FLEX 3 IN ACTION

Flex 3 实战

(美) Tariq Ahmed
Jon Hirschi
Faisal Abid 著

李松峰 译

MANNING

清华大学出版社

Flex 3 实战

FLEX 3 IN ACTION

Adobe Flex 3是一个完美的富Web开发平台，它将易用的开发环境Flex Builder与基于JavaScript的典雅编程模型结合在一起。Flex 3提供企业质量的数据以及与Java、PHP和Rails集成的服务器组件。目前，Flex的主要组件是供免费使用的开源软件！

本书首先简要概述Flex 3和ActionScript，然后以十分友好、务实的方式介绍Flex的每个主要组件。其中穿插的示例编排合理，注解清晰，引导你重点关注与每个标记或类相关的主要属性、方法和事件。本书还浓墨重彩地描述Flex中的事件，给出调试事件逻辑的提示。学习本书不要求你拥有使用Flex的经验；但如果你拥有JavaScript或ActionScript知识，那么学习起来将更加轻松。

本书主要内容

- 如何利用现有的Web开发技能
- 交互式表单、拖放和数据驱动的功能
- Flex的分析程序、AdvancedDataGrid和重构等新功能
- 如何在多个项目中共享代码

作者简介

Tariq Ahmed是美国Amcom Technology公司的产品开发经理，因创建Flex社区站点CFLEX.Net而声名远扬。Jon Hirschi是Adobe Flex社区专家，与Tariq携手在eBay启动了多个基于Flex的项目。Faisal Abid曾在Buzzspot和RazorCom公司任职。

“示例代码是本书的一大亮点，几乎每个知识点都穿插了足以满足需要的示例代码。”

——Andrew Grothe
Triware Technologies, Inc.

“本书简明易懂，是新手的指路明灯；本书内容丰富，将指引有一定基础的读者走进丰富多彩的编程世界。”

——Ken Brueck
Move Networks

“这本精彩的教程和参考书使我手不释卷、如醉如痴。”

——Christophe Bunn
Kitry S.A.S.

“将广博的Flex内容浓缩在一卷书中。”

——Charlie Griefer
Amcom Technology

ISBN 978-7-302-21892-0
9 787302 218920 >
定价：68.00元

图书分类：Web开发/Flex
读者信箱：wkservice@vip.163.com
信息网站：http://www.manning.com/ahmed
http://www.tupwk.com.cn/downpage

MANNING

Flex 3 实战

Tariq Ahmed
(美) Jon Hirschi 著
Faisal Abid
李松峰 译

清华大学出版社
北 京

EISBN: 978-1933988740
FLEX 3 IN ACTION
Tariq Ahmed, Jon Hirschi, Faisal Abid
Original English language edition published by Manning Publications, 178 South Hill Drive, Westampton, NJ 08060 USA. Copyright(c)2009 by Manning Publications. Simplified Chinese-language edition copyright (c) 2010 by Tsinghua University Press. All rights reserved.

本书中文简体字版由 Manning 出版公司授权清华大学出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。
版权所有，侵权必究。

北京市版权局著作权合同登记号　图字：　01-2009-4887

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

Flex 3 实战/(美)艾哈迈德(Ahmed, T.)，(美)赫希(Hirschi, J.)，(美)阿比德(Abid, F.)著；李松峰 译.
—北京：清华大学出版社，2010.3
书名原文：FLEX 3 IN ACTION
ISBN 978-7-302-21892-0
I. F…　II. ①艾…　②赫…　③阿…　④李…　III. 软件工具—程序设计　IV. TP311.56
中国版本图书馆 CIP 数据核字(2010)第 014052 号

责任编辑：王　军　韩宏志
装帧设计：孔祥丰
责任校对：成凤进
责任印制：何　芊
出版发行：清华大学出版社　　**地　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编**：100084
社 总 机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京富博印刷有限公司
装 订 者：北京市密云县京文制本装订厂
经　销：全国新华书店
开　本：185×260　**印　张**：32.75　**字　数**：803 千字
版　次：2010 年 3 月第 1 版　　**印　次**：2010 年 3 月第1 次印刷
印　数：1～4000
定　价：68.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：(010)62770177 转 3103　　产品编号：033731-01

序　言

创建 RIA(Rich Internet Application，富因特网应用程序)的有关技术早就出现了，而此后很久才出现术语 RIA，Flash Platform 从 v8 开始成为可行的 RIA 平台。然而，Flash Platform 的初衷是为设计人员提供一个能够发挥创意的环境，并不适合专业开发人员使用。

如果你是一名 Flash 应用程序开发人员，那么仍然需要跨越巨大的障碍。以高效及时的方式交付功能众多的富应用程序就是一项严峻的挑战，因为你必须亲手完成绝大部分工作。

问题其实清晰明了：最好是有一个纯粹的开发环境，它既能利用业已受到广泛支持、性能优越的 Flash，又能直接满足开发人员及其系统开发生命周期的需要。要为开发人员提供高效工具，再配备一个能替他们完成大多数劳神费力工作的框架，从而让他们重点关注应用程序的逻辑。这个问题的解决方案就是 Flex。

Flex 于 2004 年 3 月由 Macromedia 发布，此后历经了多年的发展。在发布第 1 版之后的同年晚些时候发布的 1.5 版取得了更好的销售业绩。Flex 最早是作为一种服务器产品开发的，广告里也把它作为在 Flash Platform 上创建应用程序的一种工具来宣传。

这个工具的推出开启了一个新时代，也点燃了 RIA 的火种。随着基于 Flex 的 RIA 日益受到人们的关注，事实已经证明确实存在对 RIA 的需求，而这种技术手段赖以发展的基础也已具备。下一步就是推进 RIA 的规模化应用。

后来，在成为 Adobe 产品后，这个产品以 2006 年 6 月发布的 Flex 2 为标志取得了巨大的飞跃。凭借对语言的重新实现(ActionScript 3)和对 Flash Player(V9)的重要升级，产品的性能得到极大的提升。

而且，开发工具转换到 Eclipse 平台，也坚定了程序员的信心。Flex 2 关注的是基本的编译与服务器端数据管理的分离问题。我们也看到了首次以免费方式发布的用于在 IDE 以外编译 Flex 应用程序的 SDK。最终，服务器端组件发展成为 Flex Data Services，即目前的 LiveCycle Data Services(该服务支持实时数据共享)。

这些改变都有助于证明 RIA 思想的合理性，即在浏览器中提供一种桌面应用程序般的体验。而此时，RIA 最需要的就是获得社区支持。

Tariq 发现了社区支持的重要性，并走在了前列。为此，他创建了第一个专注于 Flex 的门户和社区，即 Community Flex(CFLEX.Net)。

CFLEX.Net 中汇集了引导人们学习这个新框架的提示、技巧、博客帖子、新闻和技术文章。Tariq 在 Flex 社区发展初期帮助发布内容，我本人从他那里学到很多 Flex 技能。

Tariq 很早就认识到了数据服务的重要性，掌握了借助实时数据创建 RIA 的方法。他构建了很多 Flex 应用程序，从企业内部以业务为核心的系统，到很多面向消费者的各种应用程序。他了解并使用了这个平台提供的所有特性，这是他成为编写 Flex 图书的理想人选

的原因之一。我相信，这本由业内专家亲笔撰写的书，正是读者期待已久的。

Flex 3 的发布，把我们带到了这个框架和这门技术的一个激动人心的新阶段。Adobe 将 Flex 平台的很多组件作为开源软件提供，目的就是让 Flex 尽可能透明化，从而吸收来自社区的有价值的反馈(当然，也包括有价值的代码)。

随着 Adobe AIR 的发布，开发人员现在可以使用 Flex 部署桌面应用程序和基于浏览器的应用程序了。

长久以来，Adobe 凭借其提供的卓越设计工具而闻名于世。在允许设计人员与开发人员紧密协作，以便设计出漂亮美观的 Flex 应用程序方面，Flex 3 开创了先河。此外，这个框架和 Flex Builder 中包含的大量有助于提高效率的特性降低了开发新手使用 Flex 的门槛，也有助于他们开发出容易扩展的应用程序。

无论你拥有什么样的开发背景，也不管你拥有 Flex 经验的多少，相信本书一定会成为你的得力助手。应该说，任何人都可以从本书中学习到有用的东西。Tariq 与另外两位作者 Jon 和 Faisal 为我们奉献了一本必备的 Flex 佳作。Tariq 提供的 Flex 社区 CFLEX.Net 呈现了我们大家都熟知和钟爱的必备资源，同样，由这位明星人物编写的本书也给我们奉献了一道美味丰盛的知识大餐。

Ryan Stewart
Adobe 公司平台推广专家

作者自序

多年来，我一直都在寻找一种能够带给用户更美好在线体验的方式，在 Google 革命性的 Google Maps 站点引起轰动之前的很长时间里，Web 用户并不知道还能有什么更美好的在线体验。

在把 Web 当作文档发布系统使用的若干年里，用户一边使用着强大的本地桌面应用，一边又面对着乏善可陈的 HTML 应用。但是，贫乏的用户体验并没有对 HTML 和 Web 构成冲击——Web 作为平台中立的文档发布系统，似乎已无懈可击。开发人员和公司专注于 Web 是因为它支持快速应用程序开发，而用户则被动地接受现实，认为 Web 应用程序就是这个样子的。

有件事曾令我百思不得其解。每次点击都会导致后台系统执行许多代码，而结果反映到用户界面上却只有一点点变化。更糟的是对数据库服务器要进行频繁密集的访问。对一名技术人员来说，解决这个问题最简便的办法就是多加内存，使用虚拟机加载，或者买一些低价的商用服务器进行水平扩展，从而暂时渡过难关。但是，我更关心用户要为此付出什么代价。他们必须对 Web 应用程序中常见的点击加等候习以为常，而且对用户界面也没有多大的操作自由。不错，可以使用 JavaScript，但如果未采用先进的技术，也无法取得多大的进展。从投资回报角度讲，这样做往往得不偿失。

当时，Java Applet 和 Flash 问世了，而且乍一看它们正是我要找寻的东西。实际上，Applet 作为一个解决方案并不合适，它的体积太大，下载也很慢，况且不能跨平台使用。Flash 倒是挺有希望的，可是在设计人员的工作环境中创建企业级应用程序，仍然不免有缘木求鱼的感觉。

我在 eBay 的知识管理部门工作期间，也遇到了相同的问题。我需要找到一种方式，能够抽象出数据的复杂性，并且能让用户在可视的环境中更方便地操作这些数据。

Flex 于 2004 年问世(最初是 V1，很快就出现了 V1.5)。我当时拍板做出采用它的决定，我们的团队也因此得到了完全不一样的新体验。当时，我就知道 Flex 前途无量。因为 Flex 应用程序既具有高级用户需要的桌面环境，又能满足软件团队的快速开发需求。

作为 Flex 的支持者，我把推动 Flex 社区发展当作自己的一项使命。我创建了 CFLEX.Net(www.cflex.net)，并坚信这个社区的规模越大，通过知识和代码共享产生的自我推动力也将越大，借此就可以促进这项技术的更快普及。毕竟，强有力的支持网络可以降低在组织中引入新技术的风险。

作为较早采用 Flex 的人，我在学习 Flex 的过程中走过不少弯路，主要原因是当时缺少相应的书籍和读物。这种局面在 Flex 2 发布后得到了改观，大量的学习资源开始涌现。

2005 年底我离开 eBay，加入 Amcom Computer Services，并在那里组织和管理一个开发团队。与学习任何新技术一样，要熟练掌握 Flex 也不容易。因此，最好的办法就是不断

提升技能。在培训开发人员使用 Flex 的过程中，我发现市面上的多数图书常常言不及义，很多显而易见的问题都没有提到。

为了进一步推动 Flex 社区的发展，我决定编写本书，希望它能解决开发人员日常遇到的问题。本书一反按功能特性布局谋篇的常见模式，改为按创建应用程序的自然进程组织内容。为了达到尽量简化的目的，我只在必要时介绍必要的知识，不会过早地讨论复杂主题，同时还列举了大量易于理解和吸收的简短示例代码。另外，我还发现温故而知新，把新事物与已知事物联系起来，可以增强学习效果。因此，在适当的情况下，我会尽可能拿其他技术的实现原理来进行类比。

希望读者通过阅读本书能够深入理解 Flex，从而壮大 Flex 社区中——因为届时你也能够与周围的人分享自己的知识和经验。

现在是准备学习 Flex 的时候了。随着社区逐步发展壮大，越来越多的第三方厂商会发布与 Flex 有关的技术，Flex 用户组也会在世界各地如雨后春笋般涌现。

随着其他厂商的陆续跟进，RIA 领域正在迅速升温——Adobe 再次证明自己走在了前列。我们正处于一个令人振奋的时代！以 HTML 为基础的 Web 应用程序始终会占有一席之地；然而，现在是该把你的技能提升到一个新高度，以便顺应行业发展潮流的时候了。

现在请坐稳，系好安全带，我们将在求知之旅上快速前行！

Tariq Ahmed

前　言

Flex 是事件驱动的面向对象应用程序框架和编程语言，使用它可以构建出引人注目且容易扩展的运行于 Adobe Flash 环境中的 RIA。

历史上，Web 应用程序开发的优势在于能够快速推向市场，但这一优势往往以牺牲最终用户的易用性为代价。Flex 既确保了集中部署的快速转换模型，又为实现易用性提供了强大支持。

初次接触 Flex 的人往往面临一些独特的挑战。第一个挑战是，许多开发人员不熟悉事件驱动技术。虽然他们大致了解事件驱动的概念，但缺乏长期的使用经验。这种情况下，很难充分发挥这种技术的长处，更重要的是，无法让自己的工作更富有成效。

第二个挑战源于许多开发人员长期固守当前的技术领域。由于习惯于既有的思维方式，这些人往往会将自己对 HTML 的经验移植到 Flex 中。这种做法勉强可行，但束缚了开发人员的创造力。

第三个挑战是与 Flex 本身的强大功能相伴而生的复杂性。尽管 Flex 本质上并不复杂，但毕竟很多 Web 应用程序技术都是过程式、非事件驱动的，因此读者很可能在接触简单示例时感觉比较容易，而一旦深入钻研，会觉得难度陡增。

本书将面对这些挑战并给出解决方案，而这也是本书的独到之处。对于第一个挑战，本书会通过不断提醒读者该如何利用事件对象，来强调 Flex 的事件驱动本质。而且，还将针对同一任务给出多种实现方式，并分析每种方式的利弊，以加速读者的理解。

至于第二个挑战，我们不会否定读者已有的技能，也明白你可能拥有另一种技术背景。我们也有同样的经历，体验过摆脱业已习惯的思维模式时的心理转换过程。本书时不时会对如何利用某项功能给出建议。

谈到第三个挑战——复杂性，本书的使命就是让读者在短时间内成为一位高效率的 Flex 开发人员。为此，我们综合运用了一些技巧，包括设计一些容易理解的小例子。而且，我们也会尽量利用你现有的技能，将使用其他技术和使用 Flex 解决同一个问题联系起来。另外，本书所有章节都是按构建应用程序的逻辑顺序组织的，先介绍简单的内容，然后循序渐进地讨论新知识。

请放心，这本书你选对了。我们就是使你的技能和职业生涯更上一层楼的引路人。不过，有一个问题我们解决不了，那就是当用户万分感激，对你大加赞赏时，你该怎么办——这个问题必须由你自己来回答！

本书结构

第 1 章简要介绍 Flex。这一章讨论 Flex 适于解决的问题及解决方法。之后，为读者提供了说服所在部门、消费者或客户接受 Flex 的重要理由。这一章还概括介绍了 Flex 的工作原理和事件的概念，明确了 Flex 在 Adobe 产品套件中所处的位置。

第 2 章介绍构建 Flex 应用程序涉及的工具集、环境和语言。为了加深读者对事件的理解，这一章对事件的内容进行了充实。

第 3 章介绍 Flex 的核心语言——ActionScript。ActionScript 是一门符合 ECMAScript 标准的面向对象的语言，它成就了 Flex。这一章讨论了数据类型、操作符、循环和条件等概念。熟悉 ActionScript 是必需的，因此读者有必要尽早掌握这门语言。

第 4 章介绍应用程序的布局。这一章讨论如何定位显示对象和使用容器对视觉对象分组。

第 5 章开始介绍如何通过表单捕获用户输入。Flex 具有类似于 HTML 的表单输入控件，但也包含很多 HTML 中没有的控件。

第 6 章接着第 5 章的主题，继续讨论如何使用 Flex 的验证程序来验证用户输入。从易用性角度讲，事先验证可以节省用户的时间，省去不必要的麻烦。

第 7 章转而介绍使用格式化程序对原始信息进行格式化。经常与验证程序一起使用的格式化程序，解决了手工格式化数据的难题。

第 8 章探索 Flex 中最常用的结构——基于列表的组件。列表是一种数据驱动组件，它能够以列表形式自动呈现数据源中的数据。

第 9 章继续讨论列表，但专门介绍创建自定义列表的方法，包括简单的方法和使用全功能的条目渲染器自定义显示。这一章还介绍了支持行内编辑的编辑器。

第 10 章专门讨论事件。此章之前介绍过如何以最低限度标准使用事件，本章通过深入介绍事件的工作机制，把对它的应用提升到一个新的层次。

第 11 章展示如何在应用程序中添加导航组件，以便用户在不同功能间切换。

第 12 章讨论扩展应用程序导航功能的弹出式窗口。探索了应用程序如何通过来回发送信息与弹出式窗口通信。

第 13 章介绍应用程序流主题，并讨论 Flex 中被称为“视图状态”的独特功能。通过这种机制在应用程序中配置不同的视图，可以节省很多时间，同时也提供了在视图间切换的便捷方式。

第 14 章开始讨论有关数据操作的主题，尤其是在应用程序与后端服务间实现数据交换。这包括连接到支持 Flex 本地二进制协议(AMF)的服务器、HTTP 上的 XML 以及 Web 服务。

第 15 章深入讨论 XML 的使用。XML 是一种应用广泛的语言；但 Flex 率先支持了 E4X 语法，因而可以像操作本地 Flex 对象一样处理 XML。

第 16 章介绍对象和类。毕竟，Flex 是一种面向对象的语言。虽然基于标签的 MXML 语言令人愉悦，但熟悉创建和使用 ActionScript 对象的方法只会提升你的开发技能。

第 17 章详细介绍将占用大量开发时间的自定义组件。自定义组件是把应用程序分解

成小型、易管理、可重用部分的主要手段。

第 18 章专注于应用程序的结构，并介绍 Flex 的可重用特性，例如，在多个项目之间共享自定义组件与编译共享功能库。

第 19 章讨论自定义体验。这一主题包括使用 Flex 的 CSS 样式、皮肤和主题。这一章还介绍了图像和字体。

第 20 章介绍 Flex 最酷的功能——效果。效果是应用程序中吸引眼球的元素。同时，这一章还会讨论如何通过效果增强易用性。

第 21 章是讨论自定义内容的最后一章，详细介绍了拖放功能。拖放功能招人喜爱，但从易用性的角度说，拖放通过加快工作流能使用户减少一些点击操作。

第 22 章讨论图表。这一章介绍各种类型的图表并对每种类型的适用时机给出建议。此外，这一章还讨论了图表的组成部分，以及如何定制图表。

第 23 章讲述测试与调试。此时，项目临近结束，进入了 QA(质量保证)循环。知道怎样调试应用程序和如何隔离问题是关键。Flex 内置了许多相关功能，但这一章也讨论了第三方工具。

第 24 章介绍完成项目的最后步骤。项目收尾包括为应用程序添加打印功能，使用包装器加载应用程序，以及制定版本规划以便部署应用程序的生产版本。

在第 24 章之后，我们为读者提供了开发人员资源、论坛、开源项目、Flex 技术圈及北美洲用户组和国际性用户组的链接。

代码下载及格式约定

本书包含大量 Flex、ActionScript 和 XML 代码示例。所有代码示例都可以在本书网站 http://www.flexinaction.com 或出版商的网站 http://www.manning.com/ahmed 或 http://www.tupwk.com.cn/downpage 下载。

本书使用以下排版约定。

- 楷体用于表示新术语。
- 等宽字体(Courier/Fixed-Width)用于示例代码。
- 粗斜的等宽字体(Bold and Italic face Courier/Fixed-Width)用于突出显示代码中的某些部分。
- 代码片段中可能会包含注释。以❶这样的符号标出的注释，在代码后的正文中可能会有进一步的解释。
- 用“|”符号来表示应该按顺序选择的菜单项。
- 用“➥”符号表示代码行的延续。

作者在线

购买了本书意味着你已获得免费访问 Manning 出版社所运营的私有 Web 论坛的权利，在这个论坛上你可以对本书发表评论，可以提出技术问题并从作者和其他读者那里获得帮

助。要访问论坛并且订阅其内容，可以在Web浏览器中输入www.manning.com/Flex3inAction。该网页提供了各种信息，包括注册后如何访问论坛，可以获得什么类型的帮助，以及论坛的行为准则。

Manning 出版社对读者的承诺是，提供一个读者之间以及读者和作者之间能够开展有意义的交流的平台，但是并不承诺作者必须要参与到何种程度，作者对于论坛的贡献是志愿的(并且是无偿的)。我们建议你向作者多提一些富有挑战性的问题，以免他们失去兴趣！

只要本书还在印刷销售，作者在线论坛和以前讨论的存档都可以在出版社的网站上访问到。

作者简介

Tariq Ahmed 是一位卓越的 Web 应用程序的先驱人物，先后向 Bell Canada 和 Reuters 等公司引荐了下一代 Web 技术。他和 Jon Hirschi 最早将 Adobe Flex 引入 eBay，随后它又被其他团队采用。作为 Adoble Flex 社区专家，Tariq 始终致力于推广这一技术并通过各种项目为社区提供支持。另外，Tariq 因其 Community Flex (CFLEX.Net)站点而广为人知。Tariq 目前担任美国旧金山湾区的 Amcom Technology 公司的产品开发经理。

Jon Hirschi 自 Flex 第一个版本推出以来就致力于 Flex 的改进。作为一名 Adobe Flex 社区专家，他一直通过自己的博客、技术杂志文章和用户组分享其专家级的真知灼见。Jon 不仅参与向 eBay 引荐了 Flex，而且也是 eBay 负责管理服务器的前沿控制中心的一员。Jon 目前是美国旧金山湾区的 Kadoink 公司的技术经理。

Faisal Abid 是一名在加拿大多伦多求学的软件工程专业的学生，他拥有自己的 RIA 咨询公司 G-uniX Technologies。他曾为很多客户撰写过 RIA 方案，包括基于互联网的创业型公司 Buzzspot 和 RazorCom。Faisal 的社区业绩包括在各类杂志上发表文章，他也因自己拥有的技术经验而声名远扬。

关于书名

根据认知科学的研究，人们最容易记住的事情是他们在自发探索中发现的事情。

尽管在 Manning 出版社并没有专门研究认知的科学家，我们仍然确信为了使学习变得持久，必须经历探索、实践和(很有趣的)温习这几个阶段。只有在对新的事物进行了积极的探索之后，人们才能够理解并记住新的事物，也就是所谓的掌握了新的事物。人类在实践中学习，例子驱动是所有具备实战性质图书的基本部分。这个特点鼓励读者去尝试新的事物，实践新的代码，探索新的思想。

关于书名，这里还有一个更加世俗化的理由：我们的读者都很忙。他们由于工作需要或者为了解决一个问题而寻求书的帮助。他们要求图书既要深入浅出，又能在他们需要的时候为他们提供需要的内容。他们需要的是能够帮助他们增强实战能力的书。本书就是为这个目的而设计的。

封面图片

本书封面图片的名字是“An Armenian”，该插图来自于一本介绍奥斯曼帝国服饰的画册(由伦敦老邦德街的 William Miller 于 1802 年 1 月 1 日出版)。画册的扉页已经遗失，因此我们无法确定它的创作时间。画册目录中以英语和法语来标识图画，每张画有创作它的两位艺术家的名字，他们无疑会为自己的艺术能够美化 200 年后上市的一本计算机编程图书的封面而感到惊讶。

Manning 出版社的一位编辑在位于曼哈顿西 26 街“Garage”的古董跳蚤市场购买了这本画册。卖主是住在土耳其安卡拉的一个美国人，交易的时间发生在那天他准备收摊的时候。Manning 出版社的编辑随身没有携带购买这本画册所需的足够现金，并且卖主礼貌地拒绝了他使用信用卡和支票。因为卖主当天晚上要飞回安卡拉，所以似乎已经没有希望了。那么是怎么解决的呢？无非就是一个通过握手来约定的老式的君子协议。卖主提议通过银行转账付款，编辑仅仅在纸片上抄下了收款银行的信息，随后画册就到了他的手里。不用说，第二天他就将钱付给了卖主。我们感谢这位陌生人能够如此信任我们的同事。这使我们回忆起那个很久以前浪漫温情的时代。

来自奥斯曼帝国画册的图片，就像在本书封面上出现的图片一样，穿越时空将两个世纪之前服饰习俗的丰富性和多样性呈现在我们眼前。这些图片令人回忆起那个与我们隔离又与我们相距遥远的时期(以及除了通信极其发达的今天之外的其他每个历史时期)。

从那以后，服饰习俗已经改变了，那个时期不同地域的丰富多彩的服饰文化也逐渐褪色。现在甚至连区分两个来自不同大陆的居民常常都是困难的。也许我们应该尽量乐观地看待这些变化，文化上、视觉上的差异已经转变为更加多样化的个人生活上的差异，或者更加多样化和有趣的智力上和技术生活上的差异。

我们使用来自这本画册中的图片作为本书的封面，这使我们欣赏到两个世纪前丰富、多样的地域生活。Manning 出版社的编辑们使用这种方式来赞美计算机行业中的独创性和主动性以及其中无尽的趣味性。

目　录

第Ⅰ部分　应用程序基础知识

第1章　认识Flex……3
1.1　Web应用程序为何大行其道……3
1.2　为快速发展付出的代价……4
1.3　RIA解决方案……5
1.3.1　他们什么都想要……5
1.3.2　RIA的解决之道……6
1.3.3　RIA的工作原理……6
1.4　RIA的竞争技术……6
1.4.1　Adobe公司的Flex……7
1.4.2　微软公司的Silverlight……7
1.4.3　Sun公司的JavaFX……8
1.5　Flex与Web应用程序……8
1.5.1　Web应用程序基于文档……9
1.5.2　浏览器的角色：中转站和文档渲染器……9
1.5.3　RWA——终点站……10
1.5.4　跨平台问题……11
1.5.5　何来丰富性……11
1.5.6　AJAX的通信限制……12
1.5.7　并行不悖……13
1.6　熟悉Flex……13
1.6.1　利用Adobe Flash的优势……13
1.6.2　Flex与JavaScript可以一起使用……14
1.6.3　Flex技术体系……15
1.6.4　Flex的工作原理……17
1.6.5　事件驱动的环境……19
1.7　Flex 3的新功能……19
1.8　小结……20

第2章　从零开始……21
2.1　经济的Flex开发方式……21
2.1.1　建立编译环境……22
2.1.2　建立编辑环境……23
2.1.3　第三步(但愿读者还感兴趣)……24
2.2　使用Flex Builder做真正的开发……25
2.2.1　产品和价格表……25
2.2.2　获得Flex Builder……26
2.3　探索Flex Builder……26
2.4　视图与视角……28
2.4.1　预定义的视角……28
2.4.2　切换视角……29
2.4.3　自定义视角……29
2.5　第一个项目——HelloWorld……31
2.5.1　创建项目……31
2.5.2　输入代码……33
2.5.3　编译和运行……33
2.5.4　查看实际效果……34
2.6　使用设计模式……34
2.7　内置的帮助……36
2.7.1　面向对象语言及其API……36
2.7.2　访问API参考……36
2.7.3　研读API参考……37
2.8　MXML与ActionScript速写……38
2.8.1　MXML的结构……38
2.8.2　MXML与ActionScript的关系……39
2.8.3　事件由ActionScript处理……40
2.9　小结……42

第3章　使用ActionScript……43
3.1　注释……43

3.1.1 行内注释 …… 44
3.1.2 多行注释 …… 44
3.2 变量 …… 44
3.2.1 变量名 …… 44
3.2.2 严格数据类型 …… 44
3.2.3 静态类型检查与动态类型检查 …… 45
3.2.4 基本数据类型和复杂数据类型 …… 45
3.3 循环 …… 48
3.3.1 for(初始值;条件;递增) …… 48
3.3.2 for(属性名 in 数组/对象) …… 48
3.3.3 foreach(项 in 数组/对象) …… 49
3.3.4 while(条件) …… 49
3.3.5 do while(条件) …… 50
3.4 条件语句(if 和 switch) …… 50
3.4.1 if…else …… 50
3.4.2 switch …… 52
3.5 数组 …… 53
3.5.1 索引数组 …… 53
3.5.2 关联数组 …… 55
3.6 ActionScript 特色语法 …… 57
3.6.1 花括号 …… 57
3.6.2 逻辑操作符及快捷方式 …… 58
3.7 函数、类和包简介 …… 59
3.7.1 创建函数 …… 60
3.7.2 将 ActionScript 代码保存在单独的文件中 …… 63
3.8 简单的数据绑定 …… 64
3.8.1 再论事件 …… 65
3.8.2 双向绑定 …… 65
3.9 小结 …… 66

第 4 章 布局和容器 …… 67
4.1 绝对布局 …… 67
4.2 基于约束的布局 …… 70
4.2.1 基本约束 …… 70
4.2.2 增强的约束 …… 72
4.3 自动布局 …… 77
4.4 可变大小和固定大小 …… 78
4.4.1 可变大小 …… 79
4.4.2 固定大小 …… 79
4.5 容器 …… 79
4.5.1 Application 容器 …… 80
4.5.2 Canvas 容器 …… 81
4.5.3 Box、HBox 和 VBox 容器 …… 82
4.5.4 Panel 容器 …… 83
4.5.5 ApplicationControlBar 和 ControlBar 容器 …… 83
4.5.6 DividedBox、HDividedBox 和 VDividedBox 容器 …… 84
4.5.7 Form 容器 …… 85
4.5.8 Grid 容器 …… 86
4.5.9 Tile 容器 …… 87
4.6 使用 Repeater 组件实现动态布局 …… 88
4.6.1 dataProvider 简介 …… 88
4.6.2 Repeater 的属性和事件 …… 89
4.6.3 创建 Repeater …… 90
4.6.4 使用 Repeater 事件 …… 91
4.7 小结 …… 92

第 5 章 显示表单与捕获用户输入 …… 95
5.1 id 属性 …… 96
5.2 Text 控件 …… 96
5.3 Date 控件 …… 98
5.4 Numeric 控件 …… 99
5.5 Button 组件 …… 101
5.5.1 带 Bar 的 Button 控件 …… 102
5.5.2 PopUpButton 和 PopUpMenuButton …… 103
5.6 选单控件 …… 105
5.7 访问控件的值 …… 107
5.7.1 向函数传递值 …… 107
5.7.2 向函数传递事件 …… 108
5.7.3 直接访问属性 …… 109
5.7.4 应该选择哪种方式 …… 110
5.8 小结 …… 111

第 6 章　验证用户输入 ……113
6.1　验证简介 ……113
6.2　内置验证器 ……114
6.2.1　Validator ……115
6.2.2　StringValidator ……116
6.2.3　NumberValidator ……117
6.2.4　DateValidator ……119
6.2.5　EmailValidator ……122
6.2.6　CreditCardValidator ……123
6.2.7　CurrencyValidator ……125
6.2.8　PhoneNumberValidator ……127
6.2.9　RegExpValidator ……128
6.2.10　SocialSecurityValidator ……130
6.2.11　ZipCodeValidator ……131
6.3　实时验证 ……132
6.4　提交值验证 ……132
6.5　通过性验证 ……133
6.6　脚本式验证 ……133
6.7　验证技巧 ……134
6.7.1　验证器是否总是检查所有条件 ……135
6.7.2　验证输入的值与条件匹配 ……135
6.7.3　控制由什么来触发验证操作 ……135
6.8　小结 ……135

第 7 章　格式化数据 ……137
7.1　内置格式化程序 ……138
7.1.1　Formatter ……138
7.1.2　NumberFormatter ……138
7.1.3　CurrencyFormatter ……140
7.1.4　DateFormatter ……141
7.1.5　PhoneFormatter ……144
7.1.6　ZipCodeFormatter ……146
7.1.7　SwitchSymbolFormatter ……147
7.2　实时格式化 ……148
7.3　脚本式格式化 ……148
7.3.1　结合使用函数与格式化程序组件 ……148
7.3.2　结合使用函数与格式化程序类 ……149
7.4　处理格式化错误 ……151
7.5　小结 ……151

第 8 章　DataGrid、列表和树 ……153
8.1　基于列表的一系列组件 ……153
8.1.1　ListBase 和 AdvancedListBase 的属性 ……154
8.1.2　ListBase 事件 ……155
8.2　dataProvider 与集合 ……155
8.2.1　填充 dataProvider ……156
8.2.2　集合的类型 ……156
8.2.3　集合的使用者 ……156
8.3　集合的初始化 ……157
8.4　集合的填充 ……158
8.4.1　List ……158
8.4.2　HorizontalList ……160
8.4.3　TileList ……161
8.4.4　DataGrid ……163
8.4.5　Tree ……166
8.4.6　AdvancedDataGrid ……169
8.5　与列表交互 ……175
8.5.1　列表事件 ……175
8.5.2　向函数传递事件 ……176
8.5.3　向函数传递数据 ……177
8.5.4　直接访问被选定的行 ……178
8.5.5　绑定到被选定的行 ……178
8.5.6　用其他事件作为触发器 ……179
8.6　小结 ……179

第 9 章　自定义列表 ……181
9.1　标签函数 ……181
9.1.1　标签函数的类型 ……181
9.1.2　使用单列标签函数 ……182
9.1.3　使用多列标签函数 ……183
9.1.4　标签函数的用途 ……184
9.2　项渲染器 ……185
9.2.1　渲染器类型 ……185
9.2.2　创建(常规)项渲染器 ……185

9.2.3 向项渲染器中添加逻辑……186
9.2.4 创建行内项渲染器……189
9.2.5 使用混入项渲染器……189
9.3 项编辑器……191
9.3.1 启用项编辑功能……191
9.3.2 创建项编辑器……192
9.3.3 项编辑事件……194
9.4 组合项渲染器和项编辑器……199
9.5 高级项渲染器……200
9.5.1 AdvancedDataGridRenderer Provider……200
9.5.2 引用列……201
9.5.3 跨列使用……202
9.5.4 实现通行显示……203
9.6 筛选函数……204
9.7 小结……206

第II部分 应用程序流与结构

第10章 事件……209
10.1 事件系统……209
10.1.1 事件系统——类似于邮政服务……210
10.1.2 事件发送系统……210
10.1.3 设置事件传播的阶段……211
10.2 发送和接收事件……212
10.2.1 添加事件侦听器……213
10.2.2 绑定事件……214
10.2.3 移除事件侦听器……216
10.3 自定义事件……218
10.3.1 使用分派程序发送事件……218
10.3.2 创建自定义事件……220
10.3.3 停止事件传播……222
10.3.4 向组件添加事件元数据……223
10.4 小结……224

第11章 应用程序导航……225
11.1 为菜单和菜单栏提供数据……225
11.1.1 嵌套的数组……226
11.1.2 嵌套的数组集合……226
11.1.3 Model……227
11.1.4 XML 组件及类……228
11.1.5 XMLList 组件……229
11.1.6 XMLListCollection 组件及类……229
11.1.7 选定数据提供者数据类型……230
11.2 使用菜单……231
11.2.1 创建菜单……231
11.2.2 定位菜单……233
11.2.3 自定义菜单项……233
11.3 使用菜单栏……237
11.3.1 创建菜单栏……238
11.3.2 定位菜单栏……238
11.3.3 自定义菜单栏中的项……239
11.3.4 处理用户与菜单栏的交互……240
11.4 使用 ViewStack……241
11.4.1 创建 ViewStack……241
11.4.2 为 ViewStack 添加导航功能……243
11.4.3 确定当前选定的栈……244
11.4.4 处理用户与 ViewStack 的交互……245
11.5 TabNavigator……247
11.5.1 创建 TabNavigator……247
11.5.2 处理用户与 TabNavigator 的交互……248
11.6 Accordion……250
11.6.1 创建 Accordion……250
11.6.2 填充 Accordion……250
11.6.3 处理用户与 Accordion 的交互……252
11.7 小结……252

第12章 弹出式窗口简介……253
12.1 创建首个弹出式窗口……253
12.1.1 首先创建标题窗口……253

12.1.2 使用 PopUpManager 打开窗口 ……254
12.1.3 关闭弹出式窗口 ……255
12.2 控制窗口位置 ……255
12.2.1 使用 centerPopUp() 方法 ……256
12.2.2 计算窗口的位置 ……256
12.3 设置窗口的透明度 ……258
12.4 弹出式窗口与数据的结合 ……259
12.4.1 从弹出式窗口中取得数据 ……261
12.4.2 发送事件 ……262
12.4.3 取出数据 ……262
12.4.4 向窗口中发送数据 ……263
12.5 使用警告窗口 ……264
12.5.1 创建简单的警告窗口 ……264
12.5.2 挖掘警告窗口的潜力 ……265
12.5.3 高级警告窗口 ……265
12.5.4 修饰警告窗口 ……266
12.6 小结 ……268

第 13 章 视图状态 ……269
13.1 理解视图状态 ……269
13.2 Flex 中的视图状态 ……270
13.2.1 视图状态类 ……271
13.2.2 视图状态实例 ……272
13.2.3 定义视图状态 (<mx:State />) ……274
13.2.4 视图状态过渡 ……277
13.2.5 组件的状态 ……278
13.3 小结 ……280

第 14 章 使用数据服务 ……281
14.1 什么是 WSDL 文档 ……281
14.2 理解 WSDL 文档 ……282
14.2.1 理解 WSDL 文档：操作 ……282
14.2.2 理解 WSDL 文档：输入类型 ……283
14.2.3 理解 WSDL 文档：输出 ……283
14.3 使用 WebService 组件 ……285
14.3.1 用 ActionScript 创建 WebService 组件 ……286
14.3.2 调用 WebService 组件 ……286
14.3.3 处理结果 ……287
14.3.4 ResultEvent ……287
14.3.5 处理结果 ……288
14.3.6 FaultEvent ……289
14.3.7 使用异步记号 ……290
14.4 使用 HTTPService 检索结果 ……290
14.4.1 连接 HTTP Web 服务 ……291
14.4.2 显式参数绑定 ……292
14.5 使用 RemoteObject 标签 ……293
14.6 使用 URLLoader ……295
14.7 使用 Flex Import Web Service 向导 ……296
14.7.1 Import Web Service 向导 ……296
14.7.2 使用生成的 Web 服务代码 ……297
14.8 小结 ……299

第 15 章 使用 XML ……301
15.1 XML 基础知识 ……301
15.1.1 什么是 XML ……301
15.1.2 XML 的优点 ……302
15.1.3 XML 的缺点 ……302
15.1.4 XML 语法及规则 ……302
15.1.5 DTD 与 XML Schema ……305
15.1.6 名称空间 ……307
15.2 XML 组件和类 ……307
15.2.1 XML 组件 ……307
15.2.2 XML 类 ……308
15.2.3 XMLList 组件 ……312
15.2.4 XMLList 类 ……313

15.2.5 XMLListCollection 组件和类 ……313
15.3 加载外部 XML 源数据 ……314
15.4 XML 中的绑定 ……315
15.4.1 绑定自 XML ……315
15.4.2 绑定到 XML ……316
15.5 ECMAScript for XML(E4X) ……317
15.5.1 E4X 语法 ……317
15.5.2 使用 XML 结构 ……318
15.6 小结 ……322

第 16 章 对象和类 ……325
16.1 用五分钟的时间理解 OO 理论 ……325
16.1.1 对象和类的关系 ……325
16.1.2 对象拥有属性和方法 ……326
16.1.3 继承 ……327
16.1.4 封装和耦合 ……328
16.1.5 对象：无处不在 ……328
16.2 操作对象 ……329
16.2.1 进一步了解对象 ……329
16.2.2 对象的方法 ……330
16.2.3 方法的参数 ……330
16.2.4 方法返回信息 ……331
16.2.5 对象的属性 ……331
16.3 创建类 ……331
16.3.1 创建类文件 ……331
16.3.2 指定包 ……332
16.3.3 命名类 ……332
16.3.4 类修饰符 ……332
16.3.5 超类：扩展类 ……333
16.3.6 接口 ……333
16.3.7 分析类 ……334
16.4 添加属性 ……334
16.5 添加 getter/setter 方法 ……335
16.6 创建类的方法 ……337
16.7 小结 ……339

第 17 章 自定义组件 ……341
17.1 什么是自定义组件 ……342
17.2 自定义组件的工作原理 ……343
17.2.1 简单组件与复合组件 ……343
17.2.2 实现方式 ……344
17.3 简单自定义组件 ……344
17.4 传递参数 ……345
17.4.1 通过函数传递值 ……345
17.4.2 通过属性传递值 ……346
17.4.3 使用 setter 函数作为属性代理 ……348
17.4.4 使用绑定参数 ……349
17.5 检索值 ……350
17.5.1 通过函数检索值 ……350
17.5.2 通过属性检索值 ……351
17.5.3 使用 getter 函数作为属性代理 ……354
17.6 创建复合组件 ……356
17.7 创建 ActionScript 组件 ……357
17.8 自定义事件 ……359
17.9 名称空间和结构 ……361
17.9.1 mx 名称空间 ……361
17.9.2 local 名称空间 ……361
17.9.3 目录结构与名称空间 ……361
17.9.4 ActionScript 中的名称空间 ……362
17.10 小结 ……363

第 18 章 Flex 中的高级可重用性 ……365
18.1 SWC 文件 ……365
18.2 运行时共享库 ……366
18.2.1 标准 RSL ……366
18.2.2 在 Flex 应用程序中使用 RSL ……367
18.3 开发模块化的 Flex 应用程序 ……369
18.3.1 组件与模块 ……370
18.3.2 创建简单的模块 ……370
18.3.3 通过模块 API 以 MXML 方式加载模块 ……370

18.3.4 通过 ActionScript 加载和卸载模块……371
18.3.5 模块的利弊……372
18.4 在 Flex 应用程序的生命周期中添加补丁……373
18.5 重构……374
18.6 小结……375

第III部分　高级应用

第 19 章 自定义体验……379
19.1 样式……379
19.1.1 行内样式……380
19.1.2 本地样式定义……381
19.1.3 外部样式表……385
19.1.4 样式浏览器……386
19.1.5 处理颜色……386
19.1.6 透明度……388
19.1.7 使用渐变……389
19.1.8 以编程方式操作样式……389
19.2 嵌入字体……391
19.2.1 通过字体的系统名嵌入……392
19.2.2 通过字体文件嵌入……392
19.2.3 使用 CSS……393
19.3 图像和图标……393
19.3.1 图像类型……393
19.3.2 嵌入还是不嵌入……393
19.3.3 作为变量的图像……394
19.3.4 图标……395
19.4 皮肤……397
19.4.1 皮肤的类型……397
19.4.2 使用图像的图形皮肤……397
19.4.3 使用 SWF 的图形皮肤……399
19.4.4 Flex 皮肤设计扩展(SDE)……400
19.4.5 图像切割……400
19.4.6 编程式皮肤……402
19.5 小结……402
第 20 章 使用效果……403
20.1 效果的含义……403
20.1.1 创建效果的方式……403
20.1.2 Flex 内置的效果……404
20.1.3 合成效果……404
20.2 触发效果……404
20.3 以编程方式应用效果……406
20.3.1 使用 ActionScript 创建效果……406
20.3.2 只使用 ActionScript……407
20.4 创建合成效果……408
20.4.1 串行效果……408
20.4.2 并行效果……408
20.4.3 组合合成效果……409
20.5 使用 Flex 内置的效果……409
20.5.1 AnimateProperty……410
20.5.2 Blur……410
20.5.3 Dissolve……411
20.5.4 Fade……411
20.5.5 Glow……412
20.5.6 Iris……413
20.5.7 Move……413
20.5.8 Pause……414
20.5.9 Resize……414
20.5.10 Rotate……415
20.5.11 SoundEffect……416
20.5.12 Wipe……417
20.5.13 Zoom……417
20.6 缓和函数……418
20.6.1 内置的缓和函数……418
20.6.2 创建自定义缓和函数……419
20.7 字体与效果……420
20.8 小结……421
第 21 章 拖放……423
21.1 拖放过程……423
21.2 拖放事件……425
21.3 支持拖放的组件……425

