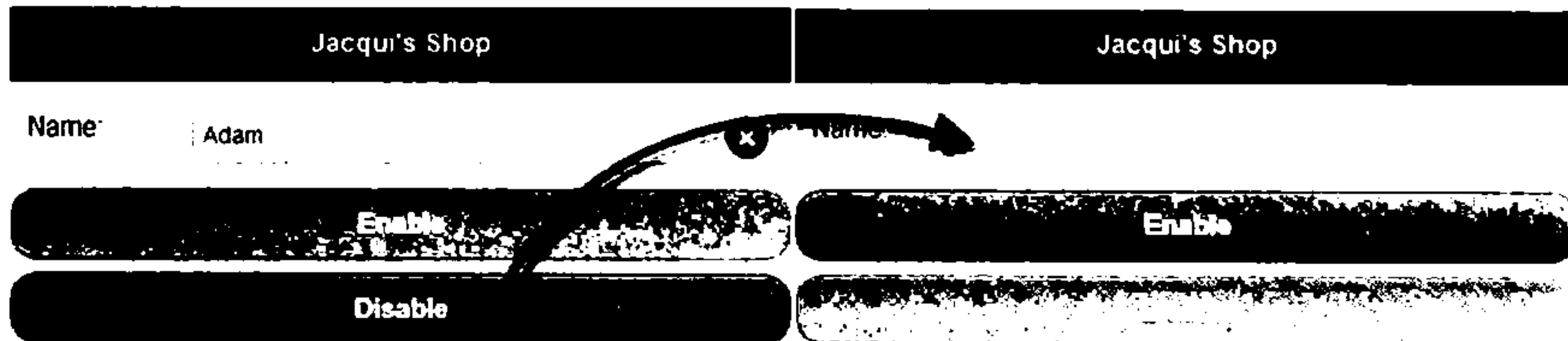


图31-4 被禁用的文本输入组件

### 31.2.3 文本输入组件事件

文本输入组件仅定义了create事件，该事件在组件创建完成时触发。



## 31.3 滑动选择器组件和范围选择器组件

若jQuery Mobile发现一个input元素的type属性值为range，它就会创建一个滑动选择器。input元素的值（即value属性的值）将作为滑动选择器的初始值，用来设定滑动选择器的滑动把手的初始位置。滑动条的最大值和最小值，则由input元素的min和max属性定义。代码清单31-4演示了滑动选择器的一个简单例子。

代码清单31-4 滑动选择器组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    div[data-role=fieldcontain] { padding: 0 10px; }
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <form method="get">
      <div data-role="fieldcontain">
        <label for="quantity">Quantity: </label>
        <input id="quantity" type="range" value="10" min="1" max="20">
      </div>
    </form>
  </div>
</body>
</html>
```

**提示** type属性的range值是input元素增强的一部分，是在HTML5规范中定义的。要详细了解HTML5，请阅读《HTML5权威指南》一书。

在这个例子中，我定义了一个type属性为range的input元素，并指定初始值为10，最小值1，最大值20。在图31-5中，你会看到jQuery Mobile如何增强这个input元素为滑动选择器组件。

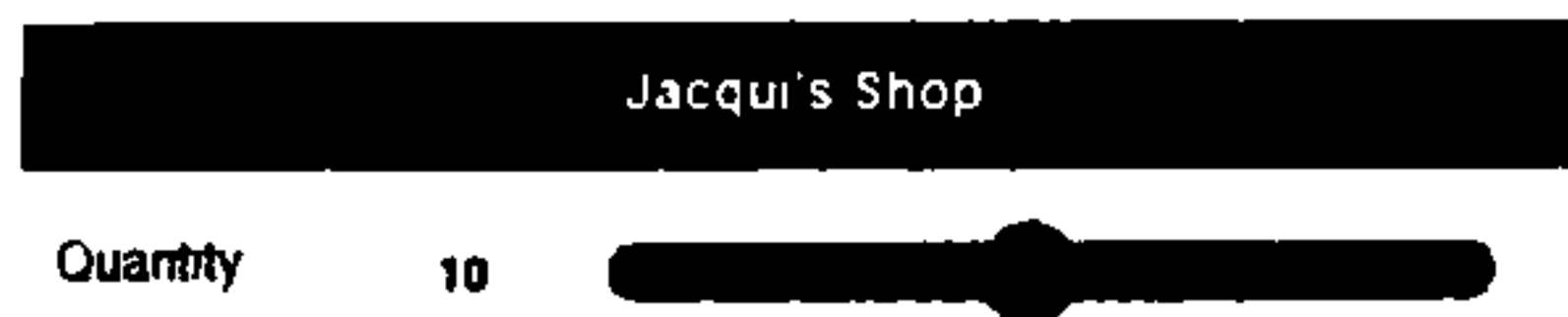


图31-5 使用滑动选择器组件

使用一对input元素，就可以生成范围选择器，允许用户在组件内选择最大值和最小值。代码清单31-5演示了如何生成范围选择器。

代码清单31-5 使用范围选择器组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    div[data-role=fieldcontain] { padding: 0 10px; }
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <form method="get">
      <div data-role="fieldcontain">
        <div data-role="rangeslider">
          <label for="quantityLow">Quantity: </label>
          <input id="quantityLow" type="range" value="10" min="1" max="20">
          <input id="quantityHigh" type="range" value="15" min="1" max="20">
        </div>
      </div>
    </form>
  </div>
</body>
</html>
```

从例子中可以看出，两个input元素被包在data-role属性为rangeslider的div元素之中。两个input元素的min和max属性的值必须相同，它们的value属性用于设置选择范围滑动把手的初始位置（图31-6）。



图31-6 使用范围选择器

**提示** 范围选择器组件并不会改变向服务器提交表单数据的方式。两个input元素的值仍然是作为两个独立的值提交给服务器的。

### 31.3.1 配置滑动选择器组件和范围选择器组件

滑动选择器与范围选择器组件支持完全相同的数据属性和配置选项，见表31-4。

表31-4 滑动选择器与范围选择器组件的属性及配置选项

data属性	选 项	描 述
data-highlight	highlight	若设置为true，则表示选中值部分会突出显示。默认值为false
data-mini	mini	若设置为true，则组件以紧凑模式显示。默认值为false
data-track-theme	trackTheme	为滑动条指定主题

**提示** 使用slider方法配置滑动选择器组件，使用rangeslider方法配置范围选择器组件。

在前面的组件介绍中，我们已经了解了data-mini属性的作用，这里不再重复介绍。我通常会把data-highlight属性设置成true，因为这会让滑动选择器更醒目，尤其是范围选择器。在代码清单31-6中，我演示了如何在范围选择器的div元素上设置这一属性。

代码清单31-6 在范围选择器组件中使用data-highlight属性

```
...
<form method="get">
  <div data-role="fieldcontain">
    <div data-role="rangeslider" data-highlight="false">
      <label for="quantityLow">Quantity: </label>
      <input id="quantityLow" type="range" value="10" min="1" max="20">
      <input id="quantityHigh" type="range" value="15" min="1" max="20">
    </div>
  </div>
</form>
...
```

图31-7展示了突出显示和不突出显示的范围选择器。



图31-7 data-highlight属性的效果

我们还可以组合使用全局属性data-theme属性和data-track-theme属性来进一步强化滑动条的显示效果。data-theme属性影响滑动把手，而data-track-theme影响滑轨。如代码清单31-7中所示，我在上一个例子的范围选择器中同时使用了这两种属性。

## 代码清单31-7 为滑动把手及滑轨指定主题

```

...
<form method="get">
  <div data-role="fieldcontain">
    <div data-role="rangeslider" data-highlight="true"
      data-theme="b" data-track-theme="a">
      <label for="quantityLow">Quantity: </label>
      <input id="quantityLow" type="range" value="10" min="1" max="20">
      <input id="quantityHigh" type="range" value="15" min="1" max="20">
    </div>
  </div>
</form>
...

```

新主题的显示效果见图31-8。



图31-8 为范围选择器及其滑轨指定主题

### 31.3.2 滑动选择器与范围选择器支持的方法

滑动选择器与范围选择器定义的方法见表31-5。

表31-5 滑动选择器与范围选择器方法

滑动选择器方法	范围选择器方法	描 述
slider("disable")	rangeslider("disable")	禁用组件，阻止用户改变滑块的值
slider("enable")	rangeslider("enable")	启用组件，允许用户改变滑块的值
slider("refresh")	rangeslider("refresh")	根据底层元素属性值的变化更新组件状态

如代码清单31-8所示，利用refresh方法，我们可以在改变底层元素各属性的值之后，刷新组件的状态。

## 代码清单31-8 refresh方法

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script>
    $(document).bind("pageinit", function () {
      $("button").tap(function (e) {
        var currentMax = Number($("#quantityLow").attr("max"));

```

```

        $("#quantityLow, #quantityHigh").attr("max", currentMax - 1);
        $("#slider").rangeslider("refresh");
        e.preventDefault();
    });
});
</script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
<style type="text/css">
    div[data-role=fieldcontain] { padding: 0 10px; }
</style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <form method="get">
            <div data-role="fieldcontain">
                <div id="slider" data-role="rangeslider" data-highlight="true"
                    data-theme="b" data-track-theme="a">
                    <label for="quantityLow">Quantity: </label>
                    <input id="quantityLow" type="range" value="10" min="1" max="20">
                    <input id="quantityHigh" type="range" value="15" min="1" max="20">
                </div>
                <div id="buttonContainer">
                    <button>Change Range</button>
                </div>
            </div>
        </form>
    </div>
</body>
</html>

```

在这个例子中，我使用一个按钮来减小底层input元素max属性的值，并调用refresh方法刷新组件以显示底层元素属性值的变化。

### 31.3.3 滑动选择器事件

滑动选择器组件与范围选择器组件定义的事件有所不同。表31-6列出了滑动选择器组件定义的事件。

表31-6 滑动选择器事件

事 件	描 述
create	在组件创建完成时触发
start	与组件开始交互时触发：点击滑轨设置值，或者拖动滑块设置值
stop	与组件交互结束时触发

在代码清单31-9中，我们利用start和stop事件来更新span元素的内容。

#### 代码清单31-9 使用滑动选择器事件

```
<!DOCTYPE html>
```

```

<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script>
    $(document).bind("pageinit", function () {
      $("#quantity").slider({
        start: function () {
          $("#message").text("Sliding");
        },
        stop: function () {
          $("#message").text(quantity.value);
        }
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    div[data-role=fieldcontain] { padding: 0 10px; }
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <form method="get">
      <div data-role="fieldcontain">
        <label for="quantity">Quantity: </label>
        <input id="quantity" type="range" value="10" min="1" max="20">
      </div>
      Value: <span id="message">Ready</span>
    </form>
  </div>
</body>
</html>

```

当我接收到start事件时，改变span元素的值，以表示滑动条的值正在变化。当接收到stop事件时，就更新span元素的值为滑块的值。注意，这里直接使用input元素获取值，而不是通过滑动选择器组件来获取

### 31.3.4 范围选择器组件事件

表31-7列出了范围选择器组件定义的事件。

表31-7 范围选择器事件

事 件	描 述
create	在范围选择器组件创建完成时触发
normalize	在范围选择器组件不得不使范围选择器重新变得正常时触发（通常是因为用户把一个滑块拖过了另一个滑块的位置）

当我写到这里时，范围选择器组件支持的事件仍然无法使用rangeslider方法正常调用。如代码清单31-10所示，事件处理函数必须使用bind方法才能创建。

代码清单31-10 范围选择器组件事件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script>
    $(document).bind("pageinit", function () {
      $("#slider").bind("rangeslidernormalize", function () {
        alert("Normalized!");
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    div[data-role=fieldcontain] { padding: 0 10px; }
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <form method="get">
      <div id="slider" data-role="rangeslider" data-highlight="true"
        data-theme="b" data-track-theme="a">
        <label for="quantityLow">Quantity: </label>
        <input id="quantityLow" type="range" value="10" min="1" max="20">
        <input id="quantityHigh" type="range" value="15" min="1" max="20">
      </div>
    </form>
  </div>
</body>
</html>
```

必须使用bind方法和rangeslidernormalize名（需要翻阅底层代码才能找出）才能接收到这一事件。在这个例子中，当接收到normalize事件时，我调用了alert函数。说实话，在实践中我尚未找到能恰当使用这个事件的场合。

## 31.4 选择菜单组件

对select元素来说，jQuery Mobile提供了两种处理方法。第一种是使用带下拉按钮的选择菜单组件，它为用户提供了一种传统的select元素界面。如代码清单31-11所示，jQuery Mobile默认把select元素加强为选择菜单组件。

## 代码清单31-11 包含select元素的示例页面

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    #buttonContainer {text-align: center}
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <form method="get">
      <div data-role="fieldcontain">
        <label for="name">Name: </label>
        <input id="name" placeholder="Your Name">
      </div>
      <div data-role="fieldcontain">
        <label for="speed"><span>Speed: </span></label>
        <select id="speed" name="speed">
          <option value="vfast">Very Fast</option>
          <option value="fast">Fast</option>
          <option value="normal" selected>Normal</option>
          <option value="slow">Slow</option>
        </select>
      </div>
      <div id="buttonContainer">
        <input type="submit" data-inline="true" value="Submit"/>
      </div>
    </form>
  </div>
</body>
</html>

```

图31-9展示了经jQuery Mobile加强之后的select元素。Opera移动模拟器无法正确显示选择菜单组件，所以我用BrowserStack获取截图。

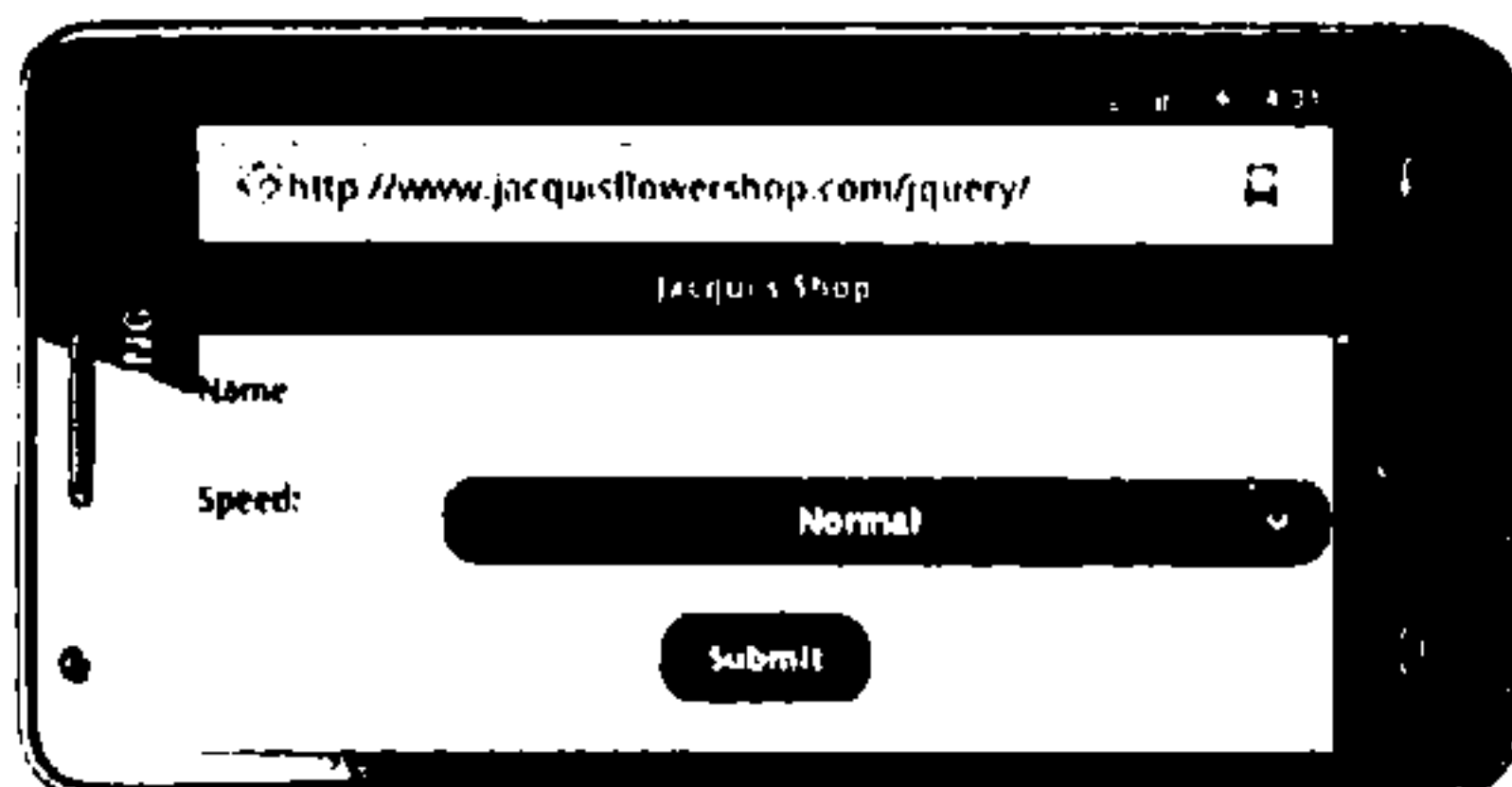


图31-9 经jQuery Mobile自动增强之后的select元素



### 31.4.1 配置选择菜单组件

选择菜单组件定义的data属性和配置选项见表31-8。这些配置选项通过调用selectmenu方法发挥作用。选择菜单组件的配置选项可分为两大区域：配置下拉按钮区域的选项，以及配置供用户选择的弹出菜单的选项。下面将对这些配置选项进行详细介绍。

表31-8 选择菜单组件定义的属性和配置选项

data属性	选 项	描 述
data-corners	corners	指定控制选项列表显示的按钮是否显示成圆角，默认值为true
data-divider-theme	dividerTheme	若nativeMenu选项设置为false，指定optgroup元素使用的主题
data-icon	icon	指定组件按钮上显示什么图标
data-iconpos	iconPos	指定组件按钮上图标的显示位置
data-inline	inline	指定组件是否以紧凑模式显示
data-mini	mini	指定组件是否以较小的尺寸显示
data-native-menu	nativeMenu	指定是否使用原生的选择菜单，默认值为true
data-overlay-theme	overlayTheme	若nativeMenu选项设置为false，指定弹出选择菜单所用的主题

#### 1. 配置选择菜单组件的按钮

选择菜单组件使用一个按钮取代了标准的select元素，使其能够与jQuery Mobile主题的其他部分相匹配。可以利用data-corners、data-icon、data-iconpos、data-inline和data-mini属性配置按钮的外观及相应属性。前面的章节已经对这些配置属性作了详细介绍，这里不再一一详述。在代码清单31-12中，我演示了其中几个属性的用法，使用它们对按钮的外观作了适当的配置。

代码清单31-12 配置选择菜单按钮

```
...
<div data-role="fieldcontain">
  <label for="speed"><span>Speed: </span></label>
  <select id="speed" name="speed"
    data-iconpos="left" data-icon="gear" data-mini="true">
    <option value="vfast">Very Fast</option>
    <option value="fast">Fast</option>
    <option value="normal" selected>Normal</option>
    <option value="slow">Slow</option>
  </select>
</div>
...
```

我使用data-icon和data-iconpos属性修改了按钮上显示的图标，并把它移到按钮的左边，同时使用data-mini属性让按钮以较小的尺寸显示。这个例子的最终显示效果见图31-10。



图31-10 配置选择菜单的按钮

## 2. 配置选择菜单组件的弹出菜单

当用户点击按钮时，弹出菜单组件默认使用浏览器内建的原生弹出菜单。这是一个明智的决定，因为用户能够充分利用浏览器对当前设备的优化，发挥设备的潜力。图31-11展示了在iOS、Android和Opera浏览器中弹出菜单组件的默认显示效果。

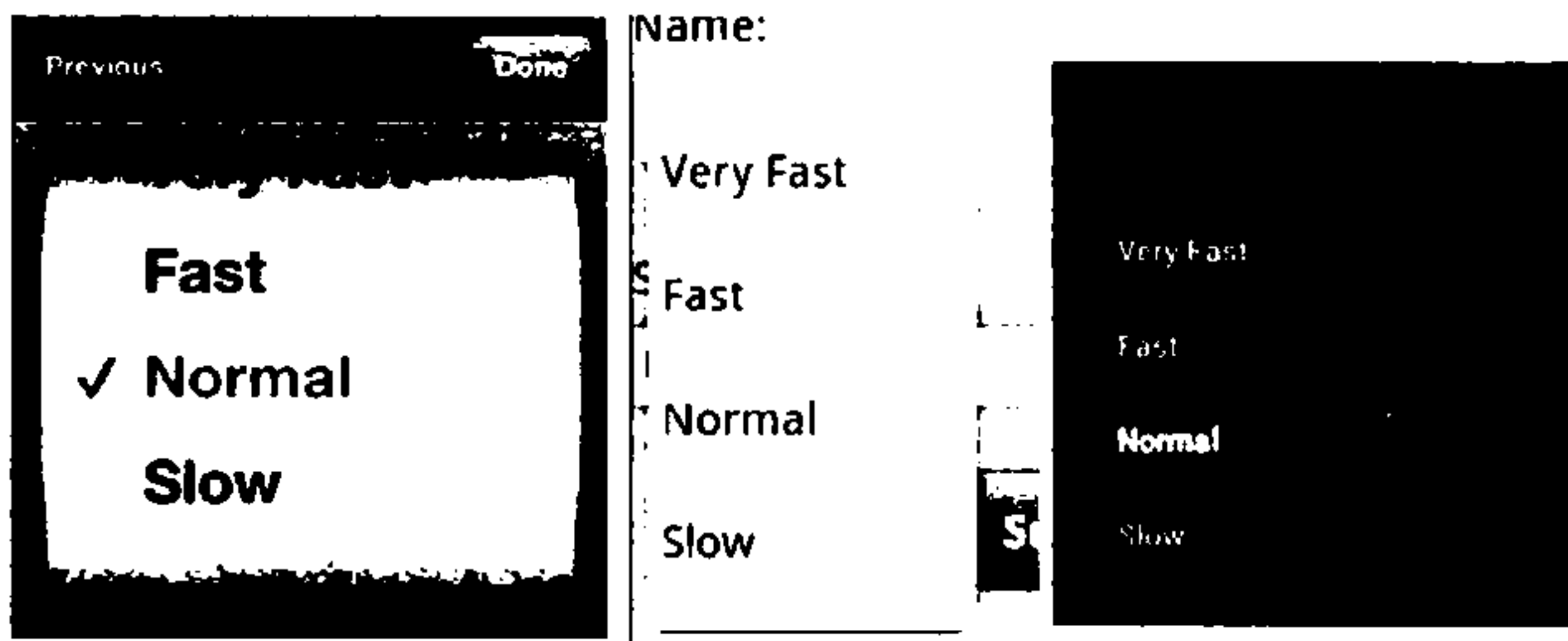


图31-11 原生选择菜单

把data-native-menu属性设置成false，就可以禁用原生菜单，让选择菜单组件与jQuery Mobile主题更加匹配。用户虽然失去了原生菜单的加强特性，界面风格却与网站的其他部分更加一致。在代码清单31-13中，我演示了如何使用data-native-menu属性禁用原生菜单，并使用data-overlay-theme属性设置弹出菜单的风格主题。

### 代码清单31-13 配置选择菜单组件的弹出菜单

```
...
<div data-role="fieldcontain">
  <label for="speed"><span>Speed: </span></label>
  <select id="speed" name="speed" data-native-menu="false" data-overlay-theme="e">
    <option value="vfast">Very Fast</option>
    <option value="fast">Fast</option>
    <option value="normal" selected>Normal</option>
    <option value="slow">Slow</option>
  </select>
</div>
...
```

这个例子的实际显示效果见图31-12。

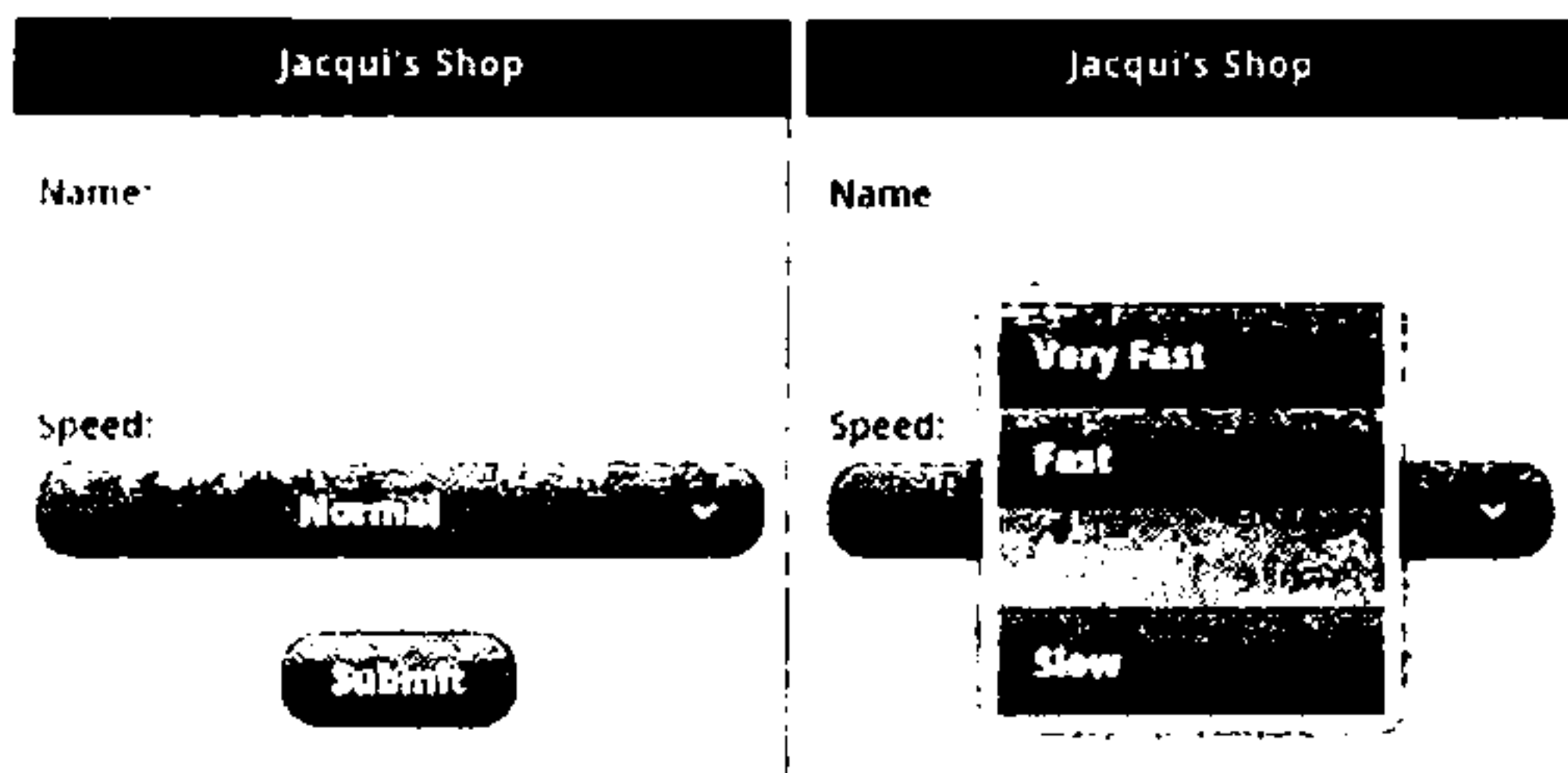


图31-12 禁用原生选择菜单

### 3. 指定占位符

要把select元素内的一个option元素当成占位元素，只需要为它添加data-placeholder属性，并把属性值设置为true。当jQuery Mobile初始化select元素时，会显示占位元素，但它不会出现在供用户选择的弹出菜单中。代码清单31-14演示了data-placeholder属性的用法。

代码清单31-14 data-placeholder属性

```
...
<div data-role="fieldcontain">
  <label for="speed">Speed:
  <select id="speed" name="speed" data-native-menu="false" data-overlay-theme="e">
    <option value="placeholder" data-placeholder="true">Select a Speed</option>
    <option value="vfast">Very Fast</option>
    <option value="fast">Fast</option>
    <option value="normal">Normal</option>
    <option value="slow">Slow</option>
  </select>
</div>
...
```

这个例子的实际显示效果见图31-13。我通常喜欢在选择菜单上使用占位元素，尤其在垂直布局且隐藏了标题元素时，占位元素特别有用，可用来向用户提供上下文信息。

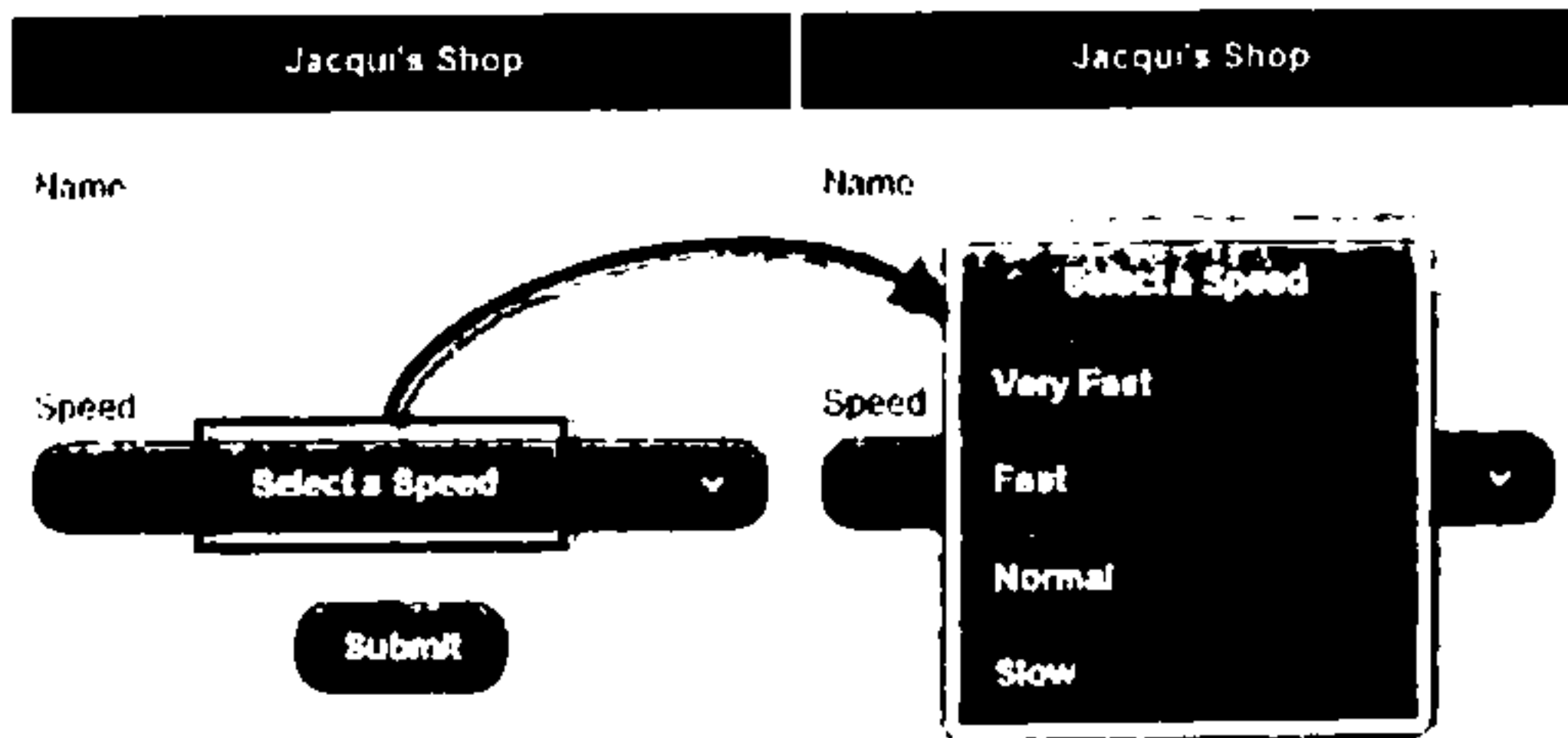


图31-13 指定占位元素

### 31.4.2 选择菜单组件的方法

选择菜单组件定义的方法见表31-9。

表31-9 选择菜单组件方法

方 法	描 述
<code>selectmenu("open")</code>	打开弹出菜单
<code>selectmenu("close")</code>	关闭弹出菜单
<code>selectmenu("disable")</code>	禁用组件，禁止用户选择值
<code>selectmenu("enable")</code>	启用组件，用户可以选择值
<code>selectmenu("refresh")</code>	根据底层元素值的变更刷新组件

在代码清单31-15中，我演示了如何使用按钮控制选择菜单组件。

代码清单31-15 利用脚本控制选择菜单

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript">
    $(document).bind("pageinit", function () {
      $("button").bind("tap", function (e) {
        e.preventDefault();
        if (this.id == "open") {
          $("#speed").selectmenu("open");
          setTimeout(function () {
            $("#speed").selectmenu("close")
          }, 3000);
        } else {
          $("#speed").selectmenu(this.id)
        }
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    [data-role=fieldcontain], .ui-grid-b { margin: 10px; }
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <form method="get">
      <div class="ui-grid-b">
```

```

        <div class="ui-block-a">
            <button id="open">Open</button>
        </div>
        <div class="ui-block-b">
            <button id="enable">Enable</button>
        </div>
        <div class="ui-block-c">
            <button id="disable">Disable</button>
        </div>
    </div>
    <div data-role="fieldcontain">
        <label for="speed"><span>Speed: </span></label>
        <select id="speed" name="speed"
            data-native-menu="false" data-overlay-theme="e">
            <option value="placeholder"
                data-placeholder="true">Select a Speed</option>
            <option value="vfast">Very Fast</option>
            <option value="fast">Fast</option>
            <option value="normal">Normal</option>
            <option value="slow">Slow</option>
        </select>
    </div>
</form>
</div>
</body>
</html>

```

在这个例子中我定义了三个按钮，分别调用open、enable和disable方法。在调用open方法时，我使用setTimeout方法让close方法在3秒钟之后执行。

### 31.4.3 选择菜单组件事件

选择菜单组件仅定义了create事件，它在组件应用于底层元素时触发。

## 31.5 轻触开关组件

如果一个select元素只有两个选项，我们更愿意生成一个轻触开关组件而非标准的选择组件。如代码清单31-16所示，只需把select元素的data-role属性设置为slider，我们就得到一个轻触开关。

代码清单31-16 生成轻触开关

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
    <script type="text/javascript" src="jquery-1.10.1.js"></script>
    <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
    <style type="text/css">

```