21.4 在基于列表的组件中启用拖放 …… 426
21.5 移动与复制 …… 427
21.6 多项拖动 …… 427
21.7 双向拖放 …… 428
21.8 使用拖放实现用户控制的排序 …… 429
21.9 使用 DragManager …… 430
21.9.1 操作值 …… 431
21.9.2 DragManager 函数 …… 431
21.10 接受或拒绝放置 …… 431
21.10.1 限制添加项 …… 431
21.10.2 阻止事件传播 …… 433
21.10.3 使用 DragEvent 对象找到放置目标 …… 433
21.11 应用自定义放置方式 …… 434
21.11.1 显示添加到组件的 dataProvider …… 434
21.11.2 隐式添加到组件的 dataProvider …… 435
21.12 为非列表组件添加拖动行为 …… 436
21.12.1 设置示例 …… 436
21.12.2 启动拖动 …… 436
21.12.3 添加视觉反馈 …… 437
21.12.4 处理放置 …… 438
21.12.5 处理退出 …… 438
21.12.6 综合起来 …… 438
21.13 自定义拖放体验 …… 440
21.13.1 修改拖动图像 …… 440
21.13.2 修改拖动代理图标 …… 441
21.14 小结 …… 442

第 22 章 图表 …… 443
22.1 图表简介 …… 443
22.1.1 图表的构成 …… 444
22.1.2 图表类型简介 …… 444
22.2 准备系列和数据 …… 445
22.3 创建图表 …… 446
22.3.1 调用图表 …… 446
22.3.2 添加图例 …… 447
22.3.3 改变图表类型 …… 448
22.4 堆叠图表 …… 449
22.5 探索图表类型 …… 450
22.5.1 面积图 …… 450
22.5.2 条形图和柱状图 …… 451
22.5.3 折线图 …… 453
22.5.4 气泡图 …… 453
22.5.5 烛台图和 HLOC 图 …… 455
22.5.6 饼图 …… 457
22.5.7 散点图 …… 459
22.6 自定义图表 …… 460
22.6.1 系列笔触 …… 460
22.6.2 系列填充 …… 460
22.7 小结 …… 462

第 23 章 调试与测试 …… 463
23.1 调试 …… 463
23.1.1 使用 Flash Debug Player …… 463
23.1.2 配置日志 …… 464
23.1.3 使用 trace()函数 …… 465
23.1.4 跟踪日志查看器 …… 465
23.1.5 把对象转换成字符串 …… 466
23.1.6 FxSpy …… 467
23.1.7 监视网络活动 …… 467
23.1.8 使用调试器 …… 468
23.2 测试 …… 471
23.2.1 测试的类型 …… 471
23.2.2 Flex 分析程序 …… 472
23.2.3 FlexUnit(单元测试) …… 474
23.2.4 Fluint(单元测试) …… 477
23.2.5 FunFX(功能测试) …… 480
23.2.6 RIATest(功能测试) …… 481
23.2.7 HP QuickTest Pro (功能测试) …… 481
23.2.8 IBM Rational Functional Tester(功能测试) …… 482
23.3 小结 …… 482

第 24 章　项目收尾 …… 483
24.1　打印 …… 483
24.1.1　在 Flex 中实现打印 ……483
24.1.2　打印工具 ……484
24.1.3　打印示例 ……484
24.1.4　控制缩放 ……485
24.1.5　添加对象的艺术 ……485
24.1.6　打印列表 ……487
24.1.7　捕获用户执行取消操作的时间 ……490
24.1.8　FlexReport ……490
24.2　自定义包装器 …… 490
24.2.1　包装器文件 ……491
24.2.2　包装器模板 ……491
24.2.3　最基本的要求 ……492
24.2.4　嵌入到 Web 应用程序中 ……492
24.2.5　在包装器中传递参数 ……493
24.3　部署 …… 494
24.3.1　创建生产版本 ……494
24.3.2　放置客户端文件 ……495
24.3.3　放置服务器端文件 ……496
24.3.4　测试效果 ……496
24.4　小结 …… 497
附录 A　在线资源 ……499

第 I 部分

应用程序基础知识

请大家准备好，马上就要踏上学习 Flex 的旅程了。这一部分的重点是通过介绍与创建 Flex 应用程序有关的基本概念，让大家进入状态。

在开始编写代码之前，首先要从全局角度审视一下 Flex，弄懂它到底是一门什么语言，它所属的技术体系，以及各个部分如何结合在一起。介绍了这些知识后，将接着讨论如何设置开发环境，以便可以创建和构建 Flex 应用程序。

学会了编译和运行 Flex 应用程序，紧接着当然就是学习编写代码(最妙不可言的部分)。我们先简要介绍 Flex 的核心语言——ActionScript。然后循序渐进地介绍每个应用程序的所有基本构件(building block)，从布局以及构建验证表单，到格式化数据，再到显示信息列表。

第 1 章 认识 Flex

本章内容：

- 通过 Flex 解决问题
- 使用 RIA 和 RWA
- Flex 与同类技术的比较
- 了解 Flex 技术体系

本章介绍 Flex 为什么是值得开发人员掌握的一项重要新技能，值得组织拥有的一项重要新技术。如今，时髦词满天飞，名字怪模怪样的网站层出不穷，与 Web 2.0 有关的技术似乎也炙手可热。面对形形色色的技术，普通的开发人员往往会无所适从，不知道自己应该学习哪门技术更好。说服客户和领导接受 Flex 是至关重要的。

本章将讨论 Web 开发人员面临的挑战和 Adobe 的 Flex 给出的解决方案。此外，本章还将深入剖析 Flex 应用程序的内部机制，并从整体上介绍 Flex 的技术体系。

1.1 Web 应用程序为何大行其道

Web 应用程序之所以如此之多，原因在于 Web 有着巨大的优势(但恰恰也是它的缺点)。设计 Web 的初衷是想让它成为一个轻量级的信息分发系统——既可以快捷地把文档发布到服务器上，又可以同样方便地从服务器中检索已有文档。

其中，最关键的优势是通过 Web 可以创建动态页面(Web 应用程序)。这一点很引人注目，就连桌面应用程序相形之下也黯然失色(见图 1-1)。

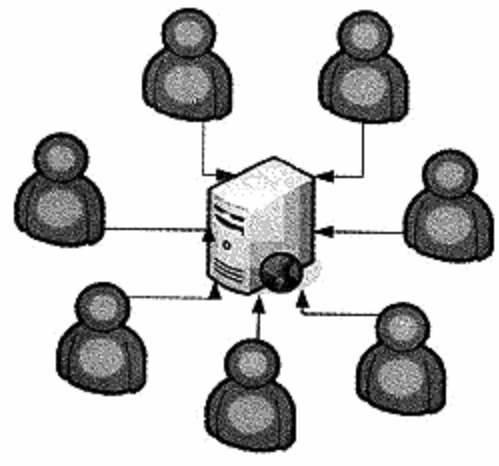


图 1-1 Web 的巨大优势——集中式部署

没错，桌面应用程序的特点是丰富而健壮，只要操作系统允许，几乎没有什么实现不了的；但是桌面应用程序的部署模型是它的瓶颈。要想保证数千个(即使不是几十万个)客户同时运行同一软件的同一版本，相应解决方案的复杂程度将是难以估量的。图 1-2 所示为一个典型的弹出窗口，该窗口正缠着用户去升级程序。

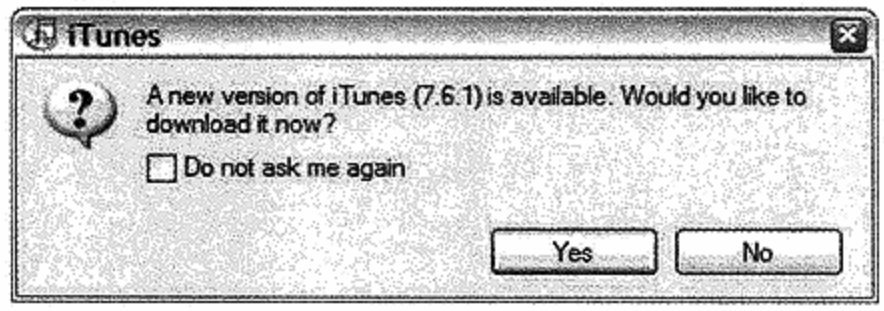


图 1-2 如果每周都要升级一次你会怎么办？每天一次呢？估计不会再用了

通常，桌面应用程序的开发人员都需要准备好支持多个版本的客户端连接到后端基础结构。对此，唯一可行的做法就是把大量增强和修正功能再集合到大的版本中。而且，这还未考虑支持多平台的情况下呈几何级数般增长的复杂性。

当然，基于 Web 的应用程序允许即时发布编写好的增强或修正功能。而且，这里的发布也只不过是改改数据库和复制粘贴一批最新文件而已。之后，所有用户就可以立即享受最新、最出色的功能了！

1.2 为快速发展付出的代价

这好像是个显而易见的问题。如果你就是一名 Web 开发人员，在享受这种优势的时候，付出了什么代价呢？

众所周知，技术升级换代的速度非常快。然而，Web 应用程序的集中式部署模型却深得大家的偏爱，其基于 HTML 的语言自 1999 年最后一次修订以来，一直还在广泛使用。这种现象难道正常吗？

或许读者从未站在一个使用 Model-Glue 框架的 ColdFusion 重要开发人员或者使用 EJB(Enterprise JavaBean)和 Struts 的 Java 开发人员的角度来看这个现象。实际上，在这么长的一段时间内，我们都忽视了一个关键要素——用户体验(见图 1-3)。对于随处可见的 Web 应用程序，用户心甘情愿地放弃了对可用性的要求。最终，我们让自己和用户接受了这个现实。

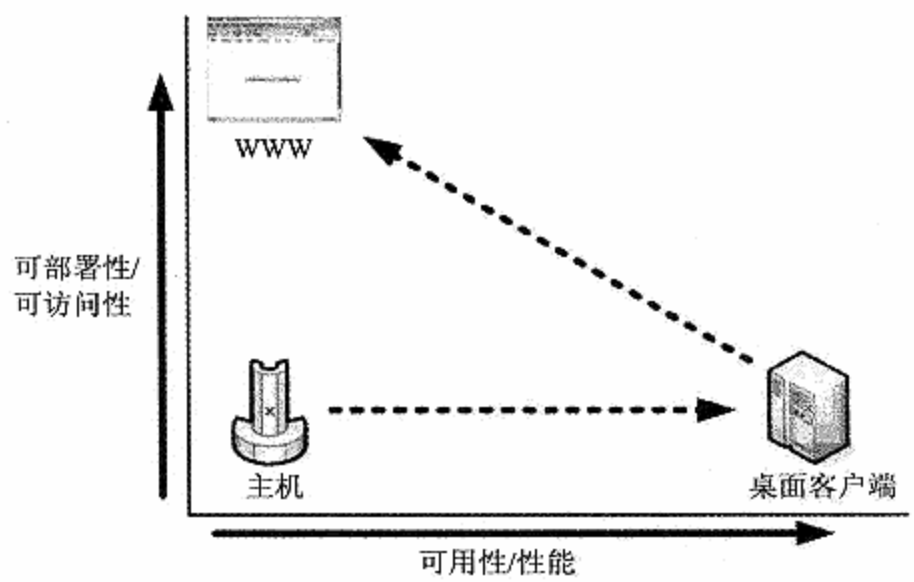


图 1-3 部署虽然方便，但可用性却大大降低

我们寄希望于使用 JavaScript

操作 CSS(Cascading Style Sheets，层叠样式表)的可见属性来解决问题。但是，结果并不理想。因为可用性问题的根源在于 Web 的本质——它的结构是为了支持文档分发机制而构建的。

开发人员投入巨大精力把尽可能多的应用程序逻辑转移到前端(客户端)，从而模仿客户端(类似桌面的)体验，但这也只能从表面上补偿一些可用性。而且，其中大部分的时间和精力，都花在跟平台的特性抗争上。

别忘了，Web 应该是与平台无关的。具有讽刺意味的是，放在客户端的逻辑越多，就要拿出越多的时间与浏览器的不兼容性抗争。这恰恰为富 Internet 应用程(rich internet application，RIA)搭建起大显身手的舞台。

1.3　RIA 解决方案

在以数据为中心的社会环境中，用户和商业组织都依赖于对信息的有效利用。用户只是想便捷地得到需要的信息，而商业组织，从留住客户的角度讲，则希望提供比竞争对手更好的用户体验，同时需要靠技术来保证员工的工作效率。

在某种意义上，我们会看到一个不可能兼顾的局面：用户向往愉悦的体验，而商业组织追求胜人一筹的周转速度和运营效率。这个靠传统技术无法解决的难题，正是投资于 RIA 的最好理由。

1.3.1　他们什么都想要

用户希望在任何一台计算机上都能访问自己的数据。他们也希望可以做像拖放这样的琐事。他们想得到丰富、流畅的图形化体验，而且音频、视频功能也要一应俱全。但是，他们不想时不时看到“请下载最新版本”的提示！

开发人员和软件团队何尝没有这些需求呢？及时将产品推向市场具有极高的战略意义，而软件开发和维护的成本却始终居高不下。不能保证所有用户同时使用相同的版本会带来极大的开销。开发人员希望专注于高效编写软件，不想为考虑让软件适应多个平台而分心。他们还盼着发布更新的过程简单而快捷。

图 1-4 概括 Web 客户端与桌面客户端在各个方面的优劣比较情况。

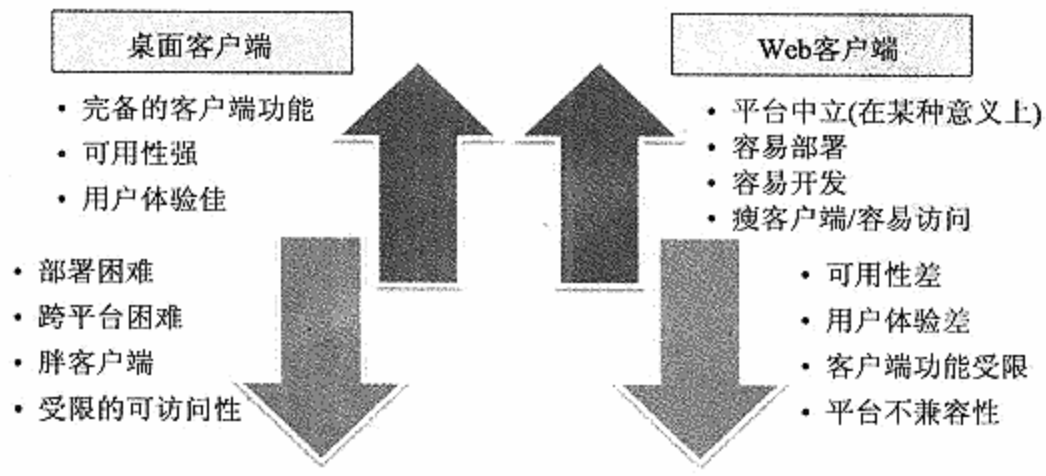


图 1-4　桌面客户端和 Web 客户端的优缺点对比——各自最突出的特点

但是问题(或挑战)也是机会，而这个机会恰好被 RIA 抓住了，它提供了两全其美的办法。

1.3.2 RIA 的解决之道

RIA 给出了兼具上述两种优势的解决方案。具体来说，RIA 技术既支持商业级的周转速度和快捷部署(通过集中式的 Internet 部署模型)，又能为用户提供桌面应用般的体验(见图 1-5)。

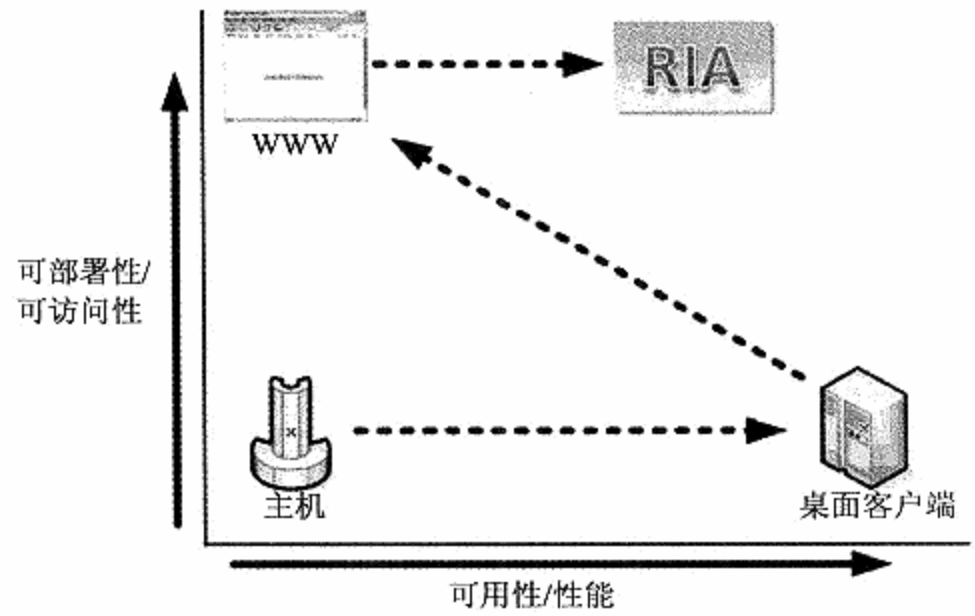


图 1-5 RIA 集合了两者的优点

开发人员通过 RIA 可以为用户提供实时应用程序般绚丽流畅的体验(与每次单击都要完全重载页面恰好相反)，而这正是满足用户需求的核心要素。

同时，部署及可访问性模式保持不变，即用户可以在任意一台计算机上加载这种应用程序，而且保证都是相同的版本。最关键的问题在于，这种应用程序实现了真正的平台无关，同一个应用程序无论在什么环境下都能使用户体验到一模一样的观感。

1.3.3 RIA 的工作原理

RIA 主要是通过为浏览器安装一个插件(作为本地运行时引擎)来达成上述目标。有了这样一个在不同操作系统的不同浏览器中可用的运行时引擎，就能做到平台无关。

运行时引擎是一个插件，因此可以依托浏览器运行。而把浏览器作为分发机制，则让这个插件具备了 Web 应用程序般的强大部署能力。

1.4 RIA 的竞争技术

现在，RIA 领域异常热闹，竞争性的技术环伺而立。其中处于领先地位的是 Adobe 公司的 Flex，与其正面对抗的竞争产品是微软公司的 Silverlight。另外，Sun 公司也打算在这个领域分一杯羹，该公司推出了一种新技术，叫 JavaFX。

下面简单介绍一下这些竞争性 RIA 技术。

1.4.1　Adobe 公司的 Flex

作为率先推出 RIA 技术的公司，Adobe 在推广 Flex 平台方面始终不遗余力。对于 Flex 3，Adobe 开放了这个框架的源代码；SDK(Software Development Kit，软件开发包)则从 Flex 2 起就免费了，而可选的 IDE 的价格定位也非常有吸引力。

Flex 具有下列优势。

- 它利用了几乎无处不在的 Flash Player，Flash Player 的安装率达到了 98%(找一台没有安装 Flash 的计算机可不容易)。
- 庞大的 Flash 社区及其知识资源。
- 与 Adobe 针对设计人员(Photoshop、Fireworks 等)、开发人员及服务器(ColdFusion、Media Streaming 等)推出的其他产品紧密集成。
- 领先其他技术 4 年时间。
- 框架源代码开放。
- SDK 免费。

但也有以下不足之处。

- 尽管 Flex 的打印功能还算令人满意，但仍有较大的改进空间(尤其是报告风格的打印方面)。
- 由于该项技术还相对较新，开发人员数量(约 15 万人)与.NET 和 Java 相比还偏少。
- RIA 应用程序从浏览器插件启动，因此该程序运行于插件的实例当中。这意味着弹出式窗口无法出现在 Flex 应用程序所在的主窗口之外(见图 1-6)。尽管 Flex 为此提供了补救措施(使用第三方工具)，但却不是平台内置的功能。

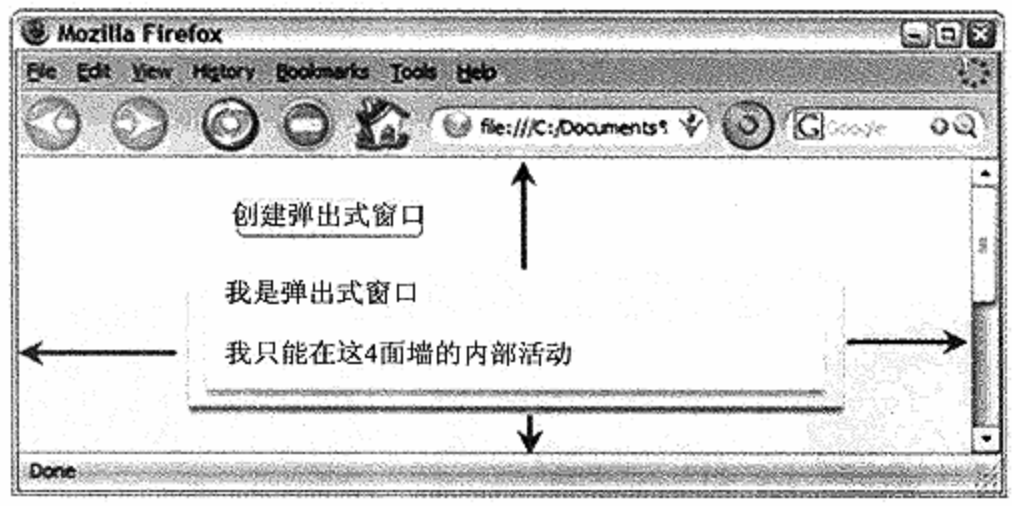


图 1-6　弹出窗口不能轻易超出运行该 Flex 应用程序的 Flash Player 实例的窗口

1.4.2　微软公司的 Silverlight

微软并不以最早提出创新性技术而著称，他们更擅长在别人创造出市场空间后，通过投入巨资获取统治地位。

Silverlight 尚不存在系列支持产品。但是从设计角度看，微软有一款名叫 Expression

Studio 的产品；从平台角度看，有一些支持 Windows 和 Apple OS 操作系统及 IE、Firefox、Safari 和 Opera 浏览器的基础。

微软借助 Silverlight 1.0 取得了快速进展，Silverlight 1.0 于 2007 年 9 月发布(2.0 版现已进入 beta 测试阶段)。2.0 版仍然致力于完善底层功能，包括布局及数据服务组件等 Flex 早期版本中就已经存在的功能。

Adobe 意识到端到端的系统开发生命周期的存在。Flex 应用程序的生命周期不是一定从 Flex 开发人员着手编码时开始，而应该从设计人员与用户体验团队模拟用户体验的时刻开始。此时，一般要用到 Fireworks、Photoshop 和 Flash 等工具。整个过程采用流水线方式，每一类用户在各自领域工作，同时还可以方便地与流水线上下的其他人员协作。

Silverlight 最大的亮点在于开发人员可以利用现有的.NET 框架、组件和语言。对于.NET 开发人员来说，使用 Silverlight 是件很自然的事。

1.4.3 Sun 公司的 JavaFX

此时此刻，JavaFX 还仅仅是一项声明。因此，除了静观其变之外，确实没有太多可陈述的内容。

然而，如果有读者想进行推测的话，恐怕要追溯一下 Sun 公司的历史。对于一家年销售收入达 130 亿美元的公司而言，它最为人们所津津乐道的产品不过两个：Java 和服务器。

曾几何时，Sun 公司有过一次借 Java 成为 RIA 领导者的机会，但当时没有把握住。因此，大体上可以把 JavaFX 看作第二次尝试。加之 Sun 公司与 Laszlo Systems 的战略合作关系，也有助于他们更快地迎头赶上。

Sun 是一家为技术人员提供技术的公司，之后，这些技术人员再向其他技术人员提供解决方案。Sun 公司的愿景及关注的领域非常有针对性，较少考虑一般用户的可用性问题。

看看 Sun 公司目前推出的工具，主要特点是功能强大且实用性强，而且假定用户已经具备了某一层次的技术专长。创意和设计绝对不是他们的强项，因此，为了取得成功，他们必须首先正视自己的缺点，然后再设法弥补。例如，为每个领域的顶级工具(如设计领域的 Photoshop)提供强大的集成支持，有助于扫清 Sun 产品在早期推广时面临的障碍。

Sun 最大的优势在于庞大的 Java 开发社区，社区成员也比较忠实于他们的平台。

Sun 最大的错误有可能是单纯地复制 Flex 和 Silverlight 的特性，拿不出独一无二的特色功能，或者说，只能以“模仿”别人技术的形象出现。

1.5 Flex 与 Web 应用程序

管理层、客户和相关团队的成员可能会站在纯 Web 手段的立场上提出反驳，但问题在于，大家讨论的并不是同一件事：Flex(RIA)不是富 Web 应用程序(RWA)。

厂商出于兜售自己产品的需要，会有意无意地混淆这两个术语，因此我们看不到相关的正式定义。不过，RWA 指基于 HTML 的 Web 应用程序，这种应用程序使用高级的 JavaScript 工具箱和 AJAX 技术(传输 XML 格式的数据)。

很多人都认为 RWA 属于 RIA，而事实上，这两种应用程序存在明显的差异。下面我

们就来解释 RWA 与 RIA 的差异。

1.5.1　Web 应用程序基于文档

Web 应用程序是基于已经有 8 年历史的 HTML 4.01 规范创建的，而 HTML 最主要的用途是分发文档。以下是几个与 Web 和 Web 应用程序有关的术语：

- Web 页面
- 书签
- 主页
- 使用索引页面的搜索引擎

Web 应用程序类似 *Choose Your Own Adventure*[1]系列中的一本书。你可曾想过要为自己在 Photoshop 或 SQL Server 中的某项操作加个书签吗？想过索引 Microsoft Project 的可执行文件吗？没有。这些都是实时运行的应用程序，当然做不到。

与 Web 应用程序有关的另一个重要因素是其渲染引擎(render engine)——浏览器。

1.5.2　浏览器的角色：中转站和文档渲染器

虽然通常使用浏览器来调用 Flex 应用程序，但浏览器相对于 Flex 应用程序而言只是一种传输机制，即根据 URL 访问或下载相应资源。

为弄清差异，读者不妨试用一下 Mindomo 中基于 Flex 构建的思维映射工具(mind mapping tool)(见图 1-7)。从中不难看出它是一个实时运行、功能丰富的应用程序。浏览器的高级用途主要有两个：一是访问和传输数据(信息)，二是渲染 HTML 文档。Flex 应用程序也要使用浏览器来加载。但是，仅凭 Flex 应用程序是在 Web 浏览器中加载的并不能说明它就是 Web 应用程序。

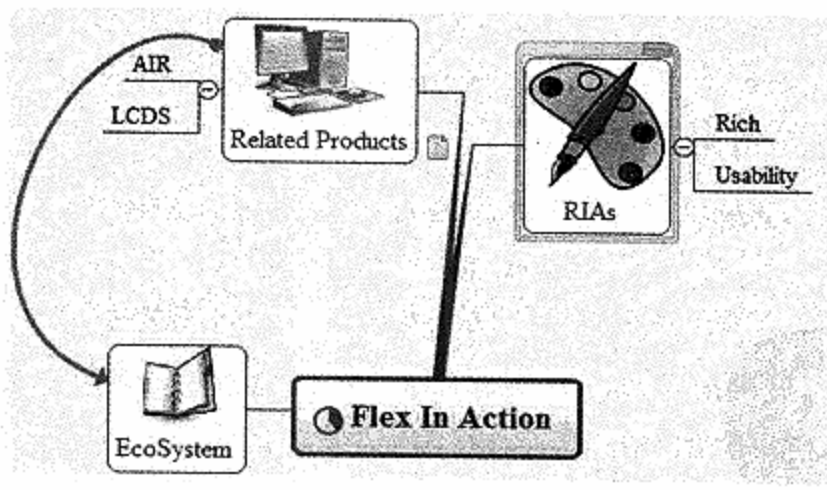


图1-7　使用Flex技术，Mindomo的思维映射工具可以让用户把构思形象化，支持团队在高度可视环境下的协作。请访问http://www.mindomo.com

1　*Choose Your Own Adventure* 是由 Bantam Books 出版的一套儿童游戏丛书。其中每一册书都以第二人称编写，故事情节根据读者的选择展开，选择不同，结局也不一样。例如，*The Cave of Time* 一书在叙述完故事之后，请读者选择故事主人公接下来的行动路线：如果你决定回家，请翻到第 4 页；如果你想继续等待，请翻到第 5 页(有关详细内容，请读者访问 http://en.wikipedia.org/wiki/Choose_Your_Own_Adventure)。在此用来比喻 Web 应用程序类似一本可以根据用户单击操作跳转到不同页面的书。

Flex 应用程序是真正的富 Internet 应用程序，它们是与平台无关、基于 Internet 部署的瘦客户端。Flex 支持多种传输协议，如文本/XML、Web 服务、RTMP/消息传递以及称为 AMF(Action Message Format，动作消息格式)的二进制格式。Flex 还具有稳健的图表引擎，内置支持视频流，等等。

此外，开发人员利用 Flex 的数据服务能力可以创建多用户、有状态(相对于 Web 的无状态)的应用程序，可以把状态变化推送到客户端(相对于 Web 应用程序通过轮询机制刷新页面)，而且能够实现记录集分页(有利于提高性能。假如要处理 1 000 条记录，可以每次只通过网络发送其中 50 条)。

有人可能会说，他们见过类似 RIA 的 RWA，还带有外观极为漂亮的用户界面小部件和效果。然而，RWA 是可以扩展的平台吗？

1.5.3 RWA——终点站

使用极为高端的 JavaScript 和 DHTML 技术，加上浏览器的 XMLHTTPRequest 功能，的确能够出现给人留下深刻印象的效果。但是，这种应用却是从文档平台中尽可能“压榨”出每一点可用性的结果。

当然，上述技术的组合如果运用得当，可以让 Web 应用程序使用起来更方便，或者成为我们所说的 RWA。

但是，AJAX 的瓶颈在于历史已长达近 10 年之久的 HTML 和 JavaScript。虽然 W3C 早就制定了改进它们的计划，但要指望近期有什么新突破是不可能的。自从 Web 应用程序诞生就存在的集合严重阻碍了 AJAX 应用程序的发展。更糟糕的是，不能通过扩展 AJAX 创建自定义集合。

而使用 Flex，则可以扩展任何已有的组件(无论是表单组件还是其他组件)，并为这些组件赋予新的功能。当然，开发人员也可以创建全新的组件。例如，HTML 和 Flex 都提供了一个复选框组件，即用于标记状态的开关。如果我们想实现一个能标记 3 种状态的复选框怎么办？图 1-8 所示的 Microsoft Excel 支持这种扩展的复选框组件，在本例中，表示当前选中的单元格已经被合并了。

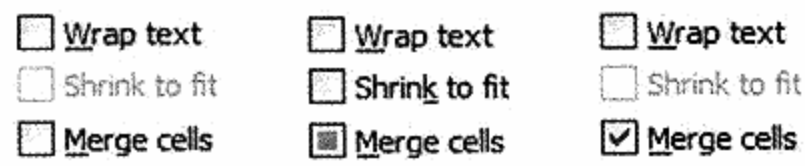


图 1-8 通过 Flex 可以创建 Microsoft Excel 中常见的 3 种状态复选框，而在 HTML 中则不存在扩展组件的选项

没错，在 HTML 中可以使用图像来仿造这种复选框，但这种情况并没有提供真正的表单字段(即包含供 JavaScript 评估相应选项的属性和值)。而且，在使用这种仿造技巧的情况下，用于复选框的任何 CSS 样式同样无效。

前面曾经提到过，给人留下深刻印象的创新源自最后一次“压榨”。但由于平台之间的不一致性，这些创新也必须付出代价。

1.5.4　跨平台问题

如果读者使用过 JavaScript，一定因支持多浏览器而头疼过。每种浏览器都有所不同，想要采用高端 AJAX/JavaScript 技术，则最后需要显著增加质量保证循环规模(图 1-9)。

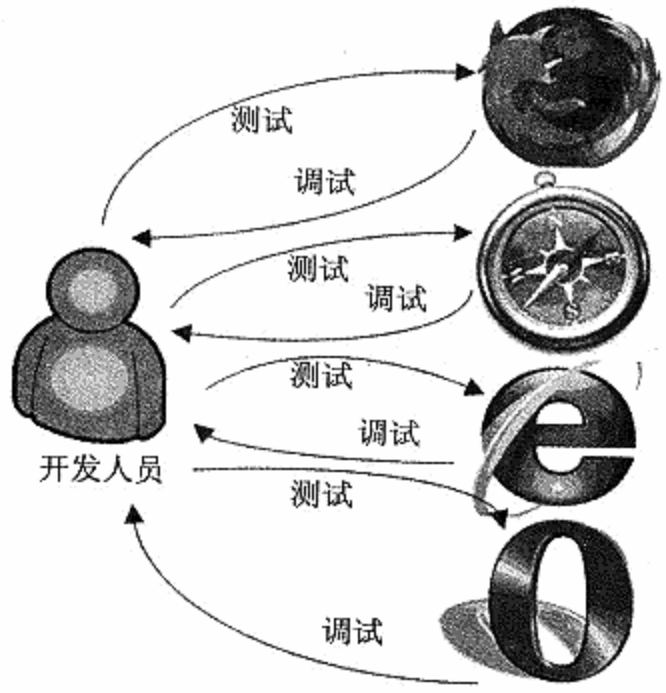


图 1-9　别忘了在使用据称免费的 JavaScript/AJAX 实现支持多浏览器时涉及的开发成本

仅这一个缺陷，就能在很大程度上削弱公司或客户实现既定 ROI(Return on Investment，投资回报)的能力，因为系统开发的最大成本是由开发时间决定的。

不可否认的是，jQuery 及 Adobe 的 Spry 等 JavaScript UI 框架确实可以节省一些开发时间。但无论如何，测试的需求仍然存在，特别是在自定义开发的情况下。

而 Flex，由于利用了无处不在的 Flash Player，抽象了各种平台的特殊细节，因此开发人员只需关注一个平台——被称为 AVM(ActionScript Virtual Machine)的 Flash Player 运行时。如果能在一个平台中运行，就可以在所有平台中运行，而且用户体验完全相同。

到目前为止，我们讨论的因素涉及了开发过程和技术性能，但作为一种丰富的技术应该……当然，还应该丰富！

1.5.5　何来丰富性

作为一种 RIA 技术，首先必须满足 RIA 中的 R(Rich)，即丰富性。JavaScript/AJAX 和 HTML 的丰富性又体现在哪些方面呢？

- 内置支持声音文件播放吗？
- 内置支持视频流播放吗？
- 能够与视频设备(如 webcam)交互吗？
- 能够实现诸如拖放之类的基本操作吗？

举例来说，你打算为公司网络运营中心的员工开发一个监控工具，那么如果应用程序能够根据触发事件的严重程度发出不同的警报声，或者有一个支持语音聊天和视频会议的业内领先的企业级协作工具，效果肯定会更好。

1.5.6 AJAX 的通信限制

AJAX 只支持一种数据传输方式——通过 HTTP 传输文本。在过去，文本一直采用 XML，而最近的趋势则是使用 JSON，以减少客户端处理的工作量。但是，无论是哪种格式，由于庞大的文本数量，都难以在传输大量信息时获得较高的效率。

试试通过 XMLHTTPRequest 传输 5 000 条记录，然后再用 JavaScript 解析这些数据。这正是 James Ward(Adobe 的技术推广人员)做过的一个试验，试验结果发布在他的博客(http://www.jamesward.org/ wordpress/2007/04/30/ ajax-and-flex-data-loading-benchmarks/)上。

如图 1-10 所示，Flex 使用其二进制格式 AMF3，要比使用 XML 或在 AJAX 中用 SOAP 快了 10 倍。导致这个结果的原因有两个：首先，如前所述，XML 本身的冗长导致了相对很小的消息有效负荷开销；其次，JavaScript 是一种解释型语言，而 Flex 应用程序则被编译为与平台无关的字节码。

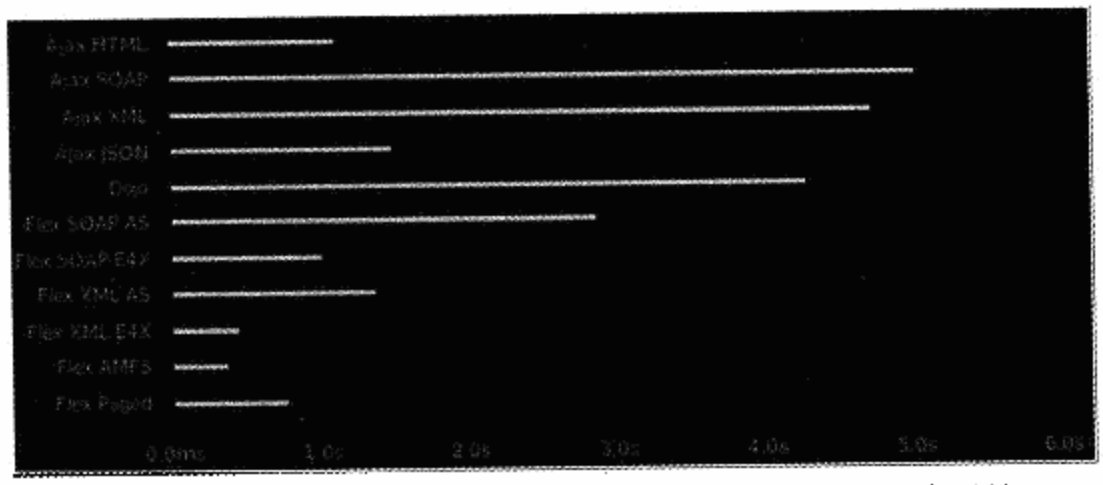


图 1-10 Flex 的二进制协议与 AJAX 的冗长方法的速度对比

除了速度之外，Flex 的压缩二进制格式还具有其他优势。更少的开销意味着更少的资源(例如使用的内存和网络)占用。

注意：

Flex 和 Flash 也支持通过 HTTP 或 SOAP Web 服务实现文本数据的简单传输。

尤其在大规模应用程序或者需要交换大量数据的应用程序中，AMF3 提供了明显优势。

秘密武器——AMF3

AMF3 使用的协议正是 AJAX 所没有的秘密武器。借助 AMF3，开发人员可以使用 RTMP(Real Time Messaging Protocol，实时消息传递协议)，该协议基于发布-订阅，而非 Web 和大多数网络应用程序中常见的请求-接收模型。

RTMP 协议允许客户应用程序注册感兴趣的特定主题，并维护与服务器的活动连接。当新的符合指定条件的主题发布时，服务器会将其推送到客户端。这种模式与 AJAX 客户端重复地向服务器轮询新信息(即使服务器未发生任何变化，也是如此)相比，显然更有效率。

使用 RTMP 和推送模型的一个经典示例是股价跟踪工具，这种工具只有在特定股票的

股价发生变化时才会传送变化。

再提一次，当涉及大规模应用时，更能体现这种模型的巨大优越性。想象一下有几十万名用户的情景。难道你愿意在没有新数据需要传输的大多数情况下，让这些用户每 5 秒钟轮询一次，不断查询数据库吗？

1.5.7　并行不悖

多年以来，JavaScript 和 HTML 并没有被迫演化，这就是它们长寿的明证！过去，它们经受了时间的考验；未来几十年，必将以这样或那样的形式继续繁荣兴盛。这一切全都缘于 Web 的伟大本质——分发平台中立文档。

Web 应用程序会在未来占有一席之地，但有一种趋势十分明显：未来的用户也会越来越苛刻。Web 应用程序无能为力的领域，必将是 RIA 技术大显身手的舞台。

在 RIA 领域中，Flex 是先行者并仍然居于领导地位。与 RWA 相比，Flex 具有很多明显的优势。随着对 Flex 理解的加深，相信读者也会充满信心。并且，能够向管理层和客户展示这些优势——至少能高度概括地介绍。接下来，我们将进一步了解 Flex。

1.6　熟悉 Flex

Flex 是以编程方式(即通过代码)利用 Flash 平台构建 RIA 的一种手段。Flash 因促销网站经常使用的交互式横幅广告、网页中炫酷的动画部分和互动营销体验而驰名。

除了 Flash Player 无处不在这个优势外，Flex 在利用这种业已得到认可而且相当成熟的技术方面，同样占有先机。不过，这并不意味着开发人员会就此与 HTML 隔绝；相反，Flex 应用程序也可以通过 JavaScript 与 Web 应用程序交互，但仍是大 Adobe 技术体系的一部分。

1.6.1　利用 Adobe Flash 的优势

支持 Flex 应用程序运行的核心是 Adobe Flash Player。它是一个极其强大、快捷、轻量级且平台无关的运行时引擎，基于面向对象的语言 ActionScript。换句话说，它带给 Mac 用户、Windows 用户及其他平台的用户(即智能手机或 PDA 等)的体验不会有任何差别。

Flex 应用程序赖以运行的 Flash Player 能够处理大量数据，具有稳健的二维图形渲染和多线程处理能力，并且支持多种通信协议。通过支持多媒体格式(如视频流、图像和音频)，Flash 成就了 RIA 的丰富性。

结果就是开发人员能够以此创建丰富多彩且类似桌面的用户体验，同时能够发挥出自己的聪明才智和创造力。从另一个角度讲，开发人员也拥有为终端用户优化工作流的独特手段。

为什么不直接使用 Flash

在 Flex 出现之前，机智的 Flash 开发人员就在开发 RIA 了。但是，对于那些具有开发背景的人来说，在进入 Flash 领域后，让他们转而采用 Flash 的思维模式做开发是相当困难的。毕竟，Flash 植根于动画设计(见图 1-11)，其设计环境的构成要素是时间轴、层、帧、每秒帧数等。让熟悉逐行编写代码的开发人员把应用程序想象成一个 Flash 影片，总归没有那么自然。

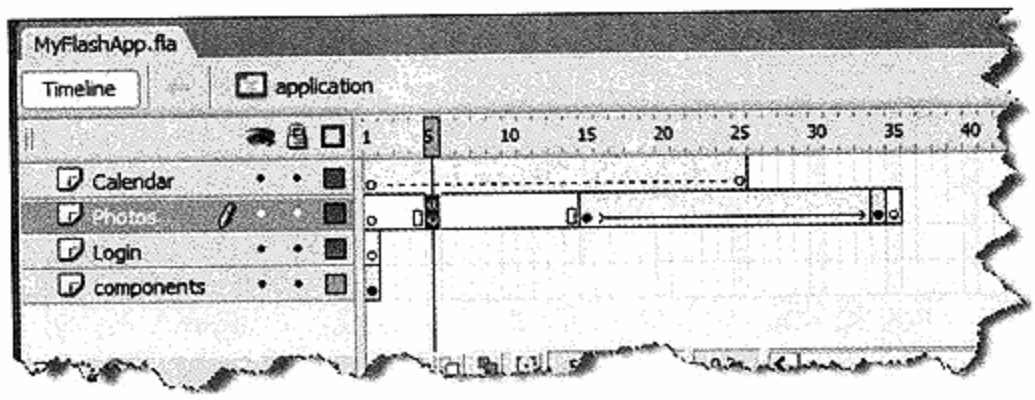


图 1-11 Flash 早就可以用来开发 RIA 了，但很多人却无法适应基于时间编写代码的方式

况且，即便是 Flash 老手，要在纯粹的 Flash 环境中做开发，也远不及在其他开发环境中效率高(很大程度上因为处理变化所需要的繁重工作)。

虽然 Flash 和 Flex 都可以作为独立的应用程序运行，但借助 JavaScript 这个桥梁，它们又都可以与 Web 应用程序交互。

1.6.2 Flex 与 JavaScript 可以一起使用

读者如果开发过基于 HTML 的 Web 应用程序，使用过 JavaScript，那么在学习 Flex 时会发现，其 ActionScript 语言与 JavaScript 极其相似。

原因在于，JavaScript 和 ActionScript 都是基于 ECMAScript 标准的实现。因此，熟悉 JavaScript，也就相当于熟悉 ActionScript。还有一个有趣的珍闻，Flex 的 ActionScript 还是实现了最新 ECMAScript 4 标准的第一个产品语言。

开发人员使用 Flex 不会受到技术上的限制。也就是说，并不是非 Flex 不可。RIA 和 RWA 虽然不同，但 Flex 允许这两种应用程序之间交互操作。为此，Flex 引入一种称为 External API(外部 API)的功能，从而使 JavaScript 应用程序能够与 Flex 应用程序通信。

针对 AJAX 环境，Flex 引入另一个名叫 Flex-AJAX Bridge(也称 FABridge)的功能，以方便 AJAX 与 Flex 应用程序的整合。如果读者已经拥有一个投入巨资开发的 AJAX 应用程序，但又希望使用 Flex 的某些功能，可以通过 Flex 在 AJAX 应用程序中生成交互式图表。

图 1-12 所示的 Nitobi 商业智能报告工具，就是在融合了 Web 和 Flex 技术的基础上开发的一种混合应用程序。

不过，这种协调能力还不止于此：Flex 是 Adobe 公司的产品，因此 Flex 也是一个大的

技术体系中的重要组成部分。

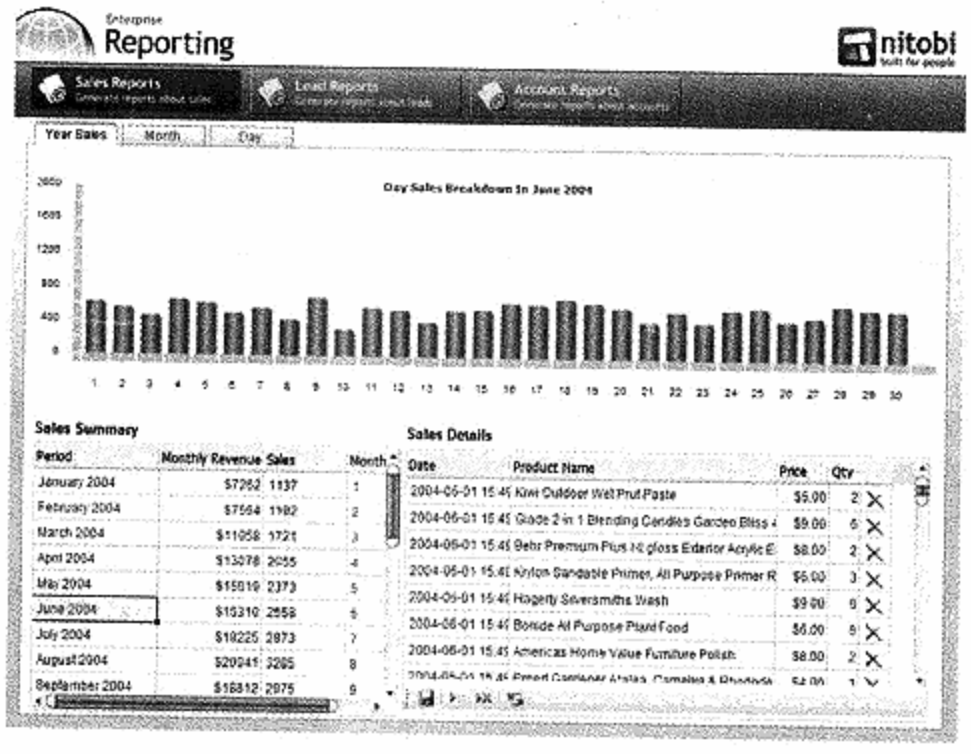


图 1-12　商业智能与报表工具公司 Nitobi 在其 AJAX 应用程序中通过 FABridge 来利用 Flex 的图表引擎

1.6.3　Flex 技术体系

Flex 内置的卓越技术只是它的优势之一。作为 Adobe 的技术，Flex 也是一整套技术体系中的一员。因此，使用 Flex 的组织可以实现从设计到开发再到部署的顺畅的工作流程。图 1-13 所示为 Flex 在整套图形工具中所处的位置。

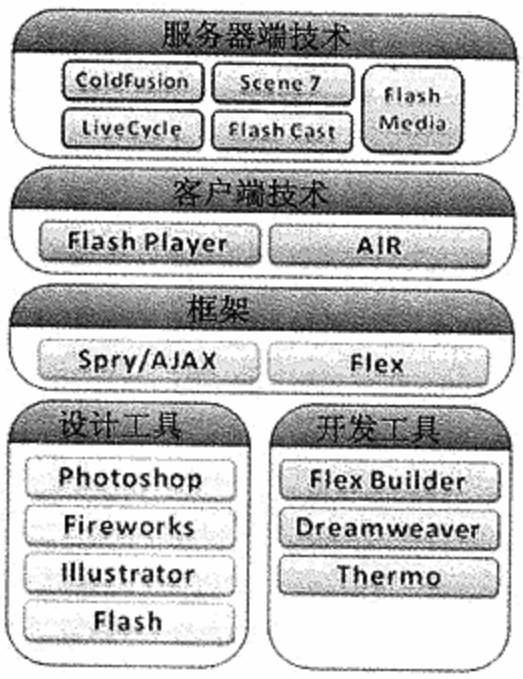


图 1-13　Flex 是一个大的技术组群中的一员

在常规的 Web 开发过程中，设计人员与开发人员之间的沟通最容易浪费时间。有读者可能知道，设计人员使用诸如 Photoshop 的工具设计应用程序的视觉元素，然后开发人员努力将设计好的图片切割成更小的单元，并生成 CSS。

但是，在 Flex 的技术体系中，设计人员可以导出主题(也叫皮肤)，而开发人员可以不周详考虑设计因素而直接导入主题。更令人期待的是，Adobe 还将发布一个名叫 Thermo 的产品，使用这个产品可以创建设计方案，然后自动生成有效的 Flex 代码。

另一种跟 Flex 有直接关系，而且也常见于各种媒体宣传的客户端技术是 AIR。AIR 可以让 Flex 应用程序作为本地桌面应用程序运行。

1. 简单了解 AIR

AIR(即以前的 Apollo)，指的是 Adobe Integrated Runtime(Adobe 集成运行时)。这种技术允许更进一步，把 Flex RIA 转换为所谓的 RDA(Rich Desktop Applications，富桌面应用程序)。尽管转换后的应用程序还没有正式的称谓，但可以把这种技术称作“混合桌面 Internet 应用程序”(Hybrid Desktop Internet Application)。

Flex 从浏览器启动，而由于安全原因，它执行诸如访问硬盘数据或操作外围设备(如扫描仪)的操作。AIR 把 Flex 应用程序从浏览器中解放出来，让它能够直接在桌面上运行(见图 1-14)，因而能为用户提供完美的桌面体验。通过 AIR 可以实现的功能如下。

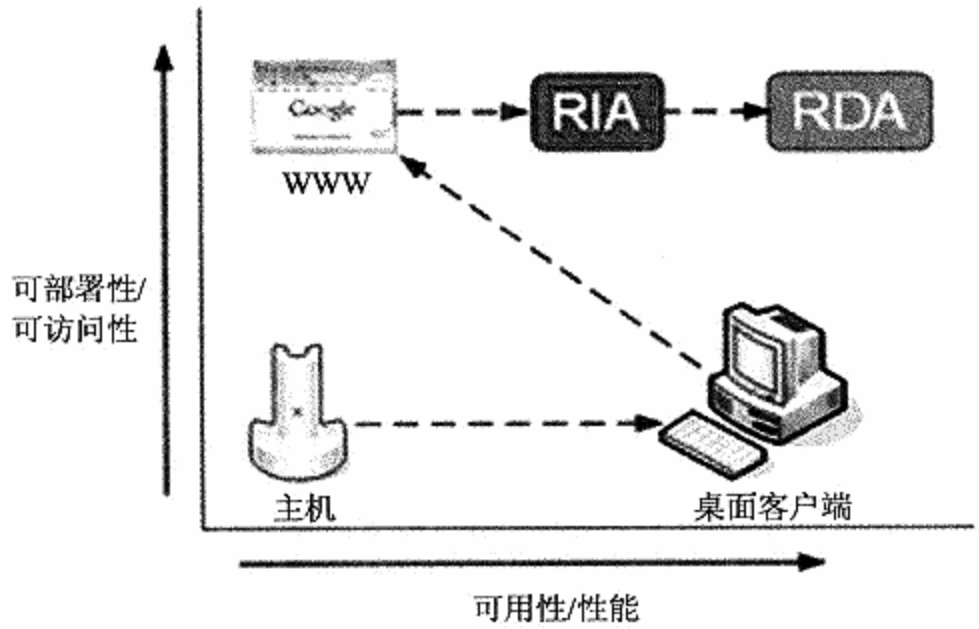


图 1-14 RDA 较 RIA 前进了一步，它能够实现完美的桌面体验。RDA 独立于浏览器存在，能够像桌面应用程序一样访问操作系统的剪贴板和本地文件系统

- 访问操作系统剪贴板中的信息。
- 从桌面到应用程序的拖放操作。
- 创建“无边框”应用程序(即不需要浏览器的方形边框)。

AIR 还提供了其他功能。

- 内置了数据库服务器。
- 透明且自动的软件更新机制，确保所有人都使用相同的版本。
- 内置 HTML 渲染引擎。

这是一个革命性的平台。如果读者希望构建不受浏览器约束的 Flex 应用程序，我们推荐阅读 Manning Publications 出版的 *Adobe AIR in Action*(Joey Lott、kathryn Rotondo、Sam Ahn 和 Ashley Atkins，2007；ISBN：1933988487)一书。

2. BlazeDS 简介

BlazeDS 是 Adobe 的产品 LCDS(LiveCycle Data Services，生命周期数据服务)的简化版。作为中间层的服务器组件，它就像是后端组件与服务(Java 和.NET 等服务器端技术)的中间人，或者数据库服务器与 JMS(Java Message Service，Java 消息服务)等消息技术的连接器。

BlazeDS 的功能如下。

- 使用二进制 AMF3 协议将后台数据传输到 Flex 客户端。
- 高性能数据传输。
- 使用 HTTP 和 AMF3 的实时数据推送(即它能在新数据产生时通知 Flex 应用程序，不需要 Flex 应用程序轮询检查新数据)。
- 发布/订阅消息传递(通过一种叫做长轮询的技术，RTMP 只能在 LCDS 产品中使用)。
- 对取自数据库的记录集分页(可以一次处理 50 条记录，或处理查询的前/后 10 条记录)。
- 数据同步(当 Flex 应用程序在客户端修改了某些数据后，它会自动通知 BlazeDS 实时更新数据库中的相应记录)。
- 最棒的是，BlazeDS 是免费的！作为 Adobe 的另一款开源软件，BlazeDS 可以基于 LGPL v3 许可分发并供使用。

1.6.4　Flex 的工作原理

Flex 的核心是免费的 SDK(Software Development Kit，软件开发工具包)，SDK 提供了构建 Flex 应用程序的框架；简言之，这个 SDK 中包含着即装即用的库和编译器。

在 SDK 之上，是基于 Eclipse 的名为 Flex Builder 的 IDE(Integrated Development Environment，集成开发环境)。因此，与使用 Flash 编辑器创建 Flash 应用程序不同，创建 Flex 应用程序要使用 Flex Builder。

Flex 由以下两种编程语言组成。

- 基于 XML 的 MXML 标签语言(没有人知道 MXML 代表什么，不过人们普遍认可的两种假设是 Macromedia XML 和 Magic XML)。
- ActionScript 脚本语言。

在使用 Flex 进行开发时，需要同时使用这两种语言。MXML 负责应用程序核心内容的主要布局(可视组件)，ActionScript 用于以脚本形成编写驱动应用程序需要的所有逻辑。

顺便说一句，MXML 在后台会被编译成 ActionScript 代码。也就是说，只使用 ActionScript 就可以创建完备的 Flex 应用程序(毕竟，最后都会编译成 ActionScript)。

提示：

刚接触 Flex 的读者可能分不清何时该用 ActionScript，何时该用 MXML。一个简单的经验法则是将 MXML 想象成 HTML，因为可以通过它构建应用程序的初始布局；把 ActionScript 想象成 JavaScript，因为可以通过它为应用程序加入逻辑思维能力。在熟练掌握 ActionScript 之后，只使用 ActionScript 就可以创建完整的应用程序。

开发时使用这两种语言，但开发完的应用程序通过编译会生成一个可执行文件，而要部署到 Web 服务器中的就是这个可执行文件(见图 1-15)。代码清单 1-1 中的简单例子展示了这两种语言之间的关系。

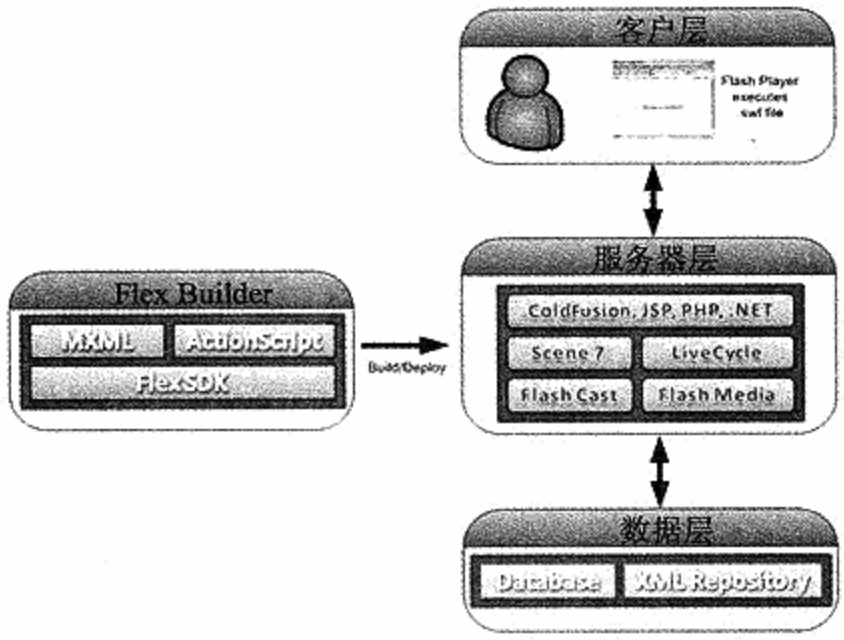


图 1-15 使用 Flex Builder 编译应用程序，然后再将其部署到服务器中

代码清单1-1 简单的Flex应用程序例子

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
   <![CDATA[
    import mx.controls.Alert;
    public function handleEvent():void
    {
     mx.controls.Alert.show("Event handled");
    }
   ]]>
  </mx:Script>
  <mx:Button label="Click on me" click="handleEvent()"/>
</mx:Application>
```

❶ ActionScript用于编写逻辑脚本

❷ MXML用于组件布局

即使读者从未看到过 Flex 代码，可能也会觉得这个示例似曾相识。原因之一恐怕就是基于标签的 MXML 语言❷衍生自 XML。原因之二则是其中的逻辑脚本❶很像 JavaScript。

下面我们简略地谈一谈 Flex 应用程序开发过程的典型生命周期。

开发生命周期

典型 Flex 应用程序的 SDLC(Systems Development Life Cycle，系统开发生命周期)如下。

- 使用 Flex Builder 或 SDK，在本地开发环境中编写 MXML 和 ActionScript 代码。
- 需要测试时，使用 Flex Builder 或 SDK 编译代码，编译输出的通常是.swf(发音为“swiff”)文件。
- 通过浏览器启动此文件，此时需要调用 Flash Player 插件；然后应用程序开始执行。
- Flex 应用程序通常要与服务器层交互以便交换数据。
- 准备发布应用程序时，将.swf 及相关文件(如图像)发布到 Web 服务器；然后用户就可以通过一个 URL 调用该应用程序了。

与使用 ColdFusion 和 PHP 等应用服务器不同，发布 Flex 应用程序不需要将源文件发

布到生产环境，只需发布编译后的文件(类似 Java 中的.class 文件，但 Flex 应用程序也包含运行应用程序需要的所有库)。

1.6.5　事件驱动的环境

一切都与事件有关。Flex 是一个事件驱动的开发框架，这很可能与有些读者原来的开发背景相去甚远。

在传统的 Web 开发技术中，事件表现为单击链接或提交按钮等操作。然后，服务器响应并执行请求的功能。结果会显示一个网页或者发回字段数据。

如果读者有 Web 应用程序的开发经验，那么肯定知道 JavaScript 可以对用户的某些手势做出响应，例如用户的鼠标在页面上的某个条目上移动时，会改变条目背景颜色，来突出显示该条目(图 1-16)。

```
<a href="" onMouseOver="alert('Howdy!')">mouse over me</a>
```

事件触发器　　事件处理程序

图 1-16　与 JavaScript 类似，Flex 中的事件同样由触发器和处理程序构成

这里的手势就是事件！只不过这个术语在传统 Web 应用程序中使用得没有那么广泛而已，毕竟传统应用程序的主体还在后端。不过，就像前面列举的 JavaScript 例子一样，Flex 是一种客户端技术，也就是说，所有操作都会发生在用户这一边。

Flex 应用程序完全是由事件驱动的，即会有一种机制触发某些事件，然后另一种机制响应这些事件并进行相应处理。事件驱动的应用程序中包含的这两种机制分别如下。

- **事件触发器**：导致事件发生。例如，用户把鼠标移到一个按钮上、应用程序装载完成、数据从 Web 服务返回，等等。
- **事件处理程序**：负责响应事件的处理程序。例如，调用函数改变用户界面元素、提交输入表单等；处理程序就是所谓的逻辑代码。

鉴于有的读者比较熟悉传统 Web 技术而对上述机制有些陌生，因此本书将循序渐进地讨论事件的用法，但你需要摆脱 Web 应用程序生成页面的概念的约束。一句话，Flex 应用程序加载完毕后，我们要做的就是捕获事件，然后做出响应。

1.7　Flex 3 的新功能

从诞生至今，Flex 技术的发展速度是非常惊人的。最初，Flex 是一种纯粹的服务器端技术，沿袭了大多数服务器端 Web 技术惯例。

Flex 2 对这门新技术进行了重大改进。具体来说，Adobe 把它分成两部分：客户端部分由应用程序框架和编译 Flex 应用程序的工具构成；服务器端部分是数据桥(data bridge)，已演化为 LCDS/BlazeDS。Flex 2 的改进还涉及技术上的重写——正是这次重写为 Flex 成为业内领先的产品平台以及将来的发展打下良好基础。

Flex 3 提供了许多重要的新功能，如下所示。

- **分析程序(Profiler)**：用于查看应用程序占用 CPU 和内存的情况，以及占用位置。

- **重构机制**：支持方便地重新命名函数、变量、类等几乎所有标识符。
- **向导功能**：提供许多引导开发人员与后端应用服务器建立连接的步进式向导程序，并能同步生成相应代码。
- **开发工作流**：允许设计人员更方便地处理应用程序的图形方面，能够大量节省开发人员把图形设计转换为技术实现的时间。
- **增强的图表功能**：支持 Adobe 提出的面向数据的实现。借此可以精确控制所要展示的信息及展示方式，包括用户与数据点的交互。
- **PFC(Persistent Framework Cache，持久性框架缓存)**：能够在客户端缓存 Flex 应用程序框架，从而把应用程序变小(因为不再需要包含支持性组件)。
- **DataGrid 组件**：显示表格数据，此功能很受欢迎。高级 DataGrid 组件还支持多列排序和跨列显示等附加功能。

在 Flex 框架和 BlazeDS 产品开源的背景下，Adobe 技术体系也在稳步向前发展。并且，Adobe AIR 几乎也在同时发布了。

1.8 小结

从历史角度看，Web 作为一个快速部署且半中立的平台牺牲了其可用性，但这是以损害用户体验为代价的。令人不解的是，用户(也包含我们自己)对这一切居然毫无怨言。每当谈起 Web 应用程序，我们都觉得受到某种限制很正常。

在如今这个信息爆炸的时代，牺牲可用性的代价超过其好处只是时间问题。用户受到折磨，公司也在追求生产效率和工作效率的过程中付出代价。

RIA 以事实证明鱼与熊掌能够兼得。从开发角度看，平台中立基础上的集中式部署模型有助于加快升级速度。从用户角度看，能够享受到类似桌面应用程序的体验，而且可以使用不同的设备访问应用程序，并享受到流畅迷人的体验。

Flex 加入 RIA 的行列，它以无处不在的跨平台 Flash Player 为依托，提供一种以编程方式创建 Flash 应用程序的新模式。这种模式使用 Flex 的核心语言——MXML(即 XML 模板语言)和 ActionScript(脚本语言)。

当前，随着第三个主版本的发布，Flex 继续保持 RIA 领域的领先地位。虽然微软仍然紧追不舍，但它没有 Flex 的技术体系——涵盖了软件开发端到端工作流的一个大型技术环境。从设计、开发到部署，Adobe 为每个阶段都提供了相应的环境和工具。

第 2 章讨论如何建立开发环境，帮助你熟悉该语言并学习如何创建 Flex 应用程序。

第2章 从零开始

2

本章内容：

- 了解 Flex 工具
- 创建第一个 Flex 应用程序
- 理解 MXML 的原理
- 使 ActionScript 配合视图

在编写 Flex 代码之前，首先要建立工作环境。Flex 提供了免费开源的 SDK，可供我们开始工作并创建 Flex 应用程序。但是，真正的开发人员往往首选基于 Eclipse 的 Flex Builder IDE 作为开发环境。

本章介绍如何使用 Flex 开源工具，来充分发挥有限预算的作用。此外，还会拿出相当篇幅介绍 Flex Builder 的诸多功能。然后我们讨论如何开始创建 Flex 应用程序。

Flex 开发人员的大多数工作时间都用在 Flex Builder 工作上。但在考虑 Flex Builder 之前，读者恐怕会问：不花钱能做 Flex 开发吗？

2.1 经济的 Flex 开发方式

有了 Flex 免费开源的 SDK，就没什么可以阻碍你开发 Flex RIA。Flex SDK 中包含把源代码转换为 Flash(.swf)文件的编译器，也包含处理应用程序组件及其核心组件(包括诸如按钮的视觉组件)的 Flex 框架。

我们的目标是轻松地开发 Flex 应用程序，而可行的方案就是排除最不常见的做法，并且专注于使你脱颖而出成为一名成功 Flex 开发人员。

在选择使用 SDK 的情况下，轻松开发很大程度上只能是一种理想，很难实现——因为使用 SDK 做开发的复杂程度远高于使用 Flex Builder。如果读者目前只是出于个人目的学习 Flex(想多掌握一门技术)，或者正准备向公司推荐 Flex，但公司并没有安排预算，免费方案还是值得探究的。

注意：

Flex Builder 有一个 60 天的免费试用版，通过试用评估可以判断它是否物有所值。

本节的目的是让读者对使用 SDK 开发涉及的问题有一个认识，从而对自己是否适合采取这种手段得出结论。如果使用 SDK，至少要考虑如下两个问题。

- **建立编译环境**：下载免费 SDK 后，可以通过命令行界面(如 MS-DOS 窗口)构建应用程序。
- **建立编辑环境**：可以使用简单的文本编辑器，但更高效的环境当然更好。

首先准备将源代码转换为应用程序的编译环境。

2.1.1 建立编译环境

可以在 Adobe 网站上下载 zip 文件格式的 Flex SDK。下载和部署这个文件以及配置命令行环境的步骤如下。

(1) 编译器需要 JRE1.4.2_06 或更高版本，因此如果系统里没有安装，就要先到 http://www. java.com/en/download/manual.jsp 下载。

(2) 导航到 Adobe 网站中的 Flex SDK 页面 http://www.adobe.com/go/flex3_sdk。在接近页面底部的位置找到下载免费 SDK(大约 80MB)的选项。

(3) 然后把下载的 zip 文件解压缩到选定位置。一般来说，接近根目录的位置比较方便(例如，在 Windows 中可以放到 c:\flex 目录下)。

(4) 如果桌面上安装了当前版本的 Flash Player，需要在继续下一步之前删除它。

(5) 安装调试版 Flash Player。它位于 runtimes/player 文件夹中。Windows 系统需要安装两个主要文件。

- Flash Player 9 ActiveX.exe，这是针对 IE 的调试播放器。
- Flash Player 9 Plugin.exe，这是针对其他浏览器(如 Firefox)的调试播放器。

(6) 把 bin 文件夹添加到 Path 环境变量中，以便将来不必提供完整路径(图 2-1)。

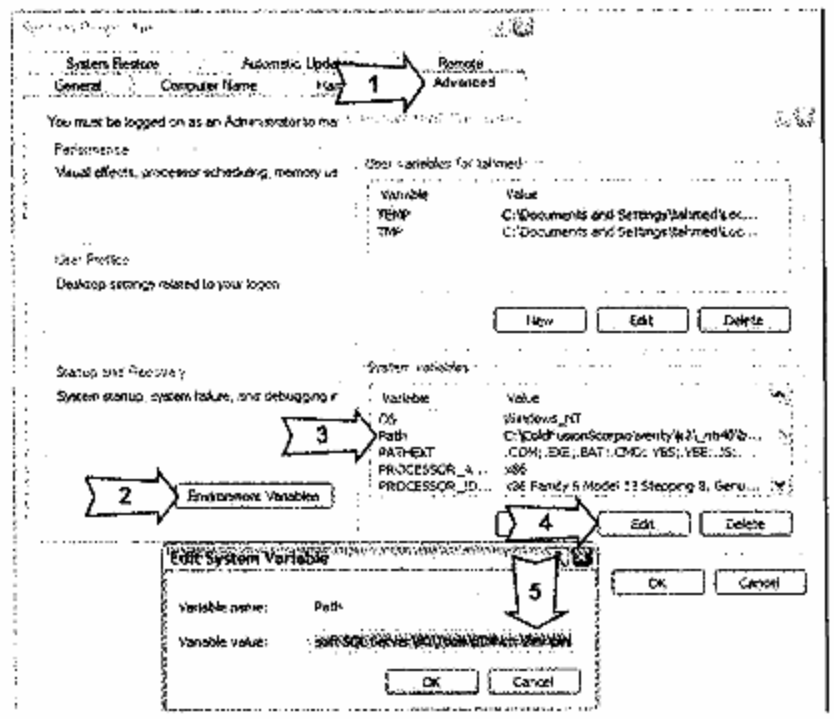


图 2-1 按照图示步骤把 SDK bin 目录添加到 Windows 的路径中

- 在 Windows XP 中，打开 Control Panel | System | Advanced。

- 单击 Environment Variables 按钮。
- 编辑 Path System 变量。
- 将安装文件夹\bin 添加到路径末尾(在这里安装文件夹是 c:\flex\bin)。

(7) 验证上述设置是否有效。

- 在想好保存 Flex 项目的位置创建一个文件夹，如 c:\apps。
- 在 c:\apps 中创建一个名为 HelloWorld 的文件夹。
- 在 c:\apps\HelloWorld 中创建一个名为 HelloWorld.mxml 的文本文件(可以使用记事本)。
- 在 HelloWorld.mxml 文件中输入下列代码。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"/>
```

- 打开命令行界面。在 Windows 中，打开 Start | Run，输入 cmd。Macintosh 或 Linux 用户打开相应的命令解释程序。
- 输入以下命令：`mxmlc.exe HelloWorld.mxml`。
- 如果一切顺利，将看到类似图 2-2 所示的结果。

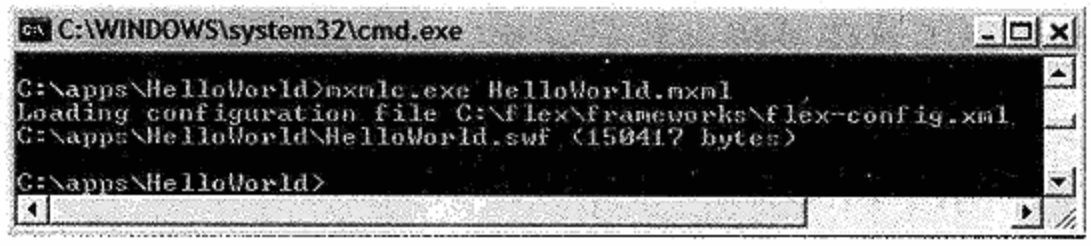


图 2-2 测试一下以确保能够编译 Flex 应用程序

也就是说，HelloWorld.mxml 所在的文件夹会生成一个名为 HelloWorld.swf 的新文件。双击可以打开这个新文件，如果打开后看到的内容与此不同也不用慌。看到蓝灰色的背景说明是正常的。因为我们现在的目的只是要验证编译器能否正常执行。

在知道了能够编译 Flex 应用程序后，也就意味着 SDK 的安装没有问题。下一步是建立编辑环境，即选择一种编写代码的有效方式。

2.1.2 建立编辑环境

除非你对 Linux 中的 VI 或 Windows 中的记事本情有独钟，否则应该选择一种最合适的代码编辑器。对于 Flex 开发，我们推荐流行的 Eclipse 编辑器。下载地址为 http://www.eclipse.org/。

Eclipse 是一个基于插件的开源编辑器和 IDE 框架，即它能很好地支持代码编辑，还允许第三方通过模块扩展其功能。商业 Flex IDE(Flex Builder)是基于 Eclipse 构建的；假如读者打算购买 Flex Builder，可以下载相应的 Eclipse 插件，以便在已有构建的基础上添加需要的功能。

Aptana Studio

接下来，应该考虑获得 Aptana Studio，这个编辑器既有商业版也有免费版。免费版叫做社区版(Community Edition)，可以从 http://www.aptana.com/studio/下载。

这个 Eclipse 插件的大量功能可以在 Aptana 网站上查到。不过，其中最适合代码编辑的组件有：

- Adobe AIR 开发
- AJAX 库
- HTML/CSS/JavaScript 开发

注意：

Eclipse 的商业版(即 Professional Edition)售价为 99 美元，添加了增强的 XML 编辑功能和对 JSON(JavaScript Serialized Object Notation)的支持。

安装好 SDK 并拥有了编译 Flex 应用程序的能力后，现在可以按更顺畅的方式编写代码了。最后还有一件事，就是实现应用程序构建的自动化，以免频繁地使用命令行。

2.1.3 第三步(但愿读者还感兴趣)

现在，读者应该有足够的信息来判断免费路线是招人喜欢，还是惹人讨厌了。如果你仍然自我感觉良好，应该再看看下面这几步。

- 学习能够发送给编译器的所有参数。查阅 MXMLC 在线文档，地址为 http://livedocs.@dobe.com/flex/3 /html/compilers_13.html。
- 创建包装文件(用于启动 Flex 应用程序的 HTML 文件)。第 24 章更详细地介绍了有关包装文件的信息。
- 使用构建脚本(如 Windows 中的.bat 文件，或 Linux 及 Mac OS X 中的 shell 脚本)，以便简化编译过程。要查看 shell 脚本的例子，请打开安装文件夹中的 samples 文件夹，其中包含一些 Flex 应用程序及相应的构建脚本。
- 必须关注开源社区；假如你遇到一个常见的问题，可能有人已经找到了解决方案。

还有一个比较棘手的问题，就是怎样自动完成重复性的任务；对此，倒是有个现成的简洁工具可用——准确地讲，这个工具名叫 ANT(Another Neat Tool，另一个简洁的工具)。

1. Ant 与 Flex Ant Tasks

使用 C++或 Java 的程序员都熟悉 make 实用程序，这是一个用于构建应用程序的常见脚本工具。但与 make 的经典与流行如影相随的则是创建和维护其配置文件的繁琐。这是使用 shell 脚本不可避免的一个问题，因为使用的是支持运行的操作系统特有的命令。

源自 Apache 软件项目的 Ant 是一个免费开源的工具，致力于简化脚本构建工作。作为使用 XML 配置文件的基于 Java 的应用程序，Ant 是平台中立的，而且也比较容易掌握和使用。

Adobe 也贡献了一个可以免费使用的基于 Ant 的构建工具，这个名叫 Flex Ant Tasks

的工具是 SDK 的标准配置，可以在 SDK 安装根目录下的 ant 文件夹中找到。

尽管刚开始创建 Flex 应用程序还用不到 Ant 或 Flex Ant Tasks，但提前了解它们是非常必要的。有关 Ant 的更多信息，请访问 http://ant.apache.org。

如果读者想深入学习 Ant，我们推荐 Steve Loughran 和 Erik Hatcher 合著的 *Ant in Action* 一书(Manning Publications，ISBN：1-932394-80-X)。这本书将教会你怎样使用 Ant 并最大限度地发挥它的威力。

2. 注意事项

Flex SDK 并没有 Flex Builder Professional 提供的所有功能，但却包含：

- 自动测试框架；
- 内存/性能分析程序(用于确定资源瓶颈)；
- 某些组件，如图表和 AdvancedDataGrid 组件(代码虽然有效，但组件上却带有水印图片)。

如果读者想真正做 Flex 开发，绝对有必要投资购买 Flex Builder 项目。使用 Flex Builder 可以立即节省时间，而省下的时间会立即转化为投资回报。

拥有标签完成、代码提示、内置 API 参考手册、自动导入库、向导、可视化布局和样式及自动构建等功能的 Flex Builder，真可谓是一个得力的助手。在软件开发中，首要的成本就是资源时间，更准确地讲，就是“开发人员的工作时间”。

2.2 使用 Flex Builder 做真正的开发

Flex Builder 是一款功能丰富的 IDE，用于设计、开发、测试和构建 Flex 应用程序。如果读者因为不想选择免费路线而跳过上一节，我们想在此简单地重复一句，Flex Builder 基于开源 IDE 框架和编辑器 Eclipse(http://www.eclipse.org)构建，并且作为插件扩展 Eclipse (因此，既可以下载 Flex Builder 插件，然后把它安装到现有的 Eclipse 安装中，也可以单独下载 Flex Builder 和 Eclipse 并分别安装)。

可是，具体定价是多少呢？Flex 3 的价格目前已经大幅下降，因此门槛比以前更低。建议读者货比三家之后再购买。

2.2.1 产品和价格表

Flex Builder 有几种不同的版本，定价也不一样(见表 2-1)。网站上公布的只是厂商的建议零售价。如果从增值零售商那里购买，可能会享受一定的折扣。要是你所在的公司与 Adobe 预先商定好了折扣度，就可能获得更大的优惠。如果你正在为客户或公司出面购买 Flex，那么拿到更低的折扣一定会增加交易成功的胜算。

表 2-1 Flex Builder 及 SDK 的价格

版 本	价格(美元)	说 明
SDK	0	N/A
Standard	250	单步调试器、代码提示/完成、代码着色、CSS 编辑、集成 API 参考、可视布局设计器、数据集成向导、Creative Suite 3 皮肤导入、自动编译等。可以创建图表，但图表中将包含水印
Professional	699	Standard 版的所有功能，外加内存和性能监视/分析程序、自动测试框架和高级组件(如图表和 AdvancedDataGrid)
Education Professional	0	对学生和教师免费

花钱买这些工具值得吗？答案是另一个问题：你的时间有多值钱？如果你或你的公司每省下 1 小时的价值是 50 美元，那么购买 Flex Builder 只需不到 1 周即可收回成本；如果使用的是 Professional 版，则只需 14 个小时。

如果读者出于个人兴趣学习 Flex Builder，还没有采用它做任何实际项目的想法，无论购买 Standard 版还是 Professional 版，都不会有立竿见影的效果。对于这些读者，我们还是建议选择前面介绍的经济的 Flex 开发方式。

否则，你或你的公司都没有理由不选择这个有效和高效的名副其实的好工具。

提示：

假如你发现领导因为价格问题犹豫不决，可以使用 Charts 组件为他们创建一个交互式的概念验证报告显示板。相信这个报告及其表现出的商业智能一定会让他们怦然心动。

如果读者想在没有任何投入的情况下体验 Flex Builder，可以选择 60 天的试用版。从测试的角度说，60 天应该足够了。

2.2.2 获得 Flex Builder

Flex Builder 有两种分发方式，一种是独立版，另一种是插件版。

独立版由 Eclipse 和插件组成，可以作为一个现成的独立产品安装。插件版则适合已经将 Eclipse 用于其他编程语言的用户，这些用户可以通过插件为现有开发环境添加 Flex 开发的功能。无论选择哪个版本，都可以在开发过程中随时添加更多的第三方插件。

如果读者还没有安装 Flex Builder，可以到 http:// www.adobe.com/go/flex_trial 下载。当然，安装的时候如果没有序列号，就只能使用 60 天。60 天后再使用必须提供序列号才行。

安装完成后，就可以启动这个程序并动手试验了！接下来，我们介绍一下 Flex Builder 的功能。

2.3 探索 Flex Builder

Flex Builder 以项目为中心运行，一个项目就是一组文件夹、文件和设置的集合。可以

创建、关闭、导入和导出项目(见图 2-3)。

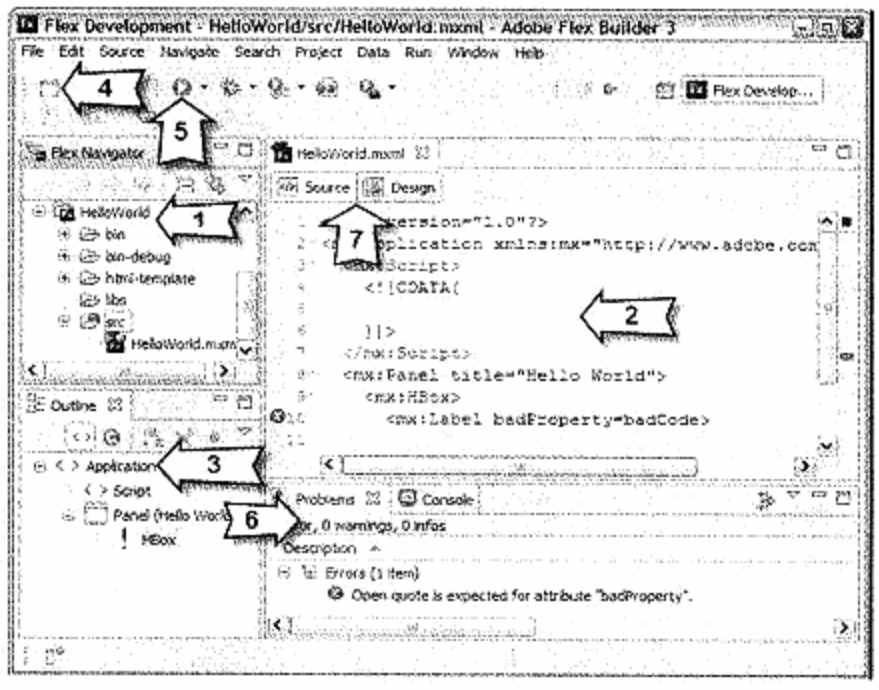


图 2-3 Flex Builder 基本布局的主工作区和窗口

以项目为中心的概念与其他编辑器(如 Dreamweaver)有所不同，其他编辑器往往采用完整文件浏览器方法，你可以在其中导航到文件系统和网络任意位置的文件夹。

在 Flex Builder 中，项目由资产(asset)组成，资产是文件夹中嵌套的各种文件，项目通常位于主项目文件夹内(也可以链接到其他位置的文件夹)。

下面我们就参照图 2-3，介绍 Flex Builder 的基本布局。

1. 导航器(1)

列出所有项目及每个项目中的资产。与标准的文件浏览器不同，它只显示位于已经配置好的项目中的文件。在这个窗口中不能在整个文件系统中导航。

导航器的一个关键功能是可以通过右击顶级项目文件夹来配置项目属性。

2. 源代码编辑器(2)

这里是所有操作的发源地。源代码编辑器当然是编写代码的地方。这个编辑器有很多极佳的功能，如文本着色、代码提示。代码提示类似于一种自动完成功能，因为该功能会根据用户已经输入的内容预测将要输入的内容。可能的匹配项会通过一个列表显示出来，然后我们可以从中选择要输入的剩余语法。这个功能有助于加快开发速度，因为可以不用记住确切的语法。

3. 应用程序大纲视图(3)

这是另一种导航器，它动态构建应用程序的映射。随着应用程序规模增大，通过这个窗格来查看各个组成部分之间的关系会非常方便。另外，在大纲视图中选定任何项，都会镜像到源代码编辑器，即编辑器会自动跳转到相应项。

4. 向导(4)

Flex 的这个功能在从头开始创建项目时尤为有用。它不仅能帮助 Flex 初学者快速创建项目，也有助于高级 Flex 开发人员加快进度。作为一个即装即用的功能，向导有很多作用，从创建 ActionScript 类，乃至生成 Flex CSS 文件。但向导最常见的用途还是创建 Flex 项目和自定义 Flex 组件。

5. 应用程序启动器(5)

这个按钮用于启动应用程序。启动应用程序的过程包括编译编写好的所有代码(如有必要)，把最新编译的代码复制到输出文件夹，然后根据配置好的 URL 启动应用程序。

6. 底部窗格(6)

底部窗格用于显示问题或控制台视图。默认的问题(Problems)视图与编辑器并行工作，显示代码中存在的问题(图 2-4)。这个窗格显示应用程序中存在的错误，并指出错误所在的文件和对应的行号。

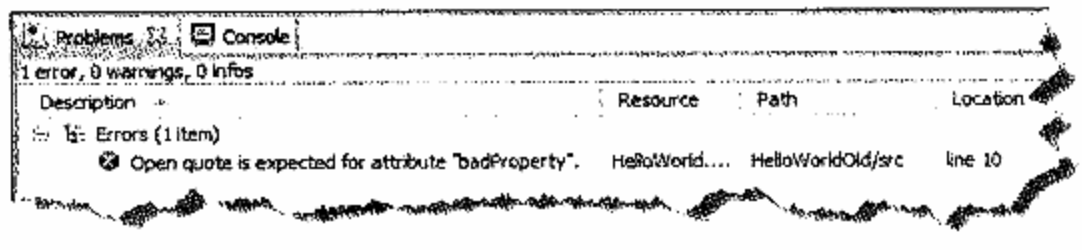


图 2-4 问题视图提醒我们代码中存在的问题，同时指出问题的位置

控制台(Console)视图在测试应用程序时使用。它会显示提醒、警告及其他指定的消息，然后把这些消息保存到一个名叫跟踪日志(trace log)的文件中(第 23 章将深入讨论如何使用跟踪日志)。

7. 源代码和设计视图(7)

这两个标签用于在源代码编辑器和交互式设计视图之间切换。设计视图用于呈现所见即所得的应用程序外观和某些视觉元素，同时还能让我们控制这些视觉元素的各种属性(如颜色、透明度、大小和位置)。