```

        #buttonContainer {text-align: center}
        [data-role=fieldcontain] { margin: 10px; text-align: center }
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <form method="get">
            <div data-role="fieldcontain">
                <label for="speed"><span>Speed: </span></label>
                <select id="speed" name="speed" data-role="slider">
                    <option value="fast">Fast</option>
                    <option value="slow">Slow</option>
                </select>
            </div>
            <div data-role="fieldcontain">
                <label for="size">Size: </span></label>
                <select id="size" name="size" data-role="slider">
                    <option value="large">Large</option>
                    <option value="small" selected>Small</option>
                </select>
            </div>
            <div id="buttonContainer">
                <input type="submit" data-inline="true" value="Submit"/>
            </div>
        </form>
    </div>
</body>
</html>

```

在本例中有两个轻触开关。图31-14展示了这两个开关在浏览器中的呈现方式。用户既可以轻触或点击文字，也可以用拖放的方式改变选项的值。

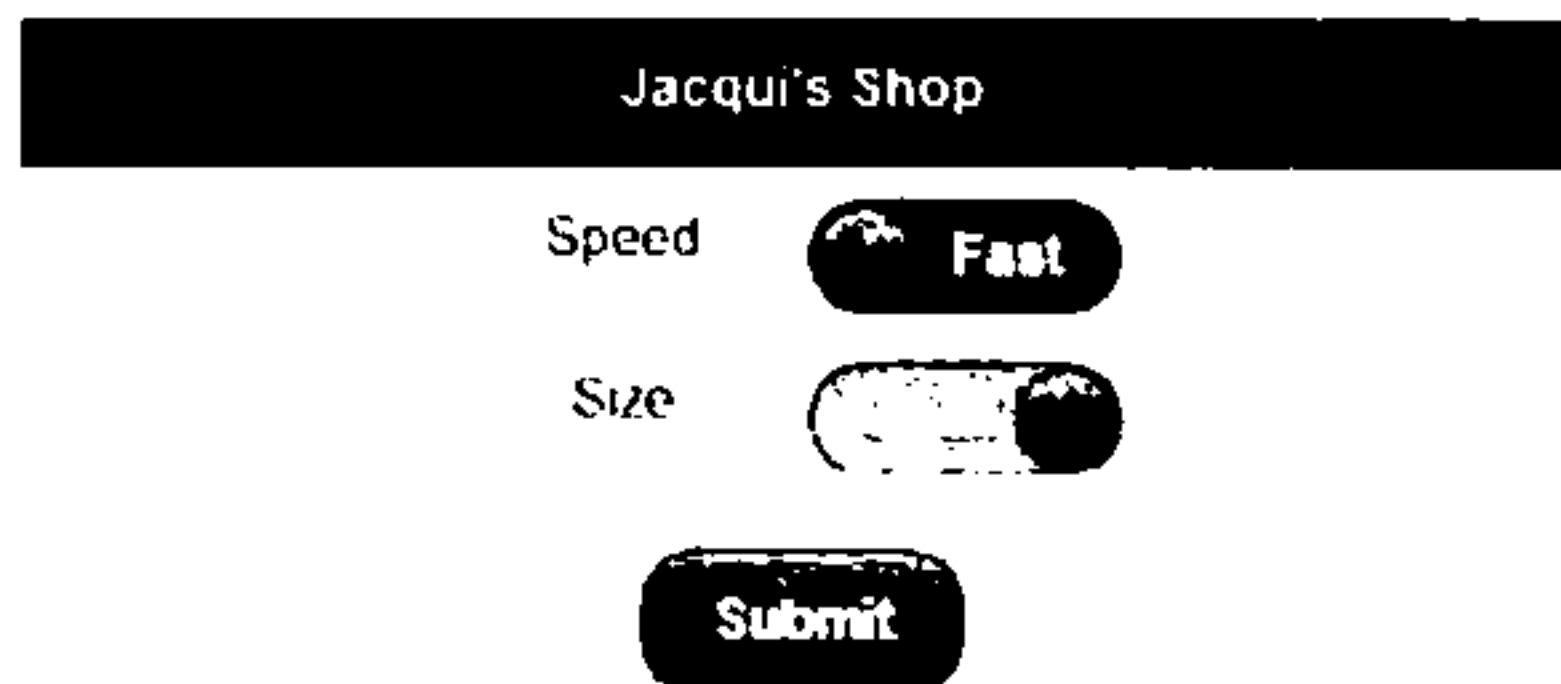


图31-14 轻触开关

## 31.6 复选框和单选钮组件

当jQuery Mobile遇到type属性的值为checkbox的input元素时，就会自动把它加强为复选框；当遇到type属性值为radio的input元素时，就会自动把它们增强为单选钮。

### 31.6.1 创建复选框

创建复选框最简单的方式，是定义一个type属性值为checkbox的input元素，并在input元素之后紧跟一个label元素，参见代码清单31-17。

代码清单31-17 创建复选框

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    #buttonContainer {text-align: center}
    form { margin: 10px; }
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <form method="get">
      <div data-role="fieldcontain">
        <label for="name">Name: </label>
        <input id="name" placeholder="Your Name">
      </div>

      <input type="checkbox" name="check" id="check"/>
      <label for="check">I agree</label>

      <div id="buttonContainer">
        <input type="submit" data-inline="true" value="Submit"/>
      </div>
    </form>
  </div>
</body>
</html>
```

图31-15展示了这种复选框。图中分别给出了复选框在选中和未选中时的样子。

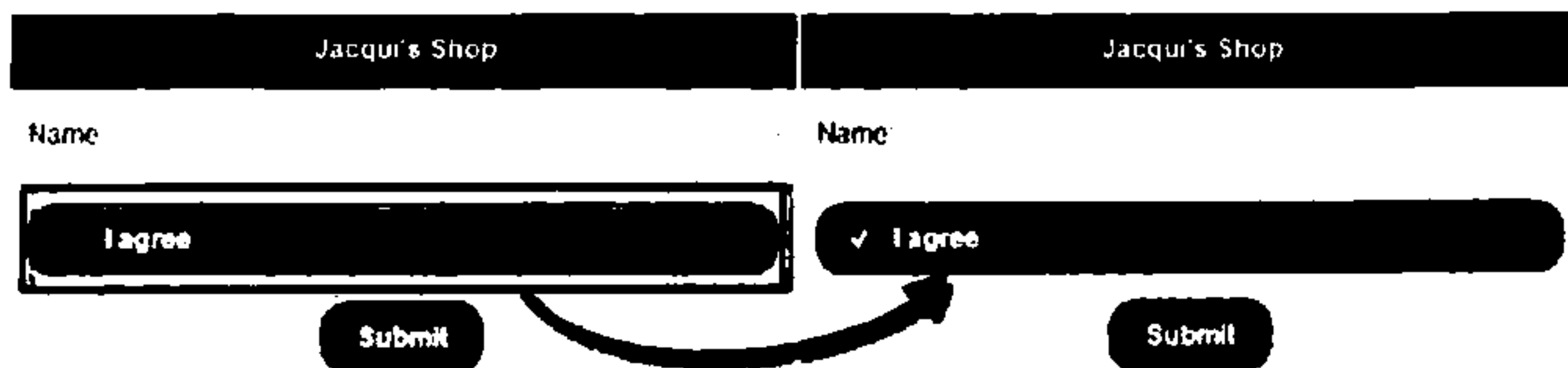


图31-15 jQuery Mobile复选框

### 1. 给复选框加一个标题

默认情况下,复选框宽度就是它父元素的宽度,这意味着本例中的复选框会与屏幕宽度一样宽。如果希望让复选框的位置与它上面的文本框相匹配,如代码清单31-18所示,我们需要使用专门的元素结构。

代码清单31-18 改变复选框的布局

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    #buttonContainer {text-align: center}
    form { margin: 10px; }
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <form method="get">
      <div data-role="fieldcontain">
        <label for="name">Name: </label>
        <input id="name" placeholder="Your Name">
      </div>

      <div data-role="fieldcontain">
        <fieldset data-role="controlgroup">
          <legend>Terms & Conditions:</legend>
          <input type="checkbox" name="check" id="check"/>
          <label for="check">I agree</label>
        </fieldset>
      </div>

      <div id="buttonContainer">
        <input type="submit" data-inline="true" value="Submit"/>
      </div>
    </form>
  </div>
</body>
</html>
```

现在你应该很熟悉包着input元素的包裹元素了——一个data-role属性值为fieldcontain的div元素。这里jQuery Mobile要面对一个问题:input元素已经关联了一个label元素,因此我们只能另辟蹊径以提供jQuery Mobile需要的信息。为此,我们可以添加一个data-role属性值为controlgroup的fieldset元素,并在input元素前增加一个legend元素(内容为我们需要显示的文本)。具体效果见图31-16。



The image shows a mobile web form for 'Jacqui's Shop'. At the top is a header bar with the text 'Jacqui's Shop'. Below the header, there is a 'Name' label followed by a text input field. Underneath the input field is a 'Terms & Conditions' label next to a checkbox. To the right of the checkbox is a button labeled 'I agree'. At the bottom of the form is a large, rounded 'Submit' button.

图31-16 改变复选框的布局

## 2. 对复选框进行分组

可以使用data-role属性值为controlgroup的fieldset元素对多个复选框进行分组。代码清单31-19演示了这种用法。

### 代码清单31-19 复选框分组

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    #buttonContainer {text-align: center}
    form { margin: 10px; }
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <form method="get">
      <div data-role="fieldcontain">
        <label for="name">Name: </label>
        <input id="name" placeholder="Your Name">
      </div>

      <div data-role="fieldcontain">
        <fieldset data-role="controlgroup">
          <legend>Choose Your Flowers:</legend>
          <input type="checkbox" name="roses" id="roses"/>
          <label for="roses">Roses</label>
          <input type="checkbox" name="orchids" id="orchids"/>
          <label for="orchids">Orchids</label>
          <input type="checkbox" name="asters" id="asters"/>
          <label for="asters">Asters</label>
        </fieldset>
      </div>
    </form>
  </div>
```

```

<div data-role="fieldcontain">
  <fieldset data-role="controlgroup" data-type="horizontal">
    <legend>Font:</legend>
    <input type="checkbox" name="bold" id="bold"/>
    <label for="bold"><b>b</b></label>
    <input type="checkbox" name="italic" id="italic"/>
    <label for="italic"><em>i</em></label>
    <input type="checkbox" name="underline" id="underline"/>
    <label for="underline"><u>u</u></label>
  </fieldset>
</div>

<div id="buttonContainer">
  <input type="submit" data-inline="true" value="Submit"/>
</div>
</form>
</div>
</body>
</html>

```

上例中一共有两组复选框。第一组垂直排列，这是复选框的默认用法。jQuery Mobile改变了组件样式，多个input元素之间未留空白，并且只有最外层的元素才有圆角。对于第二组元素，我把fieldset元素的data-type属性设置为horizontal，从而改变了复选框的布局（变为水平布局），并且使复选框隐藏，从而得到一组支持切换开关状态的按钮。具体效果见图31-17。

图31-17 复选框分组

### 31.6.2 创建并格式化单选钮

创建并格式化单选钮的方式与复选框非常相似，见清单代码31-20。

#### 代码清单31-20 创建一组单选钮

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>

```

```

<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
<script type="text/javascript" src="jquery-1.10.1.js"></script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
<style type="text/css">
    #buttonContainer {text-align: center}
    form { margin: 10px; }
</style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <form method="get">
            <div data-role="fieldcontain">
                <label for="name">Name: </label>
                <input id="name" placeholder="Your Name">
            </div>

            <div data-role="fieldcontain">
                <fieldset data-role="controlgroup">
                    <legend>Choose Your Flowers:</legend>
                    <input type="radio" name="flowers" id="roses"/>
                    <label for="roses">Roses</label>
                    <input type="radio" name="flowers" id="orchids"/>
                    <label for="orchids">Orchids</label>
                    <input type="radio" name="flowers" id="asters"/>
                    <label for="asters">Asters</label>
                </fieldset>
            </div>

            <div data-role="fieldcontain">
                <fieldset data-role="controlgroup" data-type="horizontal">
                    <legend>Choose Your Flowers:</legend>
                    <input type="radio" name="flowers" id="roses"/>
                    <label for="roses">Roses</label>
                    <input type="radio" name="flowers" id="orchids"/>
                    <label for="orchids">Orchids</label>
                    <input type="radio" name="flowers" id="asters"/>
                    <label for="asters">Asters</label>
                </fieldset>
            </div>

            <div id="buttonContainer">
                <input type="submit" data-inline="true" value="Submit"/>
            </div>
        </form>
    </div>
</body>
</html>

```

同样，我既创建了一组水平单选按钮，又创建了一组垂直单选按钮。图31-18展示了它们在浏览器中呈现的样子。

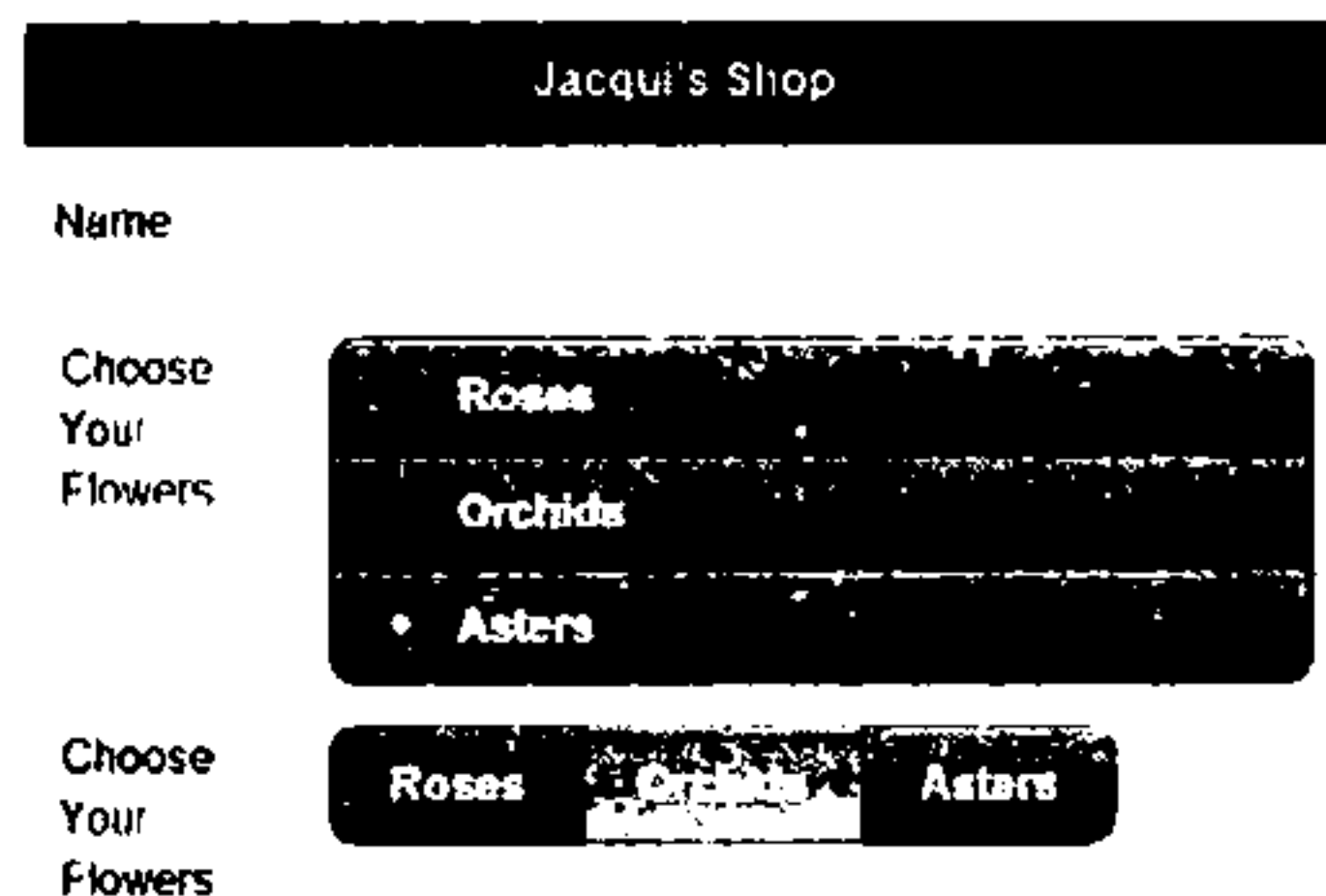


图31-18 两组单选钮

### 31.6.3 配置复选框和单选钮组件

复选框和单选钮组件只定义了一个属性，见表31-10。

表31-10 复选框与单选钮组件配置属性

data属性	选 项	描 述
data-mini	mini	若设置为true，则组件以紧凑模式显示

### 31.6.4 复选框和单选钮组件方法

表31-11展示了复选框和单选钮组件定义的方法。

表31-11 复选框和单选钮组件方法

方 法	描 述
checkboxradio("disable")	禁用组件，禁止用户选择
checkboxradio("enable")	启用组件，允许用户选择
checkboxradio("refresh")	根据底层元素的变化刷新组件

### 31.6.5 复选框和单选钮组件事件

复选框和单选钮组件仅定义了create事件，该事件在组件应用于底层元素时触发。

## 31.7 小结

在本章中，我演示了jQuery Mobile如何增强表单元素，使它们与流行的触摸操作风格一致。在提交表单时，我们无需执行任何特殊操作，就能自动使用Ajax技术，而jQuery Mobile也得以平滑地过渡到服务器返回的结果。我们尽可以信赖jQuery Mobile，让它自动增强表单元素。不过我们也有充分的理由利用一些额外的元素，以及data-role属性，特别是在需要处理select元素的时候。下一章会继续介绍jQuery Mobile组件，重点是列表组件与面板组件。

# 列表组件与面板组件

本章的主题是jQuery Mobile列表组件和面板组件。在移动应用中，列表组件是一个重要的工具，人们经常使用它创建简单醒目的导航，供用户在不同功能区之间跳转。列表组件的优点在于非常紧凑（节省空间），同时又允许单个列表项足够大以满足触摸选中的需要。它显而易见，易于理解。在列表项的右边放一个箭头按钮（jQuery Mobile的默认行为），大多数用户立刻就能明白此项拥有子项。

面板组件是一个通用组件，它通常用于在窗口的左栏或右栏根据上下文显示内容。面板可用于显示任意内容，不过人们最常把它们用于显示选项或者全局导航。表32-1列出了本章概要。

表32-1 本章概要

问 题	解决方法	代码清单
如何创建列表	定义一个包含1个或多个li元素的ul或ol元素，把这些li元素的data-role属性设置为listview。这些li元素的内容应该是链接	1
如何创建内嵌列表	把ul或ol元素的data-inset属性设置为true	2
如何创建每个列表项都由两部分组成的列表	为每个li元素添加第2个链接	3
如何允许用户过滤列表内容	把ul或ol元素的data-filter属性设置为true	4、5
如何在列表中使用分隔	把用作分隔的li元素的data-role属性设置为list-divider	6
如何为列表项添加一个表示计数的泡泡	给这个li添加ui-li-count类	7
如何强调文本	使用h1-h6和p元素	8
如何为列表项添加侧栏	使用ui-li-aside类	9
如何生成面板	把页中一个与面板关联的div元素的data-role属性设置成panel即可	10
如何设置面板的位置和显示风格	使用data-display和data-position属性	11
设置面板以何种方式消失	使用data-swipe-close和data-dismissable属性	12

## 32.1 列表组件

使用jQuery Mobile内建的列表组件可处理各种各样的列表。代码清单32-1展示了一个链接到同一文件不同jQuery Mobile页的简单列表。每页都描述了一种花卉产品，使用列表组件，用户能够轻松前往任何一个页。

## 代码清单32-1 一个简单列表

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    .lcontainer {float: left; text-align: center; padding-top: 10px}
    .productData {float: right; padding: 10px; width: 60%}
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>

    <ul data-role="listview">
      <li><a href="#roses">Roses</a></li>
      <li><a href="#orchids">Orchids</a></li>
      <li><a href="#asters">Asters</a></li>
    </ul>
  </div>

  <div id="roses" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Roses</h1>
    </div>
    <div>
      <div class="lcontainer">
        
        <div>
          <a href="#" data-rel="back" data-role="button"
            data-inline=true data-direction="reverse">Back</a>
        </div>
      </div>
      <div class="productData">
        A rose is a woody perennial within the family Rosaceae.
        They form a group of erect shrubs, and climbing or trailing plants.
        <div><b>Price: $4.99</b></div>
      </div>
    </div>
  </div>

  <div id="orchids" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Orchids</h1>
    </div>
    <div>
      <div class="lcontainer">
        

```

```

        <div>
            <a href="#" data-rel="back" data-role="button"
                data-inline=true data-direction="reverse">Back</a>
        </div>
    </div>
    <div class="productData">
        The orchid family is a diverse and widespread family in the order
        Asparagales. It is one of the largest families of flowering plants.
        <div><b>Price: $10.99</b></div>
    </div>
</div>
</div>

<div id="asters" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Asters</h1>
    </div>
    <div>
        <div class="lcontainer">
            
            <div>
                <a href="#" data-rel="back" data-role="button"
                    data-inline=true data-direction="reverse">Back</a>
            </div>
        </div>
        <div class="productData">
            The name Aster comes from the Ancient Greek word meaning "star",
            referring to the shape of the flower head.
            <div><b>Price: $2.99</b></div>
        </div>
    </div>
</div>
</body>
</html>

```

这个页面的绝大部分都是展示花卉的代码。列表代码只有如下寥寥几行：

```

...
<ul data-role="listview">
    <li><a href="#roses">Ros/bes</a></li>
    <li><a href="#orchids">Orchids</a></li>
    <li><a href="#asters">Asters</a></li>
</ul>
...

```

---

**提示** 我在这个例子中使用的是ul元素，其实也可以用ol（有序列表）元素，jQuery Mobile对这两个元素同等对待。

---

这是一个标准的无序列表，由一个包含三个li元素的ul元素组成。为了应用列表组件，我把ul元素的data-role属性设置为listview。

列表组件的基本用途是提供导航，为此，我在每个li元素中都放了一个链接，指向同一文件的其

他页。在列表中点击或触摸列表项，就会显示相应的页。在图32-1中可以看到列表组件与其中一个内容页面。我在每个页中添加了一个基于链接的返回按钮，使用标准跳转（反向），方便用户回到列表

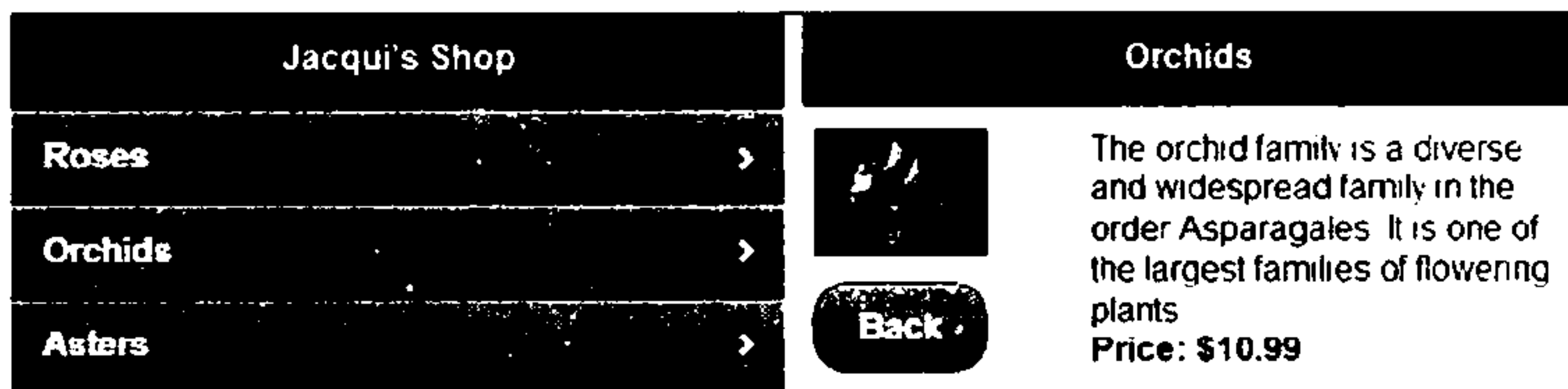


图32-1 一个简单的jQuery Mobile列表

### 32.1.1 配置列表组件

列表组件支持许多属性和配置选项，利用这些属性和选项，我们可以定制列表的外观和行为。这些属性及选项见表32-2。在下面的内容中，我将演示这些属性与选项的用法。

表32-2 列表组件定义的属性与配置选项

data属性	选 项	描 述
data-count-theme	countTheme	指定数字泡泡的显示风格
data-divider-theme	dividerTheme	指定分隔线的显示风格
data-filter	filter	若设置为true，则列表组件提供过滤器
N/A	filterCallback	指定用于过滤列表项的回调函数
data-filter-placeholder	filterPlaceholder	指定过滤器的占位内容
data-filter-theme	filterTheme	指定过滤器搜索栏的显示风格
data-header-theme	headerTheme	指定嵌套标题的显示风格
data-icon	icon	指定标题所用图标
data-inset	inset	若设置为true，则列表组件以内嵌风格显示列表
data-split-icon	splitIcon	为分栏列表指定图标
data-split-theme	splitTheme	指定分栏列表的显示风格

#### 1. 生成内嵌的列表

列表的默认布局是充满容器元素（窗口），并且四角都是直角，这与其他jQuery Mobile组件显著不同。为了让样式一致，我们可以生成内嵌风格的列表。圆角，四周留白，不会紧接屏幕边缘。如代码清单32-2所示，只需把ul或ol元素的data-inset属性设置为true，我们就得到了一个内嵌风格的列表。

代码清单32-2 生成内嵌的列表

```
...
<div id="page1" data-role="page" data-theme="b">
```



```

<div data-role="header">
  <h1>Jacqui's Shop</h1>
</div>

<div id="container" style="padding: 20px">
  <ul data-role="listview" data-inset=true>
    <li><a href="#roses">Roses</a></li>
    <li><a href="#orchids">Orchids</a></li>
    <li><a href="#asters">Asters</a></li>
  </ul>
</div>
</div>
...

```

在这个例子中，我把ul元素放在一个div元素内，并使用css padding属性设置内嵌列表与父元素之间的空白；然后，使用data-inset属性改变列表的样式。这个例子的效果见图32-2。

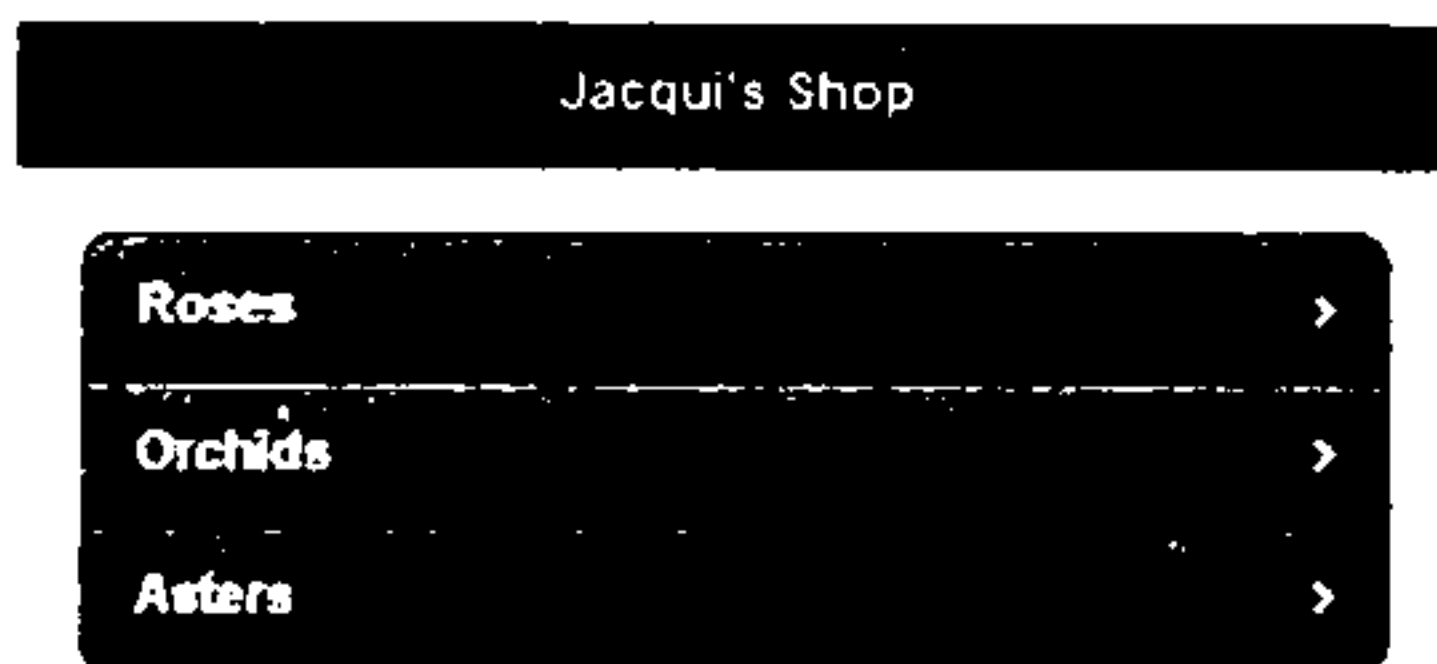


图32-2 生成内嵌的列表

## 2. 生成分栏的列表

当每个列表项都有两种行为时，分栏列表就能派上用场了。在分栏列表中，列表项被分成两个部分，点击或触摸不同的部分会导致不同的行为。代码清单32-3演示了一个分栏列表，我们既能通过它查看花卉的信息，又能直接把它添加到购物车。

### 代码清单32-3 分栏列表

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    .lcontainer {float: left; text-align: center; padding-top: 10px}
    .productData {float: right; padding: 10px; width: 60%}
    .cWrapper {text-align: center; margin: 20px}
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">

```

```

        <h1>Jacqui's Shop</h1>
    </div>
    <div id="container" style="padding: 20px">
        <ul data-role="listview" data-inset=true>
            <li>
                <a href="#basket" class="buy" id="rose">Roses</a>
                <a href="#roses">Roses</a></li>
            <li>
                <a href="#basket" class="buy" id="orchid">Orchids</a>
                <a href="#orchids">Orchids</a>
            </li>
            <li>
                <a href="#basket" class="buy" id="aster">Asters</a>
                <a href="#asters">Asters</a>
            </li>
        </ul>
    </div>
</div>

<div id="basket" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Jacqui's Shop</h1>
    </div>
    <div class="cWrapper">
        Basket will go here
    </div>
    <div class="cWrapper">
        <a href="#" data-rel="back" data-role="button" data-inline=true
            data-direction="reverse">Back</a>
    </div>
</div>

<div id="roses" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Roses</h1>
    </div>
    <div>
        <div class="lcontainer">
            
            <div>
                <a href="#" data-rel="back" data-role="button"
                    data-inline=true data-direction="reverse">Back</a>
            </div>
        </div>
        <div class="productData">
            A rose is a woody perennial within the family Rosaceae.
            They form a group of erect shrubs, and climbing or trailing plants.
            <div><b>Price: $4.99</b></div>
        </div>
    </div>
</div>

<div id="orchids" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Orchids</h1>
    </div>

```

```

</div>
<div>
  <div class="lcontainer">
    
    <div>
      <a href="#" data-rel="back" data-role="button"
        data-inline=true data-direction="reverse">Back</a>
    </div>
  </div>
  <div class="productData">
    The orchid family is a diverse and widespread family in the order
    Asparagales. It is one of the largest families of flowering plants.
    <div><b>Price: $10.99</b></div>
  </div>
</div>
</div>

<div id="asters" data-role="page" data-theme="b">
  <div data-role="header">
    <h1>Asters</h1>
  </div>
  <div>
    <div class="lcontainer">
      
      <div>
        <a href="#" data-rel="back" data-role="button"
          data-inline=true data-direction="reverse">Back</a>
      </div>
    </div>
    <div class="productData">
      The name Aster comes from the Ancient Greek word meaning "star",
      referring to the shape of the flower head.
      <div><b>Price: $2.99</b></div>
    </div>
  </div>
</div>
</div>
</body>
</html>

```

要生成分栏列表，只需要在li元素中添加第二个链接。jQuery Mobile把列表项分为两块，并在两块之间插入了一条竖线作为分隔线。点击或触摸左边的部分会访问第一个链接，点击或触摸右边部分则会访问第二个链接。分栏列表在浏览器中显示的样子见图32-3。



图32-3 生成分栏列表

在这个例子中，我把所有列表项的左侧链接都指向页面中新加的一个页面——basket。在下一章，我们还会回到这个例子，把它扩展为一个简单的购物车。对本例来说，basket页只是一个简单的占位符。

**提示** jQuery Mobile默认为右侧的按钮使用箭头图标。我们可通过给ul或ol元素添加data-split-icon属性并指定希望使用的图标，来改变这个图标。在第30章中有一个列表，列出了所有可用的图标。

### 3. 过滤列表

列表组件提供了过滤列表内容的机制。如代码清单32-4所示，给ul或ol元素添加一个data-filter属性并把值设置为true，就可以启用列表过滤功能。

代码清单32-4 列表过滤

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    .lcontainer {float: left; text-align: center; padding-top: 10px}
    .productData {float: right; padding: 10px; width: 60%}
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div data-role="content">
      <ul data-role="listview" data-inset=true data-filter=true>
        <li><a href="#roses">Roses</a></li>
        <li><a href="#orchids">Orchids</a></li>
        <li><a href="#asters">Asters</a></li>
      </ul>
    </div>
  </div>

  <div id="roses" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Roses</h1>
    </div>
    <div>
      <div class="lcontainer">
        
      </div>
      <a href="#" data-rel="back" data-role="button"
        data-inline=true data-direction="reverse">Back</a>
    </div>
  </div>
</body>
</html>
```

```

        </div>
    </div>
    <div class="productData">
        A rose is a woody perennial within the family Rosaceae.
        They form a group of erect shrubs, and climbing or trailing plants.
        <div><b>Price: $4.99</b></div>
    </div>
</div>
</div>
<div id="orchids" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Orchids</h1>
    </div>
    <div>
        <div class="lcontainer">
            
            <div>
                <a href="#" data-rel="back" data-role="button"
                    data-inline=true data-direction="reverse">Back</a>
            </div>
        </div>
        <div class="productData">
            The orchid family is a diverse and widespread family in the order
            Asparagales. It is one of the largest families of flowering plants.
            <div><b>Price: $10.99</b></div>
        </div>
    </div>
</div>
</div>
<div id="asters" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Asters</h1>
    </div>
    <div>
        <div class="lcontainer">
            
            <div>
                <a href="#" data-rel="back" data-role="button"
                    data-inline=true data-direction="reverse">Back</a>
            </div>
        </div>
        <div class="productData">
            The name Aster comes from the Ancient Greek word meaning "star",
            referring to the shape of the flower head.
            <div><b>Price: $2.99</b></div>
        </div>
    </div>
</div>
</div>
</body>
</html>

```

如图32-4所示，jQuery Mobile在列表上方添加了一个搜索框。当用户在搜索框中输入字符时，jQuery Mobile会挪走那些不包含输入字符的列表项。默认情况下，只有用户键入两个以上字母之后，

组件才会启动过滤功能。

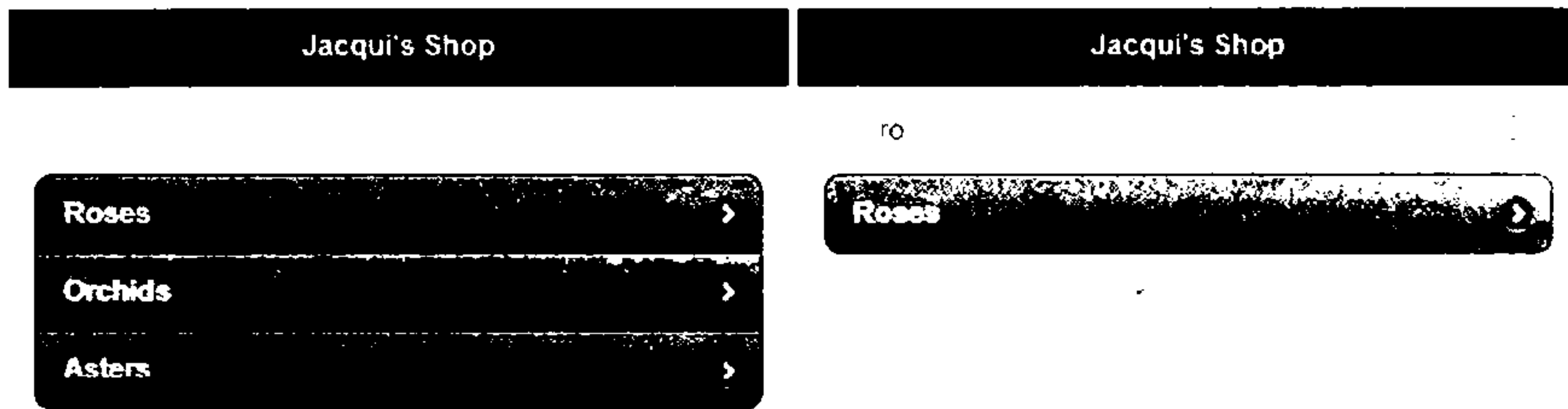


图32-4 启用列表过滤功能

**警告** 支持列表过滤固然很好，然而在小小的触摸屏幕上，这一功能并不总是可以派上用场。当用户激活一个类似过滤搜索框的文本框时，绝大多数移动设备会立刻弹出触摸键盘，以备用户输入。在小屏幕设备上，这个软键盘会占据很大的屏幕空间，以致用户很难看到过滤结果。并不是说我们因此就得放弃列表过滤功能，只是如果我们面向的目标设备是小屏幕设备，在提供列表过滤功能的同时提供其他的导航机制是非常重要的。

#### 4. 自定义过滤函数

默认的过滤机制是过滤出所有包含键入字符串的列表项，不区分大小写，也不关心匹配位置。如代码清单32-5所示，我们也可以使用jQuery UI风格的方法为jQuery Mobile提供自定义过滤函数。

##### 代码清单32-5 定制过滤函数

```
...
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript">
    $(document).bind("pageinit", function () {
      $("ul").listview("option", "filterCallback", function(listItem, filter) {
        var pattern = new RegExp("^" + filter, "i");
        return !pattern.test(listItem)
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js">
  <style type="text/css">
    .lcontainer {float: left; text-align: center; padding-top: 10px}
    .productData {float: right; padding: 10px; width: 60%}
  </style>
```

```
</head>
...
```

我们可以调用listview方法把filterCallback选项的值设置为定制的回调函数。这个函数接受两个参数，一个是列表项文本listItem，一个是用户输入的文本filter。列表中的每个项都会调用一次这个函数，若函数返回true，当前列表项就会隐藏。在本例中，我使用了正则表达式，要求过滤文本必须匹配列表项文本的起始部分（仍然不区分大小写）。这个脚本的具体效果见图32-5。当我们键入字母R时，只有Roses项通过了测试并显示出来。



图32-5 定制过滤函数

## 5. 添加分隔

列表组件可以在列表项之间添加分隔。只要把元素的data-role属性设置为list-divider，即可添加分隔。给定义列表的ul或ol元素定义data-divider-theme属性，还可以为这些分隔设置显示风格。

当遇到过长或者相当复杂的列表时，在不改变用户使用习惯的前提下，列表分隔非常有用。在代码清单32-6中，我添加了几个列表分隔，并使用data-divider-theme属性为这些分隔设置显示风格。

### 代码清单32-6 列表分隔符

```
...
<div id="page1" data-role="page" data-theme="b">
  <div data-role="header">
    <h1>Jacqui's Shop</h1>
  </div>

  <div data-role="content">
    <ul data-role="listview" data-inset=true data-theme="c"
      data-divider-theme="b">

      <li data-role="list-divider">A</li>
      <li><a href="#asters">Asters</a></li>
      <li data-role="list-divider">C</li>
      <li><a href="document2.html">Carnations</a></li>
      <li data-role="list-divider">D</li>
      <li><a href="document2.html">Daffodils</a></li>
      <li data-role="list-divider">L</li>
      <li><a href="document2.html">Lilies</a></li>
      <li data-role="list-divider">O</li>
      <li><a href="#orchids">Orchids</a></li>
      <li data-role="list-divider">P</li>
      <li><a href="document2.html">Peonies</a></li>
      <li><a href="document2.html">Primulas</a></li>
      <li data-role="list-divider">R</li>
```

```

        <li><a href="#roses">Roses</a></li>
        <li data-role="list-divider">S</li>
        <li><a href="document2.html">Snowdrops</a></li>
    </ul>
</div>
</div>
...