2.4 视图与视角

前面提到了多个视图，读者也应该注意到 Flex Builder 的主界面被分割成了不同的区域(或窗格)。每个窗格中包含一种视图，这些视图的集合叫视角(*perspective*)。

2.4.1 预定义的视角

在开发 Flex 应用程序的过程中，通常会涉及 3 个主要工作模式——开发、调试和分析(测试/调优)。视角是 Flex 预先提供的、针对每个主要阶段优化的视图和窗格集合。

1. 开发视角

开发视角是针对核心开发过程优化的一组视图，包括大纲、问题、控制台和导航器视图。

2. 调试视角

这个视角包含一组针对调试的窗格和视图。当进入调试模式时(有关调试模式的详细内容请参见第 23 章)，Flex Builder 会自动切换到该视角。调试视角增加了允许监视所跟踪变量的值的视图，同时提供控制应用程序执行的功能(暂停、取消和逐行执行代码等)。

3. 分析视角

分析(profiling)视角与调试视角相似，也有助于隔离问题。分析视角利用 Flex 的内存分析程序，为开发人员提供跟踪内存和性能信息的视图。其中包括内存使用量图表、被调用的函数列表、调用它们的频率，以及执行每个函数花费的时间。

2.4.2 切换视角

通过 Window | Perspective 菜单中的视角列表，可以随时更改视角，如图 2-5 所示。

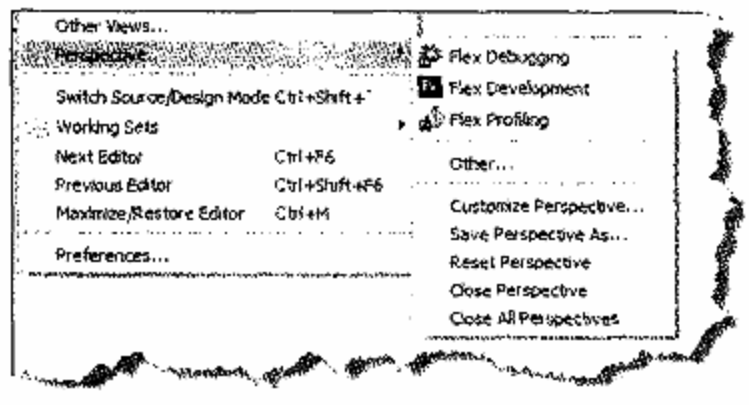


图 2-5 Flex Builder 提供了 3 种预定义视角，用户也可以定义自己的视角

随着开发过程的推进，我们可能会觉得手头的工作使用某种布局或某些视图更方便。此时，可以利用 Flex 提供的自定义视角功能创建自己的视角。

2.4.3 自定义视角

可以添加视图并在不同窗格间移动视图，然后把新的排列方式(包括任意数量的布局)另存为一个自定义视角。

1. 修改布局

下面介绍如何修改布局。请读者按照下列步骤动手尝试一下。

(1) 在 Console 标签上单击并按住鼠标左键，然后在窗口中任意拖动该标签。

(2) 在拖动过程中将会发现，有的地方出现灰色轮廓线(见图 2-6)，这表明该处可以放置此视图。

(3) 松开鼠标左键查看结果。

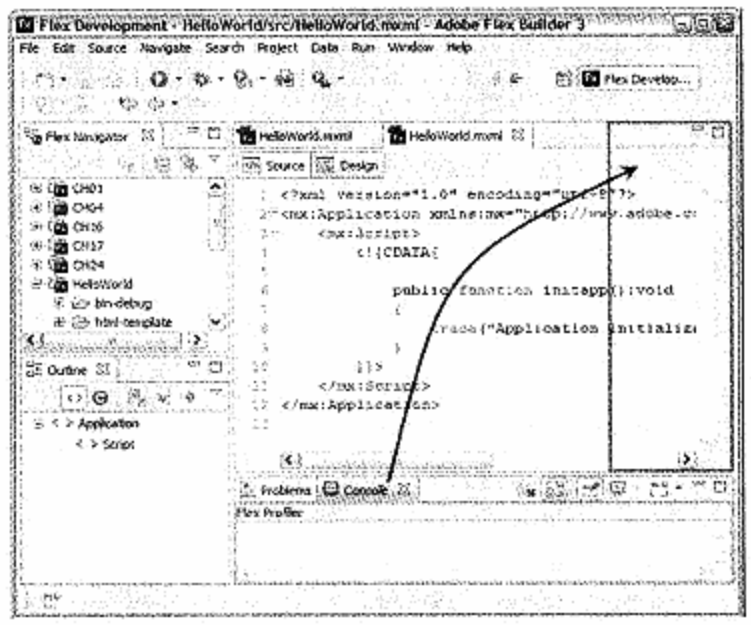


图 2-6 通过单击并按住鼠标按钮，可以改变视图的位置

重新排列现有布局不费吹灰之力。但是为了显示更多信息，往往还需要向布局中添加视图。

2. 向布局中添加视图

按照下列步骤可以向视角中添加新视图。

(1) 单击 Flex Builder 顶部的 Window 菜单，会看到一些视图选项(见图 2-7)，如 Flex Properties 视图选项。如果选择其中的 Other Views 选项，将得到一个完整的视图列表(见图 2-8)。

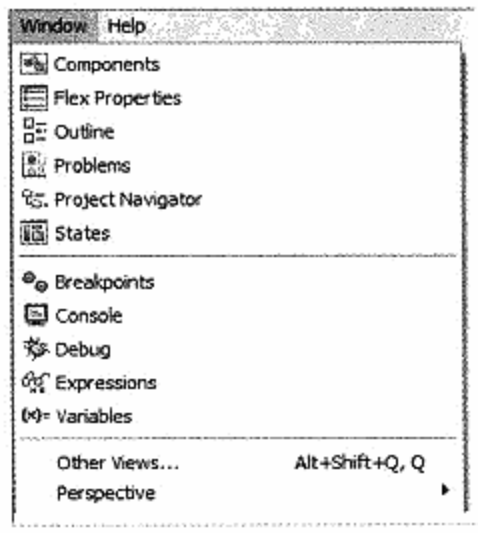


图 2-7 单击 Window 显示可以添加到布局的视图

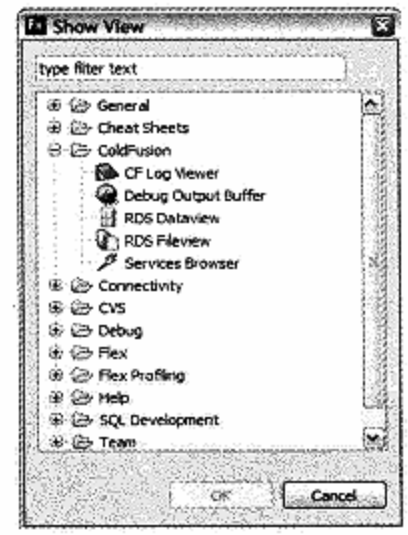


图 2-8 单击 Other Views 显示所有视图

(2) 从中选择任何一个视图选项，相应视图将出现在 Flex Builder 窗口中。

(3) 按照前面的介绍根据需要重新定位布局。

务必大致浏览一遍 Other Views 显示的所有视图。如果你是一名 ColdFusion 开发人员，很可能需要使用特定于 ColdFusion 的视图。

2.5 第一个项目——HelloWorld

如前所述，Flex Builder 是基于项目的。在开发模式下，我们所做的一切几乎都被包含在一个项目中。

Flex Builder 中的一个项目，就是所有源代码文件、资产(图像和 mp3 文件等)和配置设置的集合。下面我们就开始创建第一个项目。

2.5.1 创建项目

Flex Builder 内置了一组方便实用的向导，可以引领我们有序地完成整个过程。特别是新向导 Flex Project，可以带着我们选择项目的名称，指定保存文件的位置，甚至选择应用程序所要使用的后端服务器技术。接下来我们就利用这个向导，步骤如下。

(1) 调用向导选择器。

在 Flex Builder 中，单击左上方的按钮启动 Flex Builder 的向导(见图 2-9)。

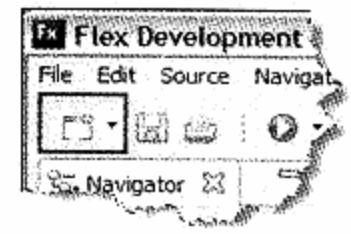


图 2-9 单击向导图标启动 Flex 项目向导

(2) 选择 Flex Project 向导。

选择 Flex | Flex Project 向导(见图 2-10)，然后单击 Next 按钮。

(3) 为项目命名。

接下来向导会要求填写项目名称，选择应用服务器类型(见图 2-11)。项目名称可以随意命名，不过这里命名为 HelloWorld。应用服务器类型选择 None。单击 Next 按钮，继续下一步。

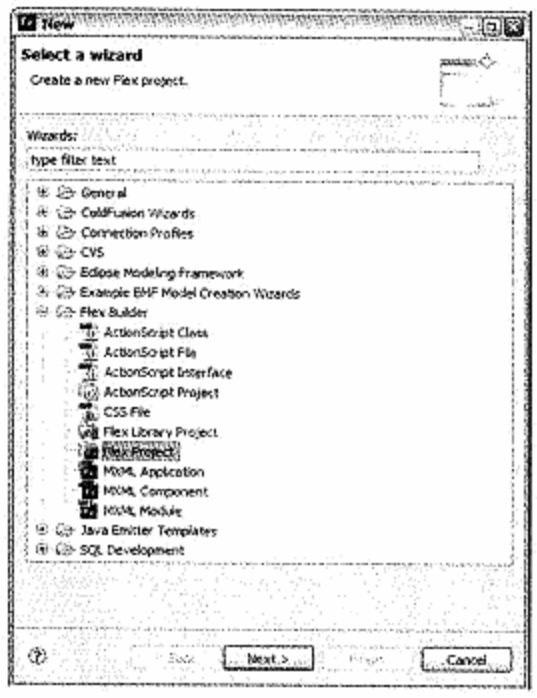


图 2-10 向导会带着我们完成新项目的创建

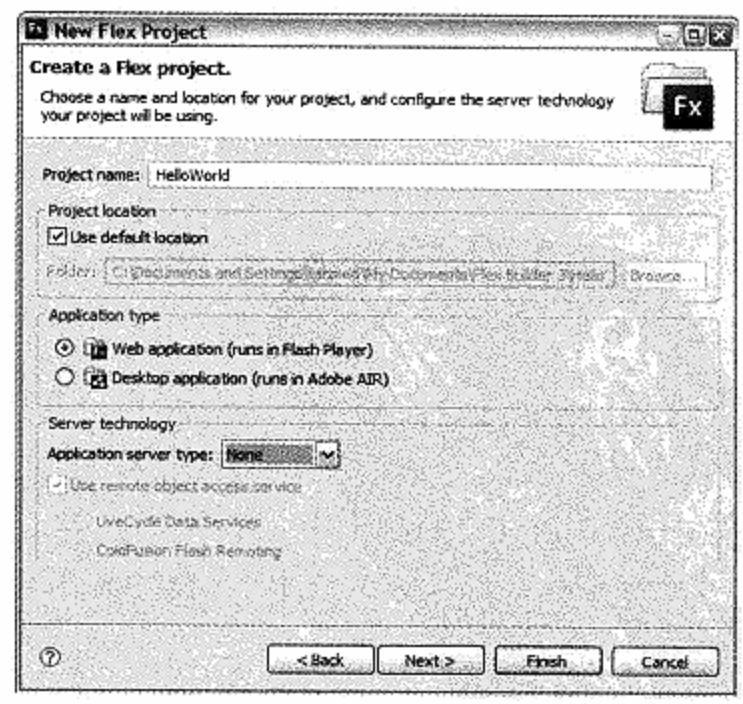


图 2-11 创建 Flex 项目的向导

注意：

第 14 章将讨论如何使用应用服务器类型功能。Flex Builder 的这个选项能为我们节省很多步骤，让我们能轻松连接到支持 Flash 二进制通信协议(AMF)的后端系统。当前，New Flex Project 向导支持 ColdFusion、ASP.Net、J2EE(Java 2 Platform, Enterprise Edition)和 PHP。眼下不需要使用其中任何协议，因此选择 None。

(4) 配置输出目录。

这一步选择在哪里创建输出目录(见图 2-12)。

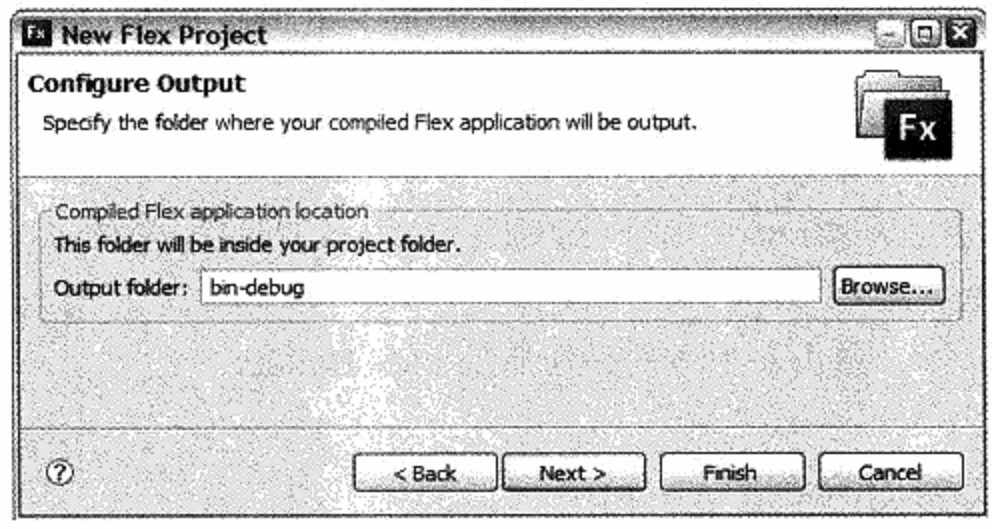


图 2-12 指定在其中构建和部署应用程序调试版本的输出目录

默认情况下，输出目录位于主项目文件夹内。不过，也可以指定任何更方便的位置。

Flex Builder 将把应用程序编译后的调试版本(测试时必需使用)发布到指定文件夹中。第 24 章将详细介绍创建生产版本的过程。

提示：

读者大可随意指定一个认为合适的位置。在熟悉了 Flex Builder 以及确定了公司内部的项目标准位置后，这很可能成为常事。

(5) 完成创建过程。

默认情况下，我们为项目起的名字会作为应用程序根文件夹和 Flex 应用程序主文件名称。

单击图 2-13 所示向导窗口底部的 Finish 按钮。Flex Builder 会为我们初始化一个新项目。

对这个例子而言，项目会保存在一个名为 HelloWorld 的文件夹中，而主文件也将被命名为 HelloWorld.mxml。

现在项目已经创建完成，只等我们开始输入代码了。

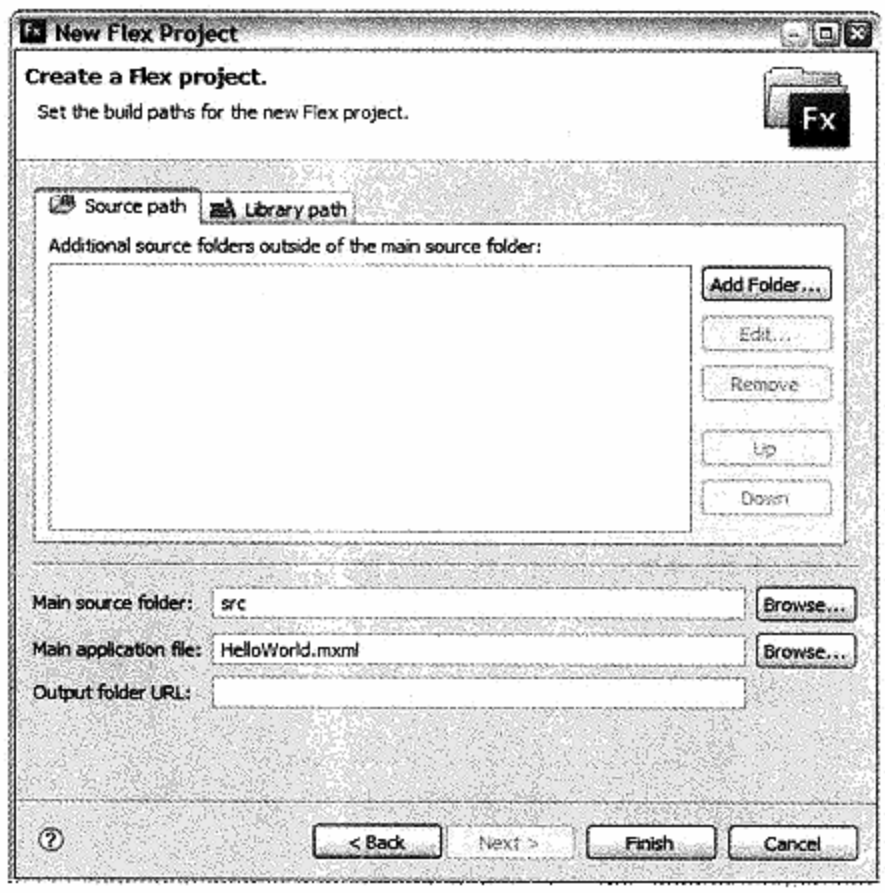


图 2-13 最后一步允许选择多个选项，例如将项目中的哪个文件作为主文件

2.5.2 输入代码

单击 Finish 按钮，Flex Builder 会自动打开项目主文件 HelloWorld.mxml，文件内容如代码清单 2-1 所示。

代码清单2-1 最基本的应用程序

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  ➥layout="absolute">
</mx:Application>
```

代码清单 2-1 显示 Flex 应用程序的基本外壳，MXML 语言用于容纳应用程序中的主元素。此时，我们正好可以利用这个机会验证一下是否可以编译和执行项目。

2.5.3 编译和运行

编译和运行 Flex 项目无非就是点一下鼠标的事。在 Flex Builder 工具栏上，找到一个带向右箭头的绿色按钮。单击该按钮，会显示一个指向 HelloWorld.html 文件的新浏览器窗口。此时页面中什么也没有，但同时也表明我们的第一个 Flex 应用程序诞生了！

浏览器中打开的 HTML 文件只是一个调用 Flex 应用程序的包装器。其中包含了附加的 JavaScript 代码，用于检测 Flash Player 版本和设置 Flex 应用程序将在其中运行的客户端环境。

2.5.4 查看实际效果

要看到真正的 Hellow World!示例，还要返回 Flex Builer，在源代码中添加一行，如代码清单 2-2 所示。

代码清单2-2 HelloWorld!

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute">
  <mx:Button label="Hello World!" fontSize="40"/>
</mx:Application>
```

现在运行一下看看有什么变化，结果类似图 2-14 所示。

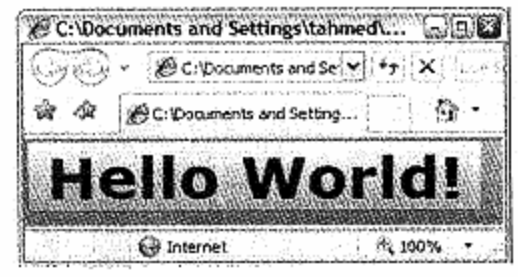


图 2-14 第一个 Flex 应用程序诞生！祝贺你成为 Flex 开发人员

感觉还不错，是吗？你刚才添加了一个按钮(没问题，确实出现了一个按钮)。事实上，通过 MXML 添加的一切都可以归类为组件(component)。像按钮这样的视觉组件也称为控件(control)。Flex 提供了很多内置组件，如果这些内置组件都不适用，也可以创建自定义组件(有关创建自定义组件的内容请参见第 17 章)。

在这个示例中，我们通过编程方式添加了一个按钮。Flex Builer 也支持通过设计方式创建组件，通过设计方式创建组件便于我们快速创建原型和验证想法。

2.6 使用设计模式

Flex Builder 提供了称为设计模式的有趣功能，让开发人员以可见的方式创建应用程序(单击图 2-3 所示第 7 项中的 Design 标签，启用设计模式)。在设计模式下，可以把视觉组件拖放到应用程序中，摆放它们的位置，而且可以调整它们的属性(如背景颜色)。

仍以 HelloWorld 项目为例，单击 Design 模式按钮。此时，我们注意到窗口右侧又多了两个视图，分别是 States(状态)和 Flex Properties(Flex 属性)。

提示：

组件是可重用的对象，可以在 MXML 中调用。组件是一种重用代码的方式，类似于 Java 类或 ColdFusion 组件(CFC)。可见的组件也称为控件。

下面向应用程序中添加另一个组件。首先，找到 Components(组件)视图，应该位于 Flex Builder 的左下角(图 2-15)。单击展开树状目录，可以看到其中包含的所有可供使用的组件。我们想添加一个 Panel 组件——如果想继续设计应用程序的布局，可以继续这么做。

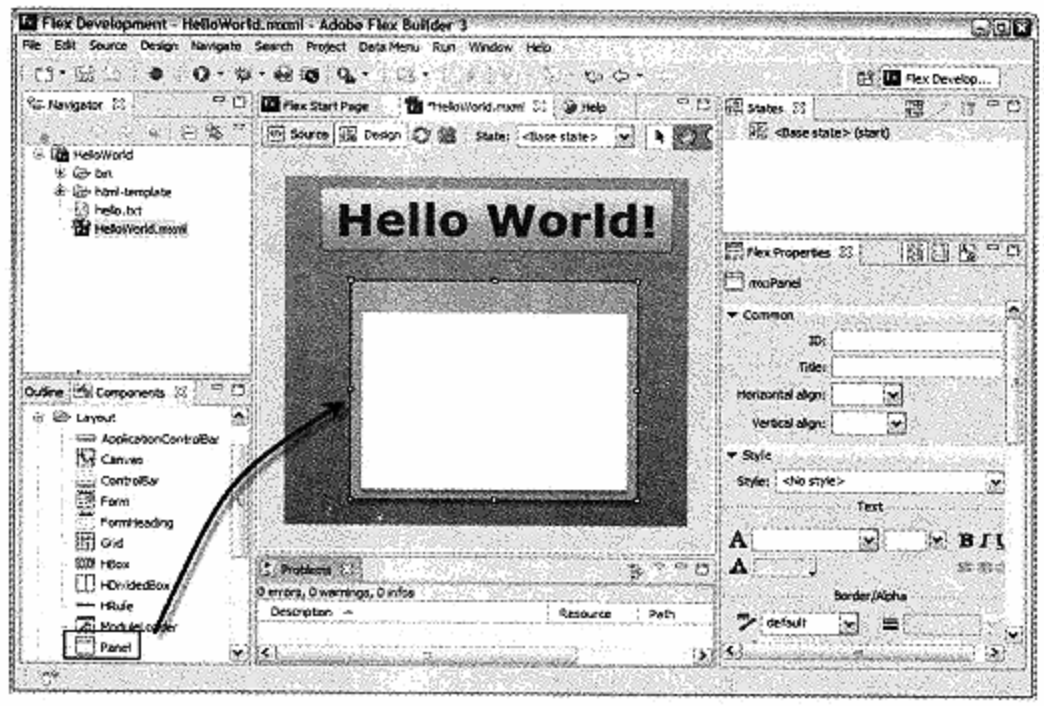


图 2-15 通过 Flex Builder 的设计视图以可见方式布局应用程序

在把组件拖动到位后，接下来将注意力集中在 Flex Properties 视图上。在设计模式中选择一个组件，可以在 Flex Properties 视图中看到和编辑该组件的所有属性。

通过 Flex Properties 视图可以修改组件的视觉特征，如颜色、透明度和位置。在其标准视图中，只会显示一部分属性。单击 Category 标签，可以看到所有可用属性(图 2-16)。

图 2-16 使用 Flex Properties 视图中的分类视图可以查看所有可编辑的组件属性

基于图形界面设计和构建应用程序的最大好处，就是能进行概念验证(proof-of-concept，POC)。在开始编码之前，跟客户及最终用户以互动方式讨论应用程序的外观时，采用设计模式能最大限度地节省沟通时间。因为所有人都能对最终目标一目了然。

在软件开发生命周期中，通常使用简单的线框模型来提供 POC，但这种方式无法让用户体验到应用程序具有什么样的外观。况且，要整理一份较为逼真的 POC 也相当费时间。Flex Builder 允许使用以最少的时间创建逼真 POC，让这两个问题迎刃而解。

虽然可以完全在设计模式下构建小型应用程序，但设计模式的主要价值还是体现在创建 POC、对应用程序进行初始布局和调整视觉属性以查看外观效果上。

在实际做项目时，真正要做的则是像使用其他任何编程语言一样，直接编写代码。

Flex Builder 不仅为创建项目提供了强大的向导、拥有模拟布局的能力，而且还与任何优秀的集成开发环境一样，内置了帮助系统，对提高编码效率提供了有力支持。

2.7 内置的帮助

Flex 是 OOP(Object-Oriented Programming，面向对象编程)语言。第 16 章将深入探讨什么是 OOP。不熟悉这个概念的读者无需担心，我们将在随后各章循序渐进地讲解这个概念，以便逐步提高你的理解水平。

要想成为一名多产的 Flex 开发人员，善于查阅 Flex API 帮助是一项必备技能。下面将介绍 Flex 帮助系统。

2.7.1 面向对象语言及其 API

面向对象语言的一个特征，就是其 API 参考的使用频率要高于非面向对象语言。主要因为非面向对象语言拥有可变类型和函数，而它们之间没有什么相关性。例如，对于 ColdFusion 而言，cfquery 标签返回一个带有某些属性的查询结构(仅此而已)。

在面向对象语言中，有对象的概念，而对象中包含函数(也称为方法)和属性。可以不断地从一个对象扩展另一个对象，从而使扩展对象继承上一个对象的所有特性(例如，动物 > 狗 > 斗牛犬)。

提到这些跟帮助有什么关系呢？由于我们要处理的对象实在太多，因此与使用非面向对象语言相比，肯定需要更频繁地查阅 API 参考。

贯穿全书，我们将分期分批向读者介绍在 API 参考的哪些地方可以查到相关信息。

可以说，越熟悉 API 参考，就越能够有效地编写 Flex 代码。

2.7.2 访问 API 参考

鉴于我们才刚开始学习 Flex，暂时也不会太多地体会到 API 参考的价值。因此，现在读者唯一要做的，就是记住怎样访问 API 参考。

1. 动态帮助

访问 API 参考的最简便方式，是使用 Flex Builder 自动化的动态帮助(Dynamic Help)功能。相应的步骤如下。

(1) 在 Flex Builder 工具栏中，选择 Help | Dynamic Help。

(2) 在代码编辑器中，单击<mx:Button>代码行中的任意位置，以便在该行上定位光标。

(3) 此时，右侧会出现一个新窗格(图 2-17)。第一部分是指向 API 的链接。

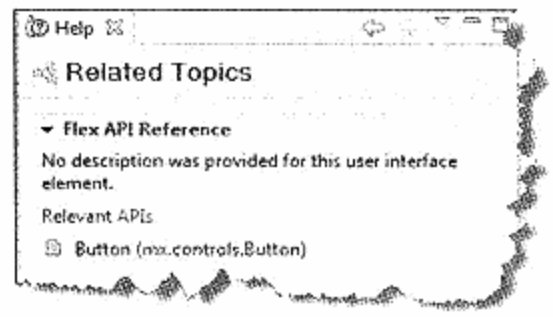


图 2-17 动态帮助根据光标在应用程序中的位置，自动显示指向 API 参考的链接

(4) 这里显示的链接指向 mx.controls.Button，即解释按钮定义和正式位置。单击该链接，看一看有关这个组件的 API 参考都会显示什么内容。

有一个实时的文档可以在需要时搜索(并显示相关信息)，当然有助于节省时间；况且，这种动态帮助功能也不是每个 IDE 都有的。不过，动态帮助并不是查阅 API 参考的唯一方式。例如，还有一个应该牢记的快捷键，只要一按就能立即调出 API 参考。

2. Shift+F2 快捷键

与动态帮助相似，如果搜索(Search)窗格没有出现在界面中，或者在想保留动态帮助下显示的内容那么，按下组合快捷键，也能快速调出 API 参考，步骤如下。

(1) 在代码编辑器中，在包含<mx:Button>的代码行的任意位置单击。

(2) 按 Shift+F2。

(3) 然后会出现针对按钮的 API 参考。

适应这个组合键恐怕需要经过一段时间，不过，相信只要读者这样做了一次，一定会经常使用它。

此时此刻，我们所做的只是找到 API 参考。接下来，就是要看懂 API 参考显示的内容。

2.7.3 研读 API 参考

我们来看一看 API 参考页面中包含的元素。无论使用动态帮助还是 Shift+F2 访问按钮组件的 API 参考页面，都会看到图 2-18 所示的内容。

Package	mx.controls
Class	public class Button
Inheritance	Button → UIComponent → FlexSprite → Sprite → DisplayObjectContainer → InteractiveObject → DisplayObject → EventDispatcher → Object
Implements	IDataRenderer, IDropInListItemRenderer, IFocusManagerComponent, IListItemRenderer, IFontContextComponent, IButton
Subclasses	AccordionHeader, CheckBox, LinkButton, PopUpButton, RadioButton, ScrollThumb, SliderThumb

图 2-18 API 参考页面的第一部分描述特定对象的相关信息

下面我们就以通常意义上“对象”这个术语(与面向对象术语相对)为着眼点来简要说明图 2-18 所示的 5 个类别。

- **Package(包)**：表示当前对象所在的目录。
- **Class(类)**：表示这个 API 参考页面描述的特定对象。
- **Inheritance(继承)**：当前对象继承了哪些父对象？子对象通常都会继承其祖先的属性和函数。这一点非常重要，因为知道父对象能做什么，就可以了解子对象能做什么。
- **Implements(实现)**：这个条目中列出的项叫做接口。接口类似对象承诺遵守的协议。具体来说，对象会实现接口所要求的属性和函数/方法。
- **Subclasses(子类)**：当前对象的子对象，即当前对象能做到的，这些子对象同样也能做到。

API 参考页面的下一部分(图 2-18 中没有显示)包含了 Flex 开发人员在操作一个对象或组件时，需要频繁参阅的所有重要的信息片断。下面对这些部分予以简要介绍。

- Properties(属性)：可以检索和设置的值，如宽度和高度。
- Methods(方法)：支持函数，例如有的函数会导致组件重绘自身。
- Events(事件)：对象会对此有所反应的操作(如被单击)，当某种操作发生时会触发特定指令的执行。
- Styles(样式)：类似于在 HTML 中使用的 CSS 样式，这里显示的是样式表参考。
- Effects(效果)：与事件一起使用，以便指定播放效果。例如，可以指定当鼠标放在一个对象上时，让该对象播放发光效果以突出显示该对象。

事实上，每个 API 参考页面都有着与上述类似的结构和条目。有的甚至还包含一些示例，如演示对象或组件的用法。关键在于，如果读者习惯了对这些条目的分析，并能从中筛选出对自己有用的信息，就能够毫不费力地在代码编辑器与 API 参考之间来回切换。

2.8 MXML 与 ActionScript 速写

Flex 基于 MXML 和 ActionScript。因此对于初学者来说，往往会对什么时候该使用哪种语言感到困惑。而且 Flex 新手也经常会问是不是有一些应该遵循的成熟的最佳实践。

下面我们先来了解一下这两种语言，稍后再来回答前面的问题。

2.8.1 MXML 的结构

在前面的示例中，我们使用纯 MXML 编写了一个 Flex 应用程序。MXML 是一种基于标签的编码系统，它派生自 XML。为此，MXML 遵循了 XML 所有规则和约定，特别是以下几条。

- XML 文件以及 MXML 文件要求必须有 XML 声明，用以定义版本和字符编码。

```
<?xml version="1.0" encoding="utf-8"?>
```

- XML 文件必须有一个顶级的根标签(及位于文件末尾的对应的结束标签)。
- MXML 语言区分大小写。
- 每个标签都必须有一个开始标签和一个结束标签。例如，<myTag>…</myTag>。也可以使用缩写形式，即<myTag/>。
- 标签间的嵌套关系必须正确。HTML 对标签的嵌套关系要求不严格，但 XML 则非常严格，下面分别列举了正与误例子。

  有效：`<a><b>…</b></a>`

  无效：`<a><b>…</a></b>`
- 属性的值必须加引号。例如，<tag attribute="value">。
- 每个空白符都算数；HTML 会忽略连续出现的空白符，只把它们看做一个。

如果读者曾大量编写过 XML，应该不会觉得上述规则和约定很新鲜。如果你没有 XML 的经验，恐怕会像一名 HTML 开发老手一样，觉得上述约定有点挑剔。

注意：

Flex Builder 会自动指出违反 XML 规则的地方。而且，如果你不解决这些问题，就不让你编译。

总而言之一句话，就是要保证文件内容清晰、一致而且采用结构化方式。

2.8.2 MXML 与 ActionScript 的关系

让某些人感到不太理解的一个问题可能是，Flex 为什么要由两种语言构成呢？请读者这样想一想：一名 Web 开发人员，会使用 HTML 描述视觉元素(如表格、表单输入和图像)。但这些元素如果没有 JavaScript，就不能执行客户端上的任何功能(如验证表单)。

对于 Flex 而言，情况也是如此。我们使用 MXML 来布局基本的视觉元素，定义一些不可见的内容(如格式化程序和验证程序)，以及访问数据服务(稍后介绍)。

提示：

把 MXML 与 ActionScript 之间的关系想象成 HTML 与 JavaScript 之间的关系，就容易理解了。

在 Flex 中，我们使用 ActionScript 添加功能、做出判断、提供交互性及构建业务逻辑。为说明问题，我们仍以 HelloWorld！项目为例，但这次使用 ActionScript 对其进行修改，如代码清单 2-3 所示。

代码清单2-3 结合使用ActionScript和MXML

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute">
  <mx:Button label="Marco" fontSize="40"
   click="mx.controls.Alert.show('Polo')"/>
</mx:Application>
```

运行代码清单 2-3 所示的代码，然后单击标签为 Marco 的按钮，就会看到一个弹出式警告框(图 2-19)。

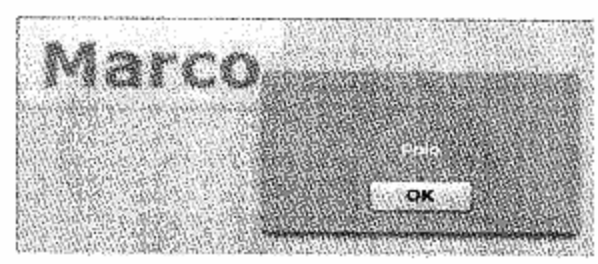


图 2-19 通过 ActionScript 可以为应用程序嵌入脚本逻辑

添加以上功能时，如果读者能把在 Flex 应用程序中编写 ActionScript，想象成在 HTML 中编写 JavaScript，就比较容易理解了。

一些 Web 开发人员理解这一关系的思想障碍，主要源自他们使用诸如 JavaScript 的客户端逻辑的经验十分有限。因为传统的 Web 技术和 Web 文档模型，更倾向于把大多数应用程序逻辑放在服务器端。

例如，ColdFusion 开发人员不仅习惯于在 HTML 中使用标签，同样倾向于用标签式语言来表达逻辑。

```
<cfif itsAllGood>
 <b>It's all good</b>
</cfif>
```

于是在编写 Flex 应用程序时，很可能会不由自主地写出下面代码。

```
<mx:if>…</mx:if>
```

因此，只要记住 MXML 与 ActionScript，相当于 HTML 与 JavaScript，很多问题就会迎刃而解。如果把 ActionScript 想象成 JavaScript，那么由 JavaScript 能处理事件，就能推断出 ActionScript 一定也支持某种形式的事件机制。

2.8.3 事件由 ActionScript 处理

Flex 应用程序是由事件驱动的，这意味着程序中发生的一切都是事件引发的结果。这种情况在 Flex 这种客户端技术中再常见不过了。即先有某种操作触发事件，然后再由代码处理相应的事件。

处理事件的“设备”叫做事件处理程序，事件处理程序由 ActionScript 来实现。回顾代码清单 2-3，就会发现当鼠标按钮触发 click 事件时，会执行一段 ActionScript 代码。

```
<mx:Button label="Marco" fontSize="40" click="mx.controls.Alert.show('Polo')"/>
```

在此我们所说的事件处理程序，就是位于双引号中的 mx.controls.Alert.show('Polo')。实际上，可以在这里放入任何 ActionScript 事件机制。通常，在事件发生时，调用事件处理函数来处理所有要执行的逻辑。

1. 几乎与 JavaScript 一模一样

这与 JavaScript 中的执行方式几乎没有任何区别。JavaScript 被大量用于处理鼠标进入和鼠标离开事件。例如，在把表单数据发送给服务器处理前，你需要验证输入表单的数据。

下面我们来看代码清单 2-4 中的代码是如何满足要求的。

代码清单2-4 跟ActionScript一样，JavaScript也是由事件驱动的

```
<script language="JavaScript">
 img_over =new Image(); img_over.src ="over.gif";
 img_off=new Image();  img_off.src="off.gif";

 function handleOver()
 {
  document.myImg.src=img_over.src;
 }

 function handleOut()
 {
  document.myImg.src=img_off.src;
 }
</script>
<a href="#" onMouseOver="handleOver();" onMouseOut="handleOut();">
 <img name=myImg src="off.gif">
</a>
```

❶

❷

这是 JavaScript 中经典的鼠标悬停技术。其中用到两个重要事件。

- onMouseOver：鼠标进入目标对象时触发❶。
- onMouseOut：鼠标离开目标对象时触发❷。

从 JavaScript 角度看，这与前面的 Flex 示例非常相似。不过比较而言，还是有一点差别的，因为 Flex 还有一个事件对象。

2. 几乎相同，但 ActionScript 有一个事件对象

JavaScript 的事件处理机制几乎相同，但不完全相同。主要区别在于 ActionScript 的事件处理机制还提供了一个事件对象。

在 ActionScript 中，所有事件都发端于一个通用事件对象，事件通过继承添加其他详细信息。有了事件对象，就能轻松编写出具有高度可重用性的事件处理程序，因为事件处理程序可以从事件对象中获得事件的详细信息。这些信息包括：

- 事件的来源；
- 数据的引用；
- 事件的类型(例如，单击还是鼠标进入)。

下面我们通过示例了解工作方式，如代码清单 2-5 所示。

代码清单2-5 使用事件对象可以让事件处理程序更通用

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="white">
<mx:Script>
  <![CDATA[
    import mx.controls.Alert;
    public function clickHandler(clickEvent:Event):void
    {
  Alert.show("Event Type:" + clickEvent.type +
             " came from:" + clickEvent.currentTarget.id);
    }
  ]]>
</mx:Script>
<mx:Button label="Handle Click" id="myButton"
        click="clickHandler(event)"/>
</mx:Application>
```

接收事件对象

显示事件类型

引用作为事件来源的属性

触发事件

代码清单 2-5 显示用户单击鼠标按钮时调用事件处理函数的过程。在鼠标单击事件发生时，会有一个事件对象被创建并被传入事件处理程序中(图 2-20)。

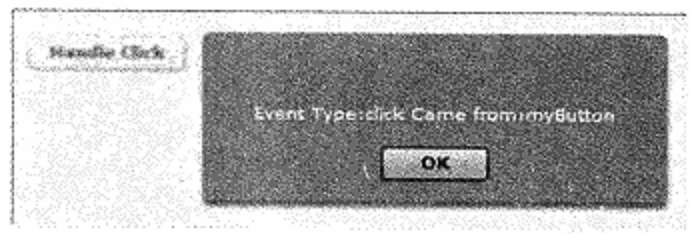


图 2-20 事件处理程序通过事件对象，可以访问包含事件来源在内的详细信息

事件处理函数并不知道是谁调用了它，但只要使用事件对象，就能访问到触发事件的组件属性，还有事件类型。

事件对象对创建可重用的事件处理程序非常关键，因为处理程序无需知道如何访问相关

组件的详细信息。

2.9 小结

我们在前面两章中介绍了很多基础性内容，而涉及代码越多，问题也会变得越复杂。在讨论复杂的问题之前，先来回顾一下本章的要点。

如果有人因使用 Flex 要花钱而质疑你的提议，你可以指出 Flex 的 SDK 是免费的，而且使用同样免费的 Eclipse IDE 及相应插件，同样能创建完整的 Flex 应用程序。但是，从提高生产效率的角度看，还是升级到 Flex Builder(是一款基于 Eclipse 的 IDE)的投资回报率更高，因此应该极力推荐。

通过使用视图、窗格和视角，可以完全重新定义 Flex Builder 界面。虽然有默认视角，但你可以修改布局，向其中添加更多视图(可供选择的视图非常广泛)，然后将其另存为新视角。这样，我们就拥有定制在这个 IDE 中工作方式的充分自由。

Flex Builder 内置了一组向导，可以利用它们执行各种任务，包括创建新项目。而 New Project 向导可以帮助我们定义主文件夹、源目录和编译后的应用程序的保存位置。

Flex 是面向对象语言，为此就必须熟悉 API 参考，因为在 Flex 中使用 API 参考的频率远高于过程式语言。对象之间存在继承关系，而且它们都包含内置属性和函数，知道了对象的父对象和子对象，就能了解它们的功能。而动态帮助和 Shift+F2 快捷键可以帮助我们达到这个目标。

ActionScript 之于 MXML，就像 JavaScript 之于 HTML。在 Flex 应用程序开发中，MXML 用于布局组件，ActionScript 用于添加所有业务逻辑。ActionScript 与 JavaScript 非常相似，它们都遵循 ECMAScript 脚本标准，都是客户端技术，而且都基于事件驱动模型。

一个重要的不同之处是，ActionScript 中创建的事件对象，可以让事件处理程序在不知道事件实际来源的情况下，访问所需的几乎任何信息。

第 3 章将介绍 ActionScript，为今后向应用程序中添加逻辑奠定基础。

第 3 章 使用 ActionScript

3

本章内容：

- 注释
- 变量类型
- 操作符
- 循环
- 条件
- 数组
- 函数

第 2 章简单介绍了 ActionScript，但可以用它来做些什么呢？稍后我们就会发现，它的功能非常强大。

ActionScript 是一种极其强大的面向对象语言，透彻地讲解这门语言至少得需要一整本书。本章将主要介绍 ActionScript 的核心概念；毕竟，这些都是应该在深入 Flex 之前必须掌握的。说到这儿，有读者可能会担心回头再学习 Flex 又会不太适应。事实上，只有打下良好的 ActionScript 基础，我们才可能脚踏实地，大步向前。

任何语言都有的一个基础概念，就是注释。下面，我们就从 Flex 支持的为代码添加说明的方式开始本章的讨论。

3.1 注释

支持在代码本身加入说明应用程序机制的信息，是所有编程语言的一种基本结构，而加入说明信息就要用到注释。

从实现的角度讲，所谓注释就是界限明确的一系列文本，Flex 编译器对相应的文本会忽略不计。Flex 的 ActionScript 语言支持两种主流注释形式：与普能代码放在一起的行内注释，和包含多个注释行的多行注释。

3.1.1 行内注释

第一种注释形式是行内注释，行内注释以双斜杠标识开头。

```
// one comment
var i:int;
var x:int; // another comment
```

可见，注释可以自成一行，也可以跟代码位于同一行(即行内)。使用行内注释有一定的限制，即编译器认为从双斜杠到当前行末尾之间的文本是一条注释。换行符意味着注释结束，而后面的内容仍然会被识别为程序代码。