```

这些分隔符的显示效果见图32-6。

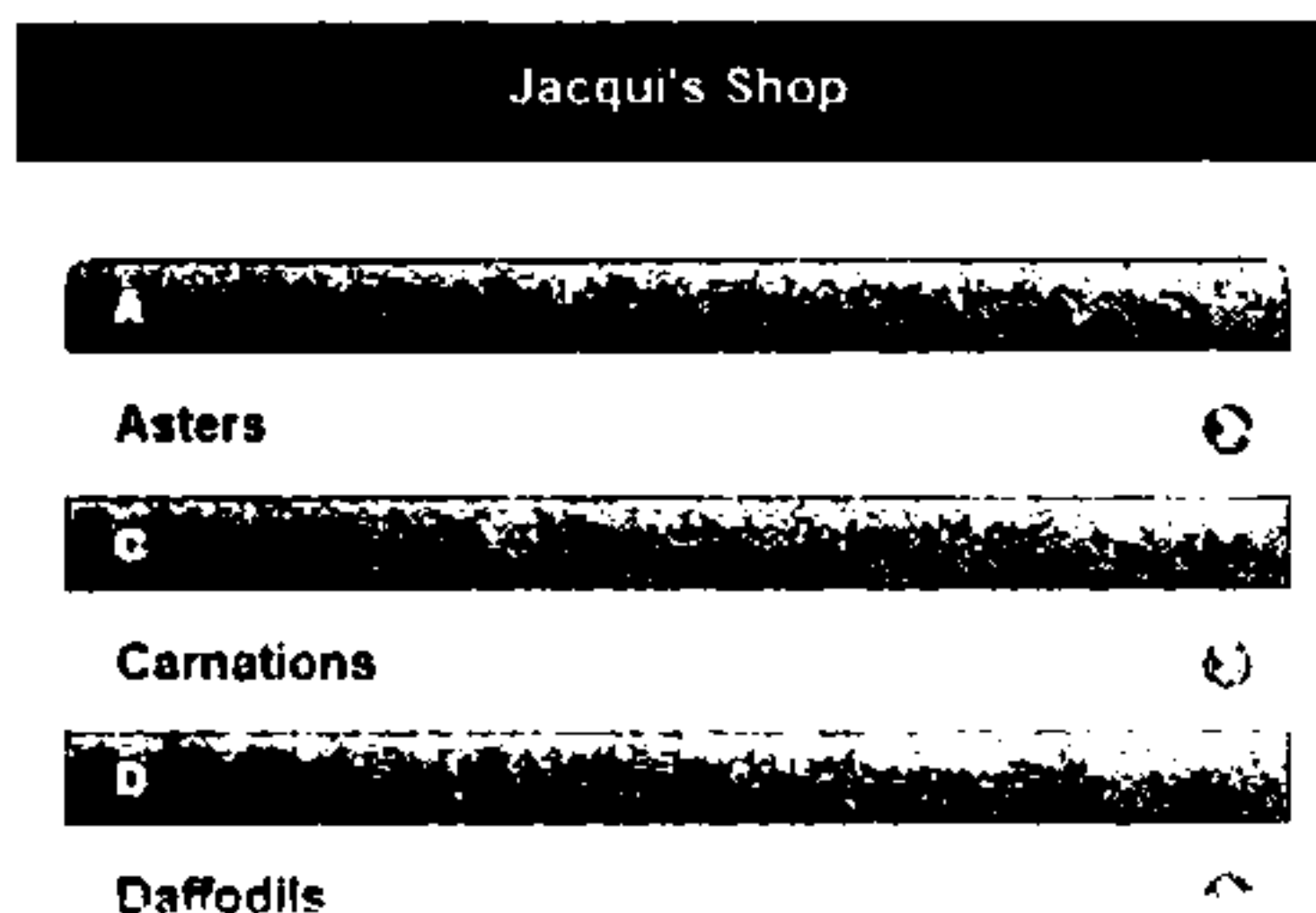


图32-6 为列表添加分隔符

**提示** 如果你希望特别显示某一项，也可以为该li元素单独指定data-theme属性。

## 6. 基于约定的格式

jQuery Mobile中的一些配置选项通过约定而不是配置处理。前面的分栏列表就是这样一个例子。如果我们为li元素添加第2个链接，jQuery Mobile就会自动生成分栏列表项。我们无需使用data属性就能实现这样的效果——结构决定了结果。在本节中，我将演示三种控制列表项的约定格式：计数泡泡、强调文本和边栏。

### ● 计数泡泡

我们可以为每一个列表项添加一个数字指示。当列表项表示某种数据的分类，而我们希望提供分类的总数时，这种称为计数泡泡的技术就非常有用。举例来说，如果我们的列表项表示Email邮件夹，我们可以用计数泡泡表示每个邮件夹里有多少封邮件。也可以使用计数泡泡来表示在一个电子商务网站的仓库里有多少种商品。

这种效果通常被用来显示数字，其实它可以显示任何信息。由于我们除了值以外没有空间显示解释信息，这些值必须能够做到望文知义。

在li元素中额外增加一个后代元素<sup>①</sup>，就创建了一个计数泡泡。这个子元素必须有内容，并且设置了ul-li-count属性。代码清单32-7演示了如何定义计数泡泡，包括一个非数字值的计数泡泡。

### 代码清单32-7 为列表项添加计数泡泡

```

...
<div id="page1" data-role="page" data-theme="b">
    <div data-role="header">

```



```

    <h1>Jacqui's Shop</h1>
  </div>
  <div data-role="content">
    <ul data-role="listview" data-inset=true data-filter=true>
      <li><a href="#roses">Roses<div class="ui-li-count">23</div></a></li>
      <li><div class="ui-li-count">7</div><a href="#orchids">Orchids</a></li>
      <li><a href="#asters">Asters</a><div class="ui-li-count">Pink</div></li>
    </ul>
  </div>
</div>
...

```

注意jQuery Mobile并未限制这个额外添加的后代元素的位置。它可以位于li元素内的任何位置，不一定是最后一个子元素（尽管通过都把它放到那里）。计数泡泡的显示效果见图32-7。

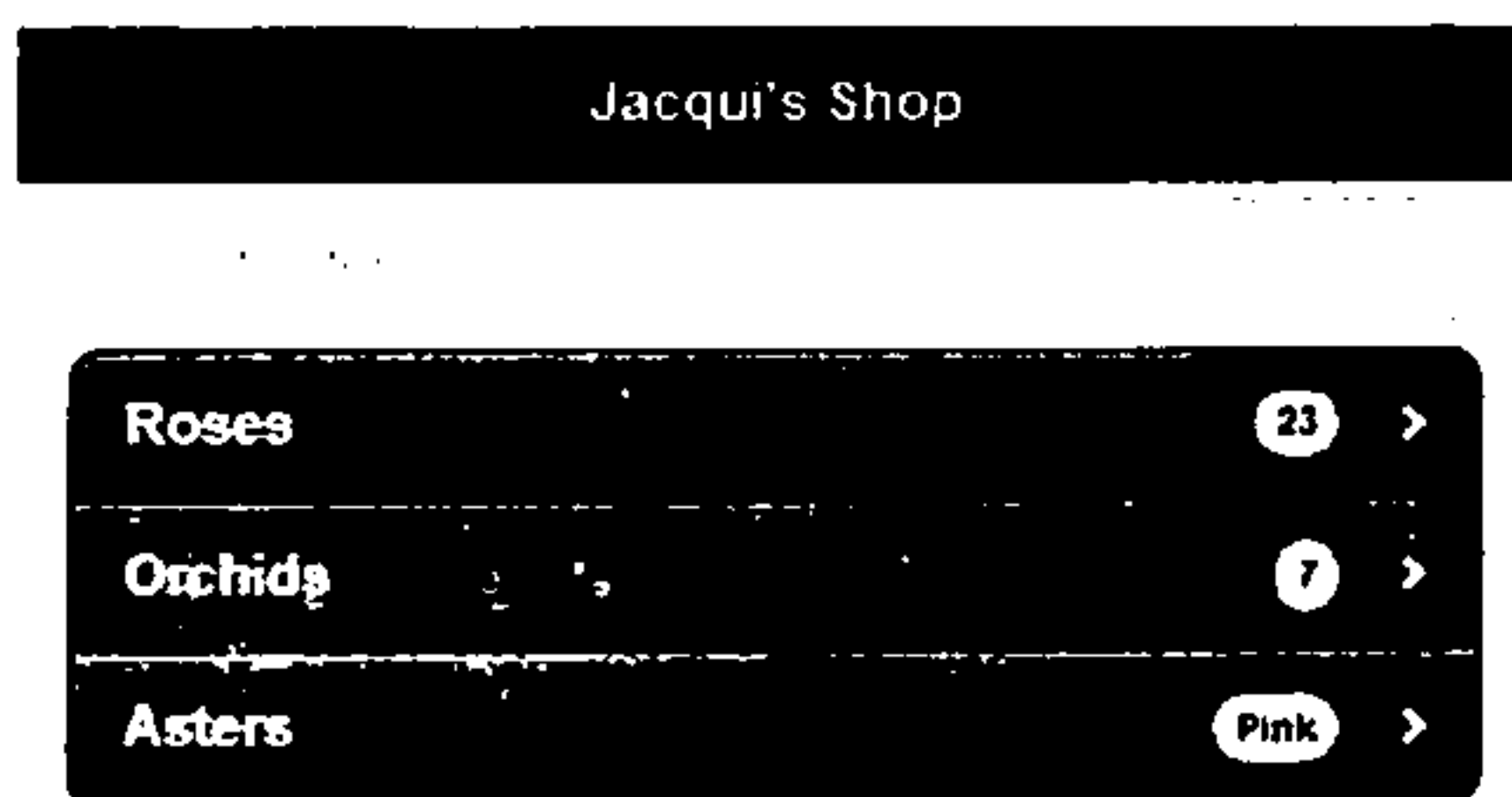


图32-7 计数泡泡

### ● 强调列表文本

如果我们在列表项内容中使用表示标题的h1-h6元素而不是表示段落的p元素，列表组件会自动添加不同级别的强调样式。如代码清单32-8所示，这样我们就生成了既包含标题又包含描述文本的列表项。

### 代码清单32-8 添加强调文本

```

...
<div id="page1" data-role="page" data-theme="b">
  <div data-role="header">
    <h1>Jacqui's Shop</h1>
  </div>

  <div data-role="content">
    <ul data-role="listview" data-inset=true data-filter=true>
      <li>
        <a href="#roses"><h1>Roses</h1>
          <p>A rose is a woody perennial within the family Rosaceae.</p>
          <div class="ui-li-count">$4.99</div></a>
      </li>
      <li><div class="ui-li-count">7</div><a href="#orchids">Orchids</a></li>
      <li><a href="#asters">Asters</a><div class="ui-li-count">Pink</div></li>
    </ul>
  </div>

```

```

    </div>
  </div>
  ...

```

在本例中，我使用h1元素表示产品的名字，使用p元素表示产品的详细信息。我在列表项中包含了一个计数泡泡，来表示这个产品的价格（计数泡泡非常适合用来显示产品价格，因为货币单位为数字值提供了明确的含义）。具体效果见图32-8。

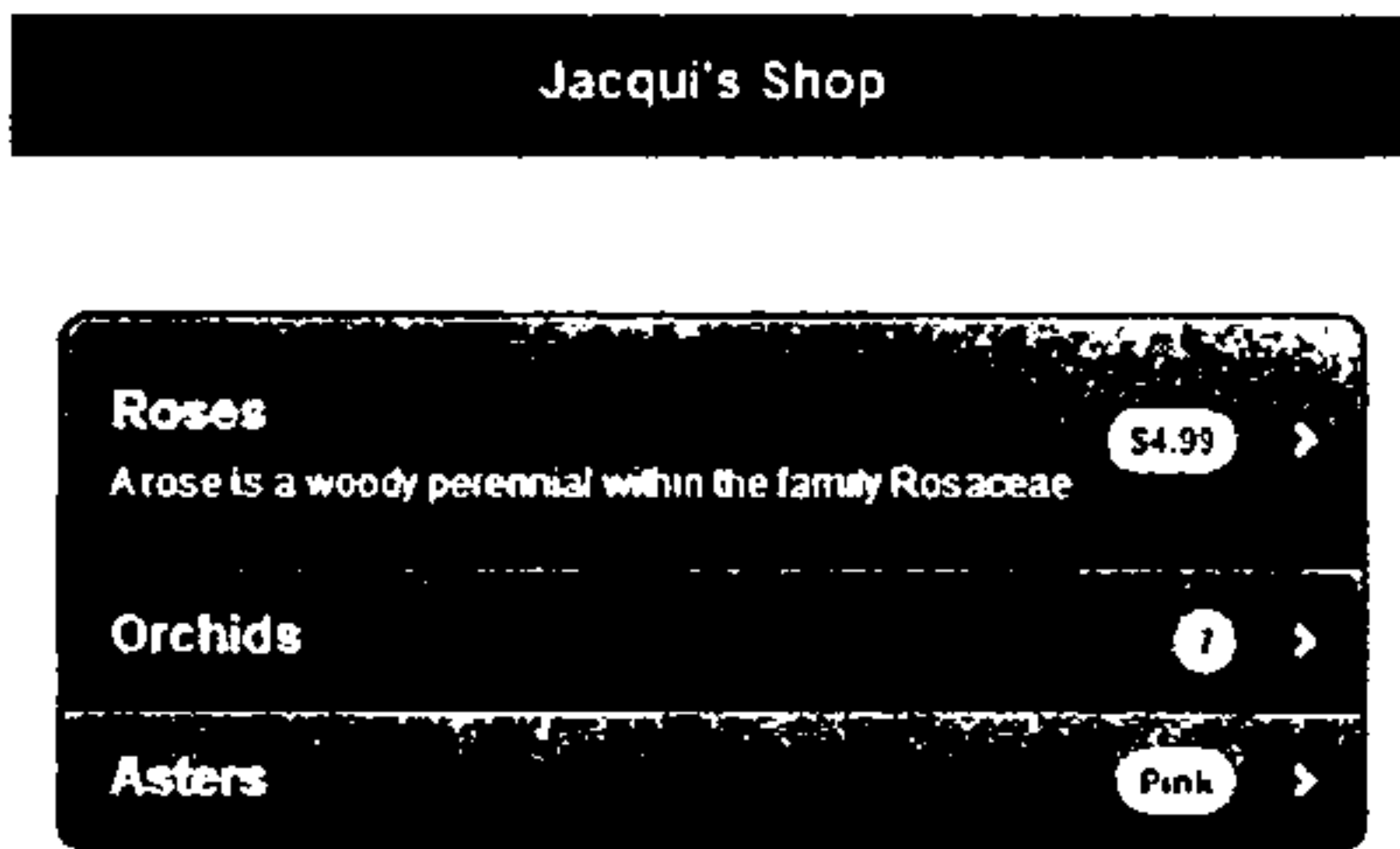


图32-8 在列表项中使用强调文本

#### ● 添加侧栏

侧栏可以用来代替计数泡泡。在li元素内添加一个包含我们打算显示信息的子元素，并为它添加ui-li-aside类，这个元素就会显示在右侧：侧栏效果。代码清单32-9展示了侧栏的用法。

#### 代码清单32-9 在一个列表项中使用侧栏

```

...
<div id="page1" data-role="page" data-theme="b">
  <div data-role="header">
    <h1>Jacqui's Shop</h1>
  </div>

  <div data-role="content">
    <ul data-role="listview" data-inset=true data-filter=true>
      <li>
        <a href="#roses">
          <h1>Roses</h1>
          <p>A rose is a woody perennial within the family Rosaceae.</p>
          <p class="ui-li-aside">(Pink) <strong>$4.99</strong></p>
        </a>
      </li>
      <li><div class="ui-li-count">7</div><a href="#orchids">Orchids</a></li>
      <li><a href="#asters">Asters</a><div class="ui-li-count">Pink</div></li>
    </ul>
  </div>
</div>
...

```

在图32-9中，你可以看到Roses产品项的侧栏信息显示效果。

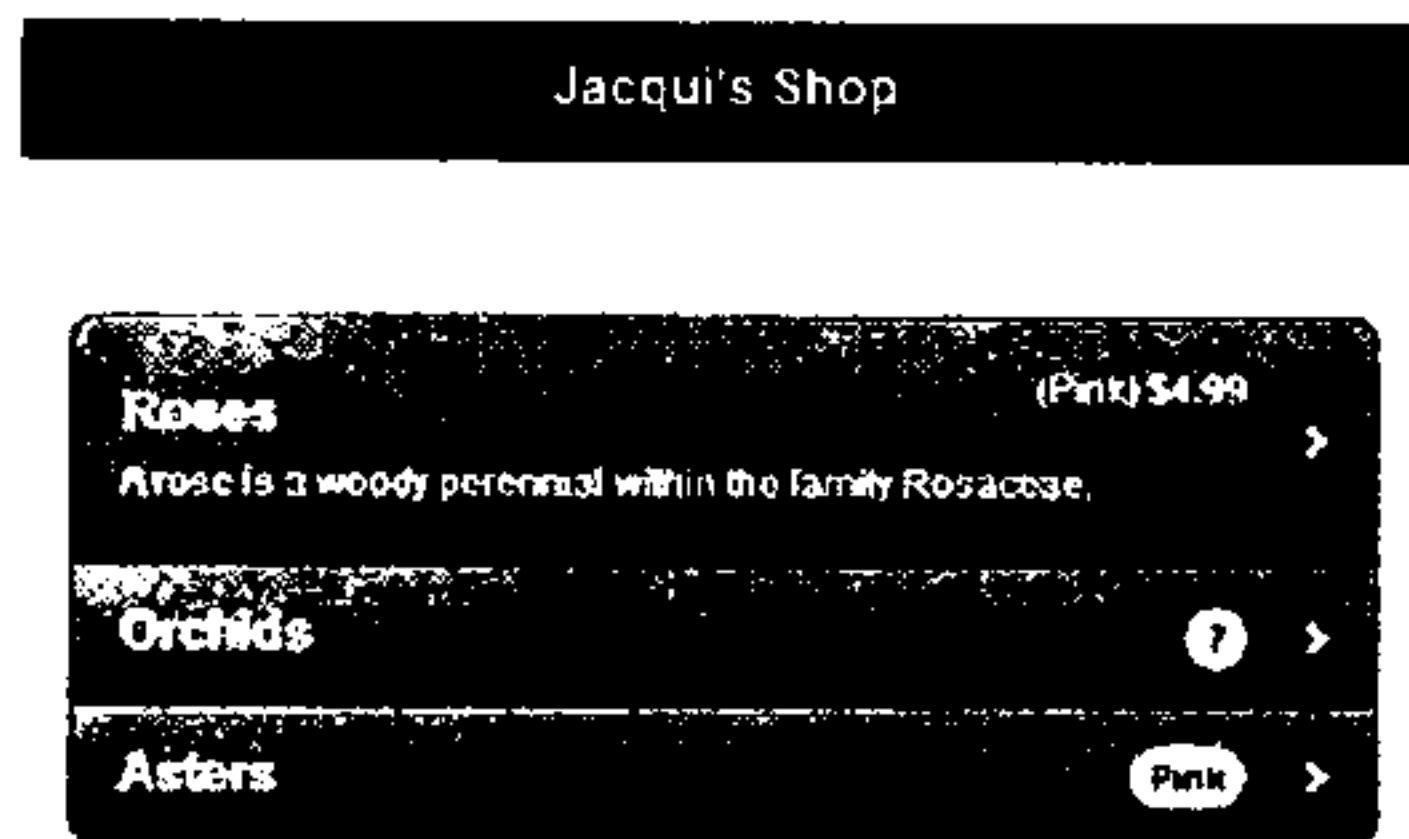


图32-9 使用侧栏

### 32.1.2 列表组件方法

列表组件只定义了一个方法，见表32-3。

表32-3 列表组件方法

方 法	描 述
listview("refresh")	根据底层元素变化刷新组件

### 32.1.3 列表组件事件

列表组件仅定义了create事件，该事件在组件应用于底层元素时触发。

## 32.2 面板组件

面板组件出现于当前面的左边或者右边，它可用于显示任何内容，通常用于访问导航选项和设置选项。如代码清单32-10所示，把div元素的data-role属性设置为panel即可生成一个面板。

代码清单32-10 生成面板组件

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style>
    .buttonContainer { text-align: center; }
  </style>
</head>
<body>

  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
```

```

    <h1>Jacqui's Shop</h1>
  </div>

  <div data-role="content" class="buttonContainer">
    <a data-role="button" data-inline="true" href="#panel">Open Panel</a>
  </div>

  <div id="panel" data-role="panel" data-theme="a">
    <div data-role="panel-content">
      <h3>Simple Panel</h3>
      <p>This is the the panel</p>
      <button data-rel="close" data-inline="true">Close</button>
    </div>
  </div>
</div>
</body>
</html>

```

面板的内容由页面内定义面板的元素决定。在这个例子中，面板包含几个简单的HTML元素，这些元素由一个data-role属性值为panel-content的div元素包裹，从而确保内容会显示在面板的适当位置。

示例中的主页包含一个href属性指向面板元素id的a元素。点击这个链接，或者说这个按钮（我把这个a元素的data-role属性设置成了button），就会打开这个面板。这个例子的实际效果见图32-10。

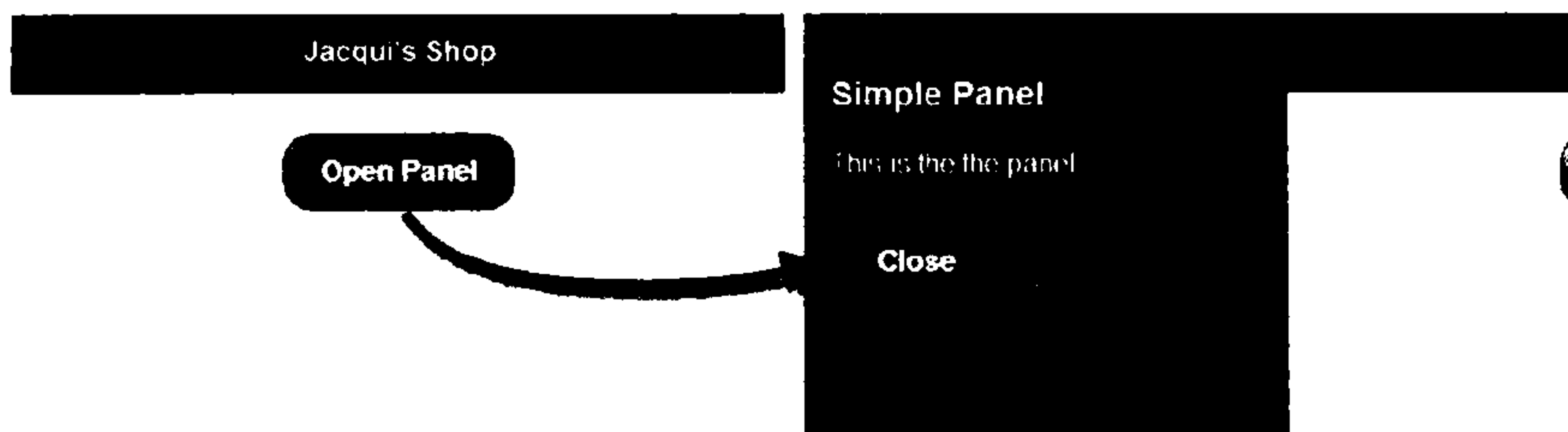


图32-10 应用面板组件

这是一个需要亲自运行才能得到第一手体验的例子。点击Open Panel按钮，主页即滑向右边，面板闪现。点击面板中的Close按钮（它的data-rel属性值是close）或者点击主页可见的部分，都会关闭这个面板。

### 32.2.1 配置面板组件

面板组件定义的属性及设置选项见表32-4。

表32-4 面板组件支持的属性与配置选项

data属性	选 项	描 述
data-animate	animate	指定面板打开或关闭时是否使用动画，默认值为true

(续)

data属性	选 项	描 述
data-dismissable	dismissable	指定点击触发面板显示的页是否可以关闭面板, 默认值为true
data-display	display	指定面板与触发面板的页之间的关系, 该选项的可取值为reveal、push和overlay, 详见表32-5
data-position	position	指定面板的显示位置, 可取值为left、right, 默认值为left
data-position-fixed	positionFixed	当用户向下滚动页面时, 面板是否保持可见, 默认值为false
data-swipe-close	swipeClose	指定面板可否使用轻扫手势关闭, 默认值为true

### 1. 面板的位置与显示

data-display和data-position属性决定面板显示位置(窗口左边还是右边), 以及它与触发显示页之间的关系。表32-5列出了data-display属性所支持的三个可取值。

表32-5 data-display属性支持的值

值	描 述
reveal	默认值, 把页推走
push	页的尺寸会变小, 与面板共享空间
overlay	面板显示在页上方(盖住页)

在代码清单32-11中, 可以看到两个属性的显示效果, 包括位置选项, 以及面板与页之间的显示层次关系。

#### 代码清单32-11 设置面板位置

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <script>
    $(document).bind("pageinit", function () {
      $("#pageContent button").tap(function (e) {
        $("#" + this.id + "Panel").panel({
          display: $("input[type=radio]:checked").attr("id")
        }).panel("open");
      });
    });
  </script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
```

```

<div id="pageContent" data-role="content">
  <div class="ui-grid-a">
    <div class="ui-block-a"><button id="left">Left</button></div>
    <div class="ui-block-b"><button id="right">Right</button></div>
  </div>

  <div data-role="fieldcontain">
    <fieldset data-role="controlgroup" data-type="horizontal">
      <input type="radio" name="display" id="reveal" checked="checked"/>
      <label for="reveal">Reveal</label>
      <input type="radio" name="display" id="push"/>
      <label for="push">Push</label>
      <input type="radio" name="display" id="overlay"/>
      <label for="overlay">Overlay</label>
    </fieldset>
  </div>
</div>

<div id="leftPanel" data-role="panel" data-theme="a" data-position="left">
  <div data-role="panel-content">
    <h3>Left Panel</h3>
    <p>This is the the left panel</p>
    <button data-rel="close" data-inline="true">Close</button>
  </div>
</div>

<div id="rightPanel" data-role="panel" data-theme="a" data-position="right">
  <div data-role="panel-content">
    <h3>Right Panel</h3>
    <p>This is the the right panel</p>
    <button data-rel="close" data-inline="true">Close</button>
  </div>
</div>
</div>
</body>
</html>

```

我在页中定义了两个按钮，用来把面板显示在左边或右边（此处用到了open方法，稍后会提到）。还定义了一组单选钮（详见第30章），用它来选择显示模式。图32-11展示了部分配置的显示效果。

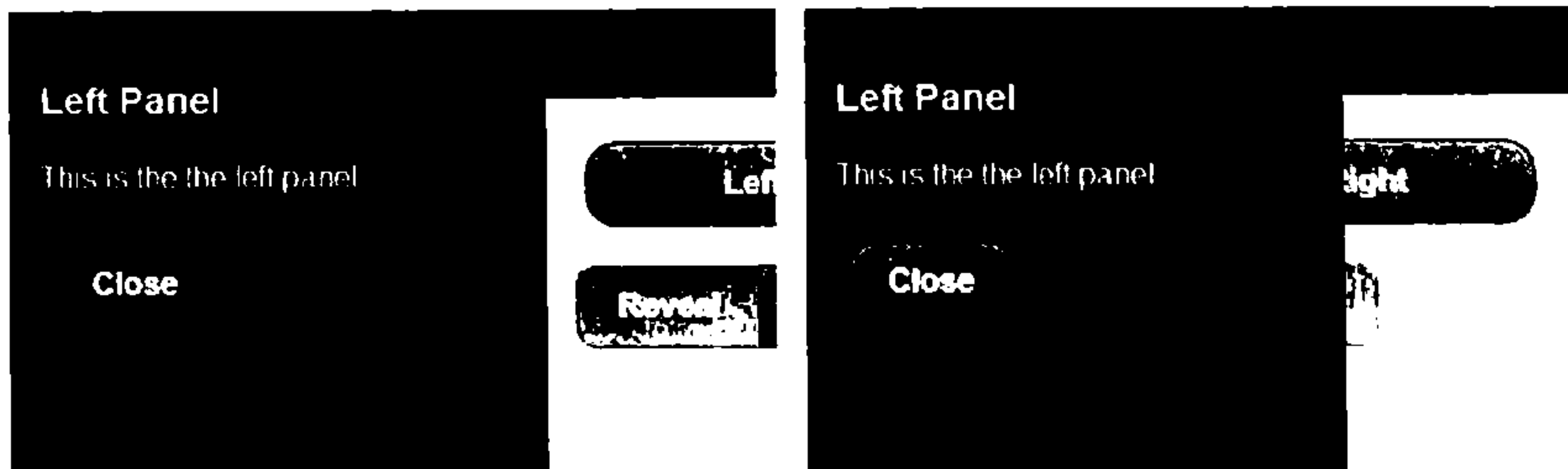


图32-11 改变面板与页之间关系的选项和显示位置选项

面板组件尚不能支持把同一份内容动态显示在左边或者右边：在准备内容的时候，只能要么根据配置显示在左边，要么显示在右边。就因为这个原因，我才在示例中准备了两个面板。

## 2. 使层消失

`data-swipe-close`和`data-dismissable`属性允许我们控制关闭面板的方式，要么使用轻扫手势，要么点击触发面板页。把这两个属性设置成`false`而生成的面板，只能通过与面板内容交互（或者使用脚本调用`open`和`toggle`方法，本章后面会讲到）来关闭。

如果你打算使用这些属性，就要确保在你的Web应用中保持一致，小心别让用户失控。如果好几个面板关闭方式各不相同，会让用户抓狂的。在代码清单32-12中，我创建了一个面板，它会显示一小段时间然后自己消失。我并不是建议你在真实项目里这么干，只是为了演示组件的功能。用户无法通过点击触发面板的页，或者轻扫手势来关闭这个面板，好在面板上的关闭按钮仍然有效，能够用来关闭面板。

代码清单32-12 创建无法点击页面或轻扫关闭的面板

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script>
    $(document).bind("pageinit", function () {
      $("a").tap(function (e) {
        var timeRemaining = 15;
        var intervalId = setInterval(function () {
          $("#remaining").text(timeRemaining--);
          if (timeRemaining == 0) {
            $("#panel").panel("close");
            clearInterval(intervalId);
          }
        }, 1000);
        $("#panel").panel("open");
      });
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>

    <div data-role="content" class="buttonContainer">
      <a data-role="button" data-inline="true">Open Panel</a>
    </div>
    <div id="panel" data-role="panel" data-theme="a"
      data-dismissable="false" data-swipe-close="false">
      <div data-role="panel-content">
```

```

    <h3>Simple Panel</h3>
    <p>This panel will close in
      <span id="remaining">15</span> seconds.</p>
    <button data-rel="close" data-inline="true">Close</button>
  </div>
</div>
</div>
</body>
</html>

```

使用open和close方法来控制面板的显示——在下一节中我会介绍这两个方法，并使用JavaScript的setInterval函数来管理倒计时，让面板在打开15秒之后自动关闭。用户可以点击面板中的关闭按钮提前关闭面板，如果你打算阻止用户关闭模板，就要确保面板不包含关闭按钮元素。如图32-12所示，面板组件可自动反映出底层元素的变化。

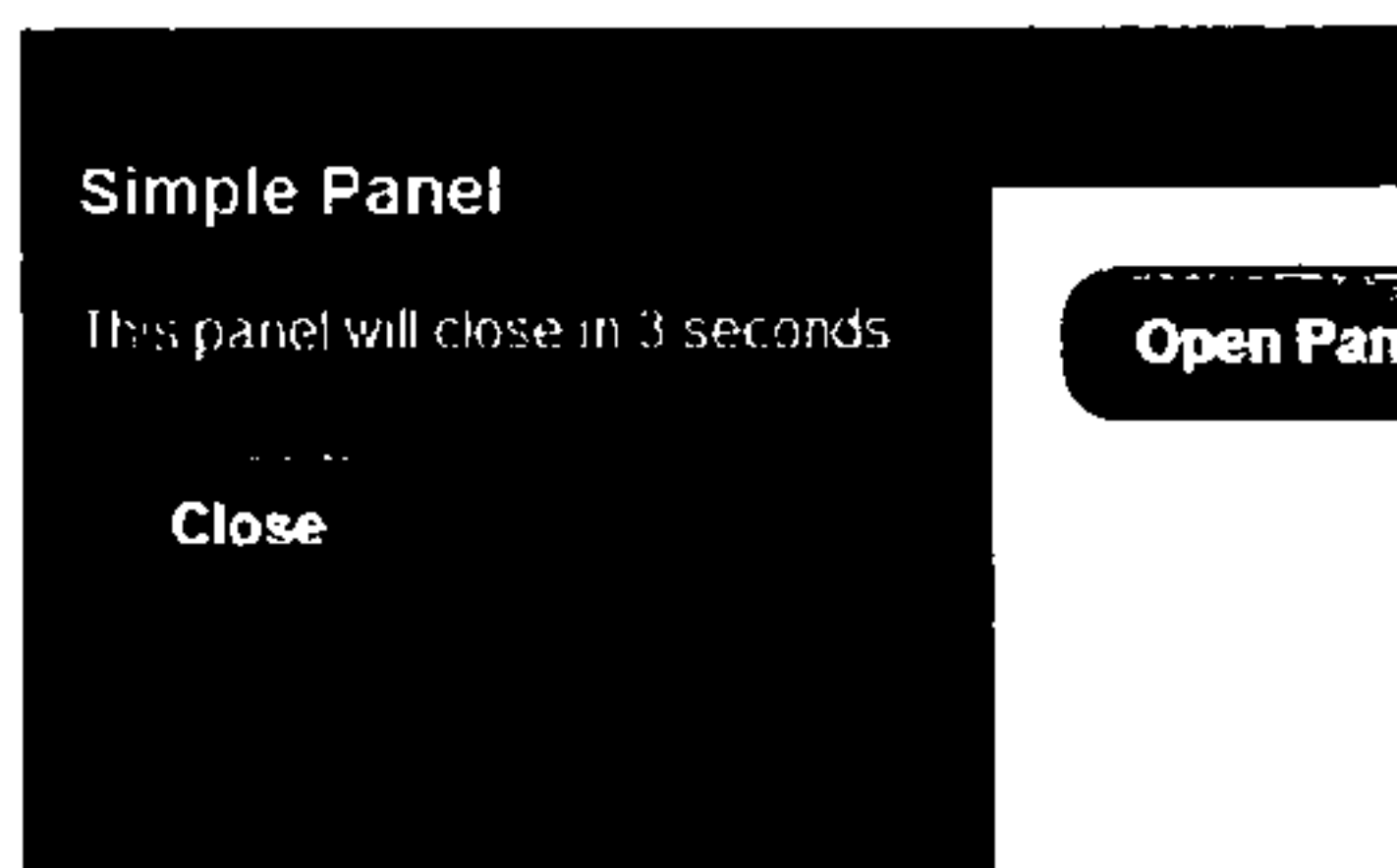


图32-12 一个自行关闭的面板

### 32.2.2 面板组件方法

面板组件定义的方法见表32-6，其中的open和close方法在前面的例子中已经用过。

表32-6 面板组件方法

方 法	描 述
panel("open")	显示面板
panel("close")	关闭面板
panel("toggle")	切换面板的显示状态：隐藏则显示，显示则隐藏

### 32.2.3 面板组件事件

面板组件定义的事件见表32-7。在我自己的项目中，我从未用过这些事件。我更喜欢处理触发面板显示或者隐藏的元素事件。

表32-7 面板组件事件

事 件	描 述
create	在组件创建完成时触发



(续)

事 件	描 述
beforeopen	在面板显示出来之前触发
beforeclose	在面板隐藏之前触发
open	在面板显示出来之后触发
close	在面板隐藏之后触发

## 32.3 小结

本章讲解了jQuery Mobile的列表组件。这个组件适用于制作移动Web项目的简单导航。我演示了列表组件提供的多种形式及显示风格，以及管理单个列表项内容的配置项与常见格式的代码约定。

在本部分前面几个章节中，我介绍了jQuery Mobile。在本章，我将利用jQuery Mobile构建一个更完整的例子。从本质上说，jQuery Mobile比jQuery UI要简单得多，因此它可用的选项要少得多。jQuery Mobile的发展也受限于移动设备开发中所面临的一些问题。

## 33.1 从基础开始

在上一章，我展示了一个分栏列表的例子。在本章，我们将利用那个例子构建更多的功能。代码清单33-1展示的是最初的页面。

代码清单33-1 最初的页面

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    .lcontainer {float: left; text-align: center; padding-top: 10px}
    .productData {float: right; padding: 10px; width: 60%}
    .cWrapper {text-align: center; margin: 20px}
  </style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>

    <div id="container" style="padding: 20px">
      <ul data-role="listview" data-inset=true>
        <li>
          <a href="#basket" class="buy" id="rose">Roses</a>
          <a href="#roses">Roses</a>
        </li>
        <li>
          <a href="#basket" class="buy" id="orchid">Orchids</a>
```

```

        <a href="#orchids">Orchids</a>
    </li>
    <li>
        <a href="#basket" class="buy" id="aster">Asters</a>
        <a href="#asters">Asters</a>
    </li>
</ul>
</div>
</div>

<div id="basket" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Jacqui's Shop</h1>
    </div>
    <div class="cWrapper">
        Basket will go here
    </div>
    <div class="cWrapper">
        <a href="#" data-rel="back" data-role="button" data-inline=true
            data-direction="reverse">Back</a>
    </div>
</div>

<div id="roses" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Roses</h1>
    </div>
    <div>
        <div class="lcontainer">
            
            <div><a href="#" data-rel="back" data-role="button"
                data-inline=true data-direction="reverse">Back</a>
            </div>
        </div>
        <div class="productData">
            A rose is a woody perennial within the family Rosaceae.
            They form a group of erect shrubs, and climbing or trailing plants.
            <div><b>Price: $4.99</b></div>
        </div>
    </div>
</div>

<div id="orchids" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Orchids</h1>
    </div>
    <div>
        <div class="lcontainer">
            
            <div><a href="#" data-rel="back" data-role="button"
                data-inline=true data-direction="reverse">Back</a>
            </div>
        </div>
        <div class="productData">

```

```

        The orchid family is a diverse and widespread family in the order
        Asparagales. It is one of the largest families of flowering plants.
        <div><b>Price: $10.99</b></div>
    </div>
</div>
<div id="asters" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Asters</h1>
    </div>
    <div>
        <div class="lcontainer">
            
            <div><a href="#" data-rel="back" data-role="button"
                data-inline=true data-direction="reverse">Back</a>
            </div>
        </div>
        <div class="productData">
            The name Aster comes from the Ancient Greek word meaning "star",
            referring to the shape of the flower head.
            <div><b>Price: $2.99</b></div>
        </div>
    </div>
</div>
</body>
</html>

```

## 33.2 用程序添加花卉产品

我要做的第一件事是把描述每种花卉的静态页替换为动态生成内容的页面。这个变动让我们拥有一个更精简的页面，并且无需重复那些HTML元素，就能很容易地添加更多的花卉。我会使用第12章讲过的数据模板插件生成页面。这个插件需要jQuery核心库，因此非常适合jQuery Mobile应用程序。我生成了一个包含所需花卉信息的文件data.json。代码清单33-2展示了该文件的内容。

代码清单33-2 data.json文件的内容

```

[
  {
    "name": "aster",
    "label": "Asters",
    "price": "$2.99",
    "text": "The name Aster comes from the Ancient Greek word meaning star..."
  },
  {
    "name": "carnation",
    "label": "Carnations",
    "price": "$1.99",
    "text": "Carnations require well-drained, neutral to slightly alkaline soil..."
  },
  {
    "name": "daffodil",
    "label": "Daffodils",
    "price": "$1.99",
    "text": "Daffodil is a common English name, sometimes used for all varieties..."
  },
  {
    "name": "rose",
    "label": "Roses",
    "price": "$4.99",

```

```

    "text": "A rose is a woody perennial within the family Rosaceae. They form a..."
  }, {
    "name": "orchid",
    "label": "Orchids",
    "price": "$10.99",
    "text": "The orchid family is a diverse and widespread family in the order..."
  }
]
```

这些数据描述了五种花卉。每一种都定义了产品名称、显示给用户看的标题、单价和描述文本。

**注意** 我并没有在列表中显示描述文本，不过它确实包含在data.json文件（本书源代码的一部分，可以从[apress.com](http://apress.com)免费获得）中。

现在我们已经有了数据，接下来是把它整合到页面中。示例页本来是一个纯静态页，代码清单33-3展示了如何利用数据模板编程生成（包含同样内容的）页面。

### 代码清单33-3 动态添加页面内容

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <style type="text/css">
    .lcontainer {float: left; text-align: center; padding-top: 10px}
    .productData {float: right; padding: 10px; width: 60%}
    .cWrapper {text-align: center}
  </style>
  <script id="flowerTpl" type="text/x-handlebars-template">
    {{#products}}
    <div id="{{name}}" data-role="page" data-theme="b">
      <div data-role="header">
        <h1>{{label}}</h1>
      </div>
      <div>
        <div class="lcontainer">
          
          <div><a href="#" data-rel="back" data-role="button"
            data-inline=true data-direction="reverse">Back</a>
          </div>
        </div>
        <div class="productData">
          {{text}}
          <div><b>Price: {{price}}</b></div>
        </div>
      </div>
    </div>
    {{/products}}
  </script>
```

```

<script id="liTpl" type="text/x-handlebars-template">
  {{#products}}
    <li>
      <a href="#basket" class="buy" id="A1">{{label}}</a>
      <a href="#{{name}}">{{label}}</a>
    </li>
  {{/products}}
</script>

<script type="text/javascript">
  var initComplete = false;
  $(document).bind("pageinit", function () {
    if (!initComplete) {
      $.getJSON("data.json", function (data) {
        $("#flowerTpl").template({ products: data })
          .filter("").appendTo("body");
        $("ul").append($("#liTpl").template({ products: data })
          .filter("")).listview("refresh")
      });
      initComplete = true;
    }
  })
</script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div id="container" style="padding: 20px">
      listview" data-inset=true</ul>
    </div>
  </div>

  <div id="basket" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div class="cWrapper">
      Basket will go here
    </div>
    <div class="cWrapper">
      <a href="#" data-rel="back" data-role="button" data-inline=true
        data-direction="reverse">Back</a>
    </div>
  </div>
</body>
</html>

```

我删除了每种花的页面，并使用数据模板插件根据data生成所需要的内容，而data是我使用getJSON（参阅第14章）方法得到的。这个改变的关键见以下代码。

```

...
<script type="text/javascript">
  var initComplete = false;
  $(document).bind("pageinit", function () {
    if (!initComplete) {
      $.getJSON("data.json", function (data) {
        $("#flowerTpl").template({ products: data })
          .filter(" * ").appendTo("body");
        $("ul").append($("#liTpl").template({ products: data })
          .filter("*")).listview("refresh")
      });
      initComplete = true;
    }
  })
</script>
...

```

得到数据之后,我先使用模板把数据渲染为HTML元素(生成内容),再把渲染结果插入到页面的body元素之中。我使用另一个模板生成花卉列表的每个条目。最后我调用了listview组件的refresh方法,告诉jQuery Mobile列表的内容已发生变化(需要刷新)。代码如下:

```

...
$("ul").append($("#liTpl").template({ products: data })
  .filter("*")).listview("refresh");
...

```

这个数据模板相当简单,使用的是我在第12章讲过的标准技术。你可以通过图33-1看到以上脚本的效果——列表项都是动态生成的,而且链接的页面也都是通过编程添加到页面之中的。



图33-1 编程生成列表页和页面

## 重用页面

我喜欢数据模板方案,因为它展示了jQuery如何支撑广泛的功能,允许我们把类似模板这样的功能和类似jQuery Mobile这样的工具包接口结合起来使用。

也就是说,我们可以采纳更优雅的方案处理每种花的页面。不同于为每一种花生成一套需要展示的元素,我们可以只生成一套元素,并在用户选择一种花的时候,通过修改这套元素展示选中花卉。代码清单33-4演示了为达成这一目标而发生的代码变化。

## 代码清单33-4 利用一个页面展示多种商品

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <script id="liTpl" type="text/x-handlebars-template">
    {{#products}}
    <li>
      <a href="#basket" class="buy" id="{{name}}">{{label}}</a>
      <a class="productLink" data-flower="{{name}}" href="#">{{label}}</a>
    </li>
    {{/products}}
  </script>
  <script type="text/javascript">
    var initComplete = false;

    $(document).bind("pageinit", function () {
      if (!initComplete) {

        $.getJSON("data.json", function (data) {

          $("ul").append($("#liTpl").template({ products: data })
            .filter("*")).listview("refresh");

          $("a.productLink").bind("tap", function () {
            var targetFlower = $(this).attr("data-flower");
            for (var i = 0; i < data.length; i++) {
              if (data[i].name == targetFlower) {
                var page = $("#productPage");
                page.find("#header").text(data[i].label);
                page.find("#image").attr("src", data[i].name + ".png");
                page.find("#description").text(data[i].text);
                page.find("#price").text(data[i].price);

                $.mobile.changePage("#productPage");
                break;
              }
            }
          });
        });
        initComplete = true;
      }
    });
  </script>
  <script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
  <style type="text/css">
    .lcontainer {float: left; text-align: center; padding-top: 10px}
    .productData {float: right; padding: 10px; width: 60%}
  </style>

```



```

        .cWrapper {text-align: center}
    </style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div id="container" style="padding: 20px">
            <ul data-role="listview" data-inset=true>

                </ul>
        </div>
    </div>

    <div id="productPage" data-role="page" data-theme="b">
        <div data-role="header">
            <h1 id="header"></h1>
        </div>
        <div>
            <div class="lcontainer">
                <img id="image" src="">
                <div><a href="#" data-rel="back" data-role="button"
                    data-inline=true data-direction="reverse">Back</a>
                </div>
            </div>
            <div class="productData">
                <span id="description"></span>
                <div><b>Price: <span id="price"></span></b></div>
            </div>
        </div>
    </div>

    <div id="basket" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div class="cWrapper">
            Basket will go here
        </div>
        <div class="cWrapper">
            <a href="#" data-rel="back" data-role="button" data-inline=true
                data-direction="reverse">Back</a>
        </div>
    </div>
</body>
</html>

```

**提示** 这是一种特别适用于jQuery Mobile的方案，因为许多页面实际上包含在单一的HTML页面之中。由于移动设备内在的局限性，作为一个基本原则，我们希望HTML页面越简单越好。

我从页面中删除了一个数据模板，并添加了一个用于每种花卉的新页（它的id为productPage）。我修改了用来生成列表项的模板，href属性不再指向目标页，而是添加了我自己的data属性，以便我知道它是哪一种花，应该链接到哪里去。当我们拿到JSON数据时，修订之后的脚本会从列表元素（我刚刚使用模板创建的）中选中所有的产品链接并为它们绑定tap事件。点击列表项时，我找出对应的数据对象并使用它的属性修改productPage页，设置要呈现给用户的文本和图片属性，代码如下所示。

```
...
<script type="text/javascript">

    var initComplete = false;

    $(document).bind("pageinit", function () {
        if (!initComplete) {
            $.getJSON("data.json", function (data) {

                $("ul").append($("#liTpl").template({ products: data })
                    .filter("*")).listview("refresh");

                $("a.productLink").bind("tap", function () {
                    var targetFlower = $(this).attr("data-flower");
                    for (var i = 0; i < data.length; i++) {
                        if (data[i].name == targetFlower) {
                            var page = $("#productPage");
                            page.find("#header").text(data[i].label);
                            page.find("#image").attr("src", data[i].name + ".png");
                            page.find("#description").text(data[i].text);
                            page.find("#price").text(data[i].price);

                            $.mobile.changePage("#productPage");
                            break;
                        }
                    }
                });
            });
            initComplete = true;
        }
    });
</script>
...
```

页面配置完成后，我使用changePage方法来触发导航。在这个例子中页面并没有发生变化，然而对移动浏览器来说，它需要管理的元素更少。同时，这还是一个维护jQuery Mobile页面结构的不错的例子。

### 33.3 生成购物车

我在这个盒子中使用了分栏列表，列表项的左栏指向购物车。在本节中，为了就地实现一个简单的购物车，我将定义一些页面元素，并添加一些JavaScript代码。代码清单33-5展示了这些变化。

代码清单33-5 实现购物车

```
<!DOCTYPE html>
```



```

        }
    }
    calculateTotals();
    $.mobile.changePage("#basket")
  });
});
});

function calculateTotals() {
  var total = 0;
  $("#basketTable tbody").children().each(function (index, elem) {
    var count = Number($(elem).find("#count").text())
    var price = Number($(elem).attr("data-price").slice(1))
    var subtotal = count * price;
    $(elem).find("#subtotal").text("$" + subtotal.toFixed(2));
    total += subtotal;
  });
  $("#total").text("$" + total.toFixed(2))
}
</script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
<style type="text/css">
  .lcontainer {float: left; text-align: center; padding-top: 10px}
  .productData {float: right; padding: 10px; width: 60%}
  .cWrapper {text-align: center}
  table {display: inline-block; margin: auto; margin-top: 20px; text-align: left;
    border-collapse: collapse}
  td {min-width: 100px}
  th, td {text-align: right}
  th:nth-child(1), td:nth-child(1) {text-align: left}
</style>
</head>
<body>
  <div id="page1" data-role="page" data-theme="b">
    <div data-role="header">
      <h1>Jacqui's Shop</h1>
    </div>
    <div id="container" style="padding: 20px">
      <ul data-role="listview" data-inset=true></ul>
    </div>
  </div>

  <div id="productPage" data-role="page" data-theme="b">
    <div data-role="header">
      <h1 id="header"></h1>
    </div>
    <div>
      <div class="lcontainer">
        <img id="image" src="">
        <div>
          <a href="#" data-rel="back" data-role="button"
            data-inline=true data-direction="reverse">Back</a>
        </div>
      </div>
    </div>
  </div>

```

```

        <div class="productData">
            <span id="description"></span>
            <div><b>Price: <span id="price"></span></b></div>
        </div>
    </div>
</div>

<div id="basket" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Jacqui's Shop</h1>
    </div>
    <div class="cWrapper">
        <table id="basketTable" border=0>
            <thead>
                <tr>
                    <th>Flower</th>
                    <th>Quantity</th>
                    <th>Subtotal</th>
                </tr>
            </thead>
            <tbody></tbody>
            <tfoot>
                <tr>
                    <th colspan=2>Total:</th>
                    <td id="total"></td>
                </tr>
            </tfoot>
        </table>
    </div>
    <div class="cWrapper">
        <a href="#" data-rel="back" data-role="button" data-inline=true
            data-direction="reverse">Back</a>
        <button data-inline="true">Checkout</button>
    </div>
</div>
</body>
</html>

```

我在购物车页添加了一个表格，用户选中的每个产品在这个表格中各占一行，用来显示产品名称、订购数量及花费小计。在表格的末尾有一行显示总额。我为左边的按钮绑定了tap事件处理函数，这样当用户轻敲按钮时，要么在表格中新增一行，要么就为表格中的现有行增加数量（和小计）。新增的行使用另一个数据模板生成，至于其他的部分，则通过读取页面中的元素内容来处理。

我决定利用DOM维护购物车的所有状态。我需要创建一个结构化的JavaScript对象来存放订单，由这个对象驱动生成购物车表格的内容。既然本书的主题是jQuery，我更喜欢把这些事情交给页面来处理。最终我们得到一个非常简单的购物车，如图33-2所示。

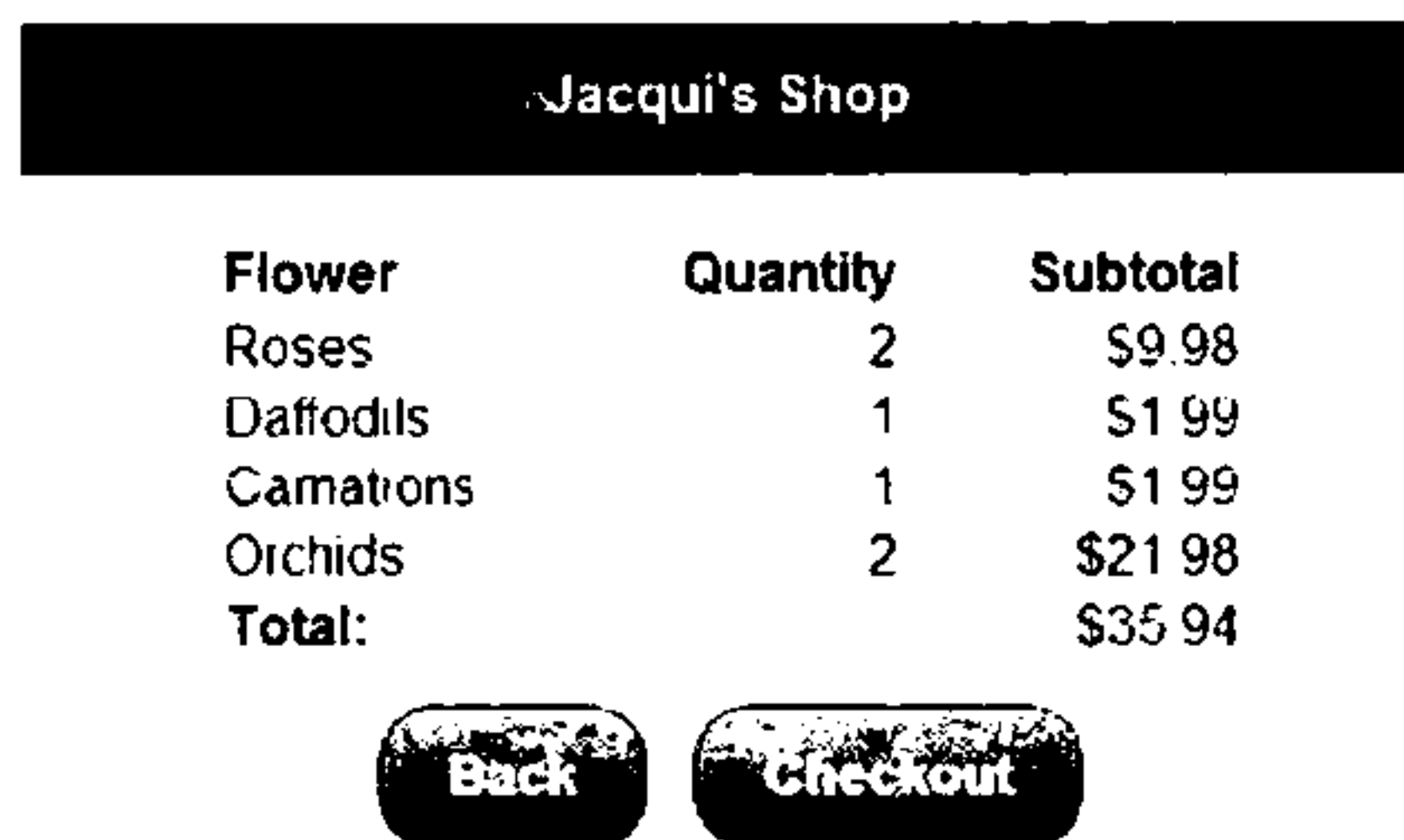


图33-2 购物车页

### 33.3.1 增加修改数量功能

购物车现在已经能用了，不过如果有人需要两支玫瑰，他只能先点一下Rose，点一下Back，再点一次Rose。这种做法实在可笑，为了便于用户修改商品的数量，我在表格中添加了一些input元素。代码变更的部分见代码清单33-6。

#### 代码清单33-6 为购物车表格添加滑动数量修改器

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <script id="liTpl" type="text/x-handlebars-template">
    {{#products}}
    <li>
      <a href="#basket" class="buy" id="{{name}}">{{label}}</a>
      <a class="productLink" data-flower="{{name}}" href="#">{{label}}</a>
    </li>
    {{/products}}
  </script>
  <script id="trTpl" type="text/x-handlebars-template">
    <tr data-theme="b" data-price="{{price}}" id="{{name}}">
      <td>{{label}}</td>
      <td id="count"><input type="number" value=1 min=0 max=10></td>
      <td id="subtotal">0</td>
    </tr>
  </script>
  <script type="text/javascript">

    var initComplete = false;

    $(document).bind("pageinit", function () {
```

```

if (!initComplete) {
    $.getJSON("data.json", function (data) {

        $("ul").append($("#liTpl")
            .template({ products: data })).listview("refresh");

        $("a.productLink").bind("tap", function () {
            var targetFlower = $(this).attr("data-flower");
            for (var i = 0; i < data.length; i++) {
                if (data[i].name == targetFlower) {
                    var page = $("#productPage");
                    page.find("#header").text(data[i].label);
                    page.find("#image").attr("src", data[i].name + ".png");
                    page.find("#description").text(data[i].text);
                    page.find("#price").text(data[i].price);

                    $.mobile.changePage("#productPage");
                    break;
                }
            }
        })

        $("a.buy").bind("tap", function () {
            var targetFlower = this.id;
            var row = $("#basketTable tbody #" + targetFlower);
            if (row.length > 0) {
                var countCell = row.find("#count input");
                countCell.val(Number(countCell.val()) + 1);
            } else {
                for (var i = 0; i < data.length; i++) {
                    if (data[i].name == targetFlower) {
                        $("#trTpl").template(data[i])
                            .appendTo("#basketTable tbody")
                            .find("input").textinput()

                        break;
                    }
                }
            }
            calculateTotals();
            $.mobile.changePage("#basket")
        })

        $(document).on("change click", "input", function (event) {
            calculateTotals();
        })
    });
    initComplete = true;
}

function calculateTotals() {
    var total = 0;
    $("#basketTable tbody").children().each(function (index, elem) {

```

```

        var count = Number($(elem).find("#count input").val())
        var price = Number($(elem).attr("data-price").slice(1))
        var subtotal = count * price;
        $(elem).find("#subtotal").text("$" + subtotal.toFixed(2));
        total += subtotal;
    })
    $("#total").text("$" + total.toFixed(2))
}
</script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
<style type="text/css">
    .lcontainer {float: left; text-align: center; padding-top: 10px}
    .productData {float: right; padding: 10px; width: 60%}
    .cWrapper {text-align: center}
    table {display: inline-block; margin: auto; margin-top: 20px; text-align: left;
        border-collapse: collapse}
    td {min-width: 100px; padding-bottom: 10px}
    td:nth-child(2) {min-width: 75px; width: 75px}
    th, td {text-align: right}
    th:nth-child(1), td:nth-child(1) {text-align: left}
    input[type=number] {background-color: white}
    tfoot tr {border-top: medium solid black}
    tfoot tr td {padding-top: 10px}
</style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div id="container" style="padding: 20px">
            <ul data-role="listview" data-inset=true></ul>
        </div>
    </div>

    <div id="productPage" data-role="page" data-theme="b">
        <div data-role="header">
            <h1 id="header"></h1>
        </div>
        <div>
            <div class="lcontainer">
                <img id="image" src="">
                <div><a href="#" data-rel="back" data-role="button"
                    data-inline=true data-direction="reverse">Back</a>
                </div>
            </div>
            <div class="productData">
                <span id="description"></span>
                <div><b>Price: <span id="price"></span></b></div>
            </div>
        </div>
    </div>

    <div id="basket" data-role="page" data-theme="b">

```



```

<div data-role="header">
  <h1>Jacqui's Shop</h1>
</div>
<div class="cWrapper">
  <table id="basketTable" border=0>
    <thead>
      <tr><th>Flower</th><th>Quantity</th><th>Subtotal</th></tr>
    </thead>
    <tbody></tbody>
    <tfoot>
      <tr><th colspan=2>Total:</th><td id="total"></td></tr>
    </tfoot>
  </table>
</div>
<div class="cWrapper">
  <a href="#" data-rel="back" data-role="button" data-inline=true
    data-direction="reverse">Back</a>
  <button data-inline="true">Checkout</button>
</div>
</div>
</body>
</html>

```

我在用于生成表格行的模板（`script#trTpl`）的数量一栏添加了一个元素，这个元素的type属性是number，一些支持number属性的浏览器会在文本输入框区域添加向上和向下的小按钮。这些按钮实在太小，以致很难点中，不过这些浏览器还是会过滤掉那些不是合法数字的字符。尽管对本章来说，这种效果可以接受，由于这种输入框也支持带小数的数字（而商品数量是不会出现小数的），因此对真实项目来说，这仍不是一个完美方案。

当我把元素添加到页面（已经被jQuery Mobile添加了各种效果）之后，调用了textinput方法：

```

...
$("#trTpl").template(data[i]).appendTo("#basketTable tbody").find("input").textinput()
...

```

如果我不执行这次调用，浏览器只会显示一个原生的文本框，主动调用textinput方法会通知jQuery Mobile增强这个元素的显示效果，尽管它并不是非常切合当前主题。因此我为元素设置了背景色样式，这样它就和当前主题融为一体了：

```

...
input[type=number] {background-color: white}
...

```

由于用户现在可以随时改变商品的数量，我需要频繁计算小计和总计。由于这些元素是在页面加载完成之后动态添加上去的，所以我使用on方法绑定事件处理函数。如果希望详细了解on方法，请阅读第9章。下面是事件处理代码。

```

...
$(document).on("change click", "input", function (event) {
  calculateTotals();
})
...

```

我使用on方法同时为change和click这两个事件绑定了同一个处理函数。由于浏览器为数字输入框添加的向上和向下按钮在被按下时会触发click事件，所以我在change事件的基础上，又额外处理了click事件。事件处理函数只是在事件发生时调用了calculateTotals函数。购物车现在的样子见图33-3。

Flower	Quantity	Subtotal
Carnations	1	\$1.99
Roses	2	\$9.98
<b>Total:</b>		<b>\$11.97</b>

Back
Checkout

图33-3 为购物车页添加input元素

### 33.3.2 在商品详情页添加购买按钮

商品详情页描述了用户选择的商品信息，却没有提供把商品添加到购物车的功能。为了让购物车的功能更加完善，我在商品页添加了一个按钮，方便用户把商品添加到购物车。商品页的代码变更见代码清单33-7。

代码清单33-7 在商品详情页添加购买按钮

```
...
<div id="productPage" data-role="page" data-theme="b">
  <div data-role="header">
    <h1 id="header"></h1>
  </div>
  <div>
    <div class="lcontainer">
      <img id="image" src="">
      <div>
        <a href="#" data-rel="back" data-role="button"
          data-inline=true data-direction="reverse">Back</a>
      </div>
    </div>
    <div class="productData">
      <span id="description"></span>
      <div>
        <b>Price: <span id="price"></span></b>
        <a href="#" id="buybutton" data-flower="" data-role="button"
          data-inline=true>Buy</a>
      </div>
    </div>
  </div>
</div>
...
```

我定义了一个a元素，jQuery Mobile会把它转换成一个按钮组件。我为它添加了一个data属性（data-flower），这样当用户点击按钮时，我就能知道当前显示的是哪种花。为了支持这个购买按钮，我对脚本做了一些修改，这些改动见代码清单33-8。

代码清单33-8 让脚本支持购买按钮

```
...
<script type="text/javascript">

    var initComplete = false;

    $(document).bind("pageinit", function () {
        if (!initComplete) {
            $.getJSON("data.json", function (data) {

                $("ul").append($("#liTpl")
                    .template({ products: data })).listview("refresh");

                $("a.productLink").bind("tap", function () {
                    var targetFlower = $(this).attr("data-flower");
                    for (var i = 0; i < data.length; i++) {
                        if (data[i].name == targetFlower) {
                            var page = $("#productPage");
                            page.find("#header").text(data[i].label);
                            page.find("#image").attr("src", data[i].name + ".png");
                            page.find("#description").text(data[i].text);
                            page.find("#price").text(data[i].price);
                            page.find("#buybutton").attr("data-flower", data[i].name);
                            $.mobile.changePage("#productPage");
                            break;
                        }
                    }
                })

                $("#buybutton").bind("tap", function () {
                    addProduct($(this).attr("data-flower"));
                });

                $("a.buy").bind("tap", function () {
                    addProduct(this.id);
                });

                function addProduct(targetFlower) {
                    var row = $("#basketTable tbody #" + targetFlower);
                    if (row.length > 0) {
                        var countCell = row.find("#count input");
                        countCell.val(Number(countCell.val()) + 1);
                    } else {
                        for (var i = 0; i < data.length; i++) {
                            if (data[i].name == targetFlower) {
                                $("#trTpl").template(data[i])
                                    .appendTo("#basketTable tbody")
                                    .find("input").textinput()

```