3.1.2 多行注释

如果想要添加的说明很多，而且格式也必须更灵活，那么可以使用多行注释。多行注释的开始位置以斜杠加星号(/*)标识，结束位置以星号加斜杠(*/)标识：

```
/*
  here is a chunk
  of text
*/
```

为代码添加注释有两个目的：一是说明代码的工作方式，二是辅助测试与调试。为代码添加完备的注释，不仅要使其他程序员了解含义，而且要有助于代码编写者回忆当时的想法。在调试程序时，可以暂时将某些代码块注释起来，从而执行开发期间的测试和诊断。

接下来介绍变量，变量可以收集和存储信息。

3.2 变量

勿庸置疑，变量是任何编程语言中的基本构件(building block)。变量用于保存数据，通过这些数据可以记录信息、维护应用程序状态，还能让用户操作这些数据。

虽然每种语言都有变量，但每种语言对变量的实现都有所差异。ActionScript 中的变量很可能与读者熟悉的某种语言中的变量大不相同。ActionScript 基于 ECMAScript 标准实现，该标准也是 JavaScript 所遵循的(只不过 ActionScript 遵循的是 ECMAScript 的更新版本)。

3.2.1 变量名

变量有名字，而且有一定的命名规则。变量名可以包含字母、数字、下划线和美元符号。只要不以数字开头，变量名可以包含上述字符的任意组合。

3.2.2 严格数据类型

与 JavaScript 或 ColdFusion 不同，ActionScript 中的变量是严格数据类型的。也就是说，在声明了某种类型的变量后，赋给该变量的值必须是声明的类型，或者与声明的类型兼容。我们来看下面的代码。

```
var myVar:Number;
myVar = "test"; //would result in an error
```

上面的代码之所以无效，是因为变量本身只接收数字值，而我们赋给它的却是文本。

3.2.3　静态类型检查与动态类型检查

Flex 检查类型匹配的方法有两种，即静态类型检查和动态类型检查。静态类型检查在编译时执行，由编译器负责维护类型规则。具有 Java 或 C++背景的程序员应该熟悉这种方法。

ActionScript 也能够执行动态类型检查。在动态类型检查的情况下，Flash Player 负责在应用程序执行期间决定变量类型。当然，这两种方法各有利弊。动态类型检查的最大好处，就是可以让开发过程更加方便，因为可以忽略编译器的警告和错误提示，集中精力快速构建应用程序。

默认情况下，编译应用程序时使用的是静态类型检查。可以在 Project | Properties | Flex Compiler 中通过 Enable strict type checking(启用严格类型检查)选项修改这一默认设置。

不过要记住，即使禁用静态模式，在执行应用程序时，Flash Player 仍然会执行类型检查；禁用静态模式只会跳过编译时检查。

虽然跳过编译时检查表面上会减少很多麻烦，但为了避免兴尽悲来的结果，我们还是建议启用静态模式。有朝一日，读者应该学会微笑面对编译器发出的各种抱怨。

3.2.4　基本数据类型和复杂数据类型

数据类型可以分为两大类：基本数据类型和复杂数据类型。顾名思义，基本数据类型是较低层次的、未加任何处理的数据，如数字和字母，复杂数据类型则完全不同，如对象、指针和结构。

我们将在第 16 章深入探讨类和 OOP 的概念。不过，在此可以提前透露一下，面向对象语言通常用面向对象的包装器包装基本数据类型。这种经过包装的基本类型更健壮，也更灵活。

1. 基本数据类型

基本数据类型与对应的复杂数据类型比，主要特点是占用资源(内存和处理能力)少。但在 ActionScript 中，一切都是对象，因此从技术上说没有多大差别。

真正的差别来自于代码中简单的语法，而简单的语法最终会转化为一定的优势。从运行时的角度看，Flash Player 确实会将基本类型与其他对象区别对待，即不会像对复杂数据类型那样进行同样多的对象管理，因而产生一些性能上的优势。表 3-1 所示为基本数据类型。

表 3-1 ActionScript 和 Flex 支持的基本数据类型

数据类型	说明
Boolean	存储 true 或 false 值。用作标记(flag)
int	整数字段，存储整数。适合作为循环计数器使用。数据存储为 32 位值
uint	无符号整数值，与 int 类似，但只能存储非负数(如 RGB 颜色值)。数据存储为 32 位值
Number	更通用的一种数值数据类型，更灵活也更健壮。保存浮点数(即小数值)时必须使用这种类型。数据存储为 64 位值
String	字符序列。用于保存文本

何时该用 int，何时又该用 Number？多数情况下，这两种数据类型可以互换，但它们还是有区别的。

2. int 与 Number

不少博客和论坛里都在讨论这两种数据类型谁在性能上更有优势。表面上看，int 类型不用处理小数精度，它的性能应该超过 Number。但在 Flash Player 中，多数情况下 Number 的优势更明显。Flash Player 在内部使用 Number 存储所有数值数据。因此，在使用 int 时，就会有一些后台的处理花在把它转换成 Number 上。这些额外的处理在不经意间抵消了 int 确实可能存在的性能优势。

事实上，在决定选择哪种类型时，不应该把性能作为唯一考量标准；更何况它们的性能差异非常之小，只有在成百上千次地循环操作数字时，其中的差异才会有所体现。更重要的是，要考虑代码的易维护性。如果所需的是一个整数(如循环计数器)，就应该使用 int 向代码将来的维护者明确传达你的意图。

3. 复杂数据类型

复杂数据类型并不是绝对的。Flex 和 ActionScript 3 提供了非常多的复杂数据类型(只不过叫做类而已)。

Flex 吸收了核心类的概念。一般情况下，必须把要使用的类导入应用程序，编译器才能知道到哪里去找所引用的类。不过，核心类的对象是随时可用的。核心类无需明确导入，可以直接使用；而且，也不必调用对象的构造函数(即初始化对象的函数)。

提示：

大多数面向对象语言都支持构造函数，这个函数会在创建对象时自动执行。因此，构造函数可以用来在使用对象以前配置对象。

核心类允许我们生成下面的代码。

```
var myString:String = "hello";
```

相反，如果 String 不属于核心类，就必须在创建该类的对象时明确使用其构造函数。

```
var myString:String = new String("hello");
```

而由于 String 是一个核心类，因此前面两种方法都可以用来创建 String 对象。

另外，所有基本数据类型都属于核心集合。表 3-2 列出了作为核心类一部分的复杂数据类型。

表 3-2　作为核心类一部分的复杂数据类型

数据类型	说　明
Object	所有对象的祖先，所有对象都从这个对象派生而来，并继承属性。它的默认值为 null
Array	一连串对象的集合，使用数字索引。例如，要保存你家所在街道的所有地址，不必为每个地址单独声明一个变量，可以声明一个地址数组，在其中保存每个地址
Date	顾名思义，这个对象保存的是与日期相关的信息。Date 为处理和转换本地时间、夏令时及 GMT/UTC 时间提供了很多函数
Error	用于以自己喜欢的方式更适当地处理异常和管理意外情况
Function	在类中又被称为方法
RegExp	正则表达式，一种用于字符串解析和模式匹配的子语言(sublanguage)
XML	用于操作包含一个根节点的 XML 数据集
XMLList	与 XML 类似，但专门用于处理有多个顶级节点集合

以上就是与变量有关的内容，但除此之外，还有一些与特殊数据类型相关的规则。

4. 特殊数据类型

有一些数据类型不能用来创建变量。本质上，这些数据类型是保留字和保留值。还有必要指出你不应尝试使用其中的任何特殊数据类型名创建变量。表 3-3 所示为特殊数据类型及其说明信息。

表 3-3　特殊数据类型

数据类型	说　明
*	一种可选的方法，用于告诉编译器将在后面设置变量的数据类型
void	在函数声明中使用，表示函数不会返回任何数据
undefined	与 void 同时使用。void 的值是 undefined
null	由复杂数据类型和 String 使用，表示目前它们还没有被赋值
NaN	非数字(Not a number)。由 Number 数据类型使用，表示目前它们还没有被赋予一个数字值

总而言之，ActionScript 变量区分大小写，要保证赋给它们的值符合或兼容它们的数据类型。ActionScript 包含基本和复杂类型，所有基本类型和一些复杂类型属于核心类集合。

我们经常会在处理数组中的值时使用循环来处理值的集合，接下来就讨论循环。

3.3 循环

循环是所有语言都提供的另一种基础结构，只是实现方法不尽相同而已。从概念上讲，循环用于迭代一定范围的数字或项的集合，直到指定条件不成立为止。

本节探讨 ActionScript 支持的循环类型及其用法，先从 for 循环开始。

3.3.1 for(初始值;条件;递增)

大多数语言中标准的 for 循环都是让开发人员先指定一个初始值，然后指定循环继续迭代的有效条件，最后根据指定的方式递增初始值。

```
for(var x:int=1;x<=10;x++)
  trace(x);
```

在这个例子中，trace()会把结果输出到日志文件中。循环首先为变量 x 赋初始值 1，然后每执行一次迭代， x 变量的值递增 1，直到 x 的值等于 10 为止。这种形式的循环适合事先知道需要执行多少次函数的情况，例如知道数组中的项数。

3.3.2 for(属性名 in 数组/对象)

对于数组和对象，使用 for...in 循环可以根据属性名迭代包含在数组或对象中的所有项。这种形式的循环适合操作 XML 或 XMLList 对象。

```
var myArray:Array = ["alpha","beta","chi"];
for(var i:String in myArray) {
  trace(i);
}
```

这里有个问题——循环是按照属性名迭代，而数组的属性名是数组的索引。也就是说，前面这个例子的输出结果应该是：

```
0
1
2
```

这种情况下，数组的第一个索引为 0。

既然可以取得索引，就可以通过下面形式取得值。

```
var myArray:Array = ["alpha","beta","chi"];
for(var i:String in myArray) {
  trace(myArray[i]);
}
```

变量 i 是包含在 myArray 中每个项的数字索引，从 0 开始。当 i 为 0 时，取得的是 myArray 中的第一个项。

因此输出应该是：

```
alpha
beta
chi
```

为什么变量 i 在这里是 String 而不是 int 呢？因为 for...in 循环返回的是属性名，而属性名是 String；如果在 for...in 循环中使用 int，大多数情况下会导致出错(在第二个例子中，Flash Player 能够自动转换成数值，因为数组的索引是数值)。

对象的情况也差不多。但与数组(其包含有序的项)不同的是，对象直接包含属性或特性。

```
var myObject:Object = {firstName:'Jeff', lastName:'Smith'};
for(var i:String in myObject) {
  trace("Property: " + i + "=" + myObject[i]);
}
```

提示：

ColdFusion 用户可以把 myObject[i]想象成访问关联数组中的一个键。

输出：

```
Property:firstName=Jeff
Property:lastName=Smith
```

这种循环形式有很多用法。但如果只想得到每项的值，使用 for each...in 循环会更方便。

3.3.3　foreach(项 in 数组/对象)

for each...in 循环用于迭代数组和对象，但测试的不是属性名，而是属性值。

```
var myObject:Object = {firstName:'Jeff', lastName:'Smith'};
for each(var i:String in myObject) {
  trace(i);
}
```

输出：

```
Jeff
Smith
```

这种循环形式是 for..in 循环的简化版。for 循环非常适合迭代数组或对象等已经知道数量的情形，但要想在满足某个条件之前始终保持迭代，则需要使用 while 循环。

3.3.4　while(条件)

while 循环会在指定条件不成立之前始终保持迭代。

```
var x:int = 0;
while(x<5)
{
  trace('x is now ' + x);
  x=x+1;
}
```

输出：

```
x is now 0
x is now 1
x is now 2
x is now 3
x is now 4
```

注意，while 循环在循环开始前检查条件；如果循环开始时条件就不成立，则不会执行任何迭代。

3.3.5 do while(条件)

do while 循环与 while 循环基本相同，但二者微妙却十分重要的区别在于，前者在循环结束后评估测试条件，而不是在开始前。这就比较适合无论测试条件成立与否，都至少要将部分代码运行一次的情况。如果想在循环前先行验证条件，就要使用 while 循环。

```
var x:int = 5;
do
{
  trace('x is now ' + x);
  x=x+1;
} while(x<5);
```

输出：

```
x is now 5
```

不同类型的循环允许根据迭代一组已知量，或在条件成立时执行代码的需要进行全面控制。接下来我们要讨论的是添加条件逻辑。

3.4 条件语句(if 和 switch)

条件语句是编程的基础。与大数语言类似，ActionScript 也实现了两种条件语句：if 和 switch。注意在此处提供的语法，ActionScript 这种情况下与其他任何语言具有相同的语法结构。

3.4.1 if...else

在需要检查条件是否成立(例如用户是否输入了密码)时，if...else 语句是最常用的一种逻辑结构。

下面的简单练习可以帮助我们理解 if...else 的工作原理。

- 加载 HelloWorld 项目。
- 如代码清单 3-1 所示修改代码。
- 在代码生成的文本框中随便填写一个密码，测试一下 if...else 语句是否有效；然后再试试不输入密码的情形。

代码清单3-1　使用if...else条件语句检查密码

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  ➥layout="horizontal">
 <mx:Script>
  <![CDATA[
   public function checkPassword():void
   {
    if(password.text.length < 1)
     mx.controls.Alert.show("Missing Password!");
    else
   mx.controls.Alert.show("Looks Good!");
   }
  ]]>
 </mx:Script>
 <mx:TextInput id="password"/>
 <mx:Button label="Check Password" click="checkPassword()"/>
</mx:Application>
```

包围 Alert 代码的花括号是可选的，但这里，由于 if 和 else 后只有一行代码，所以未予使用。

图 3-1 所示为代码清单 3-1 中新代码创建的文本输入框和提交按钮。

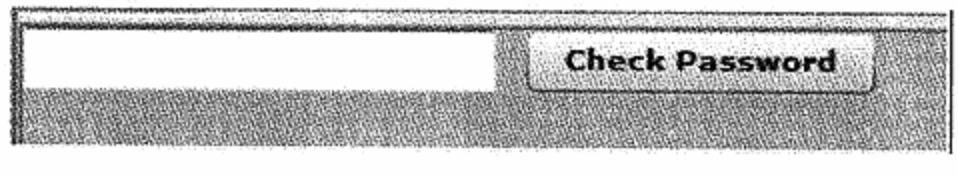


图 3-1　验证用户是否输入密码的简单密码检查程序

运行这个示例，在输入和不输入密码的情况下，会分别得到图 3-2 和图 3-3 所示的警告信息。

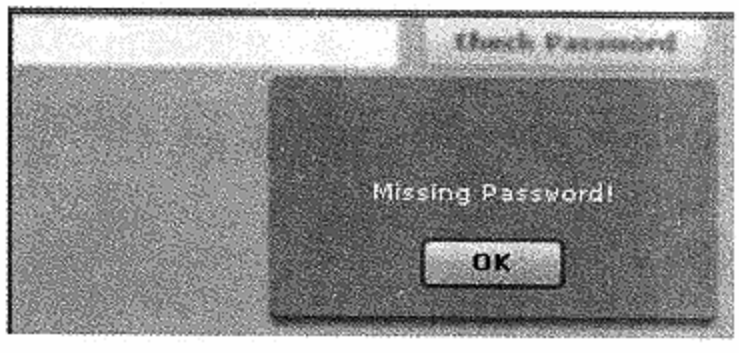


图 3-2　密码字段为空，if 语句测试条件成立，会弹出一个警告窗口

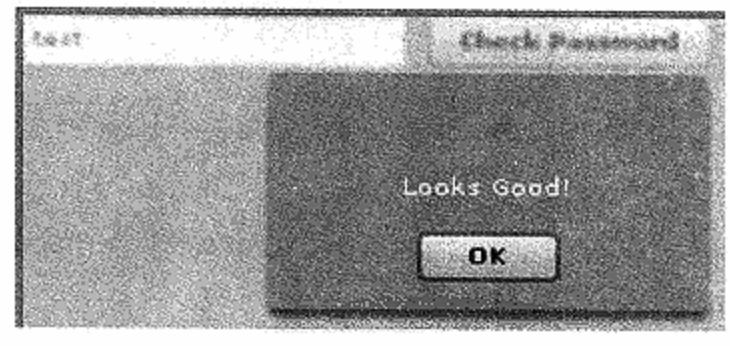


图 3-3　当 if 语句测试条件不成立时，执行 else 分支语句，显示不同的消息

在使用 if 语句时，不一定需要 else 分支语句。在前面的示例中，如果只想提示用户 Missing Password!(未输入密码！)，不想显示 Looks Good!，就可以只使用 if 分支而省略 else 分支。

```
public function checkPassword():void
{
if(password.text.length < 1)
mx.controls.Alert.show("Missing Password!");
}
```

如果还需要在当前条件不成立的情况下，继续检查更多条件，可以使用 else if 命令。我们假设测试用户是否输入了密码只是条件之一，另一个条件是检查密码包含的字符数是否过少，最后把这两个条件之外的情况当作第三个条件。每个条件测试都会向用户显示一条消息。代码清单 3-2 显示代码配置方式。

代码清单3-2 嵌套多个条件语句

```
if(password.text.length < 1)
mx.controls.Alert.show("Missing Password!");
else if (password.text.length < 5)
mx.controls.Alert.show("Not Enough Characters!");
else
mx.controls.Alert.show("Looks Good!");
```

如有必要，还可以添加更多 else if 语句；可是，如果要检查的条件实在太多，代码就会显得笨拙。此时，更好的选择是用 switch 语句取而代之。

3.4.2 switch

在需要评估很多条件时，使用 switch 语句要比重复使用 else if 更简便易行。代码清单 3-3 所示为使用 switch 语句评估多个可能值的情况。

代码清单3-3 使用switch语句可以轻松处理多个条件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            layout="horizontal">
  <mx:Script>
    <![CDATA[
      public function moodCheck():void
      {
        switch(myMood.text)
        {
          case 'happy':
            mx.controls.Alert.show("Life is good isn't it?");
            break;
          case 'sad':
            mx.controls.Alert.show("Some Prozac will cheer you up!");
            break;
          default:
            mx.controls.Alert.show("Neither here nor there eh?");
        }
      }
    ]]>
  </mx:Script>
  <mx:TextInput id="myMood"/>
  <mx:Button label="Check your mood" click="moodCheck()"/>
</mx:Application>
```

结束case块

在上述条件都不成立时执行的后备代码

switch 语句中的 case 块将每个条件组合起来，自然可以使人更加轻松地读取代码，但在使用 case 块时必须牢记几个独有的语法特征。

- 不要把 case 块放在花括号中。
- 每个 case 块必须以一个 break 语句结尾。

- 最后不使用 case 关键字，而使用 default，以便处理不满足前面所有条件的情形。

总而言之，if、else 和 switch 语句是向应用程序中添加条件逻辑的基本方式。下面我们介绍另一个重要的基础概念——数组。

3.5 数组

数组是一种能够存储多个值的变量。假设有 3 个房间，我们想保存每个房间房客的信息，那么可以为每个房客创建一个变量。

```
House1="John Doe";
House2="Michael Dorsay";
House3="Herald McGee";
```

问题在于上面的伪代码不具有扩展性。因为这种方式要求我们必须提前知道需要多少个变量(在大多数实际开发中，这是不现实的)。如果要处理的房间达到几百(或几千)个，那么维护这种代码的工作量将会非常大。

数组倒是非常适合处理这种情况。

```
HouseArray[1]="John Doe"
HouseArray[2]="Michael Dorsay"
HouseArray[3]="Herald McGee"
```

ActionScript 支持 3 种主要数组。

- **索引数组**：以数值作为每个值的键。这种数组适合按照某种顺序存储值的情况。例如，对于需要按顺序逐一执行的一系列任务，可以使用索引数组。
- **关联数组**：以任何形式的键(一般是字符串)与值建立关联。关联数组在建立某种查找机制时可以派上用场。例如，可以把社会保障号码作为键，从而将人的名字及详细资料与一个特定的数字关联。
- **多维数组**：即数组的数组。多维数组适合以键的组合存储和检索信息的情况。而且，数组的每个维度可以混合使用索引数组和关联数组。例如，可以使用一个二维数组保存一个颜色网格，以其中一维作为 x 坐标，以另一维作为 y 坐标。

下面我们深入讨论索引及关联数组的工作原理(多维数组其实就是它们的组合)。

3.5.1 索引数组

这是通常会想到的数组。索引数组通过数值索引来保存数据，并允许通过数值索引访问数据。创建索引数组的代码如下。

```
var houseArray:Array = new Array();
```

以这种方式声明的数组不包含任何项，但传入一个参数可以为数组赋予初始值。

```
var houseArray:Array = new Array(3);
```

用程序员的话来说，把项加载到数组中，就是推入(push)和压入(unshift)项；检索数组中的项，就是退出(pop)和弹出(shift)项。而为了操作更方便，还有一种叫做移接(splice)的

多功能机制，能够同时完成数组项的添加和删除。

提示：

跟我们习以为常的思维方式不同，索引数组第一项的索引是 0，而不是 1。

以下是常用的几种数组函数。

- **push()**：把项添加到数组末尾。

```
houseArray.push("John Doe");
houseArray.push("Michael Dorsay");
```

- **unshift()**：把项添加到数组开头。

```
houseArray.unshift("Herald McGee");
houseArray.unshift("Jon Hirschi");
```

- **pop()**：删除数组末尾的项。

```
houseArray.pop();
```

- **shift()**：删除数组开头的项。

```
houseArray.shift();
```

- **splice()**：从某个位置删除项，然后根据需要插入新项。下面第一行代码在数组第二个位置上插入一项，第二行代码删除第二项。

```
houseArray.splice(1,0,"Tariq Ahmed");
houseArray.splice(1,1);
```

除了 splice()外，其他函数都好理解。下面我们单独谈一谈 splice()。

1. 理解 splice()

理解 splice()函数的关键在于理解它的参数。

- **第一个参数**：指定从数组中的哪个索引位置开始操作。索引数组第一项的索引为 0，因此 1 表示从第二项开始。
- **第二个参数**：指定想要删除的数量，从第一个参数指定的索引开始删除。如果这个参数为 0，则表示不删除任何项。
- **第三个参数**：从第一个参数指定的索引开始插入的新项。插入新项的操作在删除项的操作(可能删除也可能不删除由第二个参数确定)之后执行。

下面这个简单例子从数组中第四个位置开始，删除两项，然后再添加三个新项。

```
houseArray.splice(3,2,"Ryan Stewart","Matt Chotin","Jeff Dougherty");
```

虽然数组是所有语言中都有的一种类型，但 ActionScript 对 splice()函数的实现是独一无二的。有些语言会提供初始数组的简便语法，ActionScript 也不例外。

2. 数组的简便初始化

在 ActionScript 中，可以采用下面的简便语法初始化数组并设置其初始值。

```
var houseArray:Array = new ["John Doe","Michael Dorsay","Herald McGee"];
```

前面这行代码中的方括号表示一个数组。数组中的项以逗号分隔。

对加载了值的数组，最常见的操作是循环遍历。

3. 循环遍历数组

为最大限度发挥数组的优势，往往需要循环遍历数组的一系列索引值。遍历数组最关键是要使用数组的 length 属性，该属性保存着数组包含的项数。

注意：

trace()函数是向 Flash Player 的日志文件中写入文本消息的一种简单记录机制。虽然这里的例子用到 trace()，但有关这个函数的详细内容将在第 23 章再做介绍。

前面讨论 for 循环时，我们提到 for 循环常用于遍历数组。下面看一看代码清单 3-4，顺便温习一下如何使用 for 迭代数组中的每一项。

代码清单3-4　循环遍历数组

```
var myFriendsArray:Array = ["John Doe","Marco Polo","Warren Jones"];
for (var i:int=0;i<myFriendsArray.length;i++)      ← 确定数组大小
{
 trace(myFriendsArray[i]);      ← 输出到日志文件
}
```

代码清单 3-4 所示为在渐次遍历每一项的过程中，怎样利用数组的 length 属性计算包含的项数。

数组每一项的键不一定非得是数值，字符串也可以作为数组的键。

3.5.2　关联数组

关联数组与索引数组的区别在于，与值对应的是键而不是索引。ColdFusion 开发人员可能熟悉关联数组的概念，因为 ColdFusion 中也有一个相同的关联数组机制(使用 Struct)。

创建关联数组的方法与创建索引数组一样。

```
var carManufacturersByModel:Array = new Array();
```

但是，随后没有推入和退出操作，而是像下面这样基于文本字符串赋值。

```
carManufacturersByModel["xA"] = "Scion";
carManufacturersByModel["Viper"] = "Dodge";
carManufacturersByModel["is350"] = "Lexus";
```

然后，要检索与键相关的值，通过键就可以访问。例如：

```
trace(carManufacturersByModel["xA"]);      Output: Scion
trace(carManufacturersByModel["Viper"]);   Output: Dodge
trace(carManufacturersByModel["is350"]);   Output: Lexus
```

最后，我们还要循环遍历关联数组中的所有项，为此就要使用值得信赖的 for...in 循环。

```
for (var key:String in carManufacturersByModel)
{
 trace("Key:"+key);
 trace("Value:"+carManufacturersByModel[key]);
}
```

输出：

```
Key:Viper
Value:Dodge
Key:is350
Value:Lexus
Key:xA
Value:Scion
```

或者，如果我们只关心数组中的值，对相应的键不感兴趣，则可以像下面这段代码一样使用 for each...in 循环。

```
for each (var manufacturer:String in carManufacturersByModel)
{
 trace(manufacturer);
}
```

这样，输出的就只有项的值。

```
Lexus
Scion
Dodge
```

有没有发现输出结果有点不对头？显示顺序(Lexus，Scion，Dodge)跟原来的添加顺序(Scion，Dodge，Lexus)不一致。需要记住的是，关联数组中的项是以无序方式保存的，即无法预言输出的顺序。

另一种完成所有这些工作的方式是使用所有对象的祖先——Object！即不必创建数组实例，而是创建一个对象实例。

```
var carManufacturersByModel:Object = new Object();
```

结果，除了使用对象可以让我们以简写方式初始化数组之外，没有什么太大的分别。

```
var carManufacturersByModel:Object =
   {xA:"Scion",Viper:"Dodge",is350:"Lexus"};
```

ActionScript 新手需要适应的一种语法形式，就是使用冒号赋值(is350:"Lexus")。我们相信，读者一定会适应这种赋值方式。最后，我们再介绍关联数组对点表示法的支持。

1. 使用点表示法

关联数组还有一个出色的特性，就是支持通过点表示法访问数组项。就像也可以使用冒号一样，使用点表示法也可以完成同样的任务。不过，如果习惯于使用特殊编码风格，也许读者更愿意使用以下格式。

```
trace(carManufacturersByModel.xA);
trace(carManufacturersByModel.Viper);
trace(carManufacturersByModel.is350);
```

使用点表示法时要注意一点，即 ActionScript 不允许变量中包含空格。虽然键中包含

空格没有问题，但在使用点表示法进行访问时会引发错误。

```
carManufacturersByModel["Land Cruiser"]="Toyota";
trace(carManufacturersByModel.Land Cruiser);
```

会出错的！

点表示法通常是大多数 Flex 开发人员首选的语法形式，这种形式不仅输入快，也容易看懂。谈到语法，下面我们再介绍一些使用 ActionScript 语言时应该注意的其他很多语法问题。

3.6　ActionScript 特色语法

每种语言都会支持某些特性、简写方式和语法细节。本节简单介绍几个使用 ActionScript 时应该注意的语法问题。先从花括号开始。

3.6.1　花括号

在编写脚本代码块(如函数、循环、switch 和 if 语句)时，要使用花括号告诉 Flex 这些代码块的起始和结束位置——除非该代码块只包含一行代码。

看一看代码清单 3-5 中的示例。

代码清单3-5　必须使用花括号的情况和可用可不用花括号的情况

```
for each (var manufacturer:String in carManufacturersByModel)
{
 trace(manufacturer);
}
```

此时的花括号可用可不用

```
for each (var manufacturer:String in carManufacturersByModel)
 trace(manufacturer);
```

省略了花括号

```
while(x<5)
{
 trace('x is now' + x);
 x=x+1;
}
```

此时的花括号是必需的

```
if(x==5)
 trace(x);
else
{
 x=0;
 trace(x);
}
```

此时可以不用花括号

必须使用花括号

花括号的这种使用规则与 JavaScript 中是相同的，如果读者有 JavaScript 经验，对此一定不会陌生。简言之，如果 if...else 或 for...each 语句后面只有一行代码，就可以省略花括号。如果有一行以上的代码，则必须使用花括号。有些人喜欢在任何情况下都使用花括号。应该说，这只是个人喜好问题，并没有对应的行规。因此，读者也尽可以选择自己认为最顺手的方式。

3.6.2 逻辑操作符及快捷方式

逻辑操作符是特殊字符或字符集，对编译器而言意味着某种操作。下面我们就来看看这些特殊字符，并介绍它们的用途。

1. ++和--

要递增或递减数值，使用++或--操作符是最快、最直接的方式，如下所示：

```
x=x+1;
x++;
x=x-1;
x--;
```

这样可以

但这样速度更快

这两个操作符位于变量哪边都是有效的；如果位于左边，ActionScript 会先修改变量的值，然后在后面的语句中使用变量；否则，先执行语句，再修改变量的值。

```
var x:int=0;
trace(x++);
trace(x);

var y:int=0;
trace(++y);
```

输出0，然后递增x的值！

输出1

递增y的值，输出1

接下来介绍比较操作符，比较操作符与逻辑操作符相似，用于比较值。

2. 基本的比较操作符

当出于测试目的构建一种比较关系时(在 if 或 while 语句中)，可以使用多个操作符相对于一个值比较另一个值。

表 3-4 所示为各种比较操作符及其用途。

表 3-4 比较操作符

操作符	说明	示例
==	确定两个值是否相等	if(x==y) trace("X and Y are the same");
!=	确定两个值是否不相等	if(x!=y) trace("X is not equal to Y");
<	确定一个值是否小于另一个值	if(x<y)trace ("X is less than Y");
<=	确定一个值是否小于等于另一个值	if(x<=y) trace("X is less than or the same as Y");
>	确定一个值是否大于另一个值	if(x>y) trace ("X is greater than Y");
>=	确定一个值是否大于等于另一个值	If(x>=y) trace ("X is greater than or equal to Y");
!	对操作的值求反	var x:Boolean=false; if(!x)trace ("x is not true");

最后，我们介绍用于执行计算的数学操作符。

3. 数学操作符

在操作数值、构建算法或定义公式时，一组数学操作符可以起到帮助作用。表 3-5 所示为所有数学操作符及其说明信息。

表 3-5　数学操作符

操作符	说　明	示　例
*	乘法	z = x * y;
/	除法	z = x/y;

%	取模，即第一个数除以第二个数之后的余数	trace(9 % 3); // output 0
		trace(7 % 3); // output 1

+	加法，两个值相加
	注意：这个操作符也可以用于连接字符串。用于字符串时，+操作符将文本连接在一起
		var z:int = 4 + 2; // z is 6
		var x:String="good"+"bye"; // x is"goodbye"

-	减法	var z= x-y;

下面我们回顾一下刚刚介绍的各种操作符。逻辑操作符像一种快捷方式，比较操作符根据一项比较另一项，数学操作符执行数学运算。

用于封装所有逻辑的代码元素通常是函数。接下来，我们讨论函数及其相关操作。

3.7　函数、类和包简介

在创建应用程序时，最终都要将逻辑分割成较小的模块以便于重用和维护。第 17 章和第 18 章将深入探讨这个主题，现在我们简单介绍一下相关概念。

函数是封装逻辑的代码块，用于方便应用程序中的其他元素以组件方式重用相应代码块。甚至，一个应用程序中的函数也可以被其他应用程序中的元素使用。

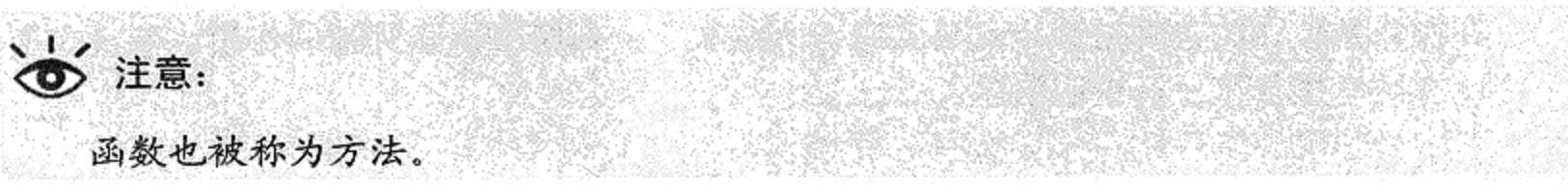
在第 2 章中，我们曾多次用到代码清单 3-6 所示的代码，其中调用了 Alert 类的函数 show()。

代码清单3-6　第2章示例中使用的函数

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute">
  <mx:Button label="Marco" fontSize="40"
          click="mx.controls.Alert.show('Polo')"/>
</mx:Application>
```

函数可以被重用，主要因为它们接收输入参数；这个示例为 mx.controls.Alert.show 传入一个文本参数 Polo。而 mx.control.Alert 则是一个类，类就是一个封装着函数和变量的独

立集合。类的实例通常叫做对象。


注意：

函数也被称为方法。

回到代码清单 3-6，show()是 Alert 类的函数，而 Alert 类又属于一个名为 mx.controls 的包。包就是类的集合。在 ActionScript 中能创建自定义函数吗？当然可以，下面我们就来介绍如何创建函数。

3.7.1 创建函数

声明一个函数很简单，相应代码的结构如图 3-4 所示。

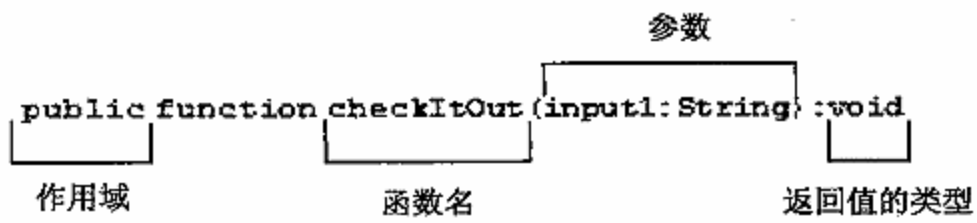


图3-4 ActionScript中的函数声明

图 3-4 中的第一个关键字(public)声明了函数的作用域，即标明谁可以访问这个函数。在这种情况下，“谁”指的是同一个类/组件(我们现在暂且把它们当作对象)中的其他代码，或者当前类/组件外部使用这个函数的代码。表 3-6 所示为与函数相关的几种作用域。

表 3-6 与函数相关的几种作用域

作用域	说明
public	所属对象及任何调用所属对象的代码都可以访问这个函数
private	只有所属对象可以访问这个函数
protected	只有所属对象及其子对象可以访问这个函数

第二个关键字(function)告诉 ActionScript 这里声明的是一个函数。

第三个关键字是函数的名字。只要不以数值开头、不包含空格和非字母非数字(下划线(_)除外)，可以随意给函数命名。

第四个关键字(括号中的部分)是类型声明参数。本章前面已经介绍了数据类型，而这恰好是应用数据类型的一个例子。这个部分的含义是需要声明函数可以接收的所有输入参数，这些参数分别是什么类型(如 int、String 等)。

第五个关键字是返回值类型。使用函数的人通过返回值类型就能知道调用函数返回的值是什么类型，如 Number、Object 或 Boolean。这个示例函数没有返回值，因此使用的是关键字 void。

下面我们就来试着创建一个函数。

1. 自定义函数

有了刚才对于函数的了解，读者可以参照代码清单 3-7 实际动手编写一个自定义函数。

代码清单3-7　创建自定义函数

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  ➥layout="horizontal">
 <mx:Script>
  <![CDATA[
   public function displayMe(input1:String):void
   {
    mx.controls.Alert.show(input1);
   }
  ]]>
 </mx:Script>
 <mx:TextInput id="thisTextInput" text=""try this"/>
 <mx:Button label="Display It" click="displayMe(thisTextInput.text)"/>
</mx:Application>
```

编译并运行这个示例后，可以看到图 3-5 所示的结果。在文本框中输入一些文字，然后单击 Display It 按钮，将刚刚输入的文字传递给函数。

我们来详细描述一下这个示例的执行过程。首先，创建一个输入文本框，让用户可以在其中输入文字。当用户单击 Display It 按钮时，Flex 调用自定义函数 displayMe()，该函数会根据 id 属性的值 thisTextInput 找到文本框，然后捕获其 text 属性的值。这个值就是用户在文本框中输入的文字。

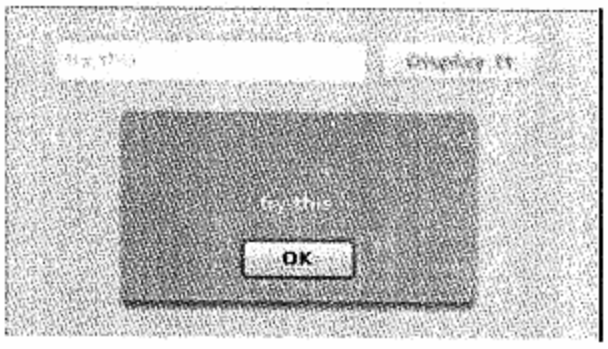


图 3-5　将用户输入的文字传递给函数

自定义函数 displayMe()接收一个参数(即在输入文本框中输入的文字)，并把该参数传递给 Alert 的 show()函数。除了自定义函数外，读者在这个示例中还会发现一些新东西——看起来有点神秘的<![CDATA[…]]>。

2. CDATA

由于 MXML 文件符合 XML 标准，因此必须遵循 XML 的规则。CDATA 在这里告诉 Flex Builder(及编译器)将方括号([])中的内容原样不动，不作为 XML 代码处理。

当然，仅就目前这个示例来说，从技术角度分析还不必使用 CDATA，因为变量的内容没有违反任何规则。但是，如果代码中包含了处理数据用的小于(<)或大于(>)操作符，而且没有使用 CDATA 指令，Flex 就会认为它们是 MXML 中的开始和结束标签。

因此，始终把 ActionScript 代码块包含在 CDATA 指令中是一种最佳实践。

另外，在使用函数时，一个令人厌烦的方面是用于从色调用它们的文本量。好在，有一种方法可以导入整个包，从而减轻了负担。

3. 导入包

假设要对代码清单 3-7 进行扩展，而扩展时还会用到 Alert.show()函数。这时，如果可以在应用程序的整个周期直接使用包含 Alert.show()的包就方便多了。有一种导入操作可以实现我们的愿望。这样一来，就不必在使用 Alert.show()函数时重复输入整个代码块了。

以下代码示范了导入包的效果。

```
import mx.controls.Alert;
public function displayMe(input1:String):void
{
 Alert.show(input1);
}
```

注意：

在执行导入操作时，导入的内容只在当前模块(也就是当前 MXML 文件或 ActionScript 类)中有效。因此不同的模块仍需在必要时再使用导入语句。但这样不会造成资源浪费，Flex 只会根据导入语句指向功能，而不会每次调用时都执行复制操作。

最后，我们要展示的是向调用函数的代码返回值。

4. 返回值

下面我们更进一步，让函数返回一种数据类型的值。代码清单 3-8 已经把 void 修改为 String，也就是说，函数声明它将返回一个字符串值。

代码清单3-8 创建返回String数据类型值的函数

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  ➥layout="horizontal">
 <mx:Script>
  <![CDATA[
   import mx.controls.Alert;
   public function textMerge(input1:String,input2:String):String
   {
    var x:String = input1 + input2;
    return x;
   }
]]>
 </mx:Script>
 <mx:TextInput id="value1"/>
 <mx:Label text="and"/>
 <mx:TextInput id="value2"/>
 <mx:Button label="Join The Two"
        click="Alert.show(textMerge(value1.text,value2.text))"/>
</mx:Application>
```

这个示例也展示了在函数中调用函数的技术。

- 单击按钮，Alert.show()调用 textMerge()，并为 textMerge()传入两个文本输入值。
- 然后，textMerge()把传入的两个字符串连接起来，并把结果返回给 Alert.show()，后者将结果显示出来(图 3-6)。

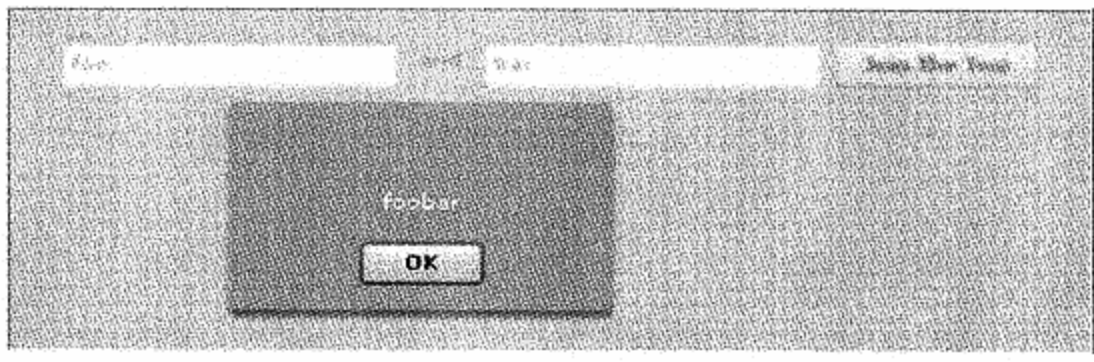


图 3-6　此函数将传入的两个独立字符串连接起来并返回结果

前面所有的示例都是在 MXML 文件中直接编写 ActionScript 代码，就像在 HTML 文件中嵌入 JavaScript 代码一样。这只是一种编写 ActionScript 的方式，另一种方式则是把 ActionScript 代码放在单独的文件中。

3.7.2　将 ActionScript 代码保存在单独的文件中

ActionScript 代码不一定非要跟 MXML 混在一起，也可以单独放到一个文件中。在代码量逐渐增多的情况下，分离 ActionScript 代码更便于维护。而且，多个应用程序通过在逻辑中引入 ActionScript 文件，也会促进函数的重用。

为此，我们把 ActionScript 代码提取到一个文件中，如代码清单 3-9 所示，然后将该文件保存为 myFunctions.as(扩展名是.as)。

代码清单3-9　myFunctions.as——分别保存ActionScript代码

```
public function textMerge(input1:String,input2:String):String
{
  var x:String = input1 + input2;
  return x;
}
```

然后在主 MXML 文件中，可以如代码清单 3-10 所示引入这个 myFunctions.as 文件。

代码清单3-10　主MXML文件引入myFunctions.as文件

```
<?xml version="1.0" encoding="utf-8"?>

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="horizontal">
  <mx:Script source="myFunctions.as"/>     ❸
  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;     ❷
    ]]>
  </mx:Script>
  <mx:TextInput id="value1"/>
  <mx:Label text="and"/>
  <mx:TextInput id="value2"/>
  <mx:Button label="Join The Two"
          click="Alert.show(textMerge(value1.text,value2.text))"/>     ❶
</mx:Application>
```

注意，由于 Button 组件调用 Alert.show()❶，因此必须在同一个文件中导入相应的库❷；这里强调在同一个文件中导入库的原因是，如果把 import 语句转移到 myFunctions.as 文件❸

中，导入内容在主 MXML 文件中❶是不可见的。也就是说，导入内容只在执行 import 语句的文件中可见。

在本章结束前，我们还要介绍一个功能，叫数据绑定(或简称绑定)。

3.8 简单的数据绑定

ActionScript 支持一种功能，能够让一个条目侦听另一个条目的值，这个功能叫做绑定。绑定不仅有助于减少需要编写的代码量，而且，由于该功能可以有效抽象信息的发布者和使用者，因此也能增进应用程序的易扩展性。

下面这个示例中使用视觉组件 TextInput，在其中，你要获得值，并将其显示在其他某个位置。在此，使用 Label 组件。根据前面介绍的技术，可以像代码清单 3-11 所示的那样去做。

代码清单3-11 以手工方式把值从一个位置复制到另一个位置

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            layout="horizontal">
<mx:TextInput id="myTextInput"/>
<mx:Button label="Copy:" click="myText.text=myTextInput.text"/>
<mx:Text id="myText"/>

</mx:Application>
```