```

        break;
    }
}
}
calculateTotals();
$.mobile.changePage("#basket")
}

$(document).on("change click", "input", function (event) {
    calculateTotals();
});
initComplete = true;
})
}

function calculateTotals() {
    var total = 0;
    $("#basketTable tbody").children().each(function(index, elem) {
        var count = Number($(elem).find("#count input").val())
        var price = Number($(elem).attr("data-price").slice(1))
        var subtotal = count * price;
        $(elem).find("#subtotal").text("$" + subtotal.toFixed(2));
        total += subtotal;
    })
    $("#total").text("$" + total.toFixed(2))
}
</script>
...

```

这些代码变更相当直白，无需解释。当用户在商品列表中选中一件商品时，我会设置a元素的data-flower属性。我注册了一个回调函数处理按钮的tap事件，在这个函数中，我以data-flower属性的值作为参数调用addProduct函数（它的代码是我从另一个事件处理函数中抽出来的）。做完这些修改之后，用户现在就能在主商品列表中通过点击列表项的左边，或者在商品详情页点击Buy按钮，把商品添加到购物车。详情页新加的Buy按钮见图33-4。

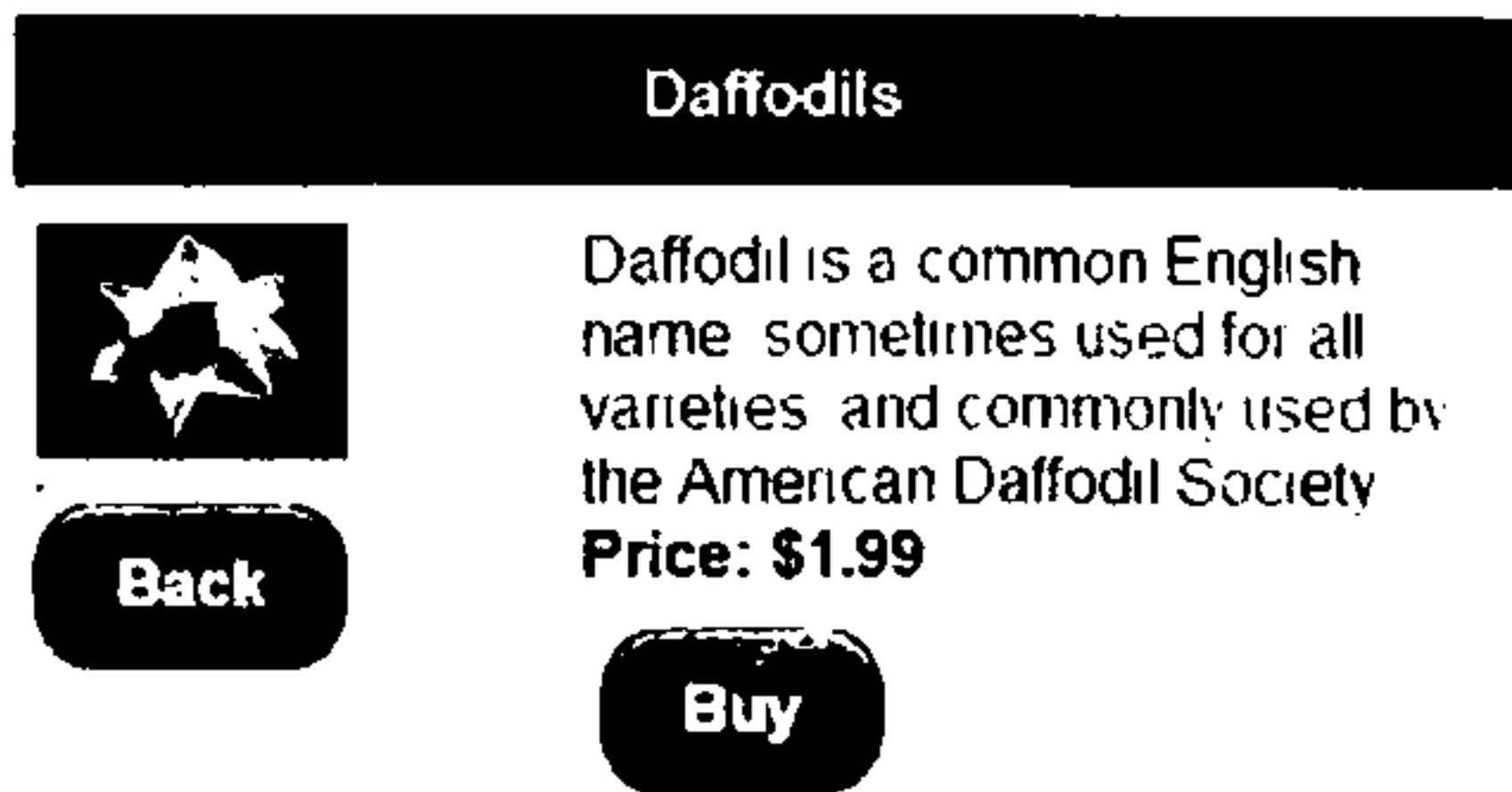


图33-4 在商品详情页添加购买按钮

## 33.4 实现支付

为了让这个例子完美收官，我要演示一下从一个表单的多个页面中收集数据并发起Ajax请求。我不会真的发起请求，也不会真的在服务器端实现支付。在jQuery Mobile中可以直接使用jQuery核心库提供的Ajax功能（详见第14、15章）。代码清单33-9是当用户按下Checkout按钮时所看到的页面代码，以及收集数据的代码。

代码清单33-9 实现支付

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.3.1.css" type="text/css" />
  <script type="text/javascript" src="jquery-1.10.1.js"></script>
  <script src="handlebars.js"></script>
  <script src="handlebars-jquery.js"></script>
  <script id="liTpl" type="text/x-handlebars-template">
    {{#products}}
    <li>
      <a href="#basket" class="buy" id="{{name}}">{{label}}</a>
      <a class="productLink" data-flower="{{name}}" href="#">{{label}}</a>
    </li>
    {{/products}}
  </script>
  <script id="trTpl" type="text/x-handlebars-template">
    <tr data-theme="b" data-price="{{price}}" id="{{name}}"><td>{{label}}</td>
      <td id="count"><input type="number" value=1 min=0 max=10></td>
      <td id="subtotal">0</td>
    </tr>
  </script>
  <script type="text/javascript">
    var initComplete = false;
    $(document).bind("pageinit", function () {
      if (!initComplete) {
        $.getJSON("data.json", function (data) {
          $("ul").append($("#liTpl")
            .template({ products: data })).listview("refresh");

          $("a.productLink").bind("tap", function () {
            var targetFlower = $(this).attr("data-flower");
            for (var i = 0; i < data.length; i++) {
              if (data[i].name == targetFlower) {
                var page = $("#productPage");
                page.find("#header").text(data[i].label);
                page.find("#image").attr("src", data[i].name + ".png");
                page.find("#description").text(data[i].text);
                page.find("#price").text(data[i].price);
                page.find("#buybutton")
                  .attr("data-flower", data[i].name);
                $.mobile.changePage("#productPage");
              }
            }
          });
        });
      }
    });
  </script>
</head>
<body>
  <div>
    <ul>
      {{#products}}
      <li>
        <a href="#basket" class="buy" id="{{name}}">{{label}}</a>
        <a class="productLink" data-flower="{{name}}" href="#">{{label}}</a>
      </li>
      {{/products}}
    </ul>
  </div>
  <div>
    <table>
      <tr>
        <td>{{label}}</td>
        <td><input type="number" value=1 min=0 max=10></td>
        <td>0</td>
      </tr>
    </table>
  </div>
</body>
</html>
```

```

        break;
    }
}
});

$("#buybutton").bind("tap", function () {
    addProduct($(this).attr("data-flower"));
});

$("#a.buy").bind("tap", function () {
    addProduct(this.id);
});

function addProduct(targetFlower) {
    var row = $("#basketTable tbody #" + targetFlower);
    if (row.length > 0) {
        var countCell = row.find("#count input");
        countCell.val(Number(countCell.val()) + 1);
    } else {
        for (var i = 0; i < data.length; i++) {
            if (data[i].name == targetFlower) {
                $("#trTpl").template(data[i])
                    .appendTo("#basketTable tbody")
                    .find("input").textinput();
                break;
            }
        }
    }
    calculateTotals();
    $.mobile.changePage("#basket")
}

$(document).on("change click", "input", function (event) {
    calculateTotals();
});

$("#submit").bind("tap", function () {
    var dataObject = new Object();
    $("#basketTable tbody").children().each(function (index, elem) {
        dataObject[elem.id] = $(elem).find("#count input").val();
    });
    dataObject["name"] = $("#name").val();
    dataObject["wrap"] = $("option:selected").val();
    dataObject["shipping"] = $("input:checked").attr("id")

    console.log("DATA: " + JSON.stringify(dataObject))
});
});
initComplete = true;
}
});

function calculateTotals() {
    var total = 0;

```

```

        $("#basketTable tbody").children().each(function(index, elem) {
            var count = Number($(elem).find("#count input").val())
            var price = Number($(elem).attr("data-price").slice(1))
            var subtotal = count * price;
            $(elem).find("#subtotal").text("$" + subtotal.toFixed(2));
            total += subtotal;
        });
        $("#total").text("$" + total.toFixed(2))
    }

</script>
<script type="text/javascript" src="jquery.mobile-1.3.1.js"></script>
<style type="text/css">
    .lcontainer {float: left; text-align: center; padding: 10px}
    .productData {float: right; padding: 10px; width: 60%}
    .cWrapper {text-align: center}
    table {display: inline-block; margin: auto; margin-top: 20px; text-align: left;
        border-collapse: collapse}
    td {min-width: 100px; padding-bottom: 10px}
    td:nth-child(2) {min-width: 75px; width: 75px}
    th, td {text-align: right}
    th:nth-child(1), td:nth-child(1) {text-align: left}
    input[type=number] {background-color: white}
    tfoot tr {border-top: medium solid black}
    tfoot tr td {padding-top: 10px}
</style>
</head>
<body>
    <div id="page1" data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Jacqui's Shop</h1>
        </div>
        <div id="container" style="padding: 20px">
            <ul data-role="listview" data-inset=true></ul>
        </div>
    </div>

    <div id="productPage" data-role="page" data-theme="b">
        <div data-role="header">
            <h1 id="header"></h1>
        </div>
        <div>
            <div class="lcontainer">
                <img id="image" src="">
            </div>
            <a href="#" data-rel="back" data-role="button"
                data-inline=true data-direction="reverse">Back</a>
        </div>
    </div>
    <div class="productData">
        <span id="description"></span>
    </div>
    <b>Price: <span id="price"></span></b>
    <a href="#" id="buybutton" data-flower="" data-role="button"

```

```

            data-inline=true>Buy</a>
        </div>
    </div>
</div>
</div>

<div id="basket" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Jacqui's Shop</h1>
    </div>
    <div class="cWrapper">
        <table id="basketTable" border=0>
            <thead>
                <tr><th>Flower</th><th>Quantity</th><th>Subtotal</th></tr>
            </thead>
            <tbody></tbody>
            <tfoot>
                <tr><th colspan=2>Total:</th><td id="total"></td></tr>
            </tfoot>
        </table>
    </div>
    <div class="cWrapper">
        <a href="#" data-rel="back" data-role="button" data-inline=true
            data-direction="reverse">Back</a>
        <a href="#checkout" data-role="button" data-inline="true">Checkout</a>
    </div>
</div>

<div id="checkout" data-role="page" data-theme="b">
    <div data-role="header">
        <h1>Jacqui's Shop</h1>
    </div>

    <div data-role="content">
        <label for="name">Name: </label>
        <input id="name" placeholder="Your Name">

        <label for="wrap"><span>Gift Wrap: </span></label>
        <select id="wrap" name="wrap" data-role="slider">
            <option value="yes" selected>Yes</option>
            <option value="no">No</option>
        </select>
        <fieldset data-role="controlgroup" data-type="horizontal">
            <legend>Shipping:</legend>
            <input type="radio" name="ship" id="overnight" checked />
            <label for="overnight">Overnight</label>
            <input type="radio" name="ship" id="23day"/>
            <label for="23day">2-3 days</label>
            <input type="radio" name="ship" id="710day"/>
            <label for="710day">7-10 days</label>
        </fieldset>

        <div class="cWrapper">
            <a href="#" data-rel="back" data-role="button" data-inline="true"

```



```

        data-direction="reverse">Back</a>
      <a href="#" id="submit" data-role="button"
        data-inline=true">Submit Order</a>
    </div>
  </div>
</div>
</body>
</html>

```

这个新页被称为checkout。我让这个表单尽量简单，提示用户输入姓名，并提供了把鲜花作为礼品包装的选项，还有商品的快递方式。具体页面见图33-5。我让这个页以竖屏方式显示，这样用户无需滚动屏幕就可看到所有内容。

图33-5 支付页

当用户按下Submit Order按钮，就会立即从当前HTML页面的各个页中收集数据，并把结果以JSON字符串的方式输出到控制台。例如：

```

{"carnation":"3","rose":"1","orchid":"1",
 "name":"Adam Freeman","wrap":"yes","shipping":"23day"}

```

## 33.5 小结

在本章中，我把jQuery Mobile的一些核心功能组装到一起，实现了一个简单的掌上花店。从本质上讲，jQuery Mobile比jQuery UI要简单得多。最大的挑战在于如何在方寸之间做好设计，以最适当的方式为用户呈现他所需要的数据。

jQuery内建了很多实用方法，这些方法能够对jQuery对象作进一步的处理或者为JavaScript语言补充一些编程语言通常具备的特征。它们为jQuery团队内部所使用，它们的公开意味着，当我们遇到了jQuery团队已经解决了的古怪问题时，使用这些方法能让你省时省力。

其中的一些方法可作用于jQuery对象，另一些则由jQuery主函数（我在第5章讲过主函数，即\$）调用。表34-1列出了本章内容概要。

表34-1 本章概要

问 题	解决方法	代码清单
如何处理希望稍后执行的一系列操作	使用常规队列	1、2
如何过滤数组元素	使用grep方法	3、4
如何判断一个数组是否包含某个对象/值	使用isArray方法	5
如何遍历处理数组的内容	使用map方法	6、7
如何连接两个数组	使用merge方法	8
如何删除jQuery对象中的重复元素，并按元素在页面中的出现顺序排序	使用unique方法	9
如何判断一个对象的类型	使用isXXX或type方法	10、11
如何为表单提交准备数据	使用serialize或serializeArray方法	12
如何把数据解析为更有用的形式	使用parseJSON或parseXML方法	13
如何删除一个字符串的前置空白和后缀空白	使用trim方法	14
如何判断一个元素是否包含另一个	使用contains方法	15

## 34.1 再访队列：使用常规队列

在第10章，我介绍过如何使用jQuery特效队列管理用于匹配元素的一连串特效。事实上，特效队列就是一个常规队列，是被广泛应用的一个通用功能。表34-2针对常规用途重新阐释了与队列有关的方法。

表34-2 队列方法

方 法	描 述
<code>clearQueue(&lt;name&gt;)</code>	删除指定队列中尚未运行的所有函数
<code>queue(&lt;name&gt;)</code>	返回即将施加在jQuery对象包含元素之上的指定函数队列
<code>queue(&lt;name&gt;, function)</code>	把一个函数追加到队列的末尾
<code>dequeue(&lt;name&gt;)</code>	为jQuery对象中包含的元素删除并执行队列中的第一个函数
<code>delay(&lt;time&gt;, &lt;name&gt;)</code>	在指定队列的效果函数之间添加一段时间延迟

使用这些方法时，如果我们没有指队列的名称，jQuery就会默认使用fx，即视觉特效队列。当然我们也可以随便指定一个队列名生成函数队列。

当我们把jQuery队列用于常规目的（而不是视觉特效）时，我们应该使用clearQueue方法而不是stop方法。stop方法针对jQuery视觉特效做了特殊优化，并不适用于常规目的。代码清单34-1演示了队列的常规用法。

#### 代码清单34-1 使用队列

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <script type="text/javascript">
    $(document).ready(function() {

      var elems = $("input");

      elems.queue("gen", function(next) {
        $(this).val(100).css("border", "thin red solid");
        next();
      });

      elems.delay(1000, "gen");

      elems.queue("gen", function(next) {
        $(this).val(0).css("border", "");
        $(this).dequeue("gen");
      });

      $("<button>Process Queue</button>").appendTo("#buttonDiv")
        .click(function(e) {
          elems.dequeue("gen");
          e.preventDefault();
        });

    });
  </script>
</head>
<body>
```

```

<h1>Jacqui's Flower Shop</h1>
<form method="post">
  <div id="oblock">
    <div class="dtable">
      <div id="row1" class="drow">
        <div class="dcell">
          <label for="aster">Aster:</label>
          <input name="aster" value="0" required />
        </div>
        <div class="dcell">
          <label for="daffodil">Daffodil:</label>
          <input name="daffodil" value="0" required />
        </div>
        <div class="dcell">
          <label for="rose">Rose:</label>
          <input name="rose" value="0" required />
        </div>
      </div>
      <div id="row2" class="drow">
        <div class="dcell">
          <label for="peony">Peony:</label>
          <input name="peony" value="0" required />
        </div>
        <div class="dcell">
          <label for="primula">Primula:</label>
          <input name="primula" value="0" required />
        </div>
        <div class="dcell">
          <label for="snowdrop">Snowdrop:</label>
          <input name="snowdrop" value="0" required />
        </div>
      </div>
    </div>
    <div id="buttonDiv"><button type="submit">Place Order</button></div>
  </form>
</body>
</html>

```

**提示** 这个例子中的styles.css文件在第二部分中也出现过。

在这个例子中，我为页面中的input元素创建了队列gen，并在队列中添加了三个函数。首先，使用val方法将所有input元素的值设为100，并使用css方法为input元素添加了边框。接着，使用delay方法在队列各函数之间添加了一个1秒钟的延时。最后，我使用val和css方法把这些input元素恢复为最初的状态。

我还在页面中加了一个按钮，点击这个按钮会调用dequeue方法。与视觉特效队列不同，常规队列需要我们亲自启动执行。图34-1展示了脚本的效果。

我们放到队列中的函数与事件队列的工作方式相同，和前面一样，不管是调用dequeue方法还是调用作为参数传递进来的函数，都需要我们亲自执行。我更喜欢使用参数函数——我在调用dequeue方法

时经常因为忘记指定队列名而失败。

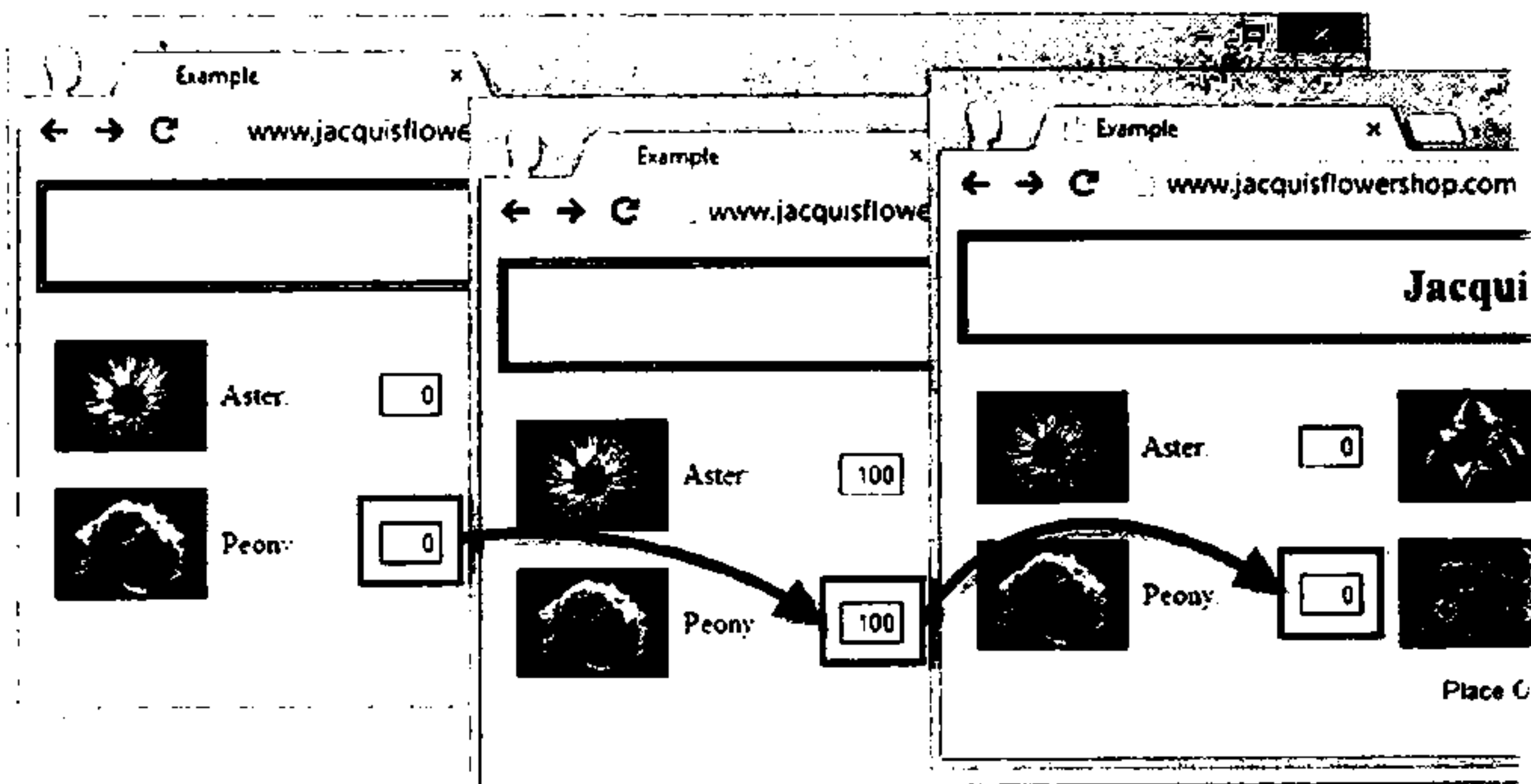


图34-1 使用常规队列

## 人工处理队列项

当然，并不是只有从一个队列函数中调用另一个队列函数这一条路。我们也能使用外部触发器来发动队列，比如让用户点击添加到页面中的那个按钮。代码清单34-2演示了这种用法。

### 代码清单34-2 明确启动队列执行

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("input").queue("gen", function() {
            $(this).val(100).css("border", "thin red solid");
        }).queue("gen", function() {
            $(this).val(0).css("border", "");
        }).queue("gen", function() {
            $(this).val(100).css("border", "thin blue solid");
            $("#dequeue").attr("disabled", "disabled");
        });
        $("<button id=dequeue>Dequeue Item</button>").appendTo("#buttonDiv")
            .click(function(e) {
                $("input").dequeue("gen");
                e.preventDefault();
            });
    });
</script>
...
```

在这个脚本中，我使用链式调用方法多次调用queue方法，每次调用都会添加一个队列函数，分别用于设置选中元素的边框和禁用button元素。执行队列中的每个函数之前需要点击一次按钮，因为此处没有自动链式调用。

## 34.2 数组实用方法

jQuery中定义了一些处理数组的实用方法(见表34-3)。大多数情况,我们都是处理HTML`Element`数组,更适合使用标准的jQuery方法处理或者过滤元素。这些实用方法擅长的,是处理其它类型的数组。

表34-3 处理数组的实用方法

方 法	描 述
<code>\$.grep(&lt;array&gt;, function)</code> <code>\$.grep(&lt;array&gt;, function, &lt;invert&gt;)</code>	使用回调函数过滤数组内容
<code>\$.inArray(&lt;value&gt;, &lt;array&gt;)</code>	判断数组中是否具有某个值
<code>\$.map(&lt;array&gt;, function)</code> <code>\$.map(&lt;map&gt;, function)</code>	使用回调函数遍历处理数组元素或者对象属性
<code>\$.merge(&lt;array&gt;, &lt;array&gt;)</code>	把第二个数组追加合并到第一个数组
<code>\$.unique(HTML<code>Element</code>[])</code>	按数组中的HTML <code>Element</code> 对象在页面中出现的顺序对数组排序,同时删除重复元素

### 34.2.1 使用grep方法

grep方法允许我们从一个数组中挑选出匹配过滤函数条件的数组元素。代码清单34-3演示了这个方法的用法。

代码清单34-3 使用grep方法

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var flowerArray = ["aster", "daffodil", "rose", "peony", "primula", "snowdrop"];

        var filteredArray = $.grep(flowerArray, function(elem, index) {
            return elem.indexOf("p") > -1;
        });
        for (var i = 0; i < filteredArray.length; i++) {
            console.log("Filtered element: " + filteredArray[i]);
        }
    });
</script>
...
```

过滤函数会得到两个参数,第一个是数组元素,第二个是元素对应的索引。数组中的每个元素都会调用一次过滤函数,如果希望保留这个元素,就让过滤函数返回true。

在这个例子中,我使用grep方法处理一个字符串数组,滤掉那些不包含字母p的元素,并把过滤后的数组内容输出到控制台,得到以下结果:

```
Filtered element: peony
Filtered element: primula
Filtered element: snowdrop
```

我们还可以多传递一个参数给grep方法。如果这个参数是true,就会对过滤过程取反,也就是说

返回那些不匹配过滤条件的元素。代码清单34-4演示了这个参数的效果。

代码清单34-4 使用grep方法执行反向选择

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var flowerArray = ["aster", "daffodil", "rose", "peony", "primula", "snowdrop"];

        var filteredArray = $.grep(flowerArray, function(elem, index) {
            return elem.indexOf("p") > -1;
        }, true);

        for (var i = 0; i < filteredArray.length; i++) {
            console.log("Filtered element: " + filteredArray[i]);
        }

    });
</script>
...
```

新的输出结果如下。

---

```
Filtered element: aster
Filtered element: daffodil
Filtered element: rose
```

---

### 34.2.2 使用inArray方法

我们可以使用inArray方法判断数组中是否包含某个元素。如果包含则返回数组的索引值，否则返回-1。代码清单34-5演示了inArray方法的用法。

代码清单34-5 使用inArray方法

```
...
<script type="text/javascript">
    $(document).ready(function() {
        var flowerArray = ["aster", "daffodil", "rose", "peony", "primula", "snowdrop"];
        console.log("Array contains rose: " + $.inArray("rose", flowerArray));
        console.log("Array contains lily: " + $.inArray("lily", flowerArray));
    });
</script>
...
```

上面的脚本检查flowers数组中是否包含rose和lily元素，结果如下。

---

```
Array contains rose: 2
Array contains lily: -1
```

---

### 34.2.3 使用map方法

map方法允许我们遍历数组元素或映射对象（map object）的属性，使用回调函数进行处理，并把回调函数的返回值存放到一个新数组中返回。代码清单34-6演示了如何使用map方法处理一个数组。

代码清单34-6 使用map方法处理一个数组

```
...
<script type="text/javascript">
  $(document).ready(function() {
    var flowerArray = ["aster", "daffodil", "rose", "peony", "primula", "snowdrop"];

    var result = $.map(flowerArray, function(elem, index) {
      return index + ": " + elem;
    });

    for (var i = 0; i < result.length; i++) {
      console.log(result[i]);
    }
  });
</script>
...
```

数组中的每个元素都会调用一次处理函数，数组元素的值和索引会作为参数传递给处理函数。map方法会利用处理函数的返回值形成一个新的数组，作为map方法的返回值。在这个脚本中，我把数组中的每个元素的索引和值（使用冒号和一个空格）连接起来，得到以下结果：

---

```
0: aster
1: daffodil
2: rose
3: peony
4: primula
5: snowdrop
```

---

我们也可以使用map方法挑选元素。如果处理函数在处理某个元素时没有返回值，返回结果中也就不会有对应这个元素的值。代码清单34-7演示了如何从一个数组中有选择地返回处理结果。

代码清单34-7 遍历一个数组，有条件地返回值

```
...
<script type="text/javascript">
  $(document).ready(function() {
    var flowerArray = ["aster", "daffodil", "rose", "peony", "primula", "snowdrop"];

    var result = $.map(flowerArray, function(elem, index) {
      if (elem != "rose") {
        return index + ": " + elem;
      }
    });

    for (var i = 0; i < result.length; i++) {
      console.log(result[i]);
    }
  });
</script>
...
```



```

    }
  });
</script>
...

```

除rose元素之外，数组中的其他元素都生成了新值，输出结果如下：

```

0: aster
1: daffodil
3: peony
4: primula
5: snowdrop

```

### 34.2.4 使用merge方法

如代码清单34-8所示，merge方法把两个数组合并为一个。

代码清单34-8 使用merge方法

```

...
<script type="text/javascript">
  $(document).ready(function() {

    var flowerArray = ["aster", "daffodil", "rose", "peony", "primula", "snowdrop"];
    var additionalFlowers = ["carnation", "lily", "orchid"];

    $.merge(flowerArray, additionalFlowers);

    for (var i = 0; i < flowerArray.length; i++) {
      console.log(flowerArray[i]);
    }
  });
</script>
...

```

第二个数组的元素被追加到第一个数组的元素之后，合并操作修改了第一个数组（以容纳更多的元素）。上面例子中的脚本输出如下。

```

aster
daffodil
rose
peony
primula
snowdrop
carnation
lily
orchid

```

### 34.2.5 使用unique方法

unique方法按它们在页面中出现的顺序对数组中的HTMLElement对象进行排序，同时删除重复的元

素。代码清单34-9演示了这个方法的使用。

代码清单34-9 使用unique方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        var selection = $("img[src*=rose], img[src*=primula]").get();
        $.merge(selection, $("img[src*=aster]"));
        $.merge(selection, $("img"));

        $.unique(selection);
        for (var i = 0; i < selection.length; i++) {
            console.log("Elem: " + selection[i].src);
        }
    });
</script>
...
```

排序操作是就地进行的，也就是说unique方法直接修改我们传入的参数。在本例中，我通过merge方法创建了一个HTMLElement对象数组，里面不但有重复元素，而且顺序也与这些元素在页面中出现的顺序不符，最后我调用unique方法排序并去除重复元素。

### 34.3 数据类型实用方法

jQuery提供了一系列检测JavaScript对象类型的工具方法。表34-4列出了这些方法。

表34-4 数据类型实用方法

方 法	描 述
\$.isArray(Object)	若对象是一个数组，返回true
\$.isEmptyObject(Object)	若对象未定义任何方法和属性，返回true
\$.isFunction(Object)	若对象是一个函数，返回true
\$.isNumeric(Object)	若对象是一个数字，返回true
\$.isWindow(Object)	若对象是一个Window对象，返回true
\$.isXMLDoc(Object)	若对象是一个XML文档对象，返回true
\$.type(Object)	返回对象所属的JavaScript内建类型

表中的大多数方法都非常简单。传递对象给方法，若对象满足检测条件则返回true，否则返回false。代码清单34-10是一段非常简单的脚本，演示了isFunction方法的使用。

代码清单34-10 使用isFunction方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        function myFunc() {
```

```

        console.log("Hello!");
    }

    console.log("IsFunction: " + $.isFunction(myFunc));
    console.log("IsFunction: " + $.isFunction("hello"));
});
</script>
...

```

在这个例子中，我使用isFunction方法检测了两个对象，结果如下。

---

```

IsFunction: true
IsFunction: false

```

---

## type方法

type方法与其他方法略有不同，它返回一个对象所属的用字符串表示的JavaScript基本数据类型。根据具体的检测对象，它的返回值会是以下字符串值中的某一个。

- ☐ boolean
- ☐ number
- ☐ string
- ☐ function
- ☐ array
- ☐ date
- ☐ regexp
- ☐ object

代码清单34-11演示了type方法的用法

### 代码清单34-11 使用type方法

```

...
<script type="text/javascript">
    $(document).ready(function() {

        function myFunc() {
            console.log("Hello!");
        }

        var jq = $("img");
        var elem = document.getElementById("row1");

        console.log("Type: " + $.type(myFunc));
        console.log("Type: " + $.type(jq));
        console.log("Type: " + $.type(elem));

    });
</script>
...

```

在这段脚本中，我使用type方法测试了一个方法、一个jQuery对象和一个HTMLElement对象，结果如下。

```
Type: function
Type: object
Type: object
```

34.4 数据有关实用方法

jQuery中还定义了一些处理各类数据的实用方法，表34-5列出了这些方法。

表34-5 处理数据的实用方法

方 法	描 述
serialize()	将表单数据编码为适合提交到服务器的查询字符串
serializeArray()	将表单数据整理到一个数组中，为编码成JSON数据做准备
\$.parseJSON(<json>)	将字符串<json>解析为JSON对象
\$.parseXML(<xml>)	将字符串<xml>解析为XMLDocument对象
\$.trim(String)	删除字符串首尾空白

34.4.1 序列化表单数据

serialize和serializeArray方法通过常规方式或Ajax方式，为从一堆表单元素中提取出提交表单所需的数据，提供了一种便利的手段。代码清单34-12演示了这两种方法的使用。

代码清单34-12 序列化表单数据

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("<button>Serialize</button>").appendTo("#buttonDiv").click(function(e) {

            var formArray = $("form").serializeArray();
            console.log("JSON: " + JSON.stringify(formArray))

            var formString = $("form").serialize();
            console.log("String: " + formString)

        });e.preventDefault();
    });
</script>
...
```

在这个例子中，我分别使用两个方法序列化文档中的表单元素，并把结果输出到控制台。serializeArray方法返回一个JavaScript数组，数组中的每个元素都是一个对象，对应着表单中的一个元素。这些对象均有两个属性：name属性是表单元素name属性的值，value属性则是表单元素的值。下

面是示例页面的输出结果。

```
[{"name": "aster", "value": "1"}, {"name": "daffodil", "value": "0"},
{"name": "rose", "value": "0"}, {"name": "peony", "value": "0"},
{"name": "primula", "value": "2"}, {"name": "snowdrop", "value": "0"}]
```

相比之下，serialize方法则是根据表单元素生成编过码的URL查询字符串，如下所示：

```
aster=1&daffodil=0&rose=0&peony=0&primula=2&snowdrop=0
```

### 34.4.2 解析数据

当我们需要处理Ajax请求的返回结果时，parseJSON和parseXML方法特别有用。对绝大多数Web应用来说，JSON是数据传输的最佳选择，至于原因我已经在第14章说过，这里不再重复。XML仍在使⽤，不过我发现自己仅在新应用程序与陈旧后端系统整合时才使用XML数据。代码清单34-13演示了parseJSON方法的用法。

代码清单34-13 解析JSON数据

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("<button>Serialize</button>").appendTo("#buttonDiv").click(function(e) {

            var jsonData = '{"name": "Adam Freeman", "city": "London", "country": "UK"}'

            var dataObject = $.parseJSON(jsonData)

            for (var prop in dataObject) {
                console.log("Property: " + prop + " Value: " + dataObject[prop])
            }

            e.preventDefault();
        });

    });
</script>
...
```

在这个例子中，我先是定义了一个简单的JSON字符串，然后使用parseJSON方法将其还原为JavaScript对象。最后，我把这个对象的属性和值迭代输出到控制台，产生以下输出：

```
Property: name Value: Adam Freeman
Property: city Value: London
Property: country Value: UK
```

### 34.4.3 去除字符串首尾空白

`trim`方法用来删除给定字符串首尾的空白，包括空格、制表符和换行符。这一字符串功能如此常用，以至于几乎所有的编程语言都将它内置为字符串核心处理功能的一部分，遗憾的是JavaScript却因为某种原因并未提供该功能。代码清单34-14演示了`trim`方法的用法。

代码清单34-14 使用`trim`方法

```
...
<script type="text/javascript">
    $(document).ready(function() {

        $("<button>Serialize</button>").appendTo("#buttonDiv").click(function(e) {

            var sourceString = "\n  This string contains whitespace  ";
            console.log(">" + sourceString + "<")

            var resultString = $.trim(sourceString);
            console.log(">" + resultString + "<")

        });
        e.preventDefault();
    });
</script>
...
```

我在这个例子中使用了`trim`方法，并在控制台分别输出了原始字符串和处理之后的字符串，结果如下：

```
> This string contains whitespace  <
>This string contains whitespace<
```

## 34.5 其他实用方法

还有一些jQuery方法，尽管不太适合单独为它们建一个分类，但这些方法也非常有用。表34-6列出了这些方法。

表34-6 其他实用方法

方 法	描 述
<code>\$.contains(HTMLElement, HTMLElement)</code>	如果第一个参数元素包含第二个参数元素，返回true
<code>\$.now()</code>	返回当前时间，相当于 <code>new Date().getTime()</code>

### 检查两个元素是否包含

`contains`方法支持我们检查一个元素是否包含着另一个。两个参数都是`HTMLElement`对象，如果第一个参数对应的元素包含第二个参数对应的元素，`contains`方法就返回true。代码清单34-15演示了

contains方法的使用。

#### 代码清单34-15 使用contains方法

```
...
<script type="text/javascript">
    $(document).ready(function() {
        $("img").hover(function(e) {
            var elem = document.getElementById("row1");
            if ($.contains(elem, this)) {
                $(e.target).css("border", e.type == "mouseenter" ?
                    "thick solid red" : "");
            }
        });
    });
</script>
...
```

在这段脚本中，我使用DOM API得到一个HTMLElement对象，并检查它是否包含传递给事件处理函数的this元素。如果包含，脚本就会修改触发事件元素的边框样式。

---

**提示** 这个方法的参数必须是HTMLElement对象。如果希望对jQuery对象进行类似的检查，可以使用我在第6章讲过的find方法。

---

## 34.6 小结

我在本章中讲解了jQuery实用方法——一整套精心挑选的功能函数，它们能有效帮助我们进一步处理jQuery对象，或者弥补JavaScript语言功能上的不足。这些都是你用一次就会喜欢的实用方法，不过对绝大多数Web应用项目而言，未必是不可或缺的。

我将在本章中重点讲解jQuery UI的两大实用功能。第一个是对jQuery已有方法的增强，我们能够以动画的形式改变颜色、显示/隐藏元素，以及修改元素的css类。另一个是一整套CSS类，利用这些类我们可以把jQuery UI主题样式应用于页面的其他部分，从而让整个应用程序具有一致的外观。表35-1列出了本章概要。

表35-1 本章概要

问 题	解决方法	代码清单
如何让改变颜色支持动画	使用增强版的animate方法	1
如何让改变元素的类支持动画	使用增强版的addClass、removeClass和toggleClass方法和switchClass方法	2、3
如何让显示/隐藏元素支持动画	使用增强版的show、hide和toggle方法	4
如何在不改变元素可见性的基础上使用效果	使用effect方法	5
如何像设置组件样式一样设置元素样式	使用组件容器类	6
如何让一个元素变成圆角	使用corner类	7
如何把可点击组件的样式应用到一个元素	使用interaction state类	8
如何提示用户元素当前的状态	使用提示（cue）类	9、10

## 35.1 使用 jQuery UI 特效

jQuery UI 扩展了一些jQuery内建方法，以支持从元素的颜色到元素的类等属性的更多过渡。如果我们谨慎地使用这些特性，可以为Web应用锦上添花。作为对这些功能的补充，jQuery UI也定义了一些额外的动画效果。

### 35.1.1 让颜色变化支持动画

jQuery UI扩展了jQuery的animate方法（参见第10章），以支持对颜色进行动画处理。新的animate方法支持对1个或几个定义元素颜色的CSS属性进行动画。表35-2列出了animate方法支持的CSS属性。



表35-2 jQuery UI版本的animate方法支持的CSS属性

属 性 名	描 述
backgroundColor	设置元素的背景色
borderTopColor、borderBottomColor、borderLeftColor、borderRightColor	分别设置元素各个边框的颜色
color	设置元素文本颜色
outlineColor	设置元素轮廓的颜色，用来增强元素

要以动画方式改变颜色，我们需要准备一个JavaScript对象，定义好动画所需的属性和目标值，把它作为参数传递给animate方法。代码清单35-1给出了一个例子。

#### 代码清单35-1 制作颜色动画

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    #animTarget {
      background-color: white;
      color: black;
      border: medium solid black;
      width: 200px; height: 50px;
      text-align: center;
      font-size: 25px;
      line-height: 50px;
      display: block;
      margin-bottom: 10px;
    }
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      $("button").click(function() {
        $("#animTarget").animate({
          backgroundColor: "black",
          color: "white"
        });
      });
    });
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>

  <div id=animTarget>
    Hello!
  </div>

  <button>Animate Color</button>
```

```
</body>
</html>
```

我为页面中的div元素设置了初始样式：白色背景，黑色文字。点击页面中的按钮时，就会执行animate方法，分别让指定的那些属性变为相对的颜色。颜色的变化是逐步进行的，而且各个属性的动画过程同步。图35-1展示了这种效果。

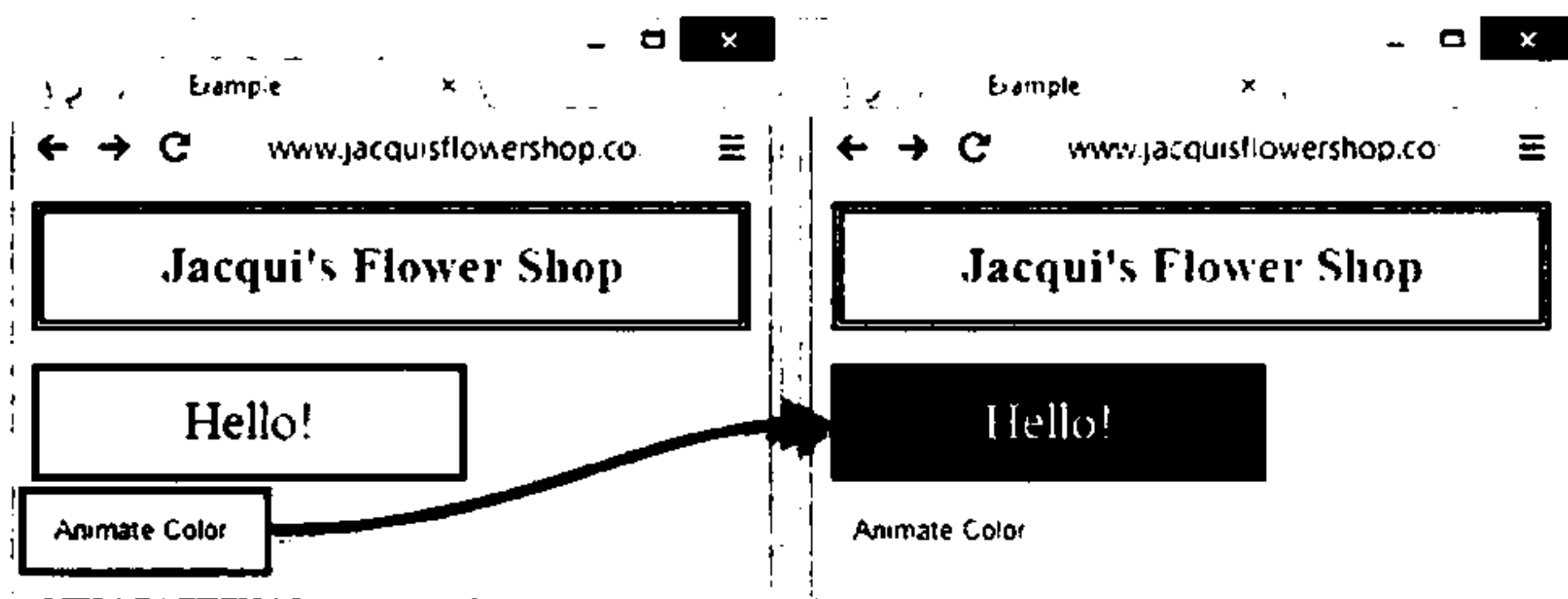


图35-1 让颜色变化支持动画

**提示** 注意在这个盒子里，我在style元素中使用了CSS标准属性名：background-color。然而我在指定同一属性的参数对象中，却改用驼峰命名法的backgroundColor作为属性名。这样我不必用引号包住这个属性术语，就能安全地在JavaScript对象中使用这个CSS属性名了<sup>①</sup>。

在这个例子中，我使用CSS颜色名称指定目标元素颜色：白底黑字。许多常用颜色可使用速记值（即颜色名称），animate方法还支持十六进制颜色值（比如#FFFFFF）和使用RGB函数表示的颜色，如rgb(255, 255, 255)。

**提示** 这个animate方法除额外支持颜色属性以外，用法与第10章讲到的animate方法完全相同。

### 35.1.2 让切换类样式支持动画

jQuery UI为使用一组CSS属性生成动画提供了一种便捷方式：使用类。不同于为每个CSS属性指定值，我们只需要使用类定义好样式，然后告诉jQuery UI为某个或某组元素添加这个类就可以了。jQuery UI会自动为元素从一个状态到另一个状态的过程生成动画。代码清单35-2演示了这一方法。

#### 代码清单35-2 使用类生成动画

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
```

① 注意，即使使用引号括住background-color这个术语，也无法实现同样的效果。使用驼峰命名法代替CSS属性中的减号是DOM的约定，当遇到CSS属性名中间带有减号时，有且仅有驼峰命名法才是合法的。——译者注

```

<script src="jquery-2.0.2.js" type="text/javascript"></script>
<script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<style type="text/css">
    .elemClass {
        background-color: white;
        color: black;
        border: medium solid black;
        width: 200px; height: 50px;
        text-align: center;
        font-size: 25px;
        line-height: 50px;
        display: block;
        margin-bottom: 10px;
    }
    .myClass {
        font-size: 40px; background-color: black; color: white;
    }
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $("button").click(function() {
            if (this.id == "add") {
                $("#animTarget").addClass("myClass", "fast")
            } else {
                $("#animTarget").removeClass("myClass", "fast")
            }
        });
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>

    <div id=animTarget class="elemClass">
        Hello!
    </div>
    <button id="add">Add Class</button>
    <button id="remove">Remove Class</button>
</body>
</html>

```

没错，jQuery UI又一次扩展了现有的jQuery方法，以增强其功能。在本例中，使用了经jQuery UI增强的addClass和removeClass方法。在第8章的时候，我们讲过这两个方法的标准版本。增强版本的方法本质上与标准版本相同，都是通过第二个参数来指定样式发生变化时动画持续的时间。

在本例中，我定义了myClass类，并且在为页面中的按钮添加或者移除myClass类时使用了命名持续时间参数fast。这个例子的实际效果见图35-2。

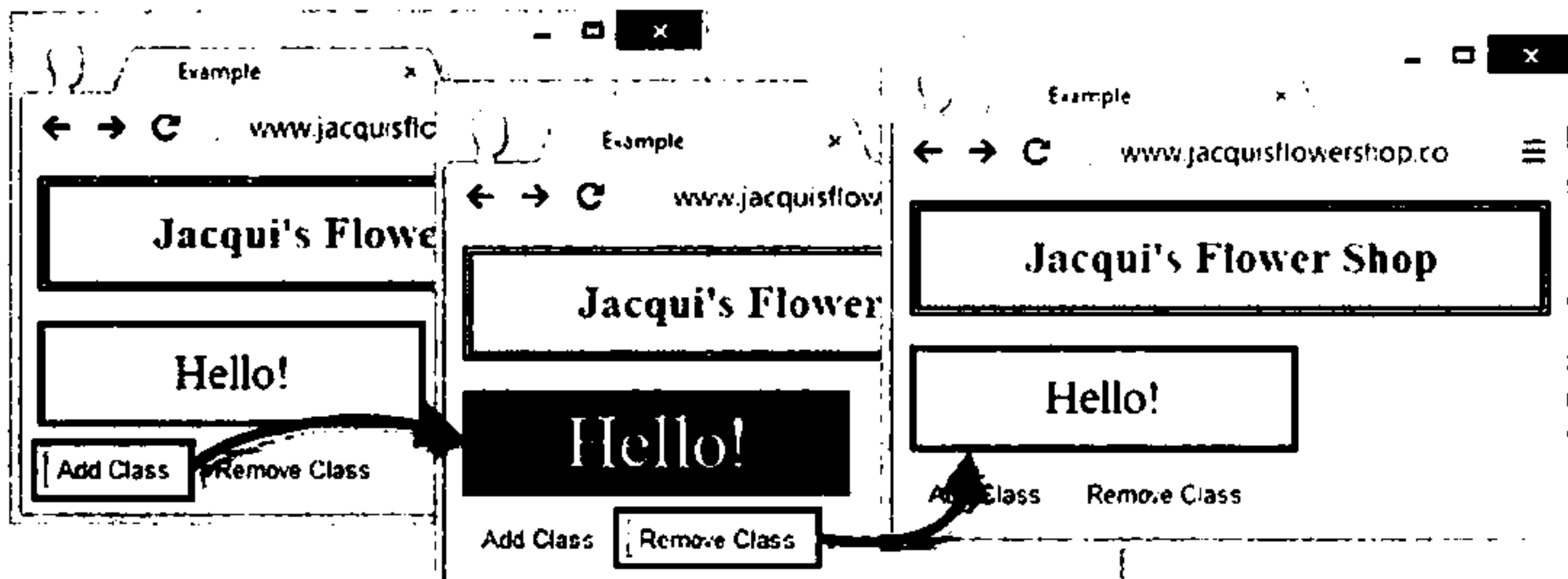


图35-2 使用类生成动画

**提示** 根据CSS样式层叠规则，只有当这个类相对于目标元素“最专一”时，类定义的样式属性才会应用于目标元素。在上一个例子中，我通过id指定了元素的初始样式。而在本例中，为便于修改，我改用了类。想详细了解层叠样式表，请阅读第3章。

jQuery UI也增强了toggleClass方法，它与我们在第8章中提到的toggleClass方法功能相同。类似于我们刚刚提到的addClass和removeClass方法，它也是通过第二个参数来控制动画的持续时间。

### 切换样式类

jQuery UI不仅仅增强了一些已有的jQuery方法，它还增加了一个方法switchClass，用来给元素删除一个类，再加上另一个类，并且让这个状态变化过程支持动画。代码清单35-3演示了这个方法的用法。

#### 代码清单35-3 switchClass方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    .elemClass {
      border: medium solid black;
      width: 200px; height: 50px;
      text-align: center;
      line-height: 50px;
      display: block;
      margin-bottom: 10px;
    }
    .classOne {
      font-size: 25px; background-color: white; color: black;
    }
  </style>
</head>
<body>
  <div class="elemClass">
    <div class="classOne">Jacqui's Flower</div>
    <div class="classOne">Hello!</div>
  </div>
</body>
</html>
```

```

        .classTwo {
            font-size: 40px; background-color: black; color: white;
        }
    </style>
    <script type="text/javascript">
        $(document).ready(function() {
            $("button").click(function() {
                $("#animTarget").switchClass("classOne", "classTwo", "fast");
            });
        });
    </script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>

    <div id=animTarget class="elemClass classOne">
        Hello!
    </div>
    <button>Switch Class</button>
</body>
</html>

```

switchClass方法支持三个参数，分别是要删除的类、要添加的类及动画持续时间。在本例中，我的两个类定义了相同的属性（不同的值），不过这并非switchClass方法的限制（仅仅是我的习惯而已）。

### 35.1.3 jQuery UI动画

jQuery UI包含大量的动画效果，几乎与我们在第10章讲到的jQuery核心库特效一样多。我建议在使用特效时保持节制。小心使用特效，的确能增强用户体验；然而如果过多过滥，反而会给用户带来烦恼。jQuery UI提供了许多动画效果，如变暗、弹跳、剪裁、掉落、爆炸、淡入淡出、折叠、高亮、喷烟、抖动、剥落、晃动、改变大小和滑动。

---

**注意** 在本章中，我只告诉你如何使用这些效果，而不是深入每个效果的细节。<http://doc.jquery.com/UI/Effects>页面很好地总结了这些特效的使用及设置。

---

#### 1. 显示与隐藏元素特效

jQuery UI增强了jQuery中的show、hide和toggle方法以支持动画效果。我们已经在第10章了解了这些方法的标准版本。为了使用这些jQuery UI增强过的版本，只需要再提供一个duration参数，设置动画的持续时间。代码清单35-4展示了这些增强方法的用法。

代码清单35-4 增强版本的show、hide和toggle方法

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>

```

```

<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<style type="text/css">
    .elemClass {
        font-size: 25px; background-color: white; color: black;
        border: medium solid black; width: 200px; height: 50px;
        text-align: center; line-height: 50px; display: block; margin-bottom: 10px;
    }
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $("button").click(function() {
            switch (this.id) {
                case "show":
                    $("#animTarget").show("fold", "fast");
                    break;
                case "hide":
                    $("#animTarget").hide("fold", "fast");
                    break;
                case "toggle":
                    $("#animTarget").toggle("fold", "fast");
                    break;
            }
        });
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>

    <button id="hide">Hide</button>
    <button id="show">Show</button>
    <button id="toggle">Toggle</button>

    <div id="animTarget" class="elemClass">
        Hello!
    </div>
</body>
</html>

```

这个例子中共有三个按钮，点击它们分别会显示、隐藏及切换显示#animTarget元素。对于这三个按钮，我都指定使用折叠动画，并将持续时间参数设为fast。除支持动画特效之外，这几个方法的工作方式与jQuery核心中定义的标准版本并无二致。

## 2. 特有的effect方法

jQuery UI定义了effect方法，它让我们无需显示/隐藏元素就能让元素动起来。正确使用动画，是在页面中抓取用户注意力的好方法。代码清单35-5就给出这样的例子。

### 代码清单35-5 effect方法

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>

```

```

<link rel="stylesheet" type="text/css" href="styles.css"/>
<link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
<style type="text/css">
    .elemClass {
        font-size: 25px; background-color: white; color: black;
        border: medium solid black; width: 200px; height: 50px;
        text-align: center; line-height: 50px; display: block; margin-bottom: 10px;
    }
</style>
<script type="text/javascript">
    $(document).ready(function() {

        $("button").click(function() {
            $("#animTarget").effect("pulsate", "fast");
        });
    });
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>

    <div id=animTarget class="elemClass">
        Hello!
    </div>
    <button>Effect</button>
</body>
</html>

```

当点击本例中的按钮时，元素#animTarget并不会消失，而是就地抖动起来。在本例中我使用了pulsate（抖动）特效，因此元素会跳跳停停。

## 35.2 使用 jQuery UI CSS 框架

jQuery UI通过一整套CSS样式类管理组件的外观。其中有一些类是公开的，这样程序员们就能利用这些类设置元素的样式，以得到与jQuery UI一致的外观。在本书第四部分的例子中，我已经用过了其中的几个类。

### 35.2.1 组件容器样式

jQuery UI组件中最核心的样式由三个最基础的类定义，如表35-3所示。

表35-3 jQuery UI 组件容器类

类 名	描 述
ui-widget	定义容器元素的一般样式
ui-widget-header	定义标题类容器元素的样式
ui-widget-content	定义内容类容器元素的样式

这几个类用于容器元素，即那些包含标题或内容元素的元素（或者，以ui-widget类为例，我们要处理的最外层元素）。代码清单35-6演示了这几个类的用法。

## 代码清单35-6 jQuery UI组件容器类

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    body > div {float: left; margin: 10px}
  </style>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>

  <div>
    <div>
      Flowers
    </div>
    <div>
      <div class="dcell">
        <label for="peony">Peony:</label>
        <input name="peony" value="0" />
      </div>
    </div>
  </div>
  <div class="ui-widget">
    <div class="ui-widget-header">
      Flowers
    </div>

    <div class="ui-widget-content">
      <div class="dcell">
        <label for="peony">Peony:</label>
        <input name="peony" value="0" />
      </div>
    </div>
  </div>
</body>
</html>

```

在这个例子中有两组元素，其中一组使用了组件容器类。具体效果见图35-3。

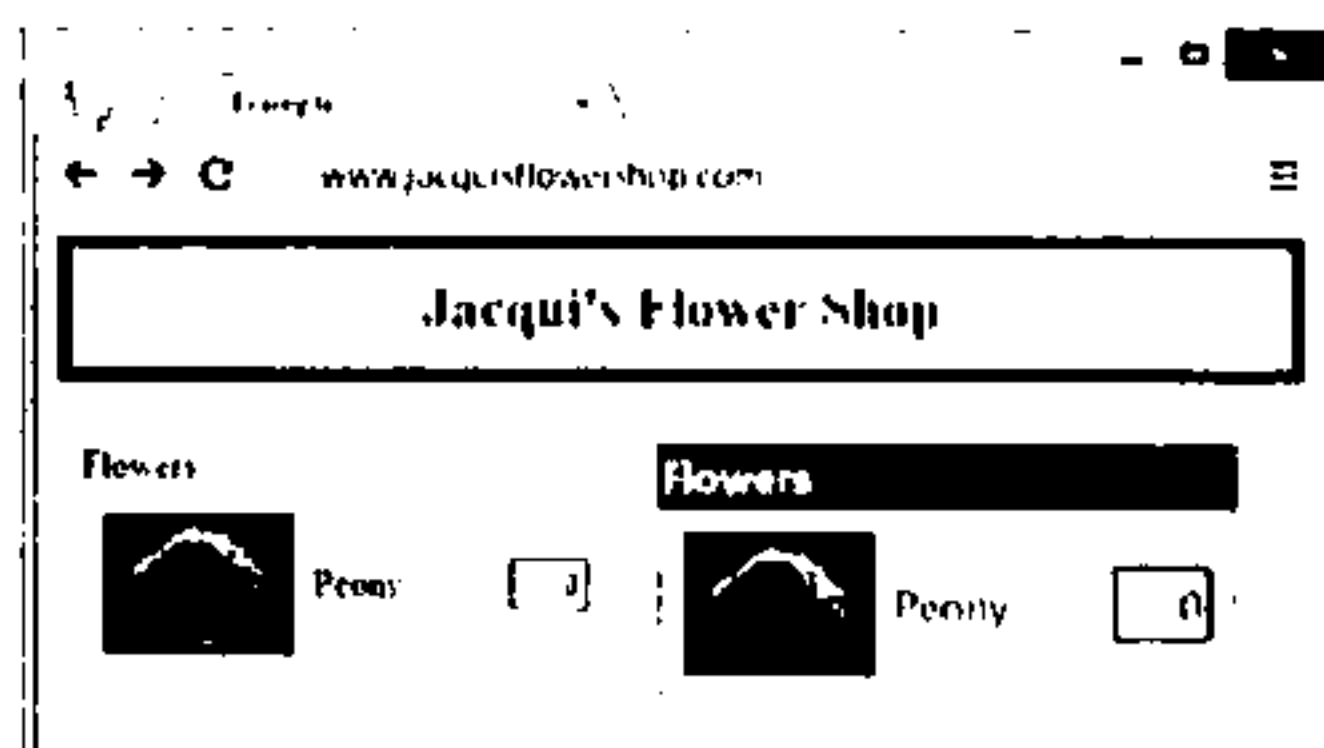


图35-3 jQuery UI组件容器类



### 35.2.2 圆角样式

接下来, jQuery UI提供的另一组类帮助我们为元素设置圆角样式。表35-4列出了这些类的用途。

35-4 jQuery UI组件圆角样式类

类 名	描 述
ui-corner-all	四个角全部圆角
ui-corner-bl	仅左下角圆角
ui-corner-bottom	左下及右下角圆角
ui-corner-br	仅右下角圆角
ui-corner-left	左上角及左下角圆角
ui-corner-right	右上及右下角圆角
ui-corner-tl	仅左上角圆角
ui-corner-top	左上及右上角圆角
ui-corner-tr	仅右上角圆角

只有当目标元素具有背景或者外边距样式时,这些类才起作用。这意味着我们可以像代码清单35-7那样,为它们应用ui-widget-header和ui-widget-content类。

代码清单35-7 使用圆角样式类

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    body > div {float: left; margin: 10px}
  </style>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>

  <div>
    <div>
      Flowers
    </div>
    <div>
      <div class="dcell">
        <label for="peony">Peony:</label>
        <input name="peony" value="0" />
      </div>
    </div>
  </div>

  <div class="ui-widget">
```

```
<div class="ui-widget-header ui-corner-top" style="padding-left: 5px">
  Flowers
</div>
<div class="ui-widget-content ui-corner-bottom">
  <div class="dcell">
    <label for="peony">Peony:</label>
    <input name="peony" value="0" />
  </div>
</div>
</body>
</html>
```

为营造整体效果，我为标题元素设置了ui-corner-top类，为内容元素设置了ui-corner-bottom类。具体效果见图35-4。注意，我给标题元素设置了一点点内边距，这为圆角和内容之间创造出一些空白，从而有效防止了内容会被圆角切掉一部分。

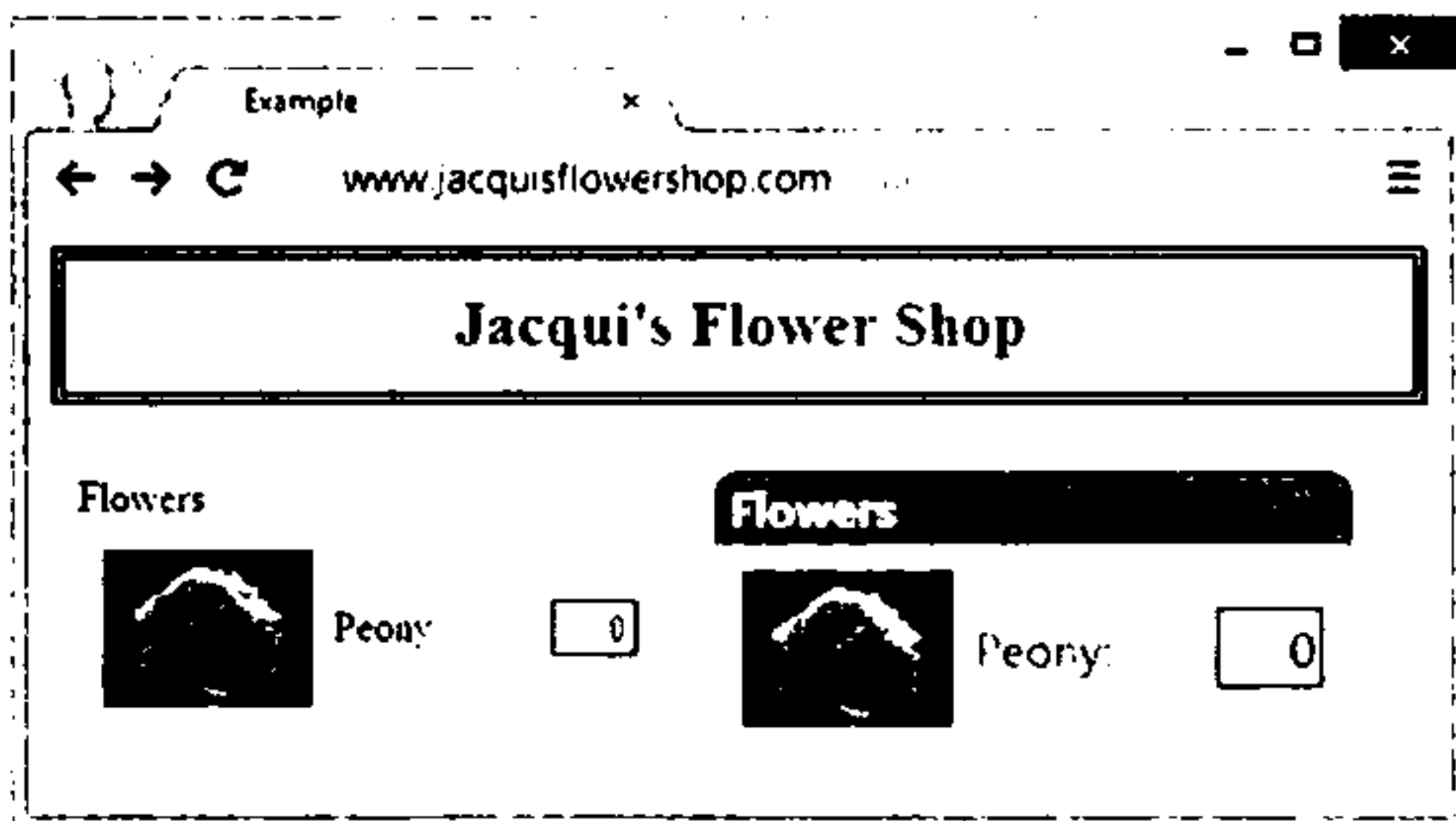


图35-4 使用圆角样式类

35.2.3 交互状态样式

jQuery UI还提供了用来表示交互状态的一些类，这些类能帮助我们生成与jQuery UI组件一致的交互界面。表35-5列出了这些类。

表35-5 jQuery UI提供的表示交互状态的类

类 名	描 述
ui-state-default	可点击组件默认样式
ui-state-hover	鼠标hover在可点击组件时呈现的样式
ui-state-focus	可点击组件得到焦点时的样式
ui-state-active	可点击组件被激活时的样式

代码清单35-8中使用了这四个类。注意在这个例子中，我为div元素内的每一个span元素设置了内边距。交互状态类也定义了内边距，瞄准内容元素（并为它设置内边距）是在容器元素与内容之间制造间距最容易的办法。

## 代码清单35-8 使用交互状态类

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    body > div {float: left; margin: 10px}
    span {padding: 10px; display: block}
  </style>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>

  <div class="ui-widget ui-state-default ui-corner-all">
    <span>Default</span>
  </div>
  <div class="ui-widget ui-state-hover ui-corner-all">
    <span>Hover</span>
  </div>
  <div class="ui-widget ui-state-focus ui-corner-all">
    <span>Focus</span>
  </div>
  <div class="ui-widget ui-state-active ui-corner-all">
    <span>Active</span>
  </div>
</body>
</html>

```

图35-5展示了每个类的效果。你会注意到例子中的效果与我使用的jQuery UI主题非常相似。如果需要加强交互状态的效果，也可以使用ThemeRoller（详见第17章）生成一个新的主题。

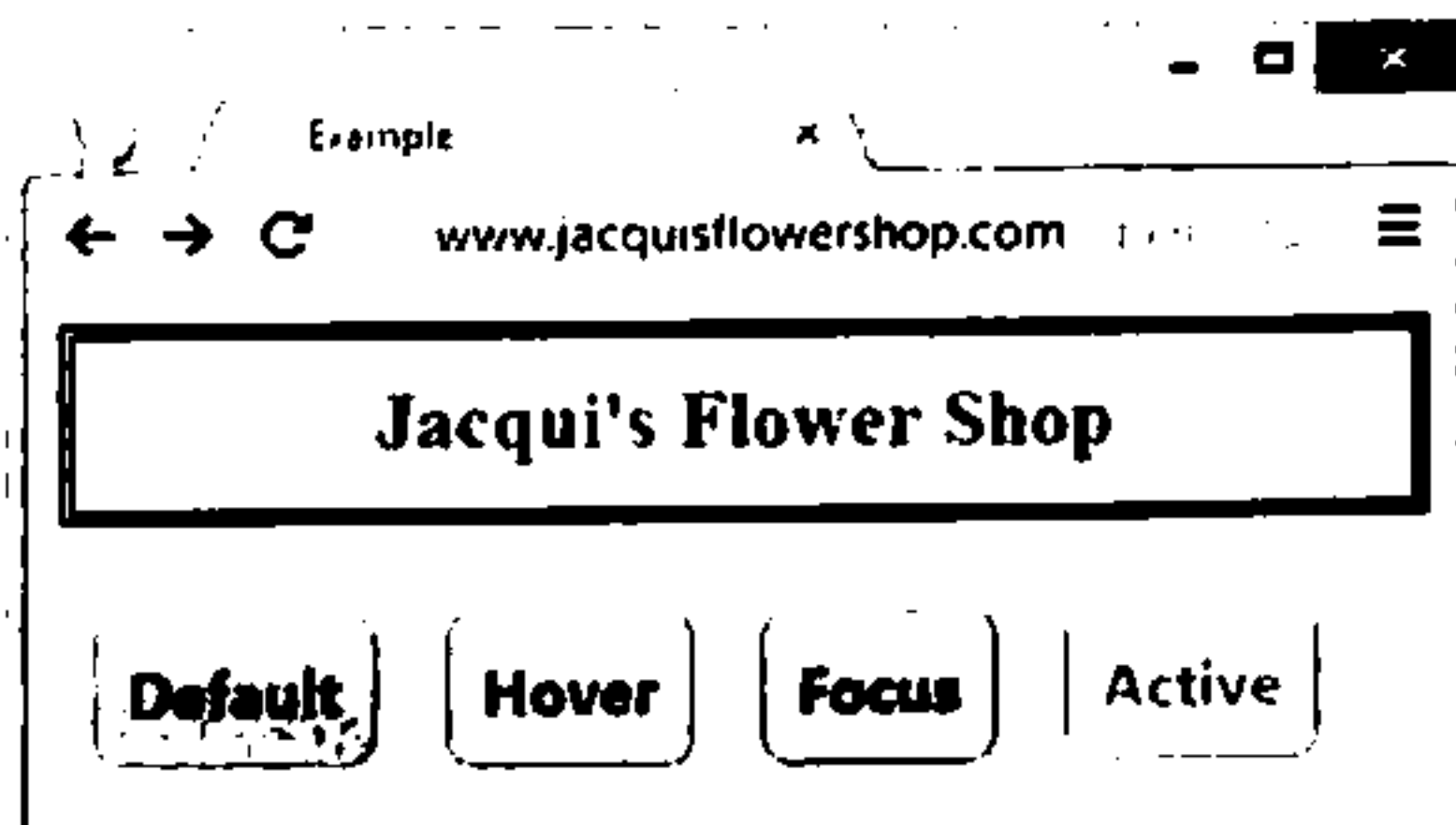


图35-5 交互状态类的实际效果

## 35.2.4 提示信息样式

jQuery UI支持我们使用预定义的提示类表示页面元素的当前状态。表35-6列出了这些提示类。

表35-6 jQuery UI的交互提示类

类 名	描 述
ui-state-highlight	高亮某个元素吸引用户注意
ui-state-error	强调某个包含错误信息的元素
ui-state-disabled	让某个元素看起来已被禁用（但并未实际禁用）

代码清单35-9演示了高亮提示及禁用提示类的用法。

代码清单35-9 jQuery UI的高亮类

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    body > div {float: left; margin: 10px}
    span {padding: 10px; display: block}
  </style>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>

  <div class="ui-widget">
    <div class="ui-widget-header ui-corner-top" style="padding-left: 5px">
      Flowers
    </div>
    <div class="ui-widget-content ui-corner-bottom">
      <div class="dcell">
        <label for="peony">Peony:</label>
        <input name="peony" value="0" />
      </div>
    </div>
  </div>

  <div class="ui-widget ui-state-highlight ui-corner-all">
    <div class="ui-widget-header ui-corner-top" style="padding-left: 5px">
      Flowers
    </div>
    <div class="ui-widget-content ui-corner-bottom">
      <div class="dcell">
        <label for="peony">Peony:</label>
        <input name="peony" value="0" />
      </div>
    </div>
  </div>

  <div class="ui-widget ui-state-disabled">
    <div class="ui-widget-header ui-corner-top" style="padding-left: 5px">
      Flowers
```

```

    </div>
    <div class="ui-widget-content ui-corner-bottom">
      <div class="dcell">
        <label for="peony">Peony:</label>
        <input name="peony" value="0" />
      </div>
    </div>
  </div>
</body>
</html>

```

图35-6展示了这些类的具体效果。注意，我在使用ui-state-highlight类的同时，还使用了ui-corner-all类，这个类为目标元素设置了圆角效果以代替默认的直角效果。如果子元素是圆角，最好让高亮元素也是圆角。

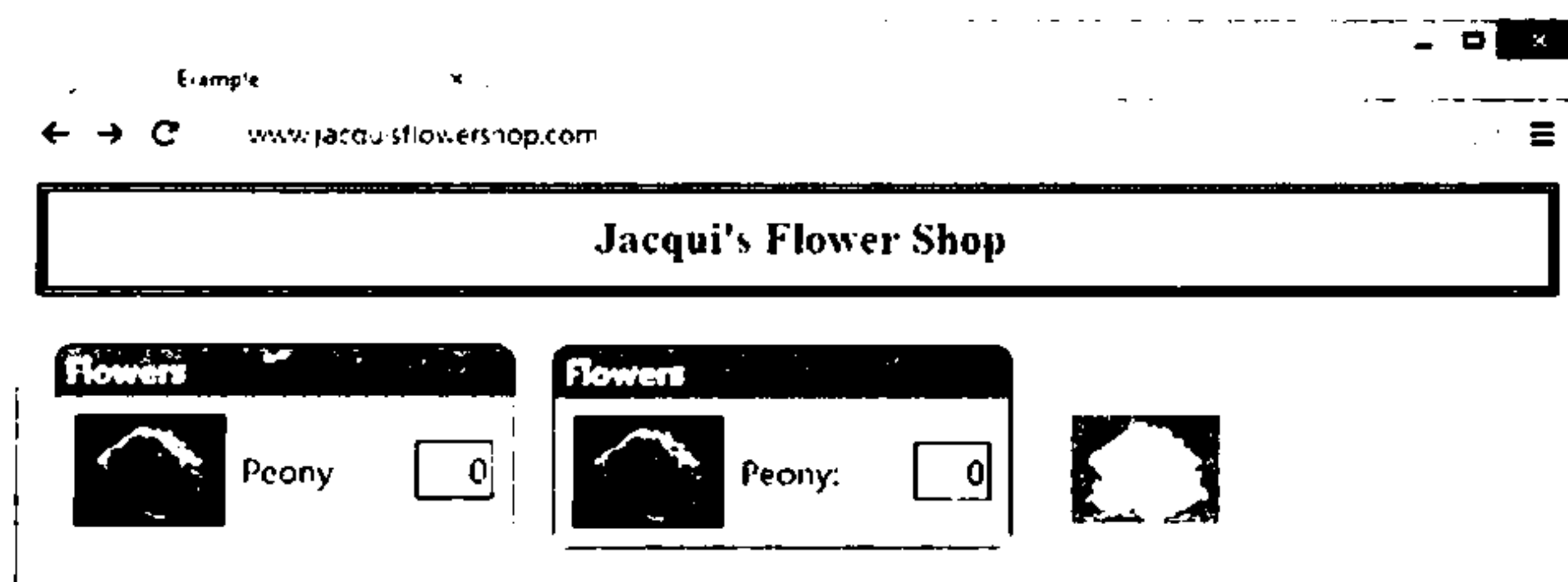


图35-6 ui-state-highlight类

代码清单35-10演示了错误提示类的用法。

#### 代码清单35-10 ui-state-error类

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    body > div {float: left; margin: 10px; padding: 20px}
  </style>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>

  <div class="ui-state-error">
    Oops! Something went wrong.
  </div>
</body>
</html>