如代码清单 3-11 所示，ActionScript 把 TextInput 组件的 text 属性复制给 Text 组件的 text 属性(图 3-7)。

图 3-7 把值从一个变量复制到另一个变量

但是，如果使用绑定则可以自动完成值的复制，如代码清单 3-12 所示。

代码清单3-12 使用绑定可以实现值的自动复制

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            layout="horizontal">
  <mx:TextInput id="myTextInput"/>
  <mx:Text id="myText" text="{myTextInput.text}"/>
</mx:Application>
```

如图 3-8 所示，使用绑定可以实现值的动态复制。

在文本框中输入的文字也会自动、实时地出现在文本框右侧，因为代码已经指定 Text 的 text 属性侦听 TextInput 的 text 属性了。

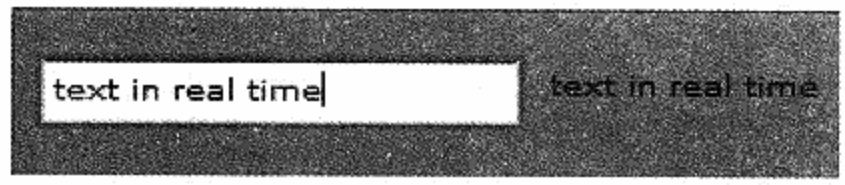


图 3-8　通过绑定实现值的实时捕获

3.8.1　再论事件

为实现绑定，Flex 极大地扩展了事件的概念(参见第 10 章)。侦听器(Listener)可以预订事件，而当事件发生时，侦听器就能侦听到。在前一个例子中，myText.text 被定义为侦听 myTextInput.text 的侦听器(通过一对花括号)。

3.8.2　双向绑定

通过指定组件之间的相互侦听，还可以实现双向绑定。代码清单 3-13 创建了两个侦听对方 text 属性的 TextInput 组件。

代码清单3-13　通过双向绑定可以在组件间实现相互侦听

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   ➡layout="horizontal">
<mx:TextInput id="myTextInput1" text="{myTextInput2.text}"/>
<mx:TextInput id="myTextInput2" text="{myTextInput1.text}"/>
</mx:Application>
```

这样，在一个文本框中输入的文字会自动出现在另一个文本框中(图 3-9)。如果想要使用自己的变量来处理这些事项，可以使用[Bindable]关键字。

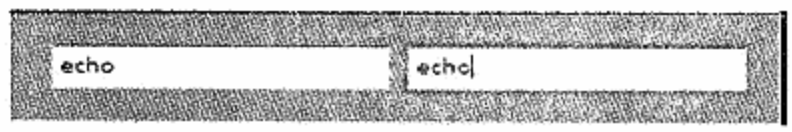


图 3-9　在一个文本框中输入的内容会自动出现在另一个文本框中

默认情况下，Flex 中的一切并不都是可以绑定的，也就是说，必须明确声明可绑定还是不可绑定。代码清单 3-14 明确声明了 String 可以被绑定。

代码清单3-14　声明可绑定变量

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="horizontal">
  <mx:Script>
    <![CDATA[
      [Bindable]
      public var s:String="";
    ]]>
  </mx:Script>
  <mx:TextInput id="myTextInput1"/>
  <mx:Button label="Update my variable" click="s=myTextInput1.text"/>
  <mx:Text id="myText" text="{s}"/>
</mx:Application>
```

在向 TextInput 中输入一些内容后，单击 Update my variable 按钮会把输入的内容复制给变量 String。然后，Text 组件会自动显示变量 String 的值，结果如图 3-10 所示。

图 3-10 先把文本复制给一个可绑定的变量，然后显示该变量的值

在真正的应用程序中，会存在各种组件通过绑定侦听众多变量的情况(我们在应用程序生命周期的不同阶段对这些变量进行配置和管理)。组件会随着数据变化做出反应，但这些组件却不必关心数据和信息的来源。一种常见的情形是，从外部数据源中取得数据，然后将结果保存在本地绑定变量中，而相应的视觉组件负责侦听这些变量并显示相应的信息。

3.9 小结

本章内容比较多，希望读者能够满怀信心地掌握这些知识点。事实上，其中很多内容都是一些基本常识，不必死记硬背。

从结构上说，ActionScript 与其他语言中的许多概念都是相同的，特别是在对数据求值、循环遍历信息和操作数组方面。

从语法上说，ActionScript 可能与读者以前熟悉的语言有很大的出入。但是，ActionScript 毕竟跟 JavaScript 一样，都遵循了 ECMAScript 标准(ActionScript 实现的是最新版本)，因此如果以前使用过 JavaScript，会有似曾相识的感觉。

Flex 提供了数据绑定这个优雅的功能。通过绑定既可以把一个变量赋值给另一个变量，还可以实现数据自动同步。利用这个功能，就可以在应用程序中节省为复制值而编写的所有代码。

在介绍完 ActionScript 之后，就可以深入了解 Flex 了。第 4 章，我们学习如何使用 Flex 的布局功能。

第 4 章 布局和容器

4

本章内容：

- 绝对布局和自动布局
- 基于约束的布局
- 可变和固定大小
- 容器
- 使用 Repeater 动态布局

第 3 章介绍了 ActionScript 的基本概念，本章我们开始视觉效果之旅，让快乐继续！本章主要讨论 Flex 应用程序中用于快速布局的视觉构件(building block)和容器类型。

容器是所有视觉组件依赖的基础。容器可以简化视觉组件的分组和定位。布局由容器承载，但只限于给定容器中的特定组件集合。这些组件通常称为所在容器的子对象，它们可能是按钮、表格、图像甚至是其他容器。

可以把容器想象成一个盒子。可以向盒子中放入各种条目(包括其他盒子)并定义位置，位置因条目而异。

4.1 绝对布局

绝对布局作为一种复合模式，允许我们控制放入容器中的元素的确切位置和大小。

容器基于二维的坐标网格实现定位。任何容器的左上角代表坐标原点(0,0)，而放入容器中的任何组件都相对于这个原点放在网格中。

在容器中使用绝对布局的优点之一是，开发人员可以精确控制对象的位置和大小。另一个优点是 Flex(从技术上说应该是 Flash Player)不必因布局组件显示方式而进行太多计算，相反，它可以使用开发人员为每个组件指定的坐标和维度。在每个容器中使用绝对布局是可选的。也就是说，可以在一个容器中使用绝对布局模式，而在另一个容器中不使用绝对布局。

绝对布局的一个缺点是开发，人员可能因为疏忽而把两个组件设置在同一位置，从而造成重叠。如果容器中盛放的是一组静态组件(即原先就有，而非动态生成的组件)，则不存在

这个问题。

下面我们就从你已经熟悉的 Application 容器开始。如代码清单 4-1 所示，通过容器的 layout 属性(粗体部分)可以告诉 Flex 我们想采用绝对布局模式。

代码清单4-1 在绝对布局中，容器不必因排列子元素而进行多余计算

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            layout="absolute">
  <mx:Button label="Button 1"/>
  <mx:Button label="This is Button 2"/>
</mx:Application>
```

运行这个示例应用程序将得到如图 4-1 所示的结果。由于没有明确指定每个按钮的坐标，因此它们都采取了在容器左上角(即原点(0,0)位置)定位的默认方式。结果，后面的按钮盖住了前面的按钮。

通过为 Button 指定 x 和 y 属性很容易解决这个问题。如图 4-2 所示，设置 Button 2 的 x 属性导致其向右移动了一段距离，下面的代码显示如何进行调整。

```
<mx:Button label="This is Button 2" x="85"/>
```

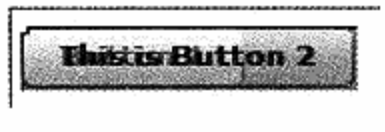


图 4-1 在绝对布局中放置组件而不指定坐标会导致组件重叠

图 4-2 明确设置组件的位置解决了绝对布局中的重叠问题

如图 4-2 所示，明确设置坐标值较好地解决了重叠问题。而且，如果再指定 y 属性的值，则可以把按钮向下移动。

```
<mx:Button label="This is Button 2"
  y="30"/>
```

当然，可以同时设置这两个属性。

```
<mx:Button label="Button 1" x="120" y="50" />
<mx:Button label="This is Button 2" x="100" y="80"/>
```

这里，我们为两个按钮都指定了适当的坐标值，从而得到它们在容器内居中的结果，如图 4-3 所示。

对于较早接触 HTML 的读者而言，应该知道 Flex 环境提供了发挥创意的更大空间。例如，图 4-1 所示的重叠现象如果出现在 HTML 中，通常会被认为是一个 bug。因此，以前很多 Web 开发人员都使用 HTML 表格设置页面的布局来避免重叠发生。在 Flex 中，则可以利用重叠创建新颖的效果。

每个人的创意有所不同，因此重叠不一定是坏事。很可能有读者想在自己的设计中利用重叠创造独特的视觉体验。例如，如图 4-4 所示以逐渐偏移的方式定位文件夹图标，可

以形象地表现出文件夹的结构或深度。

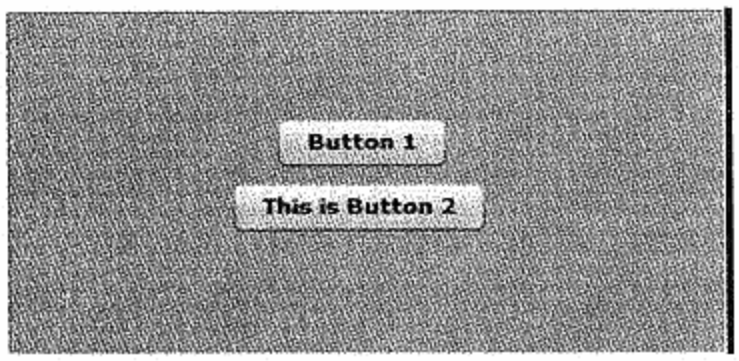


图 4-3　同时指定 x 和 y 属性可以更精确地控制按钮位置

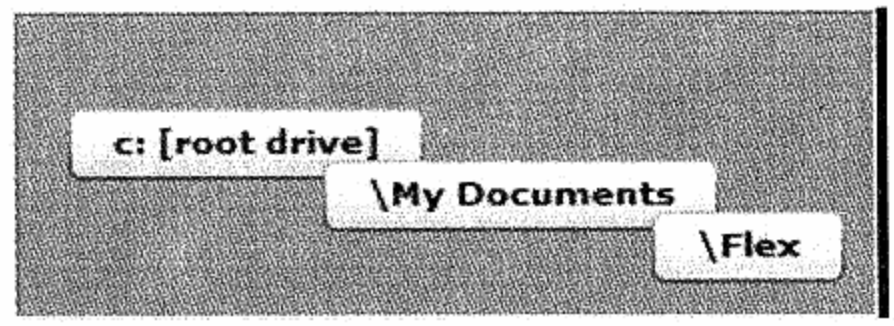


图4-4　通过绝对布局可以实现某些创意

另一个有利于发挥创意的特性是绝对布局中允许 x 和 y 取负值。这样，我们可以事先把组件移到视觉范围(或容器屏幕)之外，然后到必要时再显示它们。代码清单 4-2 所示为这样一个示例。

代码清单4-2　通过为x属性指定负值将组件隐藏在屏幕外部

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            layout="absolute">
  <mx:Button id="button1" label="Show the other button" x="120" y="50"
              click="button2.x=170;"/>
  <mx:Button id="button2" label="Hello!" x="-100" y="80"/>
</mx:Application>
```

ActionScript响应

操纵x参数

图 4-5 所示为单击 Show the other button 按钮后，显示标签为 Hello!的按钮(原来被定位在视图外部)的过程。可见通过操作这些定位属性，可以在 UI 中实现交互性。

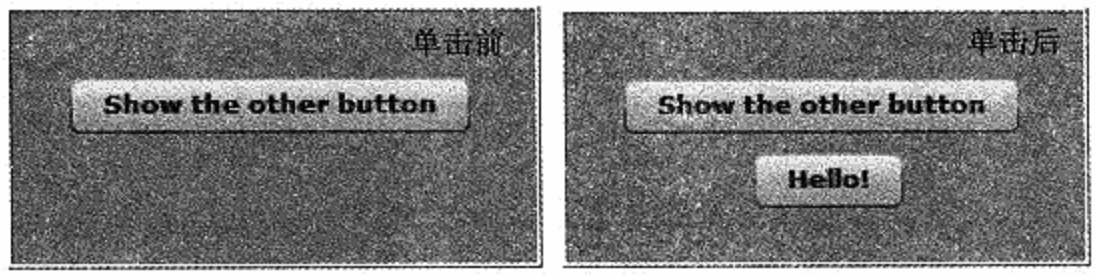


图 4-5　下面的按钮早已存在，只不过修改它的水平坐标后才得以进入视图

提示：

通过 id 属性可以为组件实例起一个唯一的名字(与在 HTML 和 JavaScript 中非常相似)，然后就可以通过该属性访问并操作组件的属性及函数。

使用绝对布局必须事先明确各个组件的位置关系。还有一种基于约束的布局方式，这种方式能相对于某个锚点定位组件。

4.2 基于约束的布局

如果确定应用程序中组件的精确坐标不太现实，可以使用绝对布局模式中基于约束的布局方式。基于约束的布局不是相对于容器左上角通过指定 x 和 y 坐标来定位组件，而是相对于容器的 4 个边或容器中心点来定位组件。

这种布局方式的优点是，可以在用户调整窗口大小时，保持组件与容器间的相对位置关系。例如，有一个距离窗口右下角 5 像素的按钮，用户可以调整窗口的大小，但无论怎么调整，按钮都会保持在距窗口右边和下边 5 像素的位置上。

如果通过固定坐标值来实现上述效果，必须编写很多 ActionScript 代码执行相应计算，并在窗口大小变化时更新组件的 x 和 y 值。

4.2.1 基本约束

使用基于约束的方式，则只需告诉 Flex 让按钮与窗口右边和下边保持 5 像素距离即可。然后，Flex 就可以在后台替我们完成这一任务。表 4-1 所示为指定约束用的所有属性。

表 4-1 基于约束的属性

属 性	类 型	说 明
top	Number	容器上边与组件上边间的像素数
bottom	Number	容器下边与组件下边间的像素数
left	Number	容器左边与组件左边间的像素数
right	Number	容器右边与组件右边间的像素数
horizontalCenter	Number	容器中心点与组件中心点间的水平像素数。注意：指定这个属性时，top 和 bottom 属性无效
verticalCenter	Number	容器中心点与组件中心点间的垂直像素数。注意：指定这个属性时，left 和 right 属性无效

下面我们就在前面示例的基础上应用表 4-1 所示的属性。代码清单 4-3 所示为把按钮锚定到页面右下角的代码，结果如图 4-6 所示。

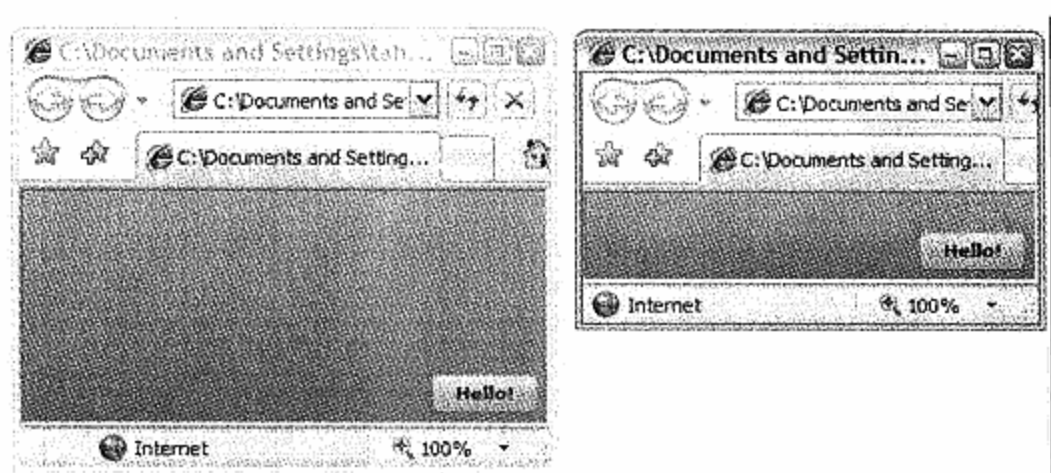


图 4-6 使用基于约束的布局方式把按钮定位在页面的右下角

代码清单4-3　把按钮定位在距离窗口右下角5像素的位置上

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            layout="absolute">
  <mx:Button label="Hello!" bottom="5" right="5"/>
</mx:Application>
```

如图 4-6 所示，无论窗口大小或形状如何变化，按钮始终保持与窗口右下角的相对位置关系。如果读者感觉编写代码不够方便，还可以通过 Flex Builder 提供的另一个功能来方便地配置组件(图 4-7)。

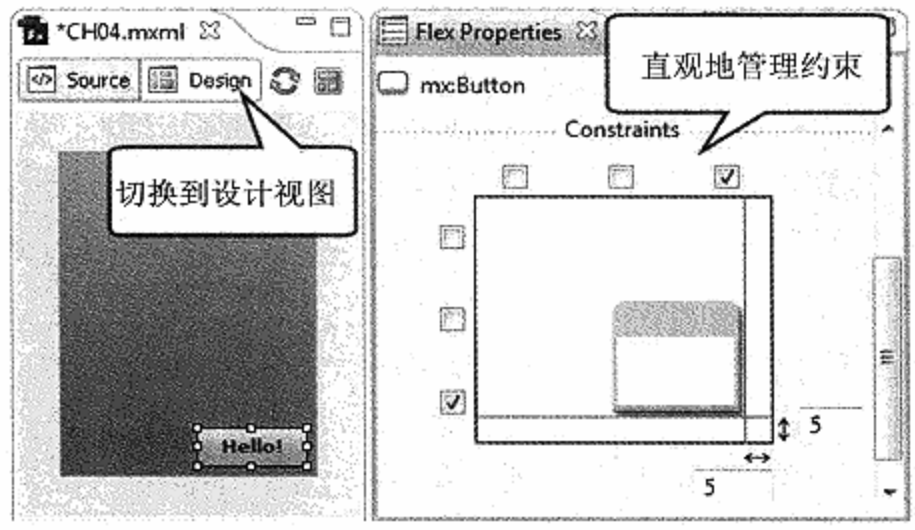


图 4-7　在 Flex Builder 的设计视图中可以通过交互方式控制约束

(1) 在 Flex Builder 中，单击 Design View 按钮切换到设计模式。
(2) 单击 Button 组件。
(3) 在 Flex Properties 窗口中，向下滚动到 Layout 部分。
(4) 注意布局窗口上边和左边的复选框，其中最右和最下边的复选框是被选中的。
(5) 随机选择其中的复选框并输入相应数值，即可看到按钮在窗口中定位的效果。例如，要想让按钮出现在页面中央，应该如图 4-8 所示单击上边和中间的复选框。

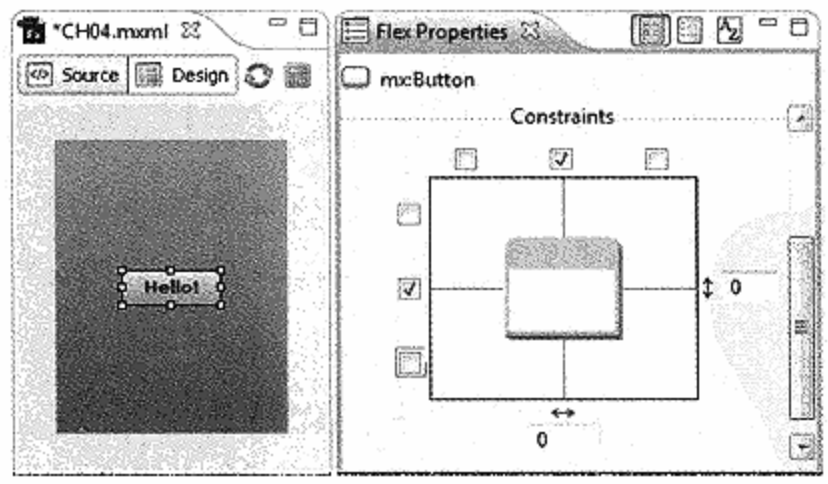


图 4-8　通过约束可以相对于中心点及窗口的四边定位对象

注意：

由于 Flash 有作为动画工具的血统，因此通常把页面称为舞台(stage)。Flex 开发人员有两类，一类是传统的开发人员，一类是 Flash 开发和设计人员。在相关论坛或博客帖子中，

有 Flash 背景的人经常使用舞台这个术语。因此，他们所说的“在舞台上居中”，意味着在容器或页面居中。

综上所述，要在容器中精确地定位组件(无论浏览器大小)，使用绝对布局；要想相对于窗口大小变化调整组件的位置，同时保持组件与浏览器边界(或中心点)的固定距离，则使用基于约束的布局可以自动实现这一效果。

如果读者觉得基于约束的布局更方便，但还想使控制更精确，那么增强的约束布局可以提供更大灵活性。

4.2.2 增强的约束

Flex 3 新增的增强约束功能在基本约束的条件下更进了一步，允许开发人员在水平和垂直方向上任意创建隐藏的辅助线，然后对照辅助线定位组件。这些辅助线分别叫做约束行和约束列。基本的基于约束的布局只允许相对于容器边界或中心(锚点)布局，而辅助线扩展了对定位的控制能力。

约束行和约束列可以按照 3 种方法来使用。

- **固定约束**：位置由绝对数字(像素)来指定。
- **相对约束**：位置根据相对于容器大小的百分比来确定。
- **内容大小约束**：与相对约束类似，但位置相对于内容确定(即随容纳内容而变大或变小)；而相对约束则基于容器大小变化。

表 4-2 和表 4-3 所示为用于实现约束辅助线的两个标签(<mx:ConstraintColumn>和<mx:ConstraintRow>)所支持的属性。

表 4-2 ConstraintColumn 的属性

属 性	类 型	说 明
id	String	约束列的唯一标识符(名字)
width	Number	可选，用于确定列宽。既可以是固定数值，也可以是相对于容器总宽度的百分比。如果未指定，则约束列的宽度会基于条目大小自动缩放
minWidth	Number	可选，指定列的最小宽度
maxWidth	Number	可选，指定列的最大宽度

表 4-3 ConstraintRow 的属性

属 性	类 型	说 明
id	String	约束行的唯一标识符(名字)
height	Number	可选，用于确定行高度。既可以是固定数值，也可以是相对于容器总高度的百分比。如果未指定，则约束行的高度会基于条目大小自动缩放
minHeight	Number	可选，指定列的最小高度
maxHeight	Number	可选，指定列的最大高度

乍一看，增强约束模式的属性似乎不太好理解。下面，我们不解释使用这些属性的所有规则及 Flex 如何应用它们，而是首先来看几个示例，以便直观地理解它们的含义和作用。

提示：

由于我们假设布局控件带有约束，因此基本上只能在 Canvas 容器中使用这种模式。

理解增强约束的最方便方式，就是把它们看成一种分割容器的机制。接下来，我们从分割容器的角度展示几个示例。

1. 两列固定分割

首先把容器分割为两个区域。如代码清单 4-4 所示，通过<mx:ConstraintColumn>标签可以在容器中创建两个列。

代码清单4-4　使用两个约束列对齐两个按钮

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  backgroundColor="white">
 <mx:Canvas width="100%" height="100%">
  <mx:constraintColumns>
   <mx:ConstraintColumn id="col1" width="200"/>
   <mx:ConstraintColumn id="col2" width="50" />
  </mx:constraintColumns>
  <mx:Button label="Button 1" left="col1:0"/>
  <mx:Button label="Button 2" left="col2:0"/>
 </mx:Canvas>
</mx:Application>
```

代码清单 4-4 生成两个按钮：Button 1 位于宽度为 200 像素的第一个约束列，Button 2 位于右侧宽度为 50 像素的第二个约束列(图 4-9)。

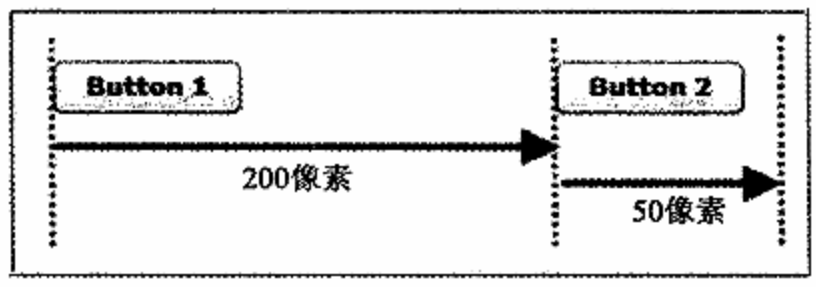


图 4-9　这两个按钮分别与命名的约束对齐

有关代码清单 4-4 的说明如下。

- 通过指定每个按钮的 left 约束属性，使它们基于相应的 ConstraintColumn 定位。
- 语法 col1:0 的第一部分指定将 col1 用作 ConstraintColumn，而第二部分的 0 表示按钮相对于 ConstraintColumn 偏移 0 像素，即无偏移。

这个示例很直观，因为约束列看起来像是图像处理工具中为对齐物体而创建的辅助线。

2. 带左右约束的两列固定分割

接下来，我们打算把按钮宽度拉伸到与其所在的约束列同宽。为此，可以指定 right 约束属性，但仍然使用原来的 ConstraintColumn。

```
<mx:Button label="Button 1" left="col1:0" right="col1:0"/>
<mx:Button label="Button 2" left="col2:0" right="col2:0"/>
```

如图 4-10 所示，left 属性会让按钮与 ConstraintColumn 的左边对齐，而 right 属性则会让按钮与 ConstraintColumn 的右边对齐。

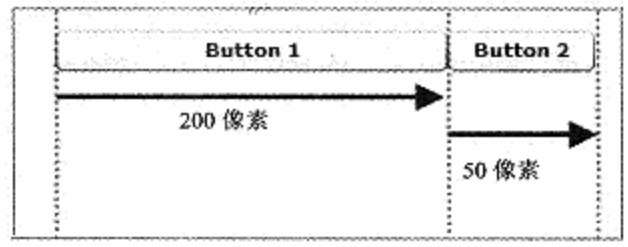


图 4-10 这两个按钮都拉伸到与所在的约束列同宽

仅仅拉伸一下按钮还算不了什么。如果再添加上约束行，这个示例就会变得更有意思。

3. 带上下约束的两行固定分割

使用约束行可以在页面的垂直方向实现类似效果。代码清单 4-5 同样定位了两个按钮，一个位于另一个上方，而且使用了相对和固定 ConstraintRow。

代码清单4-5 创建一个相对和固定大小的ConstraintRow

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:Canvas width="100%" height="100%">
    <mx:constraintRows>
      <mx:ConstraintRow id="row1" height="50%"/>      ← 相对大小
      <mx:ConstraintRow id="row2" height="100"/>      ← 固定大小
    </mx:constraintRows>
    <mx:Button label="Button 1" top="row1:0" bottom="row1:0"/>
    <mx:Button label="Button 2" top="row2:0" bottom="row2:0"/>
  </mx:Canvas>
</mx:Application>
```

这种方法展示了混合使用绝对和相对大小的技术。图 4-11 所示为这个示例的结果，其中的关键如下。

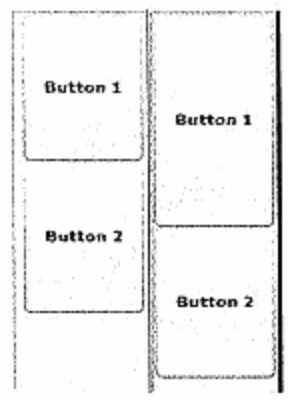


图 4-11 因为处于相对约束行中，Button 1 的高度会变化，Button 2 的高度则始终固定

- Button 1 会随着用户调整浏览器窗口的大小缩放，始终保持其高度等于画布(canvas)高度的 50%。
- Button 2 则永远保持 100 像素高，并在垂直方向上从 Button 1 的结束位置开始布局。

至此，我们已经看到单独使用约束列和约束行的示例。接下来这个示例展示了混合使用约束行和约束列。

4. 有偏移的两行加两列分割

如果应用程序需要，也可以组合使用约束行和约束列。代码清单 4-6 所示为这种用法；不过，其中也展示了偏移参数的使用。

代码清单4-6　创建约束行和约束列

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white">
  <mx:Canvas width="100%" height="100%">
    <mx:constraintColumns>
      <mx:ConstraintColumn id="col1" width="100"/>
    </mx:constraintColumns>
    <mx:constraintRows>
      <mx:ConstraintRow id="row1" height="50"/>
      <mx:ConstraintRow id="row2" height="50"/>
    </mx:constraintRows>
    <mx:Button label="Button 1" left="col1:0" right="col1:0"
                        top="row1:0" bottom="row1:0"/>
    <mx:Button label="Button 2" left="col1:10" right="col1:10"
                        top="row2:0" bottom="row2:0"/>
  </mx:Canvas>
</mx:Application>
```

图 4-12 所示为执行代码清单 4-6 中代码的结果。由此可见，通过混合搭配相应的约束，读者可以尽情发挥自己的聪明才智，从而生成和控制各种视觉效果。

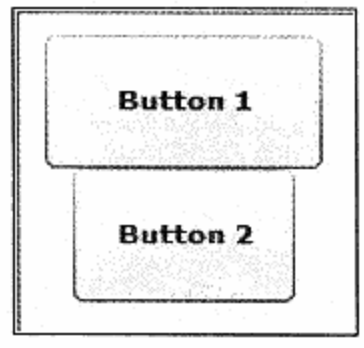


图 4-12　使用行和列约束定位按钮

如图 4-12 所示，通过为 Button 2 的偏移参数指定值，使该按钮从列左右两侧各缩进 10 像素。

在实际做项目时，虽然还可以设计出更有创意的布局，但如此使用约束也存在不容易理解的问题。最后，我们看一看基于内容的约束。

5. 基于内容的约束

基于内容的约束是在不指定约束的 height 和 width 属性的值时自动启用的。在基于内容约束的方式下，Flex 会把所有条目缩放至列宽或行高，而列宽和行高则由容器中最大的条目决定。代码清单 4-7 所示为这一过程。

代码清单4-7 使用基于内容的约束

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:Canvas width="100%" height="100%">
    <mx:constraintColumns>
      <mx:ConstraintColumn id="col1"/>
      <mx:ConstraintColumn id="col2"/>
    </mx:constraintColumns>
    <mx:constraintRows>
      <mx:ConstraintRow id="row1" height="50"/>
      <mx:ConstraintRow id="row2" height="30"/>
    </mx:constraintRows>
    <mx:Button label="Button 1" left="col1:0" top="row1:0" width="200"  />
    <mx:Button label="Button 2" left="col1:0" top="row2:0"/>
    <mx:Button label="Button 3" left="col2:0"/>
  </mx:Canvas>
</mx:Application>
```

这个示例产生的结果就像在 HTML 表格中把列宽设置为 0 的技巧，此时的列收缩为能够容纳其中图像的最小尺寸。与 HTML 表格不同的是，Flex 中的“表格”可不是一个 hack，我们拥有对它的全权控制，如图 4-13 所示。

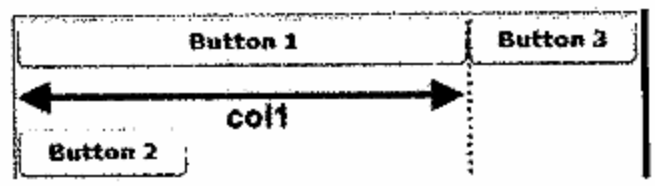


图 4-13 在基于内容的约束下，列拉伸到与其中最大组件同宽

Button 1 和 Button 2 使用了相同的 ConstraintColumn，但由于 Button 1 是该容器中最大的组件，因此 ConstraintColumn 默认与其保持同宽。

6. 加深理解

要深入理解基于约束的布局，最好的办法就是亲手试验。如果把各种可能的约束组合都列举出来，恐怕占用整个一章的篇幅。因此，读者尽可发挥自己的想象力，反复试验各种随机组合，并查看得到的结果。这里需要指出另一个重要功能：可以使用 ActionScript 来操作约束属性。例如，可以基于用户操作或特定事件链改变元素大小(第 13 章将讨论如何使用视图状态做到这一点)。

基于约束进行布局虽然强大，但为保证应用程序具有理想的布局仍然要做不少工作。如果读者并不十分在意组件的实际位置，也可尝试一下自动布局。

4.3　自动布局

自动布局与绝对布局大小不同，差别在于绝对布局要求明确指定容器中元素的位置和布局方式。而在自动布局模式下，则是由容器替我们决定元素的位置。自动布局有两种布局模式(也称为布局方向)：vertical(垂直)和 horizontal(水平)。换句话说，容器会根据指定的布局方向一个接一个地摆放子对象(代码清单 4-8)。

代码清单4-8　使用自动水平布局并列摆放两个按钮

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="horizontal">
<mx:Button label="One"/>
<mx:Button label="Two"/>
</mx:Application>
```

图 4-14 和图 4-15 所示为由容器控制的水平和垂直布局模式的差异。在图 4-14 所示界面中，容器以并列方式定位按钮，因为代码清单 4-8 中把布局模式设置为 horizontal。如果把这个属性修改为 vertical，将得到图 4-15 所示的结果。

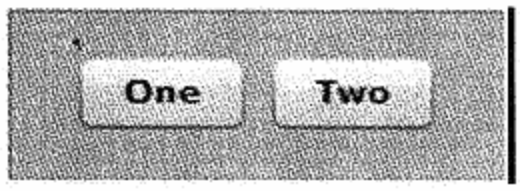


图4-14　应用程序的容器自动并列摆放按钮(沿水平方向)

图4-15　把布局属性设置为vertical，会导致按钮沿垂直方向摆放

显然，自动布局组件比手工定位轻松得多。另一个控制选项是 Spacer。可以把 Spacer 想象成 HTML 中的一个空列(<td width="100%"></td>)，或者在 HTML 中使用的 spacer.gif 技巧，这种技巧有时用于平均分布元素。表 4-4 所示为 Spacer 的属性及说明。

表 4-4　Spacer 的属性

属　性	类　型	说　明
width	Number	确定水平方向占用的空间宽度。接受绝对数值或百分比
height	Number	确定垂直方向占用的空间高度。接受绝对数值或百分比
maxWidth	Number	指定允许 Spacer 占用的最大宽度。接受绝对数值或百分比
maxHeight	Number	指定允许 Spacer 占用的最大高度。接受绝对数值或百分比
minWidth	Number	指定允许 Spacer 占用的最小宽度。接受绝对数值或百分比
minHeight	Number	指定允许 Spacer 占用的最小高度。接受绝对数值或百分比

代码清单 4-9 指定 Spacer 尽最大可能占用所有空间(100%)，即明确指定的 300 像素。

代码清单4-9 使用Spacer分离两个按钮

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            layout="horizontal">
  <mx:Button label="One"/>
  <mx:Spacer width="100%" maxWidth="300"/>
  <mx:Button label="Two"/>
</mx:Application>
```

图 4-16 显示 Spacer 如何通过应用元素之间(在这种情况下，就是两个按钮之间)的可用空间来分离两个按钮。

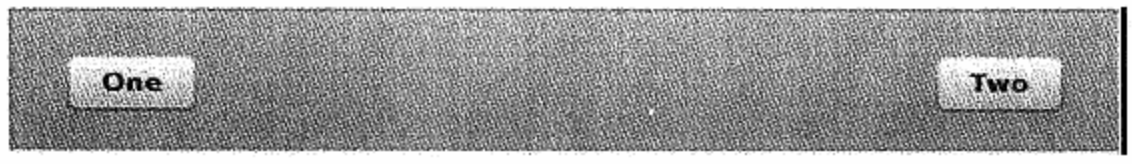


图 4-16 通过 Spacer 把按钮分别放在两侧

在自动布局中，可以使用任意数量的 Spacer 实现布局目标。代码清单 4-10 所示为如何平均分隔 3 个按钮，同时又在容器侧配置 10 像素的偏移。

代码清单4-10 使用多个Spacer可以实现对多个组件的平均分布

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            layout="horizontal">
  <mx:Spacer width="10"/>
  <mx:Button label="One"/>
  <mx:Spacer width="50%"/>
  <mx:Button label="Two"/>
  <mx:Spacer width="50%"/>
  <mx:Button label="Three"/>
  <mx:Spacer width="10"/>
</mx:Application>
```

使用多个 Spacer 加上相对大小(百分比)，可以达到平均分布组件的目的(图 4-17)。

图 4-17 中间距离相等，两侧偏移大小固定的设置让按钮各就各位

无论为项目使用自动布局还是绝对布局，组件本身的大小都是可以单独控制的。

4.4 可变大小和固定大小

所有 Flex 视觉组件支持可变大小和固定大小；这一点与 HTML 相似，却远远超出 HTML。

将元素的 width 和 height 属性设置为固定数值(像素)，即可实现固定大小；将元素的 width 和 height 属性设置为百分比数值，即可实现相对于所在容器的大小缩放。

4.4.1　可变大小

为示范可变大小，我们仍然以任劳任怨的 Button 组件为例。如果我们希望按钮按比例缩放，但最大不超过所在容器宽度的 80%，可以使用以下代码。

```
<mx:Button label="Hello!" width="80%"/>
```

如图 4-18 所示，会导致按钮宽度达到浏览器窗口宽度的 80%。

图 4-18　缩放到 80%会导致按钮随页面大小变化，并且始终占据 80%的宽度

同样的技术也适用于垂直方向，或者也可以同时设置组件的 width 和 height 属性。

```
<mx:Button label="Hello!" width="80%"
➥height="80%"/>
```

如果想让按钮在窗口内居中，可以设置 horizontalCenter="0"和 verticalCenter="0"。

以上就是可变大小的用法。接下来看一看与此相对的固定大小。

4.4.2　固定大小

固定大小跟可变大小的设置方法差别不大。仍然是设置相同的参数，只不过要使用整数(像素)指定组件大小，而不再指定百分比。

要得到一个宽 120 像素、高 80 像素的按钮，可以这样编写代码。

```
<mx:Button label="Hello!" width="120" height="80"/>
```

可见，固定大小的设置非常简单，没有什么好说的。在掌握了如何控制组件的大小和位置，理解了自动布局的工作原理后，将继续探讨利用这些功能的容器。

4.5　容器

容器为应用程序提供了一种视觉结构，即可在其中放置显示组件的框架(frame)。Flex 内置了多种容器，这些容器各有特色。但容器的目的只有一个，即从视觉而非编程方面辅助对组件的布局。

4.5.1 Application 容器

我们一直在用的 Application 容器比较特殊。所有 Flex 程序都包含一个 Application 容器。但由于应用程序以该容器为根元素，所以每个创建的应用程序都只能包含一个 Application 容器。从布局的角度看，通过 Application 容器的 layout 属性可以将其设置为自动或手动定位(表 4-5)。

表 4-5 layout 属性接收 3 个值

值	说 明
absolute	指示容器使用绝对定位模式，即明确指定每个子组件的位置
horizontal	指示容器自动沿水平方向排列其子组件
vertical	指示容器自动沿垂直方向排列其子组件

按与下面类似的方式将这些值添加到代码中。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  ➥layout="absolute">

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  ➥layout="vertical">

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  ➥layout="horizontal">
```

根容器 Application 的另一个特色属性是 usePreloader，该属性用于指定启动 Flex 应用程序时是否显示进度条。以下代码要求不显示进度条。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  usePreloader="false">
```

Application 是应用程序中的顶级对象，它可以保存全局变量和函数，并且支持应用程序中的代码访问这些变量和函数。

虽然自定义组件是第 17 章将要详细讨论的主题，但我们在此简单介绍一下。自定义组件是代码模块化的一种手段，代码清单 4-11 所示为调用自定义组件的方式。

代码清单4-11 调用自定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:local="*">
 <mx:Script>
  public var myString:String = "hello!";  <-- 消息
 </mx:Script>
 <local:customComponent/>
</mx:Application>
```

这个主应用程序文件中也包含一个公共变量(任何对象或代码都可以访问它)，该变量保存着一条消息。在名为 customComponent.mxml 的文件中，使用自定义组件访问该变量(代码清单 4-12)。

代码清单4-12　customComponent.mxml——访问应用程序中的变量

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Button label="{mx.core.Application.application.myString}"/>
</mx:Canvas>
```

运行以上应用程序后，我们会看到一个标签为 hello!的按钮。但是，一定要记住，无论文件中包含多少其他类型的容器，一个应用程序只能使用一个 Application 容器。为了简明起见，下面介绍核心容器 Canvas。

4.5.2　Canvas 容器

Canvas 是一种最基本的容器，但是由于它的轻量级设计，导致其功能不那么强大。虽然使用 Canvas 容器可以提高性能，但代价是功能减弱。具体来说，性能提高以自动布局其中的任何可视组件的能力的削弱为代价。Canvas 容器假设我们会使用本章前面介绍的绝对和基于约束的布局方式，手工设置布局。

如果需要为组件指定具体大小，或者包含的组件较为简单而无需自动布局，那倒是可以利用 Canvas 容器来有目的地增进应用程序性能。

上一个示例充分展示了 Canvas 容器的好处。其中，将 Canvas 用作自定义组件，该组件被嵌入到更大的元素中。使用 Canvas 要注意的是，如果 Canvas 容器中包含多个条目，必须仔细设置这些组件的位置，否则很容易造成组件的重叠。代码清单 4-13 所示为如何使用多个 Canvas 容器分组组件。

代码清单4-13　使用多个Canvas容器分组组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            layout="vertical" >
 <mx:Canvas width="200">
   <mx:Text text="A"/><mx:Button label="B" x="20" y="0"/>
 </mx:Canvas>
 <mx:Canvas width="200">
   <mx:Text text="C"/><mx:Button label="D" x="20" y="0"/>
 </mx:Canvas>
</mx:Application>
```

这里比较有意思的地方有几处。一是通过为 Application 标签指定 layout="vertical"属性导致两个画布(Canvas)元素被自动定位(图 4-19)。二是虽然两个按钮都使用了相同的 x 和 y 属性，但由于它们各自相对于父容器左上角(而非屏幕或应用程序左上角)定位，因此这两个按钮并没有重叠。

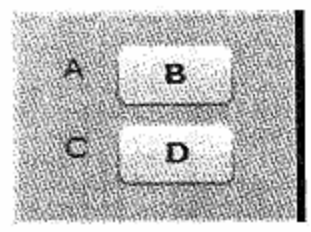


图 4-19　多个 Canvas 用于分组多个组件

如图 4-19 所示，两个画布元素由于 Application 容器的自动布局功能而上下堆叠起来。在每个画布内，位置则是手工控制的。

如果读者希望实现自动布局，可以使用专门为此设计的 Box 容器。

4.5.3 Box、HBox 和 VBox 容器

组合使用 Box 容器可以相当灵活地排列应用程序中的组件。特别是在可变宽度的应用程序中(例如，预期用户会改变窗口大小)，我们希望让 Flex 计算处理布局的最佳方式时，使用 Box 容器尤其方便。

Box 是一种方形容器，默认会沿垂直方向布局其子对象(代码清单 4-14)。

代码清单4-14 Box容器用于沿垂直方向自动布局组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >
  <mx:Box>
    <mx:Button label="A"/><mx:Button label="B"/>
  </mx:Box>
</mx:Application>
```

代码清单 4-14 中的代码生成了图 4-20 所示的两个上下堆叠在一起的按钮组件。虽然可以通过使用 direction="horizontal" 属性改变布局方向，但更直接的方式是使用 HBox(Horizontal Box，水平盒子)和 VBox(Vertical Box，垂直盒子)容器。

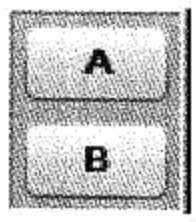


图 4-20 Box 容器会自动处理布局

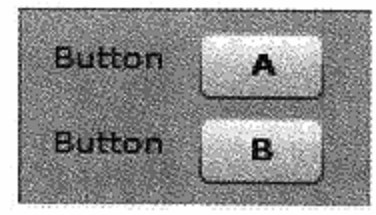


图 4-21 嵌套盒子可以沿水平和垂直方向布局组件

通过嵌套盒子还能实现更理想的效果，如代码清单 4-15 所示。

代码清单4-15 使用嵌套盒子实现更复杂的布局

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >
  <mx:VBox>
    <mx:HBox>
      <mx:Text text="Button"/> <mx:Button label="A"/>
    </mx:HBox>
    <mx:HBox>
      <mx:Text text="Button"/> <mx:Button label="B"/>
    </mx:HBox>
  </mx:VBox>
</mx:Application>
```

在编译应用程序之后，可以看到图 4-21 显示的包含文本和按钮的类似表格的布局形式。

注意，如果嵌套过深，会明显影响性能。这是因为 Flex(通过 Flash Player)计算元素在屏幕上的位置时，组件及其所有子对象之间需要进行复杂的交流。对此，一条经验性的规

则是，如果的确需要嵌套，尽管使用；但是，如果还有其他选择，绝对应该优先考虑其他选择。

4.5.4 Panel 容器

作为大众选择的结果，Panel(面板)容器常作为整个应用程序的顶级容器使用(尽管 Panel 中还能嵌套 Panel)。该容器的一个奇妙之处在于，在窗口顶部添加了标题和状态栏，而且该容器默认带有边框(用于包围子对象)。

Panel 容器的默认行为是垂直布局组件。但是，如代码清单 4-16 所示，通过 layout 属性可以将默认行为修改为水平或绝对布局。

代码清单4-16　使用layout属性修改组件布局方式

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Panel title="My Application" status="Welcome"
       width="200" layout="horizontal">
<mx:Text text="Button"/> <mx:Button label="B"/>
</mx:Panel>
</mx:Application>
```

由于把 layout 属性设置为 horizontal，导致 Text 和 Button 组件并排显示(图 4-22)。

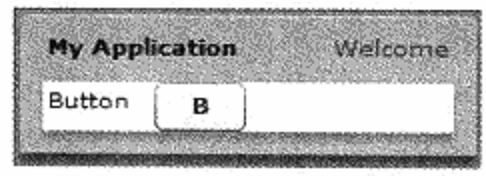


图 4-22　使用水平布局导致组件并排显示

除了好看的边框和顶部的标题之外，Panel 容器还支持诸如 ControlBar 的功能。

4.5.5 ApplicationControlBar 和 ControlBar 容器

这两个容器的特点是不能单独使用。顾名思义，两个容器用于为应用程序添加控制条区域。下面将进行解释。

ApplicationControlBar 容器创建的区域在 Windows 应用程序中很常见。例如，可以通过 ApplicationControlBar 对“文件”、“编辑”和“查看”等菜单项进行分组。代码清单 4-17 所示为如何组合使用 ApplicationControlBar 和 Application 容器。

代码清单4-17　ApplicationControlBar可以方便用户对功能的访问

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:ApplicationControlBar width="300">
    <mx:Button label="Back"/>
    <mx:Button label="Forward"/>
    <mx:TextInput width="60"/><mx:Button label="Search"/>
  </mx:ApplicationControlBar>
</mx:Application>
```

类似地，ControlBar 容器可以与 Panel 或 TitleWindow(更通俗的名称是弹出式窗口)结合使用。代码清单 4-18 所示为在面板中添加 ControlBar 的例子。

代码清单4-18 在面板中使用ControlBar容器

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Panel title="My Application" layout="vertical" width="300">
<mx:Text text="What's your email?"/>
<mx:TextInput/>
<mx:ControlBar>
<mx:Button label="Cancel"/>
<mx:Button label="Save"/>
</mx:ControlBar>
</mx:Panel>
</mx:Application>
```

组合代码清单 4-17 和代码清单 4-18，最终得到的布局效果如图 4-23 所示。

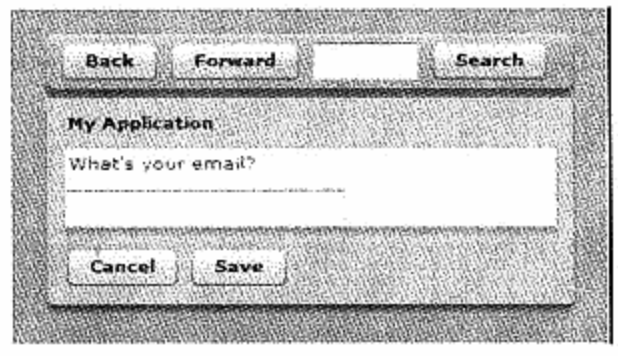


图 4-23 ApplicationControlBar 用在顶部；面板也使用一个 ControlBar

控制条区域是添加工作流及访问应用程序范围的功能的简便方式。

4.5.6 DividedBox、HDividedBox 和 VDividedBox 容器

可以把这组容器看成 Box 的“老大哥”。它们通过分割条(divider bar)来分割盒子，而用户拖动分割条可以调整盒子的大小。这种特性类似启用了框架大小调整功能的 HTML 框架(frame)。

与 Box 容器类似，DividedBox 支持通过 direction 属性设置水平和垂直(默认)布局方式。而 HDividedBox(水平布局)和 VDividedBox(垂直布局)是两种更直接的方式。

用户通过鼠标拖动分割条可以调整窗格大小，从而自定义视觉空间。另外，嵌套使用这些容器可以实现更有趣和更有用的布局。

代码清单 4-19 通过使用多个 DividedBoxe，达到分割应用程序的目的。

代码清单4-19 使用分割盒子为用户提供调整屏幕分区的功能

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Panel title="Reports Dashboard" width="300">
    <mx:DividedBox direction="horizontal" width="100%">
      <mx:VBox backgroundColor="#eeeeee" height="100%">
        <mx:Text text="Categories" fontWeight="bold"/>
        <mx:Button label="Finance" width="100%"/>
        <mx:Button label="Operations" width="100%"/>
        <mx:Button label="I.T" width="100%"/>
```

```
      <mx:Button label="Support" width="100%"/>
      <mx:Button label="Network" width="100%"/>
    </mx:VBox>
    <mx:VDividedBox width="50%" height="100%">
      <mx:VBox backgroundColor="#eeeeee" width="100%" >
        <mx:Text text="Finance Reports" fontWeight="bold"/>
        <mx:Text text="2008 Q1 Sales"/>
        <mx:Text text="2008 Q2 Sales"/>
      </mx:VBox>
      <mx:VBox backgroundColor="#eeeeee" width="100%" >
        <mx:Text text="2008 Q1 Sales" fontWeight="bold"/>
        <mx:Text text="North America: $2,832,132"/>
        <mx:Text text="Europe: $1,912,382"/>
        <mx:Text text="Asia: $3,112,987"/>
      </mx:VBox>
    </mx:VDividedBox>
  </mx:DividedBox>
 </mx:Panel>
</mx:Application>
```

这种类型的容器特别适于显示数据透视图(如财务报告)。

如果需要显示的信息很多，需要分区(图 4-24)，可以使用这种机制来方便用户调整每一部分的版面，从而确保用户更多地查看重要的内容。

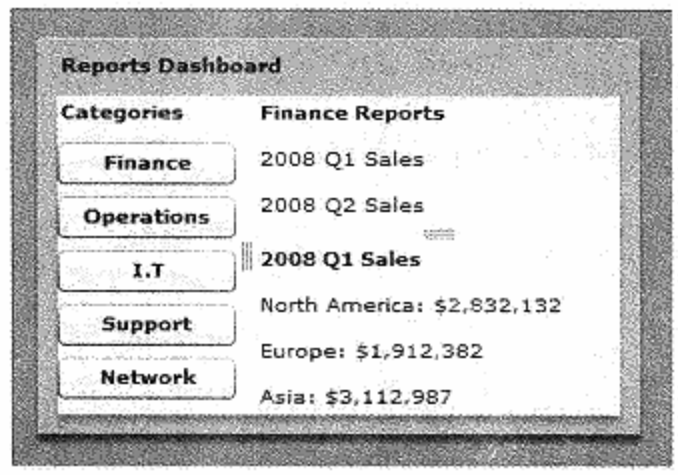


图 4-24　分割盒子允许用户调整应用程序界面分区的大小

使用表单始终是任何应用程序的一个方面，而 Flex 为此提供了一个容器。

4.5.7　Form 容器

Form 容器的用途是方便在应用程序中布局表单。Form 容器与在 HTML 中创建表单类似，但 Flex 中的 Form 容器支持在每个表单输入字段旁边添加标签(label)。此外，与 HTML 要求表单元素必须包含在 form 标签中不同，Flex 中的 Form 容器纯粹用作布局目的，不一定包含表单条目。

Form 容器包含下列 3 个标签。

- Form：主容器。
- FormHeading：可选组件，用于为表单中相应分区添加标题。
- FormItem：用于将文本与每个表单输入字段关联的组件。

代码清单 4-20 使用这些构建一个捕获用户输入的表单。

代码清单4-20 使用与表单相关的容器

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Form>
    <mx:FormHeading label="Contact Info"/>
    <mx:FormItem label="First Name">
      <mx:TextInput/>
    </mx:FormItem>
    <mx:FormItem label="Last Name">
      <mx:TextInput/>
    </mx:FormItem>
  </mx:Form>
</mx:Application>
```

图 4-25 所示为这个常见于众多应用程序的基本登录窗口。可见，使用表单相关的容器，可以方便地创建字段标签和输入字段。

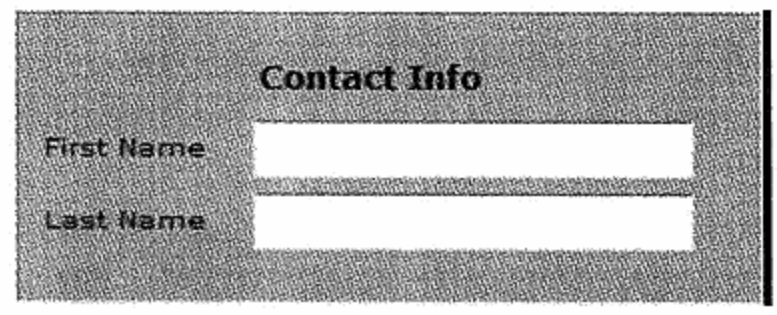


图 4-25 使用表单容器可方便地布局输入组件，并给每个组件添加一个标签

如果读者感觉两栏模式的表单不够灵活，我们再介绍一种 Grid 容器。

4.5.8 Grid 容器

Grid 容器非常类似于 HTML 中的表格，主要表现为顶级的 Grid 标签用于标识网格的开始，GridRow 标签用于插入行，而 GridItem 标签用于向每个网格中输入数据。下面我们使用 Grid 容器构建一个 MP3 播放器的控制面板(代码清单 4-21)。

代码清单4-21 使用Grid容器

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Grid>
    <mx:GridRow>                    ← 创建第一行
      <mx:GridItem>
        <mx:Button label="Rewind"/>
      </mx:GridItem>
      <mx:GridItem>
        <mx:Button label="Play"/>
      </mx:GridItem>
      <mx:GridItem>
        <mx:Button label="Forward"/>
      </mx:GridItem>
    </mx:GridRow>
    <mx:GridRow>                    ← 创建第二行
```

```
    <mx:GridItem colSpan="3">
      <mx:Button label="STOP" width="100%"/>
    </mx:GridItem>
  </mx:GridRow>
 </mx:Grid>
</mx:Application>
```

如果想让一行跨过多列(图 4-26)，可以通过在 GridItem 中指定 rowSpan 属性实现。

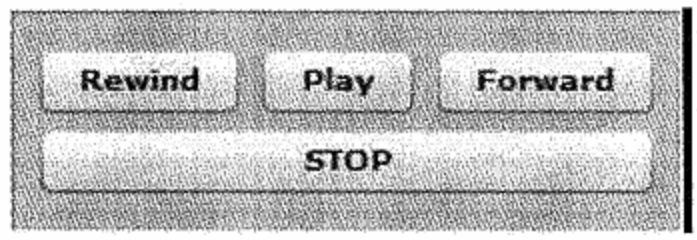


图 4-26　控制面板演示了 Grid 容器的布局能力，结果类似 HTML 中的表格

在熟悉 Grid 容器后，接下来介绍另一种支持类似于表格的布局方法的容器——Tile 容器。

4.5.9　Tile 容器

Tile 容器除了不允许跨行跨列之外，其他特性都与 Grid 相似。Tile 容器在单元格，或者说拼贴块(tile)中放置子对象，拼贴块就像我们在地上铺设瓷砖。Tile 容器最大的优点是，在你要显示数量不定的条目(例如一组缩略图)时，可以根据需要创建拼贴块。

注意：

在根据数据生成拼贴块的情况下，Tile 容器能随着数据变化而创建或删除拼贴块。

与其他容器类似，Tile 容器也支持 direction 属性；默认值为 horizontal(水平)，也可明确指定为 vertical(垂直)。这个属性的目的是告诉 Tile 容器沿指定方向生成新拼贴块，直至空间不足为止；空间不足时，新拼贴块会生成在下一行或下一列中。

下面我们看一个使用 Tile 容器显示一系列气象图标的示例(代码清单 4-22)。

代码清单4-22　使用Tile容器自动布局气象图标

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
 <mx:Tile direction="horizontal" >
  <mx:Image source="heavyrain.png"/>
  <mx:Image source="partlycloudy.png"/>
  <mx:Image source="sunny.png"/>
  <mx:Image source="thunder.png"/>
  <mx:Image source="cloudy.png"/>
  <mx:Image source="drizzle.png"/>
  <mx:Image source="snow.png"/>
  <mx:Image source="clearnight.png"/>
 </mx:Tile>
</mx:Application>
```

运行这个应用程序得到图 4-27 所示的结果。其中，气象图标以网格形式显示出来，非常整洁。

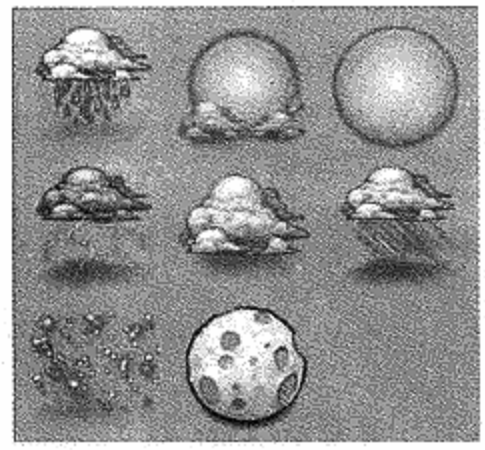


图 4-27 Tile 容器以拼贴模式排列其子对象

在需要通过细目(catalog)形式显示不定数目的项时，Tile 容器非常合适。另外一种手段是使用 Repeater 组件。

4.6 使用 Repeater 组件实现动态布局

显示已知数量的项比较容易，如果项数不定该怎么办呢？此时，可以考虑使用 Repeater。Repeater 是一种不可见的组件，它能够迭代遍历一组信息，在迭代同时动态创建组件。此类功能一个最常见的用法是动态表单或与细目类似的界面等。在此类情况下，往往涉及查询数据库并取得一组与上下文关联的信息，然后基于这些信息生成 UI。

4.6.1 dataProvider 简介

dataProvider 表示一种数据源，是指向在哪里可以检索到信息来源的一个引用(类似于指针)。第 8 章将深入介绍 dataProvider，鉴于 Repeater 的应用需要 dataProvider 支持，我们在此简单了解它，以便在随后的示例中使用。

DataProvider 代表的数据类型大致相当于一个数组。虽然 Flex 中有数组这个类型，但它使用 dataProvider 中的集合。简单地说，集合是增强的数组，它比数组包含更多内置功能，如排序和筛选等。

提示：

ColdFusion 程序员可以把集合想象成类似于结构数组。Java 程序员可以将集合看成类似于 Hashmap 的 ArrayList。

即使为 dataProvider 指定一个纯数组也完全没有问题，因为 Flex 会在后台把这个数组包装成集合。代码清单 4-23 使用前面的拼贴块例子，通过 ArrayCollection 集合类型，基于

一系列气象图标创建集合。

代码清单4-23　ArrayCollection变量中保存了图像文件名

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:local="*">
<mx:Script>
 <![CDATA[
   import mx.collections.ArrayCollection;
   [Bindable]
   public var images:ArrayCollection =
      new ArrayCollection(
            [
             {source:"heavyrain.png"},
             {source:"partlycloudy.png"},
             {source:"sunny.png"},
             {source:"thunder.png"},
             {source:"cloudy.png"},
             {source:"drizzle.png"},
             {source:"snow.png"},
             {source:"clearnight.png"}
            ]
           );
]]>
</mx:Script>
</mx:Application>
```

在变量中配置图像

以下是对代码清单 4-23 中的配置的说明。

- 首先导入集合库，以便使用它。
- 创建 ArrayCollection(一种集合类型)的一个实例。
- 代码[Bindable]在发生变化时通知与变量绑定的任何对象——当发生变化时，变量会提示绑定它的任何对象做出适当响应，例如更新显示。
- 通过简写的数组表示法创建一个 ArrayCollection。
- 方括号[]表示数组。
- 花括号{}表示对象。

结果传入的是一个对象数组。在真实的项目中，可能会查询数据库来获得相应的信息，而 ArrayCollection 是查询结果。

在解决了数据来源的问题之后，接下来介绍用于连接 Repeater 的属性和事件。

4.6.2　Repeater 的属性和事件

为了有效地使用 Repeater，应该知道它有哪些属性以及如何应用这些属性。表 4-6 列出了相应的属性，其中最重要的是 dataProvider，用于引用一个数据源。

表 4-6　Repeater 的属性

属　性	类　型	说　明
id	String	Repeater 的标识符
dataProvider	Object	对要迭代数据的引用

(续表)

属 性	类 型	说 明
startingIndex	Number	开始循环时的索引。默认值为 0
count	Number	设置要执行的循环数。默认值为 dataProvider 包含的项数
currentIndex	Number	存储循环中的当前位置(从 0 开始)，你可以使用此属性来检查当前在循环中的位置。只读
currentItem	Object	对 currentIndex 所指对象的引用
recycleChildren	Boolean	一个性能设置，值可以是 true(默认值)或 false，在允许时允许 Flex 重用子组件

表 4-7 所示为可以通过 Repeater 执行代码的一组事件。

表 4-7 基于 Repeater 的事件

事 件	说 明
repeat	在每次循环迭代后发生
repeatStart	在 Repeater 开始循环前发生
repeatEnd	在最后的迭代完成后发生

如果要使用事件，需要访问 currentIndex 或 currentItem 等属性，以便了解这些属性的状态。本章后面会展示一个相关的示例。

4.6.3 创建 Repeater

要创建 Repeater，下一步需要使用<mx:Repeater>标签并指定要使用的 dataProvider。代码清单 4-24 显示相关代码。

代码清单4-24 ArrayCollection为Repeater提供dataProvider

```
<mx:Tile direction="horizontal" >
  <mx:Repeater id="myRepeater" dataProvider="{images}">

  </mx:Repeater>
</mx:Tile>
```

因为我们想通过 Repeater 循环遍历 ArrayCollection，从而在 Tile 中创建子对象，所以把 Repeater 放在 Tile 内部。添加前面介绍的 Repeater 属性。最后添加想在每次迭代中创建的组件——在本例中，是 Image 组件。通过 Flex 的绑定机制，可以从 currentItem 属性中取得任何时间点的当前图像。

```
<mx:Image source="{myRepeater.currentItem.source}"/>
```

代码清单 4-25 把这些代码放到一起。

代码清单4-25　在循环中创建Image组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:local="*">
  <mx:Script>
    <![CDATA[
      import mx.collections.ArrayCollection;
      [Bindable]
      public var images:ArrayCollection =
          new ArrayCollection([
                {source:"heavyrain.png"},{source:"partlycloudy.png"},
                {source:"sunny.png"},{source:"thunder.png"},
                {source:"cloudy.png"},{source:"drizzle.png"},
                {source:"snow.png"},{source:"clearnight.png"}]);
    ]]>
  </mx:Script>
  <mx:Tile direction="horizontal" >
    <mx:Repeater id="myRepeater" dataProvider="{images}">
      <mx:Image source="{myRepeater.currentItem.source}"/>
    </mx:Repeater>
  </mx:Tile>
</mx:Application>
```

保存所有文件名

创建图像

图 4-28 所示的结果与图 4-27 完全一样；不同的是，可以随时为 ArrayCollection 添加项或从中删除项，但不必因此修改处理输出的代码。

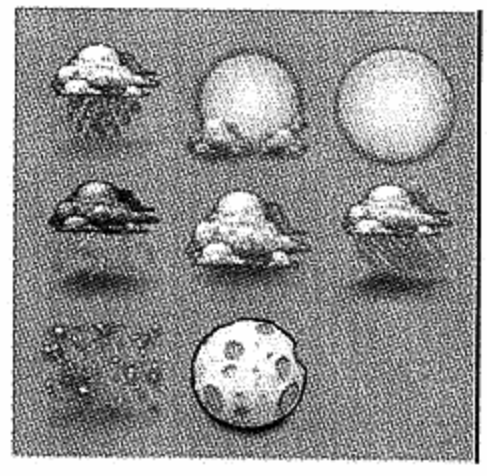


图 4-28　通过 Repeater 根据 ArrayCollection 变量动态生成 Image 组件

可以把 Repeater 想象为一个迭代遍历数组的循环语句。迭代过程中的当前数组项可以通过 Repeater 的 currentItem 和 currentIndex 属性访问。

Repeater 的主要目的是生成显示效果。但是，在 Flex 中还可以利用 Repeater 事件响应用户操作。

4.6.4　使用 Repeater 事件

通过指定要执行的 ActionScript 代码，可以方便地关联 Repeater 事件。接下来，将在 Repeater 中指定几个事件处理程序，供在发生相应的事件时调用。

```
<mx:Repeater id="myRepeater" dataProvider="{images}"
          repeatStart="handleRepeatStart()"
          repeat="handleRepeat()"
          repeatEnd="handleRepeatEnd()">
```

代码清单 4-26 所示为以上几个事件处理程序的代码。

代码清单4-26 用于处理Repeater事件的函数集合

```
public function handleRepeatStart():void
{
 mx.controls.Alert.show("Repeater starting!");
}

public function handleRepeat():void
{
  trace("Currently on index:" + myRepeater.currentIndex);
  trace("Image is:" + myRepeater.currentItem.source);
}

public function handleRepeatEnd():void
{
 mx.controls.Alert.show("Repeater finished!");
}
```

在运行这些脚本的过程中，会看到响应开始和结束事件的两个弹出式警告框，而输出日志中也将针对每幅图像生成一条记录(代码清单 4-27)。

代码清单4-27 针对代码清单4-26的Repeater事件的输出

```
Currently on index:0
Image is:heavyrain.png
Currently on index:1
Image is:partlycloudy.png
Currently on index:2
Image is:sunny.png
Currently on index:3
Image is:thunder.png
Currently on index:4
Image is:cloudy.png
Currently on index:5
Image is:drizzle.png
Currently on index:6
Image is:snow.png
Currently on index:7
Image is:clearnight.png
```

注意：

要了解与配置日志有关的信息，参见第 23 章。

可见，Repeater 是基于一组变化的数据生成组件的便捷方式。

4.7 小结

Flex 中的用户界面以布局为基础。在对界面组件进行布局时，可以选择比较简单的绝对布局模式——需要手工控制每个组件的坐标；或者选择绝对布局模式下的几种布局方式，

例如基于约束的布局方式要求开发人员为元素指定相对于窗口自身(边界或中心点)的相对位置。

为了适应用户对浏览器窗口大小的调整，可以使用自动布局模式让 Flex 帮助我们计算组件的位置。自动布局不可能适应一切需要，但也绝非无足轻重；而使用容器可以混合和搭配使用自动及绝对布局。

Flex 内置了很多容器，每种容器都具有布局和定位 UI 组件的特殊用途。

在最初组件并不存在的情况下，如果需要基于一组数据动态生成组件，可以使用 Repeater。

Flex 中的 Form 容器仅仅是一个标准容器。第 5 章将深入探讨实际创建表单和捕获用户输入的方式。

第5章 显示表单与捕获用户输入

本章内容：

- 在 Flex 中创建表单
- 使用输入组件(控件)
- 捕获用户输入

本章我们将把已经掌握的布局和 ActionScript 知识，运用于任何应用程序中都会有的基础性操作——捕获用户输入。

在第 4 章曾经提到，虽然 Flex 提供了 Form 组件，但并非一定要使用它；你会发现，将其用作布局工具时效果最好。在 Flex 开发中，可以使用控件组件(一般简称为控件)来显示信息并接收用户输入。与控件如影随形的是事件和事件处理程序，用于识别和响应用户操作(如单击鼠标)。

在事件处理函数运行时，它们会从指定的来源中取得数据；不存在像 HTML 中那样包含所有输入的 Form 标签。

注意：

跟 HTML 中的表单概念说再见吧。在 Flex 中，可选的<Form>标签的目的仅限于布局我们称之为控件的 UI 组件。

还记得吗，我们可以使用的控件不止 Flex 提供的那些。与 HTML 中的控件被局限在 HTML 规范和浏览器实现的范围内不同，Flex 鼓励开发人员扩展现有控件并增加更丰富的功能，甚至从头开始创建全新的 UI 控件。

本书的一种讲解方式是向读者展示达到同一目标的多种手段。读者看到的每个示例都将有所不同，尤其是后一个示例可能会借用前一个示例的思想，但又在前一个示例基础上经过部分修改展示新的实现手段。新的实现手段不一定更好，也许只是另一种思路，但最重要的是能够向读者渗透 ActionScript 的思维模式。

理解 ActionScript 核心思想的一个重点是理解重要的 id 属性。MXML 组件的 id 属性是 ActionScript 代码用来访问组件所包含值的重要途径。

5.1 id 属性

我们在第 2 章已经接触过 id 属性，但有必要在这里回顾并扩展一下对 id 的认识。任何组件都可以包含 id 属性，而且也可以像访问其他任何变量一样访问该属性。它是一种为组件实例指定唯一名称的机制，因此，通过这个唯一标识符可以明确引用相应组件。

从角色和用途来看，HTML 的 Form 标签也有类似的 id 属性。但与 HTML 不同的是，MXML 不要求使用 JavaScript 的 getElementByID()之类的函数访问 id。

谈到构建表单，从目标组件中取得值的关键在于 id 属性。

5.2 Text 控件

任何应用程序 UI 的基本用途都是显示文本。Flex 提供了不同的组件供捕获和显示文本信息。表 5-1 显示这些控件及其说明。

表 5-1 用于捕获和显示文本的文本控件

控　件	说　明
Text	显示文本信息的一个简单控件。其作用是使文本像是嵌入在应用程序中，而不像浮动于容器中的其他视觉元素。这意味着它没有滚动条(其他组件在空间不够用时会显示滚动条)、没有边框，而且背景是透明的(默认)。Text 控件中保存的文本数据会自动换行，而且支持 HTML 文本
Label	与 Text 相同，只不过它包含的文本数据不会自动换行。这个控件常用于显示应用程序某一部分的标题，或输入字段的标签
TextInput	单行文本输入字段。它与 HTML 中的<input type="text">标签相似
TextArea	多行文本输入字段。它与 HTML 中的<textarea>标签相似
RichTextEditor	顾名思义，这是一个可供用户设置文本格式的可靠编辑器。这个控件的参数包括颜色、字体和文本大小。HTML 没有与 RichTextEditor 类似的内置功能

代码清单 5-1 在同一个应用程序中展示这些 Text 控件，也构成一个表单。这个表单使用 Panel 组件自动布局所有控件。

代码清单5-1　此代码使用所有基于文本的控件组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
 <mx:Script>
  public function showMsg(msg:String):void
  {
   mx.controls.Alert.show(msg);
  }
 </mx:Script>

 <mx:Panel title="Profile" width="400" height="400">

 <mx:Label text="Enter your name" fontWeight="bold" />

<mx:TextInput id="yourName"
            width="250" valueCommit="showMsg(yourName.text)"/>

<mx:Text text="Profile Summary"/>

<mx:TextArea id="aboutYou" textAlign="center" width="250"
           valueCommit="showMsg(aboutYou.text)"/>

<mx:Label text="Enter your profile" fontWeight="bold" color="#ff0000"/>
<mx:RichTextEditor id="rte" height="150"
               valueCommit="showMsg(rte.text)"/>

 </mx:Panel>

</mx:Application>
```

显示包含传入消息的弹出式窗口

把组件的文本值传给showMsg函数

编译并运行这个应用程序可以看到与图 5-1 所示类似的结果。

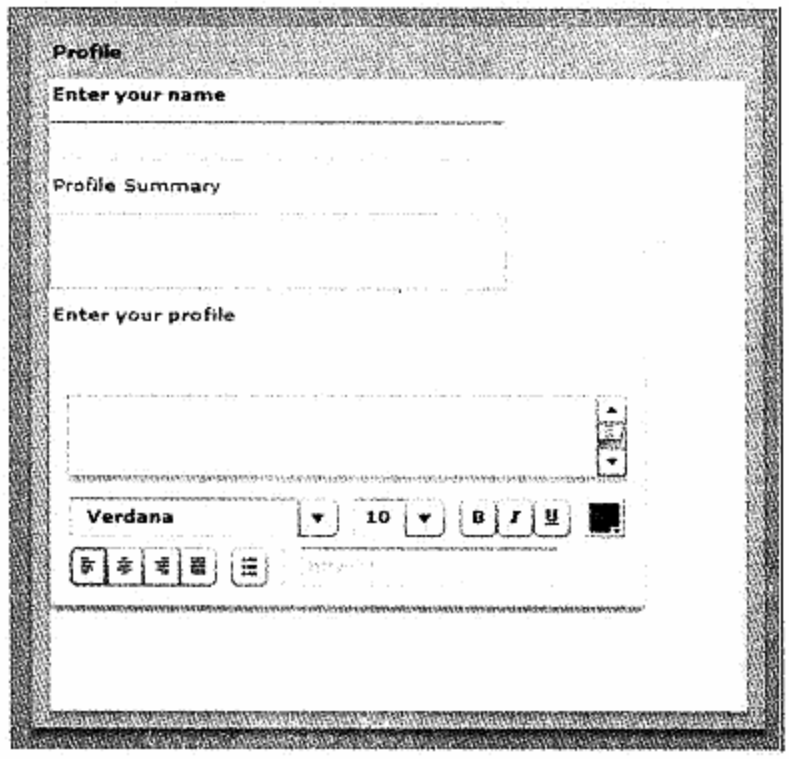


图 5-1　使用 Flex 中基于文本的组件构建的表单

下面我们就来看一看创建的表单及其工作原理。Label 组件用于在每个输入控件上方显示描述性信息。这里为展示 Label 的工作方式，修改了它的一些样式属性。当用户输入值并提交这个表单时(例如，按回车键、Tab 键，或者单击诸如提交按钮的元素)，Flex 会接收到相应事件按照指令并运行 showMsg()函数。

通过这些文本字段可以捕获各种数据。甚至有读者使用它们捕获日期值。对于有 HTML 背景的读者，这很可能是一种自然反应，毕竟 HTML 中没有内置的日期 Form 输入功能。实际上，要想捕获日期值，在 Flex 中最好使用 Date 控件。

5.3 Date 控件

在使用 HTML 时，常用的一个捕获日期输入的技巧是使用表单的<input type="text">标签，在标签后面直接显示一个日历图像。被调用时，日历会出现一个弹出式窗口中。用户从日历中选择日期后，会通过一些 JavaScript 代码把选择的值复制到输入字段中。

但是，Flex 中有一种捕获日期值的更简便方式，这种方式提供两个与日期相关的字段(表 5-2)。

表 5-2 基于日期的控件

控　件	说　明
DateChooser	显示一个与大多数 HTML 日历选择小工具类似的的小日历。值得一提的是，可以控制这个日历显示的日期范围，从而限制可选的日期。这个功能适合屏蔽某些日期，如节假日
DateField	这个输入控件类似于 HTML 中使用的<input type="text">标签加日历图标。用户可以在其中输入日期，或者用户单击图标则会显示 DateChooser 控件

代码清单 5-2 显示如何使用这两个组件。

代码清单5-2 使用Flex提供的两个基于日期的组件捕获日期信息

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    public function showMsg(msg:String):void
    {
    mx.controls.Alert.show(msg);
    }
  </mx:Script>
  <mx:Panel title="Profile" width="400" height="400">
  <mx:DateField text="12/05/2008" id="thisDateField"
            change="showMsg(thisDateField.selectedDate.toString())"/>
<mx:DateChooser maxYear="2010" minYear="2006"
 selectedDate="{new Date(2008,10, 15)}"
               id="thisDateChooser"
          change="showMsg(thisDateChooser.selectedDate.toString())"/>
  </mx:Panel>
</mx:Application>
```

设置默认日期

访问这个组件的实例

设置对年份的限制

设置默认选择的日期

触发对showMsg()的调用

这个示例通过 id 属性的值访问相应组件的值。这样做的目的纯粹是为了向读者展示 id 属性的用途，本章后面将介绍另一种具有相同作用的方法。

有读者可能已经注意到，组件的事件是 change 而非 valueCommit。这两个特定事件的区别在于，valueCommit 会在用户执行某种指示已经设置好值的操作(如按回车键)时被触发。

而 change 响应的则是值的变化，因此用户每敲击一下键盘，都会发生 change 事件。

编译运行这个应用程序会看到如图 5-2 所示的结果，图中有两个基于日期的组件。

图 5-2　Flex 中的两个基于日期的组件显示在一起

注意：

有读者可能会问，为什么不默认地使用简单的文本表示 DateChooser 的日期呢？

查看 Flex 的 API 参考就会发现，selectedDate 的值必须是一个 Date 对象——这个示例以行内方式动态创建了一个。DateField 控件也支持这个属性。

和使用基于日期的控件能够比使用文本输入框更有效地获得日期值一样，Flex 还提供了用于捕获数字值的控件。

5.4　Numeric 控件

Numeric 控件在说明 Flex 胜过 HTML 方面非常具有典型性。在传统 Web 开发领域中，根本不存在与 Numeric 控件类似的功能。

表 5-3 所示为可以用来捕获用户输入数字值的控件。显然，可以使用这些控件执行较简单的任务，如获得用户的年龄；但是，也可以使用它们来实时缩放一幅图像或筛选一组数据。

表 5-3　用于捕获数字值的控件

控　件	说　明
NumericStepper	用户可以方便地递增或递减数字值的控件。可以指定最大值、最小值及每次递增/递减的步长值
HSlider	用户可以水平拖动滑块来选择数字值。可以指定最大值、最小值、snapInterval 增量(滑动条上滑块的对齐位置)及可视增量(称为刻度)。另外，如果想让用户选择多个值(如一个范围)，也可以设置多个滑块
VSlider	除了滑动条的方向是垂直的以外，其余跟 HSlider 相同

下面我们通过代码清单 5-3 展示一下这些控件的用法，并捕获一些数值。

代码清单5-3 使用Flex的Numeric输入控件捕获值

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    public function showMsg(msg:String):void
    {
      mx.controls.Alert.show(msg);
    }
  </mx:Script>

  <mx:Panel title="Profile" width="360" height="240" layout="horizontal">
    <mx:VBox>
      <mx:Text fontWeight="bold" text="How many kids do you have?"/>

      <mx:NumericStepper id="kids"
                 minimum="0" maximum="15"
                 stepSize="1"
                 change="showMsg(kids.value.toString())"/>

        <mx:Text fontWeight="bold" text="How long is your commute (mins)?"/>

      <mx:HSlider id="commuteTimeRange" minimum="0" maximum="180"
          snapInterval="5"      ❶
          tickInterval="15"     ❷
          labels="[0 mins,180 mins]"   ❸
          thumbCount="2"
          change="showMsg(commuteTimeRange.values.toString())" />

</mx:VBox>
<mx:VBox>
      <mx:Text fontWeight="bold" text="How tall are you (cm)?"/>
      <mx:VSlider id="yourHeight" minimum="0" maximum="300"
              tickInterval="50" snapInterval="1"     ❹
              labels="[0,50,100,150,200,250,300]"    ❺
                 change="showMsg(yourHeight.values.toString())"/>
    </mx:VBox>
  </mx:Panel>
</mx:Application>
```

设置年龄介于0～15岁之间

增量/减量等于1

两个滑块

在水平滑动条中，❶表示用户每次移动滑块时以 5 为单位步进，❷表示以 15 为刻度线的显示间隔。❸指定显示在两端的两个标签(每端一个)。在垂直滑动条中，❹表示刻度线间隔为 50(比水平滑动条的更大)。最后，平均分布着显示了 7 个标签❺。

在编译和运行这个应用程序后，应该得到图 5-3 所示的结果。这些用户输入控件在基于 HTML 的 Web 应用程序中不可能看到，甚至在桌面应用程序中也是不多见的。

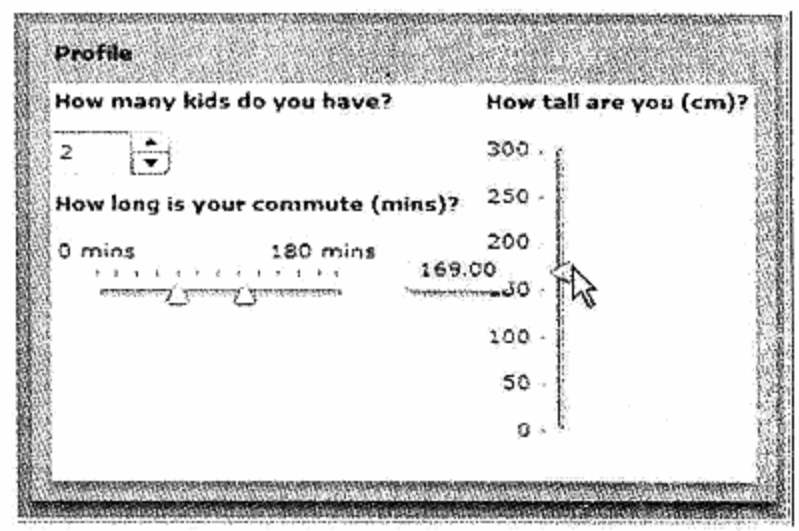


图 5-3 Flex 的基于数值的组件允许以独特的方式捕获值

许多有 HTML 背景的开发人员由于过去不曾接触到这些控件，所以经常不能充分利用它们。这种情形很令人遗憾，毕竟使用基于数字值的组件要比简单地捕获数值更能发挥开发人员的创意。你可以让最终用户实时调整数字值。再举一个例子，在设计一个综合报表时，可以在用户来回拖

动滑块实时改变显示的数据——有效地扩大或缩小筛选数据的口径。

5.5 Button 组件

用整节篇幅介绍按钮？对于 Flex 而言，肯定的回答掷地有声，当然！因为 Flex 已经把 HTML 中的普通按钮远远地抛在了后面，而且增加了大量更加有趣的选项(见表 5-4)。按钮是 GUI 中的主打元素，但有时候也会变得有点儿令人讨厌。Flex 提供的多种按钮组件在某种程度上解决了这个问题。

表 5-4　Flex 提供了多种按钮组件

控　件	说　明
Button	用于接收鼠标单击交互的标准、通用按钮。Flex 内置了支持功能，允许你将图像方便地添加到按钮中。而在 HTML 中，只能使用伪文本加图像来创建这样的按钮，因此文本是无法修改的。在 Flex 中，即使图像按钮中的文本也是动态的
ButtonBar	可以基于数组动态生成一组按钮。虽然你自己也可以动态生成多个 Button，但与使用 ButtonBar 相比，一方面后者更容易实现，另一方面 ButtonBar 借用了菜单栏的外观，在把按钮分开的同时还能让最外侧的按钮呈现圆角
LinkButton	Flex 中的 HTML 链接。如果读者查看这个对象的类文档(在 Flex API 参考中)，会发现它是 Button 的后代，只不过应用了一些样式并扩展了一些行为，使它像是没有背景和边框的链接而已
LinkBar	与 Button 和 ButtonBar 的关系类似，LinkBar 可以用来方便地创建一组 LinkButton
ToggleButtonBar	与 ButtonBar 很相似，只不过其中的按钮能够保持被选状态，即单击一个按钮会导致该按钮处于被选状态(并突出显示)，单击其他按钮后，该按钮才会恢复常态。可以把其想象成内部带标签的单选按钮
PopUpButton	组合了两个按钮功能的“双重按钮”。按钮左侧是一个常规的 Button，右侧在被单击时会调用另一个 UI 对象(即任何派生自 UIComponent 的对象)。这个控件最适合创建多选 Button，它看上去将 Button 与 ComboBox 合二为一
PopUpMenuButton	PopUpButton 的子类，非常适合创建下拉菜单

代码清单 5-4 所示为如何调用所有这些按钮选项。

代码清单5-4　使用Flex中一组基于Button的组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
 <mx:Script>
  <![CDATA[
   public function showMsg(msg:String):void
   {
    mx.controls.Alert.show('You just clicked on ' + msg);
   }
  ]]>
 </mx:Script>
 <mx:Panel title="Profile" width="360" height="240" layout="vertical">

  <mx:VBox>

   <mx:Button id="thisBtn" label="Button" click="showMsg('button')"/>

   <mx:LinkButton id="thisLinkBtn" label="LinkButton"
              click="showMsg('linkbutton')"/>
```

```
    </mx:VBox>

    <mx:VBox>

      <mx:ButtonBar id="thisBtnBar"
                dataProvider="{['One','Two','Three']}"
                itemClick="showMsg(event.label)" />

      <mx:LinkBar id="thisLinkBar"
              dataProvider="{['One','Two','Three']}"
              itemClick="showMsg(event.label)" />

      <mx:ToggleButtonBar id="thisToggleBar"
                    dataProvider="{['One','Two','Three']}"
                    itemClick="showMsg(event.label)"/>

    </mx:VBox>
  </mx:Panel>
</mx:Appliclation>
```

DataProvider填充组件的显示内容

Flex 作为一门 OOP 语言，倾向于以自身的某些组件为基础创建各种派生组件。本节介绍的 Button 家族极好地印证了这一策略。

5.5.1 带 Bar 的 Button 控件

ButtonBar、LinkBar、ToggleButtonBar 这几个 Button 控件就像亲兄弟一样，抛开外观上的差异，它们的功能实际上是完全相同的(见图 5-4)。更重要的是，这几个带 Bar 的控件都使用了 dataProvider 属性生成显示内容，该属性的值是类似于数组的数据(参见第 4 章关于 Repeater 及其 dataProvider 属性的讨论)。

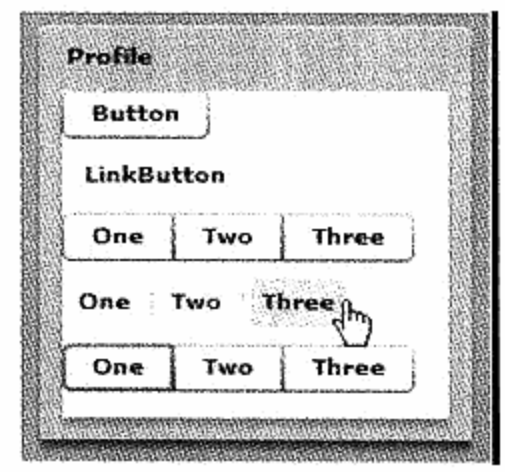


图 5-4 Flex 提供了很多基于 Button 的组件

跟其他应用程序一样，Flex 也提供数组类型，但另一个相似的类型 ArrayCollection 则显得更强大一些。在上面的例子中，我们使用 ActionScript 代码动态创建了一个硬编码的数组。但更好的方法是创建一个 ArrayCollection 变量，然后将该变量赋值给 dataProvideras 属性。如下：