```

ui-state-error类的效果如图35-7所示。



图35-7 ui-state-error类

### 35.3 小结

在本章中，我讲解了jQuery UI库对jQuery动画功能的增强：支持颜色、可见性及CSS类。尽管这些功能都很有用，我们还是应该谨慎使用它们，避免过多的特效对用户造成不必要的麻烦。我还讲解了jQuery UI样式框架中的关键类。利用这些类设置页面样式，我们不但能得到与jQuery UI一致的外观，而且即使有一天我们改进了jQuery UI主题，我们的页面样式也会随之自动改变。

贯穿本书，我们看到很多依赖回调函数的例子。我们事先定义好一个函数，然后等到事件发生时执行。回调函数的典型应用是事件处理，比如我们调用click方法，传入一个函数作为它的参数。该函数体的代码一直在休眠等待，直到用户触发click事件后才执行。

延迟对象（deferred）是jQuery术语，它定义了一种加强版的回调机制，帮助我们更好地使用回调函数。使用延迟对象，我们不必局限于事件处理，能够在任何场合使用回调函数。除这个特点之外，延迟对象还提供了许多用于控制回调时机及调用方法的回调选项。尽管并非必须，延迟对象经常用于后台任务。

在本章中，我从一个相当简单的例子开始，不断进行扩展，展示延迟对象的功能以及对管理延迟对象和后台任务非常有用的模式。

由于延迟对象的使用隶属异步编程（或者说并发编程）的领域，我承认这些例子相当简单。精通高效的并发编程本身就是一项很难掌握的技术，而JavaScript由于缺少一些其他语言（如Java和C#）先天具有的高级特性，因此更加困难。绝大多数项目都不需要延迟对象，如果你是初次接触并发编程，我建议你跳过本章，等你手头的项目需要这一功能时再来学习也不晚。

表36-1列出了本章概要。

表36-1 章节概要

问 题	解决方法	代码清单
如何使用延迟对象的基本功能	使用done方法注册一个回调函数，然后调用resolve方法触发执行	1
如何在后台任务中使用延迟对象	使用setTimeout函数生成一个后台任务，当任务结束时调用resolve方法	2~4
如何通知任务失败	使用reject方法触发fail回调函数	5、6
如何同时注册成功回调函数和失败回调函数	使用then方法	7
如何注册总会执行的回调函数（不论任务成功还是失败）	使用always方法	8
如何为同一出口定义多个回调函数	多次调用注册函数或者一次传入多个以逗号分隔的回调函数	9
如何生成一个出口依赖其他延迟对象的延迟对象	使用when方法	10
如何标识任务进度	调用notify方法，它会触发由progress方法注册的回调函数	11、12
如何得到延迟对象的状态	使用state方法	13
如何使用Ajax Promise	把jQuery Ajax系列方法的返回值视为延迟对象	14

## 36.1 延迟对象第一例

我先介绍延迟对象的工作原理，然后再教你延迟对象的使用。代码清单36-1是一个使用了延迟对象的简单例子。

代码清单36-1 一个延迟对象的简单例子

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    td {text-align: left; padding: 5px}
    table {width: 200px; border-collapse: collapse; width: 50%; float: left}
    #buttonDiv {width: 15%; text-align: center; margin: 20px; float: left}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {
      var def = $.Deferred();

      def.done(function() {
        displayMessage("Callback Executed");
      });

      $("button").button().click(function() {
        def.resolve();
      });
    });
    displayMessage("Ready");

    function displayMessage(msg) {
      $("tbody").append("<tr><td>" + msg + "</td></tr>");
    }
  </script>
</head>
<body>
  <h1>Jacqui's Flower Shop</h1>

  <table class="ui-widget" border=1>
    <thead class="ui-widget-header">
      <tr><th>Message</th></tr>
    </thead>
    <tbody class="ui-widget-content">
    </tbody>
  </table>

  <div id="buttonDiv">
    <button>Go</button>
  </div>
```



```
</body>
</html>
```

这个演示延迟对象工作原理的例子很简单。我会逐步讲解这个例子，为本章后面的内容热身。首先，执行`$.Deferred`方法，这个方法会生成一个延迟对象：

```
...
var def = $.Deferred();
...
```

`Deferred`方法返回一个延迟对象，并把这个对象赋值给变量`def`。延迟对象总是与回调函数密切相关，因此我们接下来使用延迟对象的`done`方法注册一个回调函数，如下：

```
...
def.done(function() {
    displayMessage("Callback Executed");
});
...
```

如果该回调函数得到执行机会，就会调用`displayMessage`函数，从而在页面的表格中增加一行。

最后一步，做一些安排以调用`resolve`方法，从而触发回调函数执行。这种触发回调函数执行的方式，又称为通知延迟对象任务完成。因为我们需要控制通知延迟对象的时机，所以我在页面中添加了一个按钮，并使用`click`方法为该按钮绑定了事件处理函数。在这里，我不得不用一种回调机制（事件回调）讲解另一种回调机制（延迟对象回调），这确实有点好笑。在本章中，我希望你忽略事件系统，把目光聚焦于这样一个事实：延迟对象直到点击按钮时才得到通知。下面的代码调用`resolve`方法，从而触发执行刚才使用`done`方法注册的回调函数。

```
...
$("button").button().click(function() {
    def.resolve();
});
...
```

在调用`resolve`方法之前，延迟对象处于任务尚未完成状态，因此回调函数不会执行。如图36-1所示，点击按钮，通知延迟对象任务完成，从而执行回调函数，并在表格中显示出相应的信息。

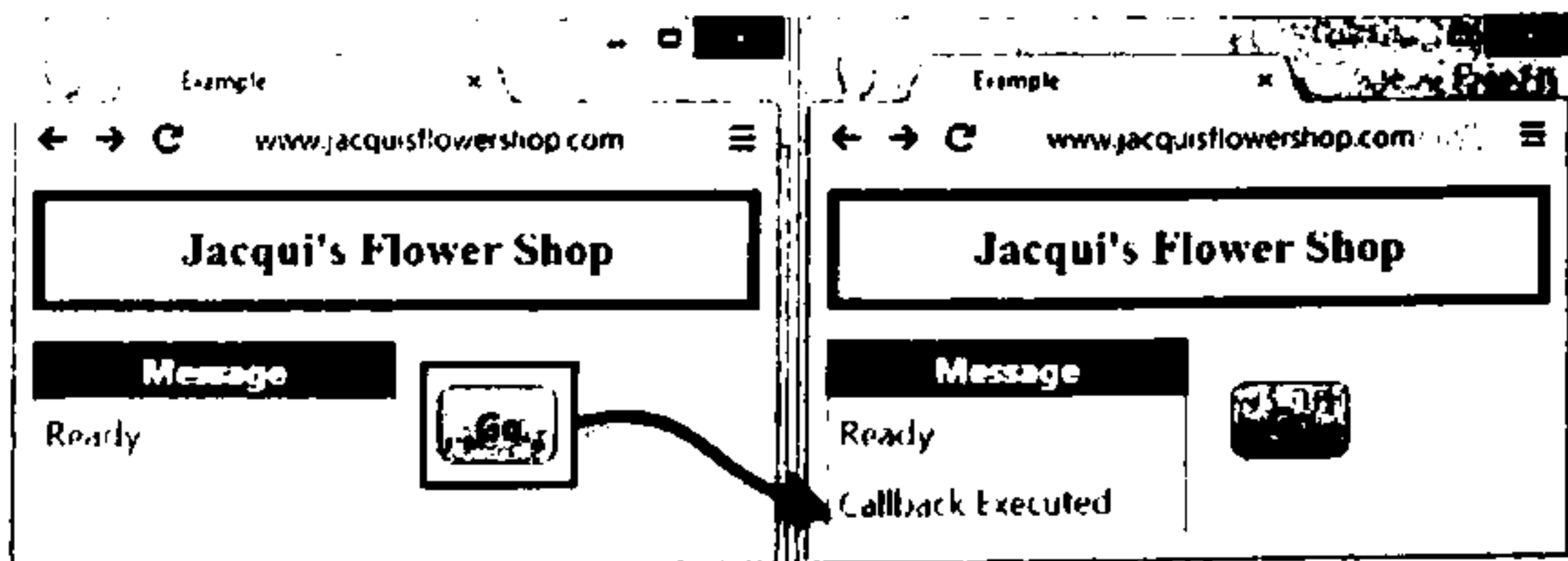


图36-1 通知延迟对象任务完成

有一点很重要，延迟对象在这个过程中没有做什么特别的事情。我们使用`done`方法注册回调函数，直到我们调用`resolve`方法时回调函数才会执行。在本例中，延迟对象直到点击按钮时才得到通知，进而触发回调函数执行，并把信息添加到表格元素。

## 延迟对象有什么用

如果我们不能直接监控任务，又希望在该任务结束时做一些别的事情，而且这个任务还是在后台执行，延迟对象就派上了用场。在代码清单36-2中，我修改了上一个例子并添加了一些功能。

代码清单36-2 在耗时任务中使用回调函数

```
...
<script type="text/javascript">
  $(document).ready(function() {
    var def = $.Deferred();

    def.done(function() {
      displayMessage("Callback Executed");
    });

    function performLongTask() {

      var start = $.now();

      var total = 0;
      for (var i = 0; i < 500000000 ; i++) {
        total += i;
      }
      var elapsedTime = (($.now() - start)/1000).toFixed(1);
      displayMessage("Task Complete. Time: " + elapsedTime + " sec");
      def.resolve();
    }

    $("button").button().click(function() {
      displayMessage("Calling performLongTask()");
      performLongTask();
      displayMessage("performLongTask() Returned");
    });
  });

  displayMessage("Ready");
  function displayMessage(msg) {
    $("tbody").append("<tr><td>" + msg + "</td></tr>");
  }
</script>
...
```

本例中定义的`performLongTask`函数，会不断地做加法运算。我需要一个任务执行好几秒钟，调用这个函数正合适。

---

**提示** 在我的电脑上，`performLongTask`函数中的这个for循环大约需要0.5秒钟执行完毕。你可能需要自己的机器上调整循环的上限，以便得到近似的结果。对本章中出现的例子来说，4到5秒钟最好。这个时间长度刚好适合演示延迟对象的功能，同时也没有久到非要去冲杯咖啡才能等到任务执行完毕。

---

现在点击按钮就会调用performLongTask函数，当该函数执行完毕时会调用延迟对象的resolve方法，从而启动预先定义好的回调函数。performLongTask函数在调用resolve方法之前先向表格中输出了任务完成信息，这样我们就能够看到整个脚本的执行进度。图36-2是该脚本的执行结果。

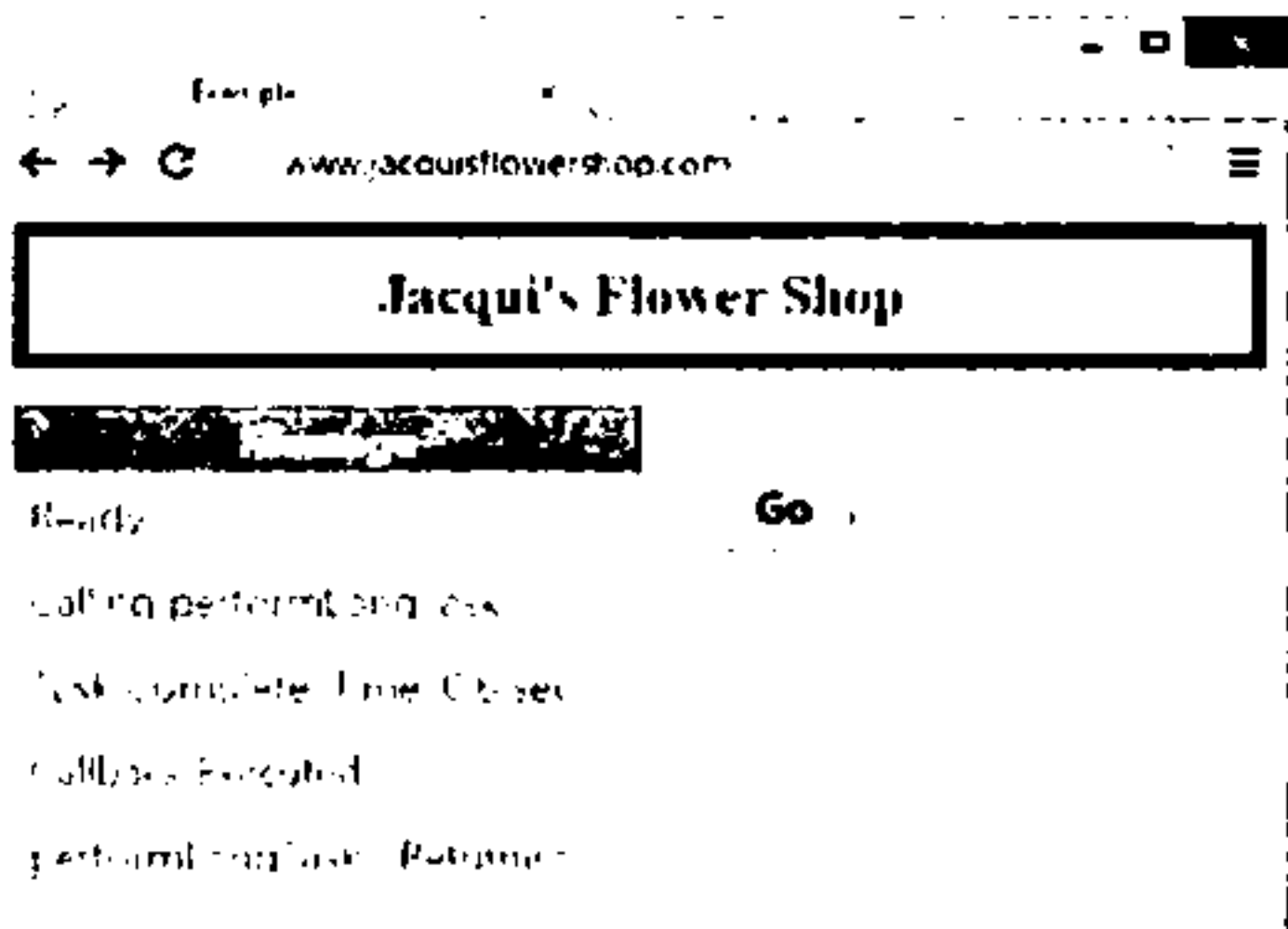


图36-2 使用延迟对象观察任务完成

这是一个同步任务的例子。点击按钮，然后等待函数执行结束。同步任务的最佳工作指示是Go按钮，在performLongTask结束之前，它会一直保持按下的状态。更强大的证据则是图36-2中显示的信息序列。这些信息先是来自click事件处理函数，接着来自performLongTask函数，最后来自回调函数。

延迟对象与异步任务配合的最大好处是，任务是在后台执行（因此不会阻塞界面）。我们不希望像前一个例子那样锁住用户界面，所以我们让任务在后台执行，同时密切关注着它们，随着任务的进展不断将信息显示给用户。

启动一个后台任务最简单的方法是使用setTimeout函数，也就是说我们要使用另一种回调机制。你也许觉得这样做有点怪异，由于JavaScript缺少其他语言天生具备的异步任务管理机制，我们不得不自己动手实现这些功能。代码清单36-3展示的是上面这个例子的修订版，让最耗时的部分，即performLongTask函数在后台运行。

### 代码清单36-3 让任务异步执行

```
...
<script type="text/javascript">
  $(document).ready(function() {
    var def = $.Deferred();

    def.done(function() {
      displayMessage("Callback Executed");
    });

    function performLongTask() {
      setTimeout(function() {
        var start = $.now();

        var total = 0;
        for (var i = 0; i < 500000000 ; i++) {
          total += i;
        }
      }, 3000);
    }
  });
</script>
```

```

        var elapsedTime = (($.now() - start)/1000).toFixed(1);
        displayMessage("Task Complete. Time: " + elapsedTime + " sec");
        def.resolve();
    }, 10);
}

$("button").button().click(function() {
    displayMessage("Calling performLongTask()")
    performLongTask()
    displayMessage("performLongTask() Returned")
})

displayMessage("Ready")
})

function displayMessage(msg) {
    $("tbody").append("<tr><td>" + msg + "</td></tr>")
}
</script>
...

```

我使用setTimeout函数让performLongTask任务在10毫秒之后执行，从图36-3中我们可以看到这个脚本的执行结果。注意，来自click事件处理函数的信息出现在performLongTask函数和回调函数的信息之前。如果你亲手运行这个例子，就会注意到按钮在按下后几乎立即就回到了初始状态，而不必等到任务完成。

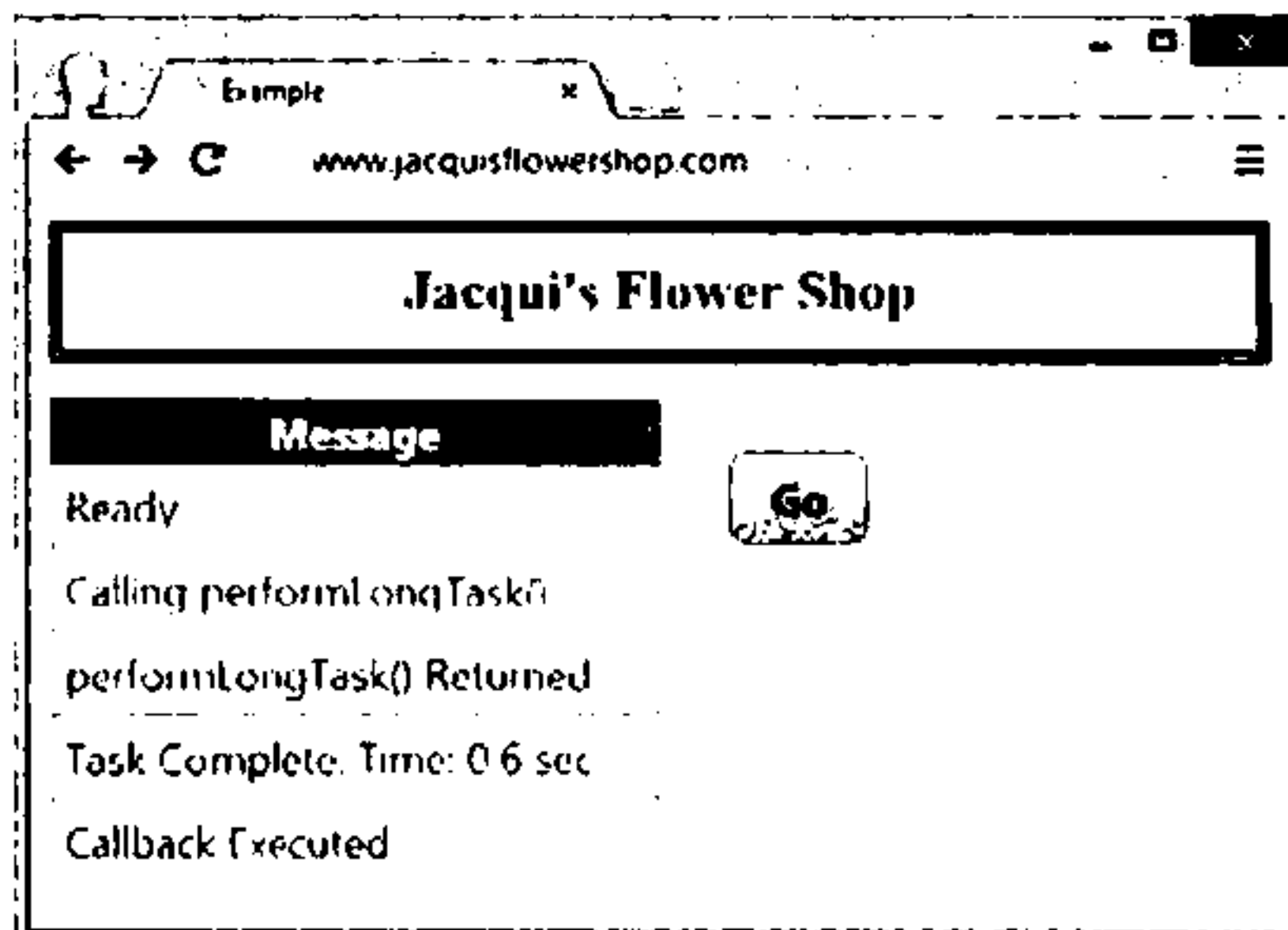


图36-3 后台执行任务

由于当执行后台任务时，我们不知道任务何时才会完成，在这种情形下回调函数机制就分外重要。我们自己实现一套信息更新机制，不过这意味着我们需要为每一个后台任务都实现一遍。不用说，这非常麻烦，而且容易出错。延迟对象允许我们使用一种标准机制指示任务已经完成，而且在后面的例子中，延迟对象还会提供大量灵活的选项来指导任务如何完成。

#### 整理这个例子

在我们深入了解延迟对象的功能之前，我会先修改这个例子，应用我在真实项目中所用到的模式。这纯属个人喜好，不过我喜欢把异步任务封装与将生成延迟对象的代码整合到函数的工作分开。代码清单36-4展示了例子修改后的版本。

## 代码清单36-4 整理后的代码

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    td {text-align: left; padding: 5px}
    table {width: 200px; border-collapse: collapse; float: left; width: 300px}
    #buttonDiv {text-align: center; margin: 20px; float: left}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      function performLongTaskSync() {
        var start = $.now();

        var total = 0;
        for (var i = 0; i < 500000000 ; i++) {
          total += i;
        }
        var elapsedTime = (($.now() - start)/1000).toFixed(1)
        displayMessage("Task Complete. Time: " + elapsedTime + " sec")
        return total;
      }

      function performLongTask() {
        return $.Deferred(function(def) {
          setTimeout(function() {
            performLongTaskSync();
            def.resolve();
          }, 10)
        });
      }

      $("button").button().click(function() {
        if ($(":checked").length > 0) {
          displayMessage("Calling performLongTask()")
          var observer = performLongTask();
          observer.done(function() {
            displayMessage("Callback Executed");
          });
          displayMessage("performLongTask() Returned")
        } else {
          displayMessage("Calling performLongTaskSync()")
          performLongTaskSync();
          displayMessage("performLongTaskSync() Returned")
        }
      });
    });
  </script>

```

```

        $(":checkbox").button();
        displayMessage("Ready")
        function displayMessage(msg) {
            $("tbody").append("<tr><td>" + msg + "</td></tr>")
        }
    </script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>
    <table class="ui-widget" border=1>
        <thead class="ui-widget-header">
            <tr><th>Message</th></tr>
        </thead>
        <tbody class="ui-widget-content">
        </tbody>
    </table>

    <div id="buttonDiv">
        <button>Go</button>
        <input type="checkbox" id="async" checked>
        <label for="async">Async</label>
    </div>
</body>
</html>

```

在这个例子中，我把一部分工作拆分到performLongTaskSync函数，它只负责计算，完全不了解后台任务或者回调函数。我喜欢拆分代码，因为在开发早期，这会让测试更容易进行。下面是计算函数的同步版本。

```

...
function performLongTaskSync() {
    var start = $.now();

    var total = 0;
    for (var i = 0; i < 500000000 ; i++) {
        total += i;
    }
    var elapsedTime = (($.now() - start)/1000).toFixed(1)
    displayMessage("Task Complete. Time: " + elapsedTime + " sec")
    return total;
}
...

```

我还分离出了执行异步任务的代码，也就是performLongTask函数。它是一个封装器，使用了延迟对象：当任务完成时触发回调函数。下面是修订后的performLongTask函数。

```

...
function performLongTask() {
    return $.Deferred(function(def) {
        setTimeout(function() {
            performLongTaskSync();
            def.resolve();
        }, 10);
    });
}

```

```

    });
}
...

```

如果调用Deferred方法时，它的参数是一个函数，该函数就会在延迟对象生成之后，以该对象（即刚生成的延迟对象）为参数立即执行。利用这一特性，我们能够实现一个简单的函数封装器，它异步执行任务并在任务完成后立即触发回调函数。

**提示** 如果你比较细心，就会注意到这中间有一个时机可以调用done方法注册回调函数。这样当任务完成后调用resolve方法就能触发这个回调。这个机会适用于一些短小的任务，不过，即便是在resolve方法之后调用done方法注册回调函数，回调函数仍然会执行。

我喜欢使用封装器的另一个原因是，延迟对象在得到通知任务完成或者任务失败之后，无法复原到原始状态。而在封装函数内生成延迟对象，就能保证我使用的总是一个“新鲜”的、尚未完成的延迟对象。

本例中还有一个变化，我添加了一个切换按钮，这样就能控制任务是同步执行还是异步执行。因为本章的主题是异步任务，因此我把这一功能提取了出来，把它用到后面的例子中，这也是处理变化的一种好方法。图36-4一并列出了两种执行方式的输出。

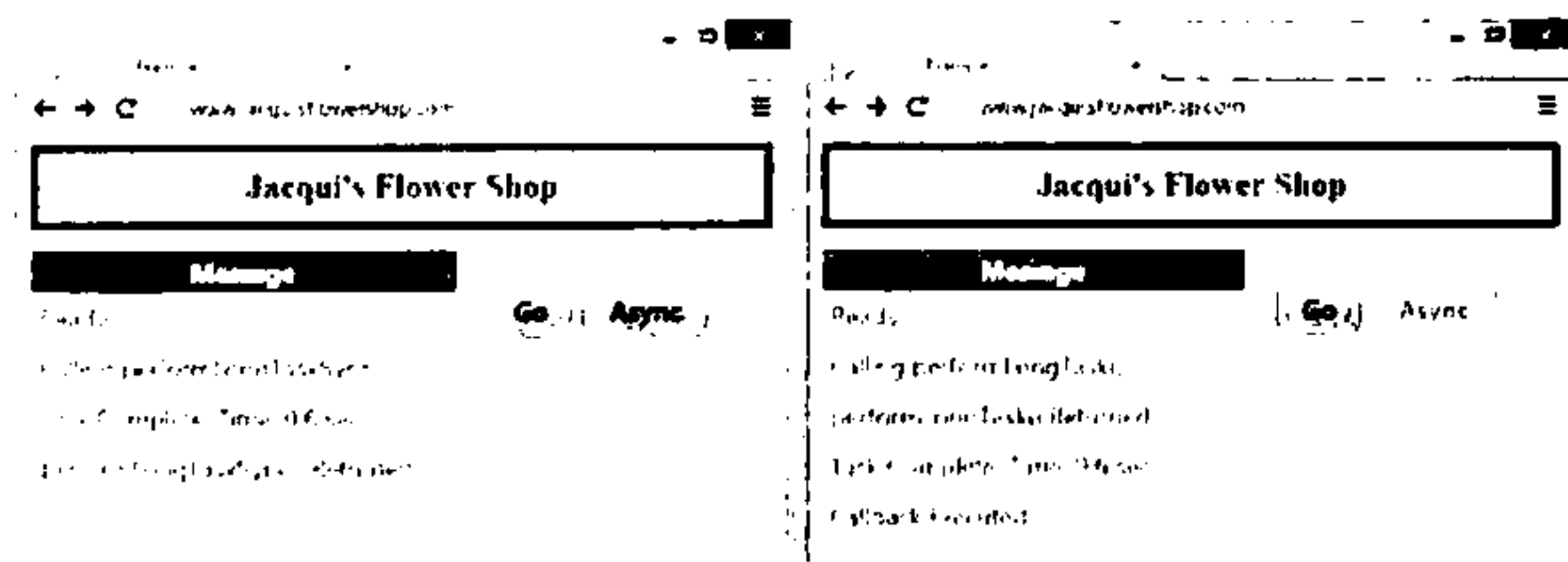


图36-4 以同步和异步方式执行同样的任务

## 36.2 其他回调

我们已经了解了一个简单异步任务的例子，接下来我们来研究延迟对象一些有用的特性。第一个特性是我们可以为任务标识不同的出口。表36-2列出了延迟对象提供的所有注册回调函数的方法以及触发相应的回调函数执行的方法。前面讲过了done方法和resolve方法，在后面的内容中，我将讲解另外几个方法。

表36-2 用来注册回调函数的方法

注册回调函数的方法	触发方法
done	resolve
fail	reject
always	resolve或reject

### 36.2.1 通知延迟对象任务失败

并非所有任务都能成功完成。若任务成功完成，我们会调用`resolve`方法通知延迟对象任务成功完成；若任务中间出了什么差错，我们就调用`reject`方法通知延迟对象任务失败（不执行`done`方法注册的回调函数）。我们可以用`fail`方法注册任务失败时需要执行的回调函数。就像`resolve`方法触发执行由`done`方法注册的回调函数，`reject`方法触发执行的是使用`fail`方法注册的回调函数。代码清单36-5展示了一个任务，它要么调用`resolve`方法，要么调用`reject`方法通知相应的延迟对象。

代码清单36-5 通知延迟对象任务失败

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    td {text-align: left; padding: 5px}
    table {width: 200px; border-collapse: collapse; float: left; width: 300px}
    #buttonDiv {text-align: center; margin: 20px; float: left}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      function performLongTaskSync() {
        var start = $.now();

        var total = 0;
        for (var i = 0; i < 5000000 ; i++) {
          total += (i + Number((Math.random() + 1).toFixed(0))));
        }
        var elapsedTime = (($.now() - start)/1000).toFixed(1)
        displayMessage("Task Complete. Time: " + elapsedTime + " sec")
        return total;
      }

      function performLongTask() {
        return $.Deferred(function(def) {
          setTimeout(function() {
            var total = performLongTaskSync();
            if (total % 2 == 0) {
              def.resolve(total);
            } else {
              def.reject(total);
            }
          }, 10);
        });
      }

      $("button").button().click(function() {
        displayMessage("Calling performLongTask()");
        var observer = performLongTask();
```



```

        displayMessage("performLongTask() Returned");
        observer.done(function(total) {
            displayMessage("Done Callback Executed: " + total);
        });
        observer.fail(function(total) {
            displayMessage("Fail Callback Executed: " + total);
        });
    })

    displayMessage("Ready");
})

function displayMessage(msg) {
    $("tbody").append("<tr><td>" + msg + "</td></tr>");
}
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>

    <table class="ui-widget" border=1>
        <thead class="ui-widget-header">
            <tr><th>Message</th></tr>
        </thead>
        <tbody class="ui-widget-content">
        </tbody>
    </table>

    <div id="buttonDiv">
        <button>Go</button>
    </div>
</body>
</html>

```

在这个例子中，我稍微调整了一下任务，在for循环中的每次迭代中，使用一个小随机数代替原来的迭代变量追加到total变量。异步封装函数performLongTask先检查同步版函数performLongTask Sync的返回值total，如果total是偶数，则通知延迟对象成功，否则通知延迟对象失败，也就是下面的代码：

```

...
if (total % 2 == 0) {
    def.resolve(total);
} else {
    def.reject(total);
}
...

```

调用performLongTask函数之后，我在click事件处理函数中分别使用done和fail方法为任务成功和任务失败注册了回调函数，也就是下面的代码：

```

...
var observer = performLongTask();
displayMessage("performLongTask() Returned");
observer.done(function(total) {
    displayMessage("Done Callback Executed: " + total);

```

```

});
observer.fail(function(total) {
    displayMessage("Fail Callback Executed: " + total);
});
...

```

注意调用`resolve`和`reject`方法时使用的参数。这些方法本身并不需要参数，不过如果你确实传递了参数给它们，它们就会把这些参数传递给相应的回调函数。这样我们就能告诉回调函数一些上下文或者细节信息。在本例中，任务成功还是失败取决于计算结果`total`的值，我把这个值作为参数传递给了`done`和`reject`方法。图36-5展示了任务成功和任务失败的两种输出。

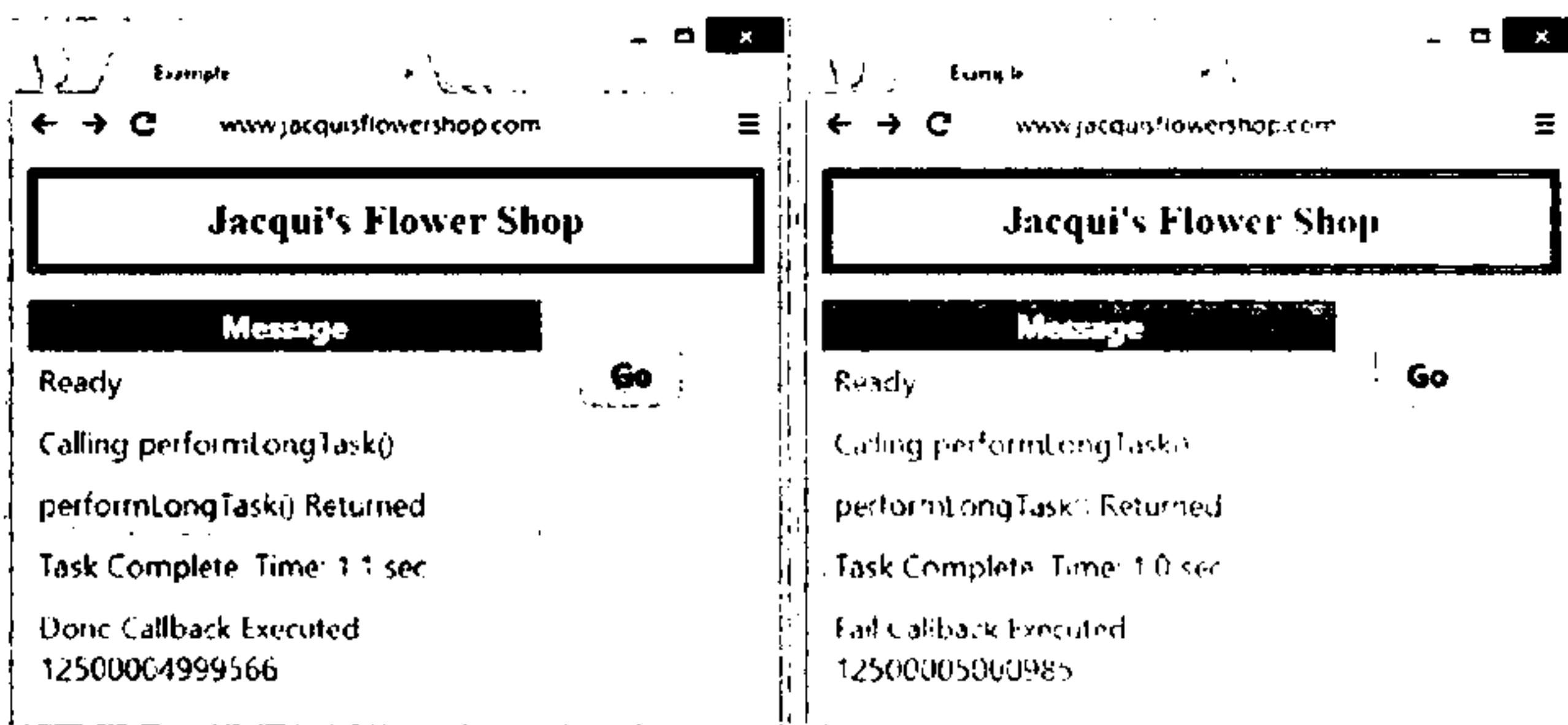


图36-5 一个有可能成功，也有可能失败的任务

### 链式调用延迟对象的方法

延迟对象的方法支持链式调用，这意味着每个方法都返回一个延迟对象，这样我们就能接着调用其他方法。这有点类似我在本书中一直实践的jQuery对象的方法。代码清单36-6展示了链式方式调用的`done`方法和`fail`方法。

#### 代码清单36-6 链式调用延迟对象的方法

```

...
$("button").button().click(function() {
    performLongTask().done(function(total) {
        displayMessage("Done Callback Executed: " + total);
    }).fail(function(total) {
        displayMessage("Fail Callback Executed: " + total);
    });
});
...

```

### 36.2.2 覆盖两个出口（任务成功和任务失败）

如果我们每个出口（成功或失败）都有回调函数需要执行，可以使用`then`方法一次注册两个回调函数。如果任务成功，就触发执行`then`方法的第一个参数，否则触发执行第二个参数。代码清单36-7演示了`then`方法的使用。

## 代码清单36-7 then方法

```

...
$("button").button().click(function() {
    displayMessage("Calling performLongTask()");
    var observer = performLongTask();
    displayMessage("performLongTask() Returned");

    observer.then(
        function(total) {
            displayMessage("Done Callback Executed");
        },
        function(total) {
            displayMessage("Fail Callback Executed");
        }
    );
});
...

```

与then方法相比，我更喜欢使用链式调用的方法注册回调函数。我发现，用链式调用为每个出口准备回调函数，代码更清晰明了。

## 36.2.3 与出口无关的回调函数

有时候，不论任务是否成功，我们总要执行一个回调函数。一个常见的应用场景是，使用always方法注册一个函数删除（或隐藏）标识后台任务正在执行的页面元素，然后使用done或fail方法注册一个回调函数，在任务完成时为用户显示下一步。代码清单36-8展示了always方法的用法，用always方法注册的回调函数，不论任务成功还是失败，都会执行。

## 代码清单36-8 使用always方法注册一个总会执行的回调函数

```

<!DOCTYPE html>
<html>
<head>
    <title>Example</title>
    <script src="jquery-2.0.2.js" type="text/javascript"></script>
    <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
    <style type="text/css">
        td {text-align: left; padding: 5px}
        table {width: 200px; border-collapse: collapse; float: left; width: 300px}
        #buttonDiv {text-align: center; margin: 20px; float: left}
    </style>
    <script type="text/javascript">
        $(document).ready(function() {

            function performLongTaskSync() {
                var start = $.now();

                var total = 0;
                for (var i = 0; i < 5000000 ; i++) {
                    total += (i + Number((Math.random() + 1).toFixed(0)));
                }
            }

```

```

        var elapsedTime = (($.now() - start)/1000).toFixed(1)
        displayMessage("Task Complete. Time: " + elapsedTime + " sec")
        return total;
    }

    function performLongTask() {
        return $.Deferred(function(def) {
            setTimeout(function() {
                var total = performLongTaskSync();
                if (total % 2 == 0) {
                    def.resolve(total);
                } else {
                    def.reject(total);
                }
            }, 10);
        });
    }

    $("button").button().click(function() {
        displayMessage("Calling performLongTask()");
        var observer = performLongTask();
        displayMessage("performLongTask() Returned");

        $("#dialog").dialog("open");

        observer.always(function() {
            $("#dialog").dialog("close");
        });

        observer.done(function(total) {
            displayMessage("Done Callback Executed: " + total);
        });

        observer.fail(function(total) {
            displayMessage("Fail Callback Executed: " + total);
        });
    });

    $("#dialog").dialog({
        autoOpen: false,
        modal: true
    });

    });
    displayMessage("Ready")
    function displayMessage(msg) {
        $("tbody").append("<tr><td>" + msg + "</td></tr>")
    }
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>

    <table class="ui-widget" border=1>

```

```

    <thead class="ui-widget-header">
      <tr><th>Message</th></tr>
    </thead>
    <tbody class="ui-widget-content">
    </tbody>
  </table>

  <div id="buttonDiv">
    <button>Go</button>
  </div>

  <div id="dialog">
    Performing Task...
  </div>
</body>
</html>

```

在这个例子中，在任务运行期间，我显示一个jQuery UI模态窗口。我用always方法注册了一个回调函数，这个回调函数负责在任务结束时关掉这个模态窗口。这样一来，我不必在任务成功和失败的出口中重复关闭窗口的代码，也一样能达到目标。

---

**提示** 回调函数的执行顺序与它们注册的顺序一致。在本例中，我先调用了always方法，后调用done或fail方法。这意味着always回调函数总是先于任务成功或失败的回调函数执行。

---

## 36.3 使用多个回调函数

使用延迟对象的一个好处，是帮助我们吧代码拆分成专门处理某种情况的小函数。除此之外，延迟对象还提供了为同一出口注册多个回调函数的机制，这进一步方便了我们分解代码。代码清单36-9就是一个这样的例子。

**代码清单36-9** 使用延迟对象注册多个回调函数

```

...
<script type="text/javascript">
  $(document).ready(function() {

    function performLongTaskSync() {
      var start = $.now();

      var total = 0;
      for (var i = 0; i < 5000000; i++) {
        total += (i + Number((Math.random() + 1).toFixed(0)));
      }
      var elapsedTime = (($.now() - start)/1000).toFixed(1)
      displayMessage("Task Complete. Time: " + elapsedTime + " sec")
      return total;
    }

    function performLongTask() {
      return $.Deferred(function(def) {
        setTimeout(function() {

```

```

        var total = performLongTaskSync();
        if (total % 2 == 0) {
            def.resolve({
                total: total
            });
        } else {
            def.reject(total);
        }
    }, 10);
});
}

$("#button").button().click(function() {
    displayMessage("Calling performLongTask()");
    var observer = performLongTask();
    displayMessage("performLongTask() Returned");
    $("#dialog").dialog("open");

    observer.done(function(data) {
        data.touched = 1;
        displayMessage("1st Done Callback Executed");
    });

    observer.done(function(data) {
        data.touched++;
        displayMessage("2nd Done Callback Executed");
    }, function(data) {
        data.touched++;
        displayMessage("3rd Done Callback Executed");
    });

    observer.done(function(data) {
        displayMessage("4th Done Callback Executed: " + data.touched);
    });

    observer.fail(function(total) {
        displayMessage("Fail Callback Executed: " + total);
    });

    observer.always(function() {
        displayMessage("Always Callback Executed");
        $("#dialog").dialog("close");
    });
});

$("#dialog").dialog({
    autoOpen: false,
    modal: true
});

displayMessage("Ready");
});

function displayMessage(msg) {
    $("#tbody").append("<tr><td>" + msg + "</td></tr>")
}

```

```
</script>
...
```

在这个例子中，我使用done方法注册了四个回调函数。如代码所示，我们可以使用注册函数一个一个地注册，也可以一批一批地注册（使用逗号分隔多个函数）。延迟对象会保证这些回调函数总是按照注册的顺序执行。

注意，在这个例子中我传递给resolve方法的参数变成了一个对象，把计算结果保存到这个JavaScript对象的total属性。之所以这样，是为了演示回调函数也能修改通过延迟对象传递的数据。如果需要在回调函数之间进行简单的通信（比如通告某个行为/事件已经发生），这个特性就很有用。多回调函数产生的效果见图36-6。

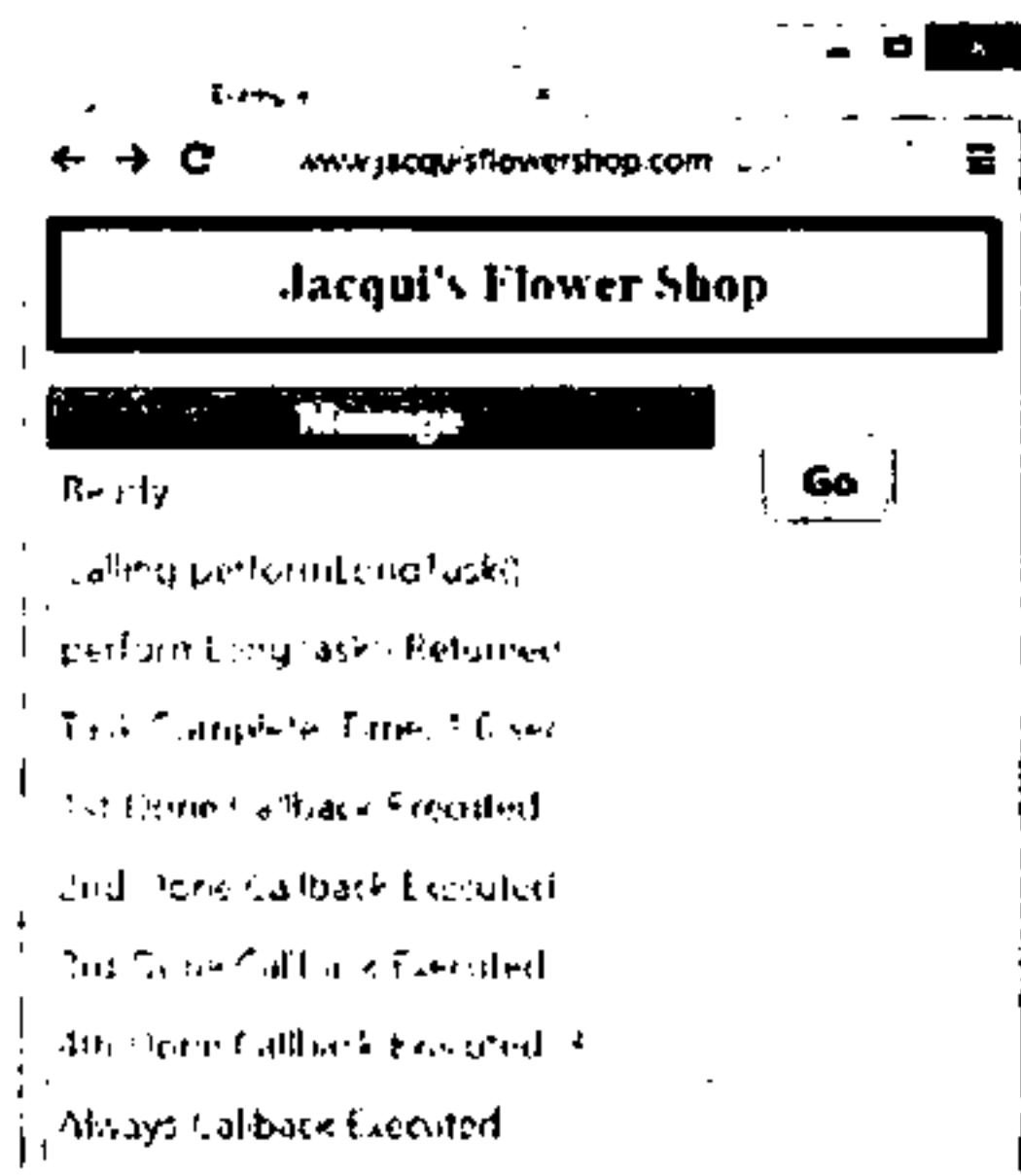


图36-6 同一出口注册多个回调函数

**提示** 我们也可以使用then方法为每个出口指定多个回调函数（以函数数组作为参数）。

## 36.4 利用多个延迟对象的出口

使用when方法能够得到出口依赖其他延迟对象的延迟对象。若你的后续行为依赖几个背景任务的结果，或者需要确保其他几个任务达到指定目标时才会进行下一步，这时就需要这一技术。代码清单36-10提供了一个示例。

代码清单36-10 when方法

```
...
$("button").button().click(function() {
    var ob1 = performLongTask()
    .done(function() {
        displayMessage("1st Task <b>Resolved</b>")
    })
    .fail(function() {
        displayMessage("1st Task <b>Failed</b>")
    })
})
```

```

    });

    var ob2 = performLongTask()
      .done(function() {
        displayMessage("2nd Task <b>Resolved</b>")
      })
      .fail(function() {
        displayMessage("2nd Task <b>Failed</b>")
      });

    var ob3 = performLongTask()
      .done(function() {
        displayMessage("3rd Task <b>Resolved</b>")
      })
      .fail(function() {
        displayMessage("3rd Task <b>Failed</b>")
      });

    $.when(ob1, ob2, ob3)
      .done(function() {
        displayMessage("Aggregate <b>Resolved</b>")
      })
      .fail(function() {
        displayMessage("Aggregate <b>Failed</b>")
      });
  });
  ...

```

在这个例子中，我生成了三个延迟对象，这些延迟对象都是performLongTask函数的返回值，并且都使用了done和fail方法注册了相应的回调函数。

我把这三个延迟对象作为when方法的参数，从而得到又一个延迟对象（该对象又称为整体延迟对象）。我接着使用done和fail方法为这个整体延迟对象注册回调函数。这个整体延迟对象的出口由其他三个延迟对象的出口决定。如果所有这三个普通延迟对象都得到通知说任务成功，则整体延迟对象就会成功，done方法绑定的回调函数就会触发执行。然而，只要有一个普通延迟对象得到任务失败的通知，整体延迟对象就会失败。图36-7列出了整体延迟对象的两种出口结果。

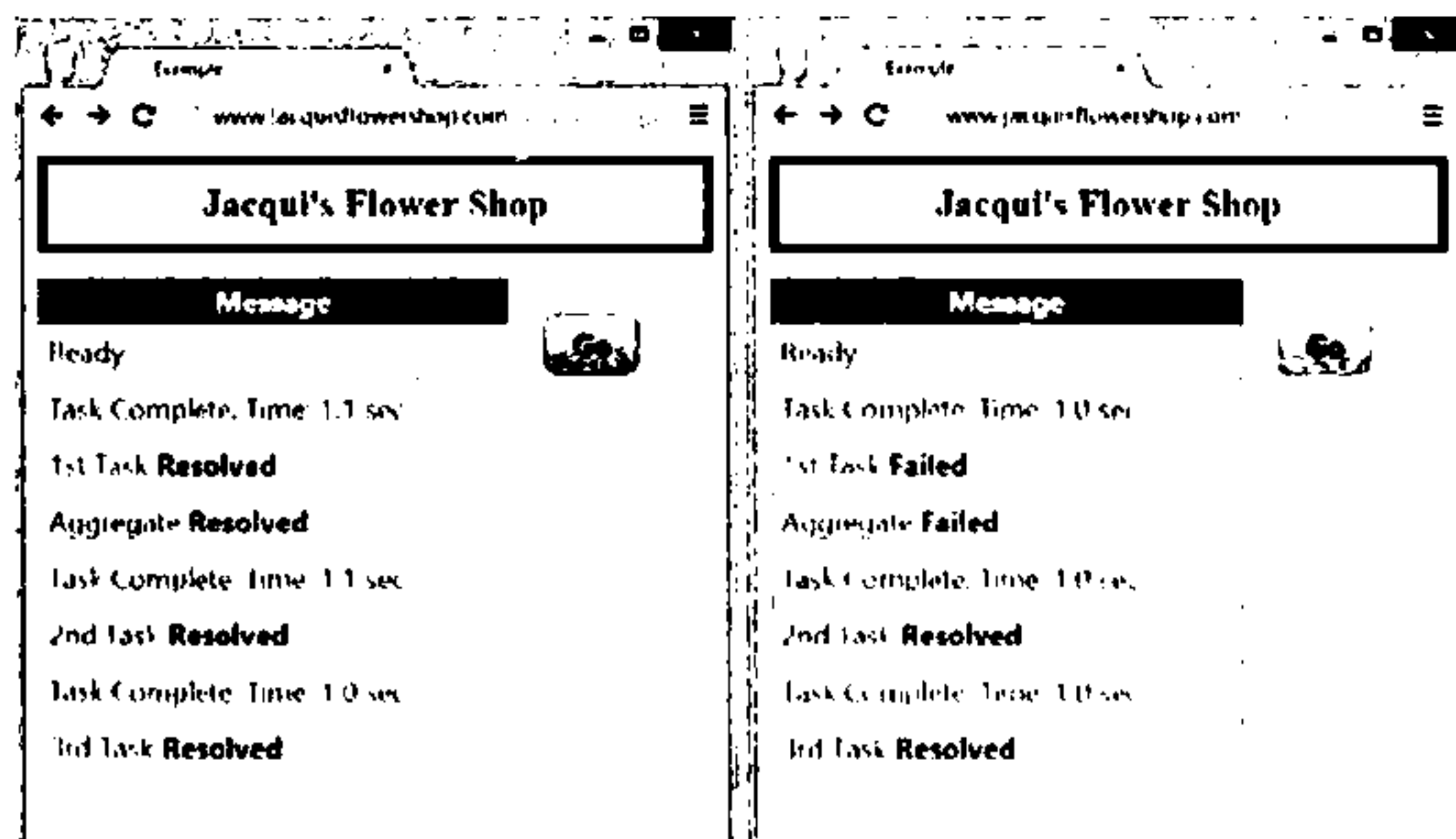


图36-7 when方法



**警告** 如果你认真观察上图，就会注意到一个反常的时间点。只要底层任意一个延迟对象得到通知说任务失败，整体延迟对象就立即失败了。这意味着用整体延迟对象的fail方法注册的回调函数很可能在有的任务尚未完成时就触发执行。当我们在处理一个得到通知说任务失败的延迟对象时，一定不要假定与它有关的所有任务都已经完成。

## 36.5 提供进度信息

如果正在执行一个耗时的后台任务，最好能为用户提供任务进度信息。延迟对象可以用来在任务和回调函数之间传递进度信息，大体上说，这和我们已经做过的传递信息给常规的出口回调函数没什么不同。我们通过notify方法制造进度信息，使用progress方法注册回调函数（代码清单36-11）。

代码清单36-11 利用延迟对象生成并展示进度信息

```
...
<script type="text/javascript">
  $(document).ready(function() {

    function performLongTaskSync() {
      var total = 0;
      for (var i = 0; i < 5000000 ; i++) {
        total += (i + Number((Math.random() + 1).toFixed(0)));
      }
      return total;
    }

    function performLongTask() {
      return $.Deferred(function(def) {
        setTimeout(function() {
          var progressValue = 0;
          for (var i = 0; i < 4; i++) {
            performLongTaskSync();
            progressValue += 25;
            def.notify(progressValue)
          }
          def.resolve();
        }, 10);
      });
    }

    $("button").button().click(function() {

      performLongTask().progress(function(val) {
        displayMessage("Progress: " + val + "%")
      }).done(function() {
        displayMessage("Task Resolved");
      });
    });

    $("#dialog").dialog({
      autoOpen: false,
```

```

        modal: true
    })

    displayMessage("Ready");
})

function displayMessage(msg) {
    $("tbody").append("<tr><td>" + msg + "</td></tr>")
}
</script>
...

```

在这个例子中，任务被执行了四次。每执行完一次，我让延迟对象调用一次notify方法，并把进度信息作为参数传递给它（你尽可以传递任意对象或者对你的程序有意义的值，简单起见，我在这里使用了百分比）。在click事件处理函数中，我使用progress方法注册了一个回调函数，它会在进度更新时自动调用。我利用这个函数往页面的表格中添加了一条信息。

这个例子演示了提供进度信息的基本能力，然而它并非以你希望的方式工作。问题在于浏览器只能等到四次循环结束之后才能更新DOM，并往表格中添加新行。这是由JavaScript处理任务的特点决定的，我们只能等到任务结束才能一次拿到所有的进度更新信息。为了解决这个问题，我们需要在任务的每一步增加一点小的延时，以便给浏览器以时间处理更新。代码清单36-12演示了如何使用setTimeout函数引入这些延迟并生成一个延迟对象链。我一般都是用for循环设置这些延时和延迟对象，不过在这个例子中，为清晰起见，我明确定义了所有的步骤。

#### 代码清单36-12 为了使DOM能够及时更新而分解任务

```

...
<script type="text/javascript">
    $(document).ready(function() {

        function performLongTaskSync() {
            var total = 0;
            for (var i = 0; i < 5000000 ; i++) {
                total += (i + Number((Math.random() + 1).toFixed(0)));
            }
            return total;
        }

        function performLongTask() {

            function doSingleIteration() {
                return $.Deferred(function(innerDef) {
                    setTimeout(function() {
                        performLongTaskSync();
                        innerDef.resolve();
                    }, 10);
                });
            }

            var def = $.Deferred();

            setTimeout(function() {
                doSingleIteration().done(function() {
                    def.notify(25);
                });
            }, 10);
        }
    });

```

```

        doSingleIteration().done(function() {
            def.notify(50);
            doSingleIteration().done(function() {
                def.notify(75);
                doSingleIteration().done(function() {
                    def.notify(100);
                    def.resolve();
                });
            });
        });
    }, 10);

    return def;
}

$("#button").button().click(function() {

    performLongTask().progress(function(val) {
        displayMessage("Progress: " + val + "%")
    }).done(function() {
        displayMessage("Task Resolved");
    });

    $("#dialog").dialog({
        autoOpen: false,
        modal: true
    })

    displayMessage("Ready");
});

function displayMessage(msg) {
    $("tbody").append("<tr><td>" + msg + "</td></tr>")
}
</script>
...

```

经过这番改造，就能够正常显示进度信息了（图36-8）。

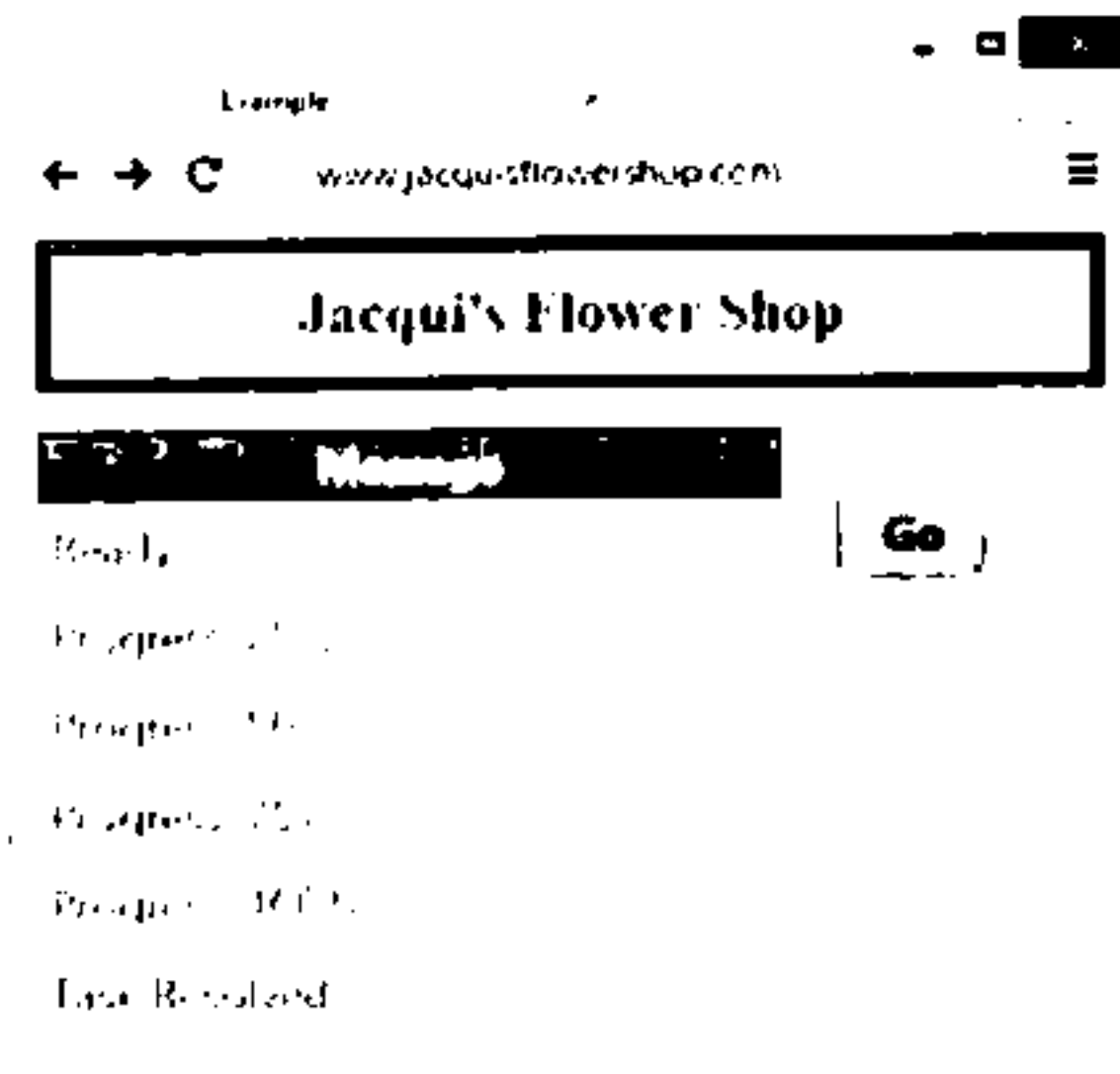


图36-8 使用延迟对象获取进度信息

## 36.6 获取延迟对象的状态

延迟对象定义了state方法，我们可以使用这个方法查询延迟对象的状态，并获知当前执行的任务。这个方法的返回值含义见表36-3。

表36-3 state对象可取的值

值	描 述
pending	延迟对象的resolve和reject方法都还没有调用过
resolved	已使用resolve方法明确通知延迟对象任务成功完成
rejected	已使用reject方法明确通知延迟对象任务失败

**警告** 要小心使用这个方法，尤其当你发现自己正在轮询一个延迟对象的状态时，请停下来想一想：这个Web应用的设计是不是有什么问题？轮询状态，特别是在while或for循环中轮询，不但增加了异步任务的开销和复杂度，而且从本质上讲，你的异步任务其实是在同步执行。

只有一次我发现state方法有用。那一次我使用always方法注册了一个回调函数，并且我想知道任务最后的输出结果。通常我都是使用done或fail方法分别定义回调函数，然而有时候我需要在任务结束后执行大量的代码，并且任务成功或失败时要执行的代码大同小异。代码清单36-13给出了一个使用state方法的例子。

### 代码清单36-13 state方法

```
<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    td {text-align: left; padding: 5px}
    table {width: 200px; border-collapse: collapse; float: left; width: 300px}
    #buttonDiv {text-align: center; margin: 20px; float: left}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      function performLongTaskSync() {
        var start = $.now();

        var total = 0;
        for (var i = 0; i < 5000000 ; i++) {
          total += (i + Number((Math.random() + 1).toFixed(0)));
        }
        var elapsedTime = (($.now() - start)/1000).toFixed(1);
        displayMessage("Task Complete. Time: " + elapsedTime + " sec");
      }
    });
  </script>
</head>
<body>
  <div id="buttonDiv">
    <button type="button" value="Perform Long Task Sync">Perform Long Task Sync
  </div>
</body>
</html>
```

```

    }

    function performLongTask() {
        return $.Deferred(function(def) {
            setTimeout(function() {
                performLongTaskSync();
                def.resolve();
            }, 10);
        });
    }

    $("button").button().click(function() {
        displayMessage("Calling performLongTask()");
        var observer = performLongTask().done(function() {
            displayMessage("Task complete");
        });
        displayMessage("performLongTask() Returned");

        displayMessage("Calling getJSON()");
        var ajaxPromise = $.getJSON("mydata.json").done(function() {
            displayMessage("Ajax Request Completed");
        });
        displayMessage("getJSON() Returned");

        $.when(observer, ajaxPromise).done(function() {
            displayMessage("All Done");
        });
    });
    displayMessage("Ready");
});

function displayMessage(msg) {
    $("tbody").append("<tr><td>" + msg + "</td></tr>");
}
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>

    <table class="ui-widget" border=1>
        <thead class="ui-widget-header">
            <tr><th>Message</th></tr>
        </thead>
        <tbody class="ui-widget-content">
        </tbody>
    </table>

    <div id="buttonDiv">
        <button>Go</button>
    </div>
</body>
</html>

```

```

<table class="ui-widget" border=1>
  <thead class="ui-widget-header">
    <tr><th>Message</th></tr>
  </thead>
  <tbody class="ui-widget-content">
    </tbody>
</table>
<div id="buttonDiv">
  <button>Go</button>
</div>
<div id="dialog">
  Performing Task...
</div>
</body>
</html>

```

## 36.7 Ajax 延迟对象

延迟对象已经整合到jQuery的Ajax功能当中（见第14章和15章），这也许是延迟对象功能中最有用的部分。ajax和getJSON方法返回的jxXHR对象已经实现了Promise接口，这个实现是常规延迟对象方法集合的一个子集。Promise接口定义了done、fail、then和always方法，并且可以与when方法配合使用。代码清单36-14展示了Ajax promise接口和延迟对象如何配合使用。

代码清单36-14 Ajax Promise与延迟对象

```

<!DOCTYPE html>
<html>
<head>
  <title>Example</title>
  <script src="jquery-2.0.2.js" type="text/javascript"></script>
  <script src="jquery-ui-1.10.3.custom.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.css"/>
  <style type="text/css">
    td {text-align: left; padding: 5px}
    table {width: 200px; border-collapse: collapse; float: left; width: 300px}
    #buttonDiv {text-align: center; margin: 20px; float: left}
  </style>
  <script type="text/javascript">
    $(document).ready(function() {

      function performLongTaskSync() {
        var start = $.now();

        var total = 0;
        for (var i = 0; i < 5000000 ; i++) {
          total += (i + Number((Math.random() + 1).toFixed(0)));
        }
        var elapsedTime = (($.now() - start)/1000).toFixed(1)
        displayMessage("Task Complete. Time: " + elapsedTime + " sec")
        return total;
      }
    });
  </script>

```

```

    }

    function performLongTask() {
        return $.Deferred(function(def) {
            setTimeout(function() {
                performLongTaskSync();
                def.resolve();
            }, 10);
        });
    }

    $("button").button().click(function() {
        displayMessage("Calling performLongTask()");
        var observer = performLongTask().done(function() {
            displayMessage("Task complete");
        });
        displayMessage("performLongTask() Returned");

        displayMessage("Calling getJSON()");
        var ajaxPromise = $.getJSON("mydata.json").done(function() {
            displayMessage("Ajax Request Completed");
        });
        displayMessage("getJSON() Returned");

        $.when(observer, ajaxPromise).done(function() {
            displayMessage("All Done");
        });
    });
    displayMessage("Ready");
});

function displayMessage(msg) {
    $("tbody").append("<tr><td>" + msg + "</td></tr>");
}
</script>
</head>
<body>
    <h1>Jacqui's Flower Shop</h1>

    <table class="ui-widget" border=1>
        <thead class="ui-widget-header">
            <tr><th>Message</th></tr>
        </thead>
        <tbody class="ui-widget-content">
        </tbody>
    </table>

    <div id="buttonDiv">
        <button>Go</button>
    </div>
</body>
</html>

```

**提示** 在一个延迟对象上调用`promise`方法，可以生成我们自己的`Promise`对象。如果你正在写一个JavaScript库，并且希望别的程序员只能注册回调函数而不能调用`resolve`或`reject`方法，就可以这么做。

在这个例子中，我调用了`getJSON`方法，并把它的返回值`ajaxPromise`视为延迟对象。我使用`done`方法注册了一个回调函数，并把`ajaxPromise`作为`when`方法的参数。这个例子的结果见图36-9。

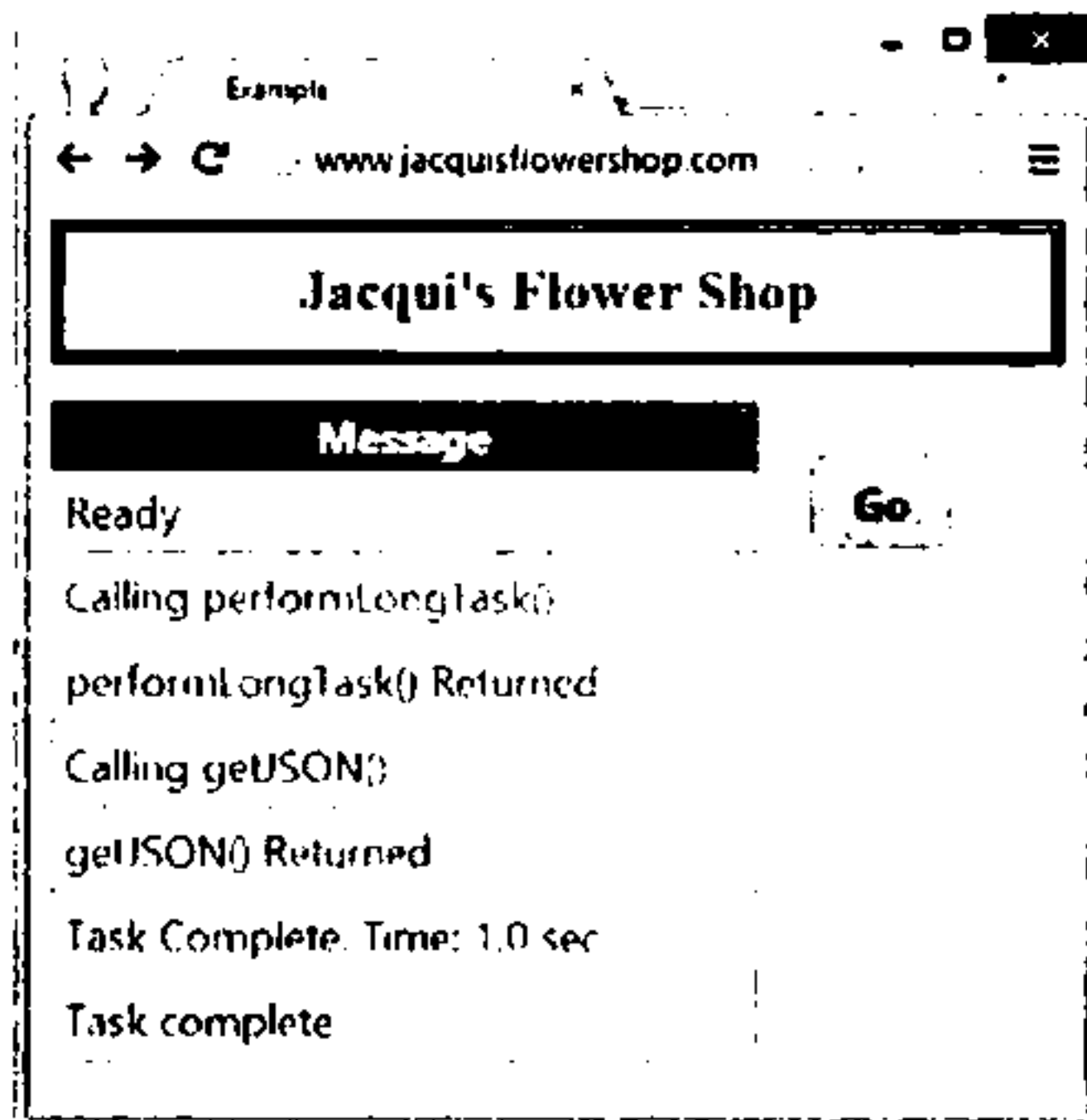


图36-9 Ajax promise对象的使用法

## 36.8 小结

在本章中，我演示了jQuery延迟对象的使用法，包括用它标识任务进度，以及任务的完成状态（出口），这都是些典型的后台任务。jQuery的Ajax组件也使用了延迟对象，这样我们就能够使用一致的方式处理Ajax请求和后台任务。延迟对象是高级技术，很可能在绝大多数Web应用中都用不到它，然而对那些用到重要后台任务的项目来说，利用这项技术我们能有效地实现及时响应，从而改善用户体验。