```
<mx:Script>
  import mx.collections.ArrayCollection;
  [Bindable]
  public var arData:ArrayCollection = new
                ArrayCollection(['One','Two','Three']);
</mx:Script>
<mx:ButtonBar id="thisBtnBar" dataProvider="{arData}" />
```

创建 ArrayCollection 变量的好处是把该变量包含的逻辑从显示组件中分离出来。这样，就可以读入配置数据并相应地填充变量了。

注意：

click 与 itemClick 有什么区别呢？区别不大。Flex 应用程序是事件驱动的，而事件是由操作触发的。不同的组件响应所有事件种类。click 指的是在一个 Button 组件上单击鼠标按钮(触发)的操作，而 itemClick 指的是单击列表中某个项的操作。

前面的示例涵盖了除 PopUpButton 和 PopUpMenuButton 外的所有按钮。虽然这两个按钮的名称就说明了它们的作用，但是为了让读者更恰当地用好这两个控件，下面对它们进行详细介绍。

5.5.2　PopUpButton 和 PopUpMenuButton

事实上，这两个控件的使用频率并不高。但是，正因为它们适合一些特殊情况，所以才需要更多地加以解释。

代码清单 5-5 所示为 PopUpMenuButton 的用法。

代码清单5-5　使用PopUpMenuButton的示例

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>
    public function showMsg(msg:String):void
    {
      mx.controls.Alert.show('You just clicked on ' + msg);
    }
  </mx:Script>

  <mx:Panel width="100" height="100">
    <mx:PopUpMenuButton id="menuBtn"
      dataProvider="{['One','Two','Three']}"
      click="showMsg('left side')"
      itemClick="showMsg('right side with ' + event.label)"/>
  </mx:Panel>
</mx:Application>
```

注意，这个示例中既使用了 click 事件也使用了 itemClick 事件。在单击右侧的“二级”按钮时(图 5-5)，会出现一个下拉菜单，提示你从中选择一项。选中一项，就会触发 itemClick 事件。但是，如果单击了左侧的“主”按钮，则会相继触发 itemClick 和 click 事件。

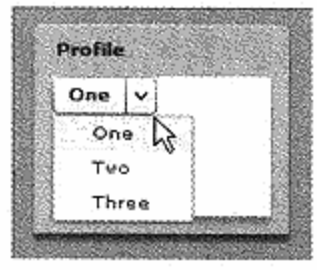


图 5-5　PopUpMenuButton 类似于 Button 和 ComboBox 二合一的产物

区分上述两个事件有助于区分用户交互操作。假设代码清单 5-5 定义的下拉菜单中包含几种信用卡品牌(如 Visa、MasterCard、AmericanExpress)。当用户改变选项时(触发 itemClick 事件)，可以把选择的信用卡品牌保存在一个变量中。但是，如果用户单击的是“主”按钮(触发 click 事件)，则可以让 Flex 提交表单。

下面我们再来看一看 PopUpMenuButton 的父类——PopUpButton。PopUpButton 能够完成与 PopUpMenu- Button 相同的任务，但功能更广泛；当然，为了实现对应的功能，就需要编写更多代码。

PopUpMenuButton 使用了一个名为 Menu 的元素(Flex 中的另一个标准对象)生成下拉菜单。但 PopUp MenuButton 只能显示 Menu。

PopUpButton 能够显示更多元素，但没有指定任何默认元素。一方面这可以使 PopUpButton 更加灵活，另一方面也要求我们必须明确指定 PopUpButton 显示什么。

要通过 PopUpButton 生成与代码清单 5-5 相同的结果，就要使用代码清单 5-6 中的代码。

代码清单5-6 使用PopUpButton生成选项菜单

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
 <mx:Script>
  import mx.events.*;
  import mx.controls.Menu;

  public var menuItems:Object =
  [{label:'One'},{label:'Two'},{label:'Three'}];

  public var thisMenu:Menu = Menu.createMenu(null,menuItems,false);

  public function handleItemClick(event:MenuEvent):void
  {
     menuBtn.label = event.label;
  }
 </mx:Script>l

 <mx:Panel width="100" height="100">
 <mx:PopUpButton id="menuBtn"
  creationComplete="thisMenu.addEventListener('itemClick',handleItemClick)
  ;"
  popUp="{thisMenu}"/>
 </mx:Panel>
</mx:Application>
```

使用类似于数组的菜单值创建变量

填充Menu对象

在菜单项被单击时调用项

设置按钮的文本标签

以下是关于代码清单 5-6 的三点说明。

- 在多数示例中，保存填充信息的变量都是一个数组。在上面这个示例中，我们使用了一个对象。第 3 章曾介绍过，这种情况下的数组和对象没有什么区别。因此，如果把这里的 menuItems 作为数组而不是对象来声明，结果是一样的。
- 这个示例用到了 creationComplete 事件。此事件告知 PopUpButton 在创建完成后调用其他代码。
- 其他代码会告诉 Menu 对象(thisMenu)侦听 itemClick 事件，如果该事件发生，则调用事件处理程序(handleItemClick)处理该事件。

看起来是麻烦一些，既然如此为什么还要这样做呢？如图 5-6 所示，其中的按钮和下

拉菜单与使用前面的 PopUpMenuButton 产生的效果几乎相同。这样做值得吗？当然。与使用 PopUpMenuButton 不同的是，“主”按钮的默认标签并不是菜单中的第一项。

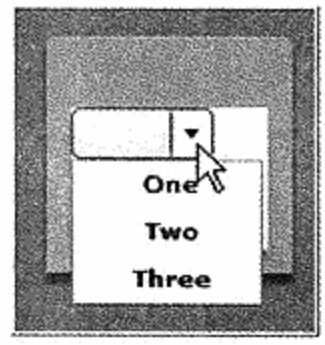


图 5-6　虽然 PopUpButton 与 PopUpMenuButton 的行为类似，但更具灵活性；同时需要的工作量也更大

PopUpButton 与 PopUpMenuButton 的主要区别，就是使用 PopUpButton 并不局限于显示菜单，而是可以显示任何能够与用户交互的视觉元素。如果应用程序中只需要下拉菜单，选择 PopUpMenuButton 没错。如果需要对展示给用户的选项拥有更多控制，就要发挥 PopUpButton 的灵活性。

除按钮之外，选单(picklist)也是一个(无论桌面还是在 Web 应用程序中)常见的用户输入控件。选单可以提供一组选项，供用户从中选择一个或多个项。

5.6　选单控件

选单控件用于呈现其他类型的数据——显示一系列供选择的项。表 5-5 所示为选单控件。

表 5-5　Flex 的选单控件

控　件	说　明
CheckBox	通用的复选框控件(在行为上与 HTML 中的复选框相同)。该控件是 Button 的子类，只不过专门设计为可以被勾选和取消勾选。CheckBox 最常用于提供一组选项，用户可以从中勾选(选择)一个或多个选项
RadioButton	与 CheckBox 类似，但在分组后的一组选项中，用户每次只能选中一个。RadioButton 同样是 Button 的子类
ComboBox	该下拉菜单控件显示当前所选项，但当被单击时会显示一个选项列表以便用户选择。ComboBox 与 HTML 中<select>框大致相同，主要区别是允许用户修改选中的文本。虽然表面上看它基于 Button 构建，但它不是 Button 的子类
ColorPicker	ComboBox 的变体。ColorPicker 会下拉显示一个颜色菜单，让用户能够像使用调色板一样从中选择颜色。通过 ColorPicker 可以让用户自定义桌面背景，或者通过它实现应用程序工作流中的某种功能，例如在汽车厂商网站上提供一个颜色选择器，以供客户实时查看使用不同颜色喷漆的汽车外观

为了了解这几个选单控件，下面我们用它们来组成一个用户简介表单的程序(代码清单5-7)。

代码清单5-7 选单控件的例子

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>

    public function showMsg(msg:String):void
    {
      mx.controls.Alert.show(msg);
    }
  </mx:Script>

  <mx:Panel width="400" height="150" title="Profile">

  <mx:HBox>
    <mx:Label text="Your hobbies:"/>
    <mx:CheckBox id="cbVideoGames" label="Video Games"
          click="showMsg('Video Games is ' + cbVideoGames.selected)"/>
    <mx:CheckBox id="cbFishing" label="Fishing"
          click="showMsg('Fishing is ' + cbFishing.selected)"/>
  </mx:HBox>
  <mx:HBox>
    <mx:Label text="Do you like spam:"/>
    <mx:RadioButtonGroup id="Spam"
                    itemClick="showMsg('User picked ' +
                    event.currentTarget.selectedValue)"/>
<mx:RadioButton id="rbYes" value="Yes" groupName="Spam"
               click="showMsg('Yes')" label="Yes"/>
    <mx:RadioButton id="rbNo" value="No" groupName="Spam"
               click="showMsg('No')" label="No"/>
  </mx:HBox>

  <mx:HBox>
    <mx:Label text="Favorite car maker:"/>
    <mx:ComboBox id="combo"
            change="showMsg('Favorite car is ' +
            event.currentTarget.selectedItem)">
      <mx:Array>
        <mx:String>Ferrari</mx:String>
        <mx:String>Porsche</mx:String>
        <mx:String>Hyundai</mx:String>
      </mx:Array>
    </mx:ComboBox>

    <mx:Label text="With the color of:"/>
    <mx:ColorPicker id="clr"
            change="showMsg('Color ' +
                  event.currentTarget.selectedColor)"/>
   </mx:HBox>

  </mx:Panel>

</mx:Application>
```

❶

❷

显示值的变化

用于填充ComboBox的行内数据

显示颜色

这个示例使用了两个 CheckBox，而且在它们被单击时会显示相应的值❶。紧接着，创建了两个分组的 RadioButton❷，它们会触发两个弹出式窗口。首先会触发 RadioButton 的 click 事件，然后触发 itemClick 事件。RadioButtonGroup 是可选的，一般在需要调用单个与

RadioButton 无关的函数时使用。

编译并运行这个应用程序，将看到与图 5-7 所示类似的结果。其中包含使用选单控件创建的几个问题。

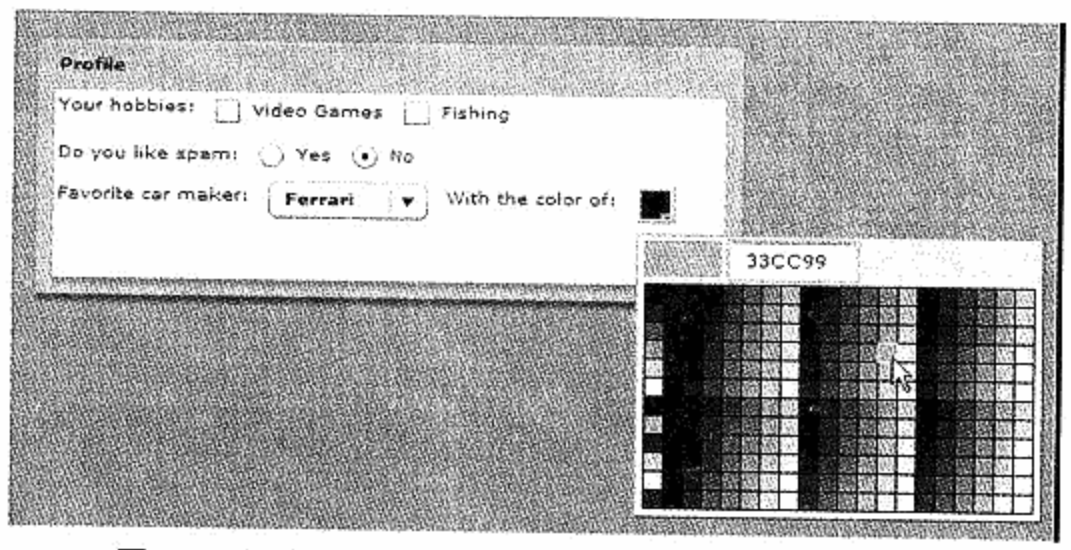


图 5-7　用户通过选单控件可以选择指定范围内的选项

将这些控件与 HTML 中的对等控件相比，它们的行为有什么大的不同；不过，颜色拾取器(color picker)的确是很新颖的。

注意：

Flex 官方文档中没有选单这个术语，这只是我们为此类控件起得通用名。

在介绍完所有用户输入组件(控件)并展示它们的用法示例之后，接下来深入讨论在与用户交互，并访问用户输入的值时应该做出的选择。

5.7　访问控件的值

与使用其他编程语言一样，在 Flex 中完成一个任务也会有多种方式。出于简单明了的考虑，前面的示例使用了最直接的方式，以便你了解后台发生的事情。实际上，对于前面每个示例来说，都有达到同样目的的其他方式(也许还是更好的方式)。

5.7.1　向函数传递值

前面的示例曾多次展示把值传递给函数的技术。虽然这样做是为了保持示例简单明了，但把值传递给独立函数仍不失为一种首选技术，因为处理数据的逻辑不用知道数据来自何处。

这样一来，函数可以被重用，而且即使将来修改实现方式(例如把 RadioButton 换成 CheckBox)，也不会影响函数。分析函数 showMsg 就可以了解到这一点：

```
public function showMsg(msg:String):void
{
  mx.controls.Alert.show(msg);
}
```

当调用函数并传递值时，其实是在响应一个导致事件发生的触发操作。每个事件都有一个相伴的事件对象，事件对象(依靠 currentTarget 属性)包含所有类型的重要信息(如事件的来源)。

因此，下面的代码通过 currentTarget(在此引用的是 RadioButtonGroup)提取出 RadioButtonGroup 的 selectedValue。

```
<mx:RadioButtonGroup
   itemClick="showMsg(event.currentTarget.selectedValue)"/>
```

如果为相应的控件指定一个 id 属性，则可以通过 id 来引用。

```
<mx:RadioButtonGroup id="Spam"
   itemClick="showMsg(Spam.selectedValue.toString())"/>
```

传递简单值在任何语言中都属于最基本的操作，但 Flex 是事件驱动的，因此提供了接触事件本身的机会。

5.7.2 向函数传递事件

相反的方式是让函数负起更大的责任，使函数更多地了解数据的起源。

为此可以把事件对象传递给函数，如代码清单 5-8 所示。

代码清单5-8 在用户单击控件时，向函数传递事件对象

```
<mx:Script>
import mx.events.ItemClickEvent;   ❶ 导入事件类
public function showMsg(anEvent:ItemClickEvent):void
{
 mx.controls.Alert.show(anEvent.currentTarget.selectedValue);
}
</mx:Script>
<mx:RadioButtonGroup id="Spam" itemClick="showMsg(event)"/>
```

可见，这样一来函数变复杂了，但调用函数的控件倒是简单了不少。控件只需传递事件对象；而事件处理函数可以按照需要自主地处理事件。传递事件对象的关键在于，必须知道事件的类型，而且要导入相应的类❶。

可以通过查看 Flex API 参考中组件(或控件)的类信息来确定事件的类型(参见第 2 章)。

提示：

查看 Flex API 类参考(在这种情况下，是显示控件定义)的最快捷方式是单击要查看的控件，然后按 Shift+F2 快捷键。

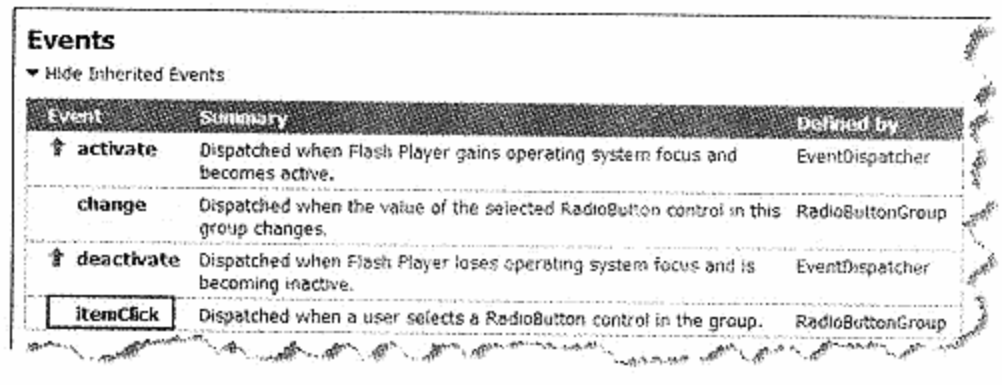
例如，通过下列步骤可以确定代码清单 5-8 中的事件是 itemClickEvent 对象。

(1) 单击代码中的 RadioButtonGroup，然后按 Shift+F2 快捷键。这个组件的 API 参考

会显示出来。

(2) 要查找事件，需要向下滚动到事件部分。

(3) 从列表中选择 itemClick(见图 5-8)。



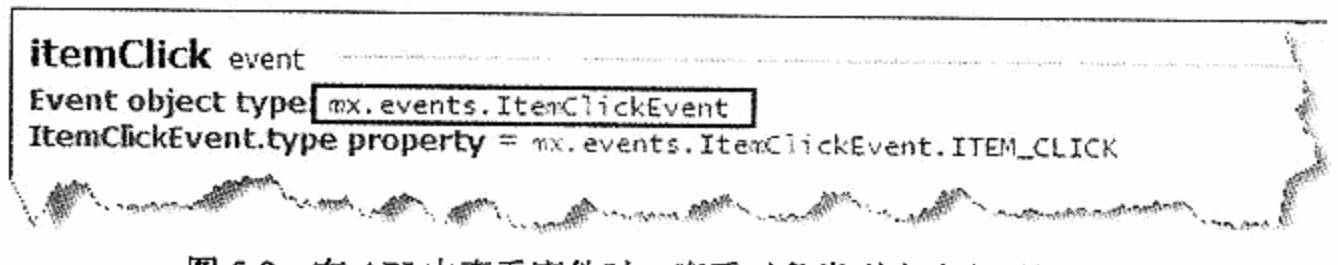
Events

Hide Inherited Events

Event	Summary	Defined by
activate	Dispatched when Flash Player gains operating system focus and becomes active.	EventDispatcher
change	Dispatched when the value of the selected RadioButton control in this group changes.	RadioButtonGroup
deactivate	Dispatched when Flash Player loses operating system focus and is becoming inactive.	EventDispatcher
itemClick	Dispatched when a user selects a RadioButton control in the group.	RadioButtonGroup

图 5-8　要了解有关特定事件的信息(如应导入哪个事件类)，可以在给定组件的 API 参考的事件列表中选择事件

(4) 相应事件的 API 参考就会出现。类名将出现在事件对象类型(event objecttype)区域(见图 5-9)。具体来说，我们要找的就是 mx.events.ItemClickEvent 类。



itemClick event

Event object type: mx.events.ItemClickEvent

ItemClickEvent.type property = mx.events.ItemClickEvent.ITEM_CLICK

图 5-9　在 API 中查看事件时，查看对象类型来确定要导入的类

在采用这种方法时，初期消耗的工作量最大，如果你要采取真正的快捷方法，可以使函数直接访问组件属性从而取得相应的值。

5.7.3　直接访问属性

最后我们介绍耦合程度最高的方式。这种方式要求函数必须非常具体地了解所访问的内容。当然，这就限制了函数的重用能力。最糟糕的是，当函数访问的属性被修改后，函数代码就会失效。

从另一方面看，这是一种非常简便的方法，因为不需要为其传递任何数据(代码清单 5-9)。

代码清单5-9　这个函数直接访问组件的值

```
<mx:Script>
 public function showMsg():void
 {
  mx.controls.Alert.show(Spam.selectedValue.toString());
 }
</mx:Script>
<mx:RadioButtonGroup id="Spam" itemClick="showMsg()"/>
```

在这种情况下，甚至可以在输入控件中响应事件；不过，由于直接访问控件的值，因

此可以在任何时候进行访问(如单击提交按钮时)。

直接访问属性的方式也可以跟前面介绍的方式结合使用。代码清单 5-10 使用一个低耦合度的函数验证用户输入的值，该函数对数据来源一无所知；同时，在用户单击提交按钮时，另一个函数会显式访问特定的 TextInput 实例。

代码清单5-10 使用组合技术访问控件的值

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
      public function checkValue(inputValue:String):void
      {
        if(inputValue.length < 5)
          Alert.show("Are you sure there's not that much new?");
      }
      public function submitClicked():void
      {
        Alert.show("User says:" + whatsnew.text + " is new.");
      }
    ]]>
  </mx:Script>
  <mx:Panel title="Profile" width="400" height="400">
    <mx:Label text="What's new?"/>
    <mx:TextInput valueCommit="checkValue(event.currentTarget.text)"
              id="whatsnew"/>
    <mx:Button label="Submit" click="submitClicked()"/>
  </mx:Panel>
</mx:Application>
```

以传送的参数形式接收控件的值

直接访问控件的值

提示：

注意<mx:Script>块中使用了<![CDATA[...]]>。这是因为此处的 ActionScript 代码使用了小于字符(<)。为保持简单，前面一些示例曾省略了这个声明，但最好是始终在 ActionScript 块中使用这个声明(具体说明参见第 3 章)。

显而易见，这具有了一定程度的灵活性。不过，哪种技术最合适呢？

5.7.4 应该选择哪种方式

纯粹论者可能坚持认为函数应该操作的是控件值，但哪种方式最合适却没有硬性和便捷的规定。就和使用其他任何编程语言一样，在编写代码前对代码重用和维护问题考虑得越周全，长期来看，节约的成本越多。

最后要说的是，没有绝对的错误方式。需要评估多种因素，如程序的规模、预计的生命周期以及其他问题。如果应用程序的目标是成为一个简单的一次性项目，或者是为了验证设计思路，那就不必在可重用性方面大伤脑筋。

5.8 小结

本章介绍了 Flex 内置的一些界面组件(也称为控件)。其中有些控件与 HTML 中对应的控件类似，有些控件的功能则远远超过对应 HTML 的控件。

例如，ColorPicker、Slider 及各种 Button 控件为构建用户界面和表单提供了极大的自由度。

请读者记住，与 HTML 中表单要封装一组表单输入不同，Flex 中的表单组件只用于辅助布局组件。换句话说，表单组件不一定出现在<mx:Form>容器中。

获取用户所选值的方式有很多，可以利用事件实时检索，也可以通过组件的 id 属性访问值。

捕获用户输入是一个问题，但怎么验证用户输入呢？比如说，要构建的应用程序要求用户输入不少于 5 位字符的密码。第 6 章将介绍使用 Flex 验证器来验证密码及其他用户输入的值。

第 6 章 验证用户输入

6

本章内容：

- 验证与执行
- 通过性验证
- 提交值验证
- 脚本式验证及实时验证

处理客户端验证是 Flex 的一个主要长处。从可用性角度讲，有必要在应用程序中尽可能多地进行事先验证，以免用户花了很多时间填写表单，而单击提交按钮后才发现输入不正确。

验证可以保证应用程序不受用户错误输入的干扰。在 HTML 环境中，你使用 JavaScript 处理数据验证。很多读者都可能有过填写表单之后，才发现密码不匹配的经历。

如果能在提交表单之前就完成对输入的验证，不就能节省宝贵的时间吗？Flex 当然可以做到这一点！Flex 为此提供了一种实时、不唐突的验证机制。

本章讨论 Flex 如何处理验证，并了解不同类型的验证器(包括验证电话号码格式及电子邮件地址结构)。最后我们会介绍处理验证任务的不同方式。

我们就从介绍 Flex 与其他应用程序中验证的不同之处开始。

6.1 验证简介

读者有没有在支持Captcha防垃圾机制的博客中发表评论的经历？Captcha有意扭曲文本字符并在背景中混入图形杂点，从而最大限度地阻止计算机识别其内容。只有人类才能区分图形并认出其中的字符(见图 6-1)。

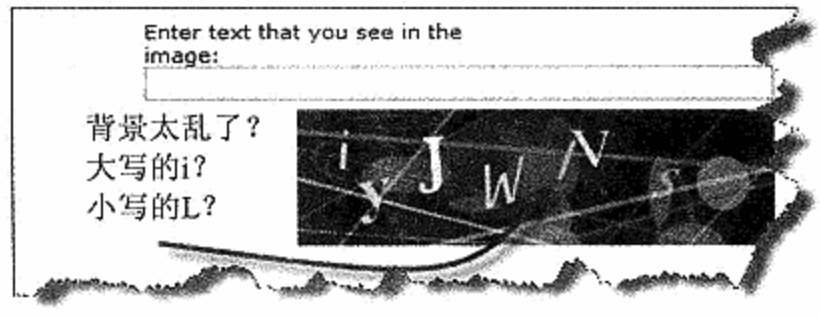


图 6-1 基于 HTML 的输入验证具有局限性，这幅 Captcha 图像会令用户讨厌

在写完评论后，应用程序要求你正确输入嵌在这幅图像中的字母。对此，用户会觉得很无聊，认为这是在重复执行不必要的步骤验证是在浪费时间。

如果打算在用户填写内容时进行验证，那么应该显示即时反馈信息向用户说明填写要求。在基于 HTML 的应用程序中，使用 JavaScript 来提供这种反馈还是比较成功的；缺点是构建验证逻辑及与用户沟通问题的责任基本上是由开发人员一肩挑的。

Flex 通过让内置组件负责处理艰难的工作并提供多种调用手段简化了验证工作。以下列出验证用户输入的几种常见方式。

- **实时验证**：应用程序基于每次键盘敲击或鼠标点击来验证输入的内容是否符合要求或正确无误。
- **提交值验证**：类似实时验证，但不验证每次键盘操作，而是在用户填写完成并提交其输入值时验证(例如按回车键或 Tab 键更改字段)。
- **通过性验证**：通常由单击提交按钮触发，这种验证会一次性检查所有表单输入字段，以确保所有输入值都是有效的。
- **脚本式验证**：使用 ActionScript 动态创建验证器，甚至在多个元素上重用同一个验证器。

在这些方式中，决定何时进行验证的都是开发人员。如果发现不一致情况，Flex 会突出显示问题字段(以不唐突的方式)并提供一个友善的鼠标悬停消息，用以说明问题所在(图 6-2)。

图 6-2 Flex 以友好的方式显示警告消息，由此进行验证

请读者注意，我们这里说的是验证，不是自动执行。验证是在未满足条件的情况下提醒用户和开发人员。而开发人员需要确定如何通过向用户说明怎样做才能满足应用程序要求，来达到验证目的。

接下来，我们看一看 Flex 都提供了哪些内置验证器。

6.2 内置验证器

Flex 提供了很多内置验证器。这些验证器都是 Validator 类的子类，因此它们具有一组公共的属性、方法和事件。

Adobe 的 Flex 团队以扩展 Validator 类的方式创建了内置验证器，以便满足常见验证环境的需要。其中每个验证器都具有与环境相关的特有属性。

表 6-1 所示为 Flex 提供的验证器，并说明每一项的特点。不要忘记，每个内置验证器都继承了它们的父类(Validator)提供的特性。

6.2.1 Validator

Validator 是所有验证器的父类，它的主要用途是作为其他派生验证器类的模板。由于 Validator 只是一个基类，因此它唯一的作用和能力就是检查用户是否为所针对的控件(如表单输入字段)提供了值。

在介绍 Validator 组件的用法之前，先来看一看表 6-1 列出的属性。

表 6-1　Validator 组件的重要属性和函数

属　性	类　型	说　明
enabled	Boolean	true 或 false。设置是否启用验证功能
required	Boolean	true 或 false。设置当前字段对用户而言是否必填
requiredFieldError	String	如果 required 为 true，通过这个属性设置显示给用户的消息。如果未设置这个属性，则显示默认的通用消息
source	Object	想要验证的对象(如 TextInput)
property	String	想要验证的对象属性(如 TextInput 的 text 属性)，判断提供的值是否符合要求
listener	Object	默认为 source 对象。如果 source 对象的属性未通过验证，Flex 会突出显示 source 对象。可以通过设置 listener 属性来修改要突出显示的对象
valid	Function	在验证通过的情况下调用的函数的名称
invalid	Function	在验证失败的情况下调用的函数的名称
trigger	Object	默认为 source 对象。是导致触发的对象的名称(如提交按钮)
triggerEvent	String	导致基于触发器执行验证的事件的名称(例提交按钮的 click 事件)。默认为 valueCommit 事件(通常在用户按 Tab 键或单击表单其他地方从而移动到另一个字段的情况下被触发)

代码清单 6-1 所示为一个使用 Validator 组件的最简单的例子。

代码清单6-1　基本的Validator组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Validator source="{username}" property="text" required="true" />
   <mx:Text text="Enter your username:"/>
   <mx:TextInput id="username" />
</mx:Application>
```

侦听valueCommit事件

编译运行这个示例，将看到与图 6-3 所示类似的应用程序：用鼠标单击要求输入用户名的字段。

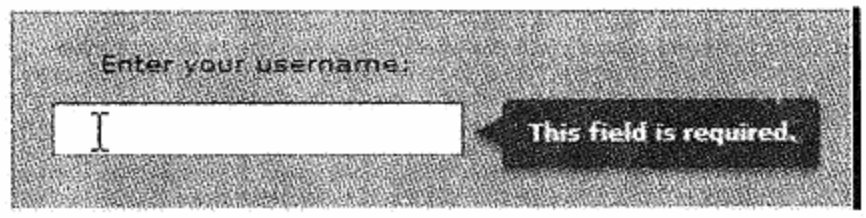


图 6-3 验证失败时，Validator 组件会突出显示字段，并显示一条消息

注意，当鼠标离开字段时，焦点随之改变。与此同时，字段边框变为红色，旁边也会显示一条弹出式消息，提示用户这个字段是必填的。

验证器的整个工作过程如下。

- 焦点改变——即焦点从一个 UI 元素转移到另一个 UI 元素(可能是鼠标单击、按 Tab 键或回车键所致)，这时一般会导致输入控件提交它的值。
- 提交文本字段中的内容——这里文本字段为空，因此值也为空。
- 由于 triggerEvent 默认为 valueCommit。验证器被设置为侦听 text 属性的 valueCommit 事件。
- 当 TextInput 对象的 text 属性为空导致验证失败后，将字段突出显示为红色。

Flex 中所有具体的验证器类都继承了父类 Validator，因此所有验证器都遵循上述过程。不同的是，每个验证器经过配置可以分别基于不同的条件进行验证。

例如，StringValidator 可以计算用户输入的字符数，而 Validator 只能执行简单的 Boolean(true/false)检查。

6.2.2 StringValidator

StringValidator 也是一个通用的验证器，它具有父类的所有特性，但增加了检查 String 是否过长或过短的能力。表 6-2 所示为 StringValidator 在 Validator 组件基础上新增的属性。

表 6-2 StringValidator 的新增属性

属 性	类 型	说 明
minLength	Number	必填的最少字符数。如果不设置这个属性，则不做此项检查
tooShortError	String	可选。用于设置在用户输入的字符数少于 minLength 值时显示的自定义消息。如果不设置这个属性，则显示标准的提示字符太少的消息
maxLength	Number	必填的最多字符数。如果不设置这个属性，则不做此项检查
tooLongError	String	可选。用于设置在用户输入的字符数多于 maxLength 值时显示的自定义消息

代码清单 6-2 所示为一个使用 StringValidator 的例子，验证条件为最少 3 个字符，最多 20 个字符。

代码清单6-2　使用StringValidator验证用户输入的字符数是否在合理范围内

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:StringValidator
   source="{username}" property="text"
   minLength="3" maxLength="20"
   trigger="{submitButton}" triggerEvent="click"
   tooShortError="Your username must be at least 3 characters"
   tooLongError="As if you'll remember that long of a username"
   />
   <mx:Text text="Enter your username:"/>
   <mx:TextInput id="username" />
   <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

用于触发验证操作

图 6-4 所示为通过单击按钮触发验证操作的结果。

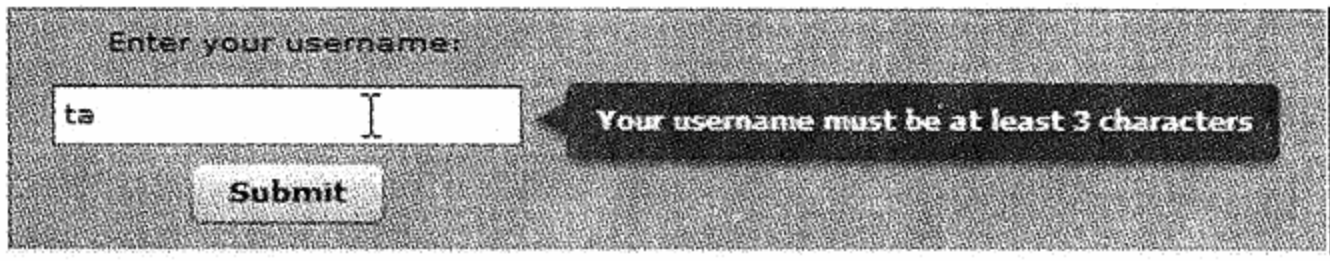


图 6-4　StringValidator 提示用户输入的字符数还没有达到最低 3 个字符的要求

这个示例与前面的示例有两处不同。

- 在一般意义上，验证器默认侦听的是发生在它所验证的组件上的触发事件(即正常情况下会留意用户提交 TextInput 中的值)。但这个示例通过将 trigger 属性设置为指向按钮，覆盖了默认行为。
- 同样是在默认情况下，触发事件应该是 valueCommit。但由于使用按钮，因此要使验证器留意按钮的 click 事件，而不是留意是否按下了回车键或 Tab 键

通过这个示例可以看出，使用 Flex 验证器是相当简单的事，你不用再为完成验证任务而前思后想、费尽周折。而且，由于这些验证器突出了实用性、针对性，使用起来也非常简单。

既然有针对字符串的验证器，那么一定也有针对数值的验证器。

6.2.3　NumberValidator

顾名思义，NumberValidator 专门用于验证数字值。这个验证器既能测试输入的值是否过大或过小，也能把输入限制为整数或非负数。另外，NumberValidator 具有自动识别千分位(如 23 543 121 72)的智能水平。表 6-3 所示为 NumberValidator 的可用属性。

表 6-3 NumberValidator 的特有属性

属 性	类 型	说 明
allowNegative	Boolean	是否允许负值？默认为 true
negativeError	String	用于 allowNegative 被设置为 false 的情况下，如果用户输入负值，显示给用户的提示消息。如果不设置这个字段，则显示默认的标准消息
domain	String	real 或 int，默认为 real。在设置为 real 的情况下，用户可以输入小数
integerError	String	用于设置在 domain 被设置为 int 的情况下，如果用户输入了小数，显示给用户的提示消息。如果未设置这个字段，则显示默认的标准消息
decimalSeparator	String	用于设置表示小数点的非数字字符。在针对目标用户本地化输入字段时有用(例如，欧洲使用逗号作为小数点)。如果不设置这个属性，则默认为句点(.)
decimalPointCountError	String	用于设置在用户输入包含多个小数点的数字时，显示给用户的提示消息。如果不设置这个字段，则显示默认的标准消息
minValue	Number	必须的最小值。如果不设置这个字段，则忽略此字段
lowerThanMinError	String	用于设置在用户输入的数值小于 minValue 值时，显示给用户的自定义提示消息。这个字段是可选的，如果不设置这个字段，则显示默认的指示数值过小的标准消息
maxValue	Number	允许的最大值。这个字段是可选的，如果不设置这个字段，则不做此项检查
exceedsMaxError	String	用于确定在用户输入的数值大于 maxValue 值时，显示给用户的自定义提示消息。这个字段是可选的，如果不设置这个字段，则显示默认的指示数值过大的标准消息
precision	Number	用于确定允许小数点后面出现几位数字。默认值为-1，即忽略此参数
precisionError	String	用于设置用户输入的小数点后面的位数超过 precision 参数指定的数量时，显示给用户的自定义提示消息。这个字段是可选的，如果不设置这个参数，则显示默认的与此错误相关的标准消息
thousandsSeparator	String	用于设置表示千分位分隔符的字符。默认为逗号
separationError	String	用于设置在用户使用不正确字符分隔千分位时，显示给用户的自定义提示消息。这个字段是可选的，如果不设置这个字段，则显示默认的与此错误相关的标准消息

如此多的属性，也说明这个验证器的用法可以非常灵活。代码清单 6-3 使用这个验证器限制用户输入的数值介于 5 至 110 之间，而且必须是整数。

代码清单6-3　使用NumberValidator检查数值的范围并确保该值是一个整数

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:NumberValidator
    source="{age}" property="text" allowNegative="false"
    negativeError="I highly doubt you're that young!"
    minValue="5" maxValue="110" domain="int"
    trigger="{submitButton}" triggerEvent="click"/>
    <mx:Text text="Enter your age:"/>
    <mx:TextInput id="age" />
    <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

图 6-5 所示为用户输入负数的结果，此时会显示一条警告消息，指示提供的输入信息是不正确的。

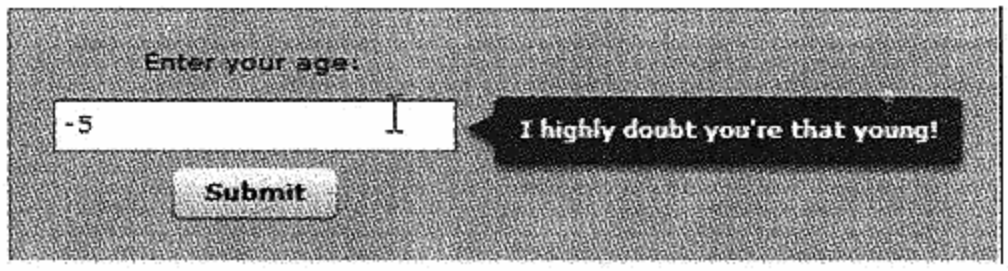


图 6-5　使用 NumberValidator 检查用户是否输入真实的年龄

前面示例代码中的 allowNegative 属性是多余的，但即使设置它也不会有什么负面影响；设置该属性只是向用户表明，不允许输入负数(见图 6-5)。因为我们已经定义了允许的最小值，所以不设置该属性也没有问题。

前面已经介绍了验证字符串和数值。接下来介绍一个最常用的验证器——验证日期的 DateValidator。

6.2.4　DateValidator

想必读者已经猜到了，DateValidator 是为确保用户输入有效的日期数据而设计的。DateValidator 一个值得称道的特性是，它能够接收 3 个独立的输入控件，这 3 个控件分别用来保存年、月、日(这区别于只能验证一个源的其他验证器)。

当然，标准的使用方式也是只验证一个源(如一个 TextInput)，而该对象保存的值应该是 08/21/85 这种格式。表 6-4 所示为在验证单个源时涉及的 DateValidator 属性。如果想通过独立的字段捕获日期数据，可以参见表 6-5 所示的另外一些必需的 DateValidator 属性。

表 6-4　DateValidator 的基本属性

属　性	类　型	说　明
allowedFormatChars	String	设置在日期中用来分割日、月和年的一系列字符。例如，/.允许的日期格式为“12/31/2008”和“12.31.2008”。默认为/\-..
invalidCharError	String	当日期中的字符与 allowedFormatChars 指定的字符不匹配时显示的消息

(续表)

属　性	类　型	说　明
inputFormat	String	设置日期格式或日、月、年先后顺序的字符串。如“mm/dd/yyyy”、“yyyy/mm/dd”等。分隔符仍然根据 allowedFormatChars 决定
formatError	String	当日期值不符合 inputFormat 指定的格式时显示的消息
validateAsString	Boolean	true 或 false。默认值为 true，即将源视为自由格式的文本字段。如果把这个属性设置为 false，则把源视为一个 Date 对象(即 DateField 控件)

如果要使用三个独立的字段存储日期，就使用表 6-5 所示的将这些字段作为一个整体进行验证的属性。

表 6-5　多字段下的 DateValidator 属性

属　性	类　型	说　明
daySource	Object	据以验证日的对象(即一个通过 id 属性指定名称的 TextInput 控件)
dayProperty	String	检查 daySource 对象的哪个属性(TextIuput 的 text 属性)来验证提供的值是否符合要求。默认为 text 属性
dayListener	Object	默认为 daySource 对象的值。在验证 daySource 的 dayProperty 失败的情况下，Flex 会默认突出显示 daySource 对象。可以使用 dayListener 属性来改变 Flex 在这种情况下突出显示的对象
wrongDayError	String	设置当用户输入的日在当月中不存在时显示的消息。这个属性是可选的，如果不设置这个属性，则显示默认消息
monthSource	Object	据以验证月的对象(即一个通过 id 属性指定名称的 TextInput 控件)
monthProperty	String	检查 monthSource 对象的哪个属性(TextInput 的 text 属性)来验证提供的值是否符合要求。默认为 text 属性
monthListener	Object	默认为 monthSource 对象的值。在验证 monthSource 的 monthProperty 失败的情况下，Flex 会默认突出显示 monthSource 对象。可以使用 monthListener 属性来改变 Flex 在这种情况下突出显示的对象
wrongMonthError	String	设置当用户输入的月不介于 1～12 时显示的消息。这个属性是可选的，如果不设置这个属性，则显示默认消息
yearSource	Object	据以验证年的对象(即一个通过 id 属性指定名称的 TextInput 控件)
yearProperty	String	检查 yearSource 对象的哪个属性(TextInput 的 text 属性)来验证提供的值是否符合要求。默认为 text 属性
yearListener	Object	默认为 yearSource 对象的值。在验证 yearSource 的 yearProperty 失败的情况下，Flex 会默认突出显示 yearSource 对象。可以使用 yearListener 属性来改变 Flex 在这种情况下突出显示的对象
wrongYearError	String	设置当用户输入的值不在 0 至 9 999 之间时显示的消息。如果不设置这个属性，则显示默认消息

代码清单 6-4 所示为单源 DateUalidator 的例子，它检查 TextInput 控件来查看输入的值是不是一个有效的日期。

代码清单6-4　使用DateValidator验证单源输入

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:DateValidator
    source="{birthday}" property="text" inputFormat="mm/dd/yyyy"
    allowedFormatChars="/"
    trigger="{submitButton}" triggerEvent="click"/>
   <mx:Text text="Enter your birth date:"/>
   <mx:TextInput id="birthday" />
   <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

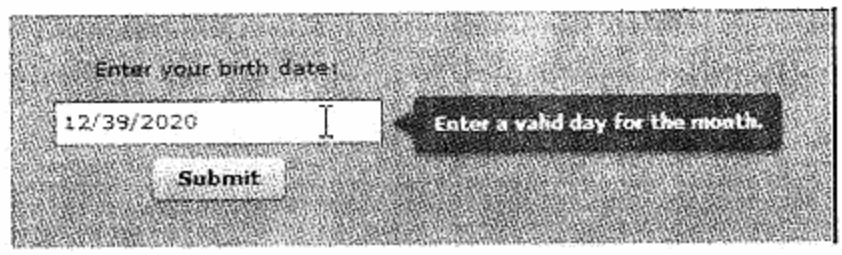


图 6-6　DateValidator 不认可输入的日期

如图 6-6 所示，由于 12 月没有 39 日，所以 DateValidator 给出了提示消息。消息中甚至明确指出错误所在。代码清单 6-5 在这个示例基础上扩展了一下，把捕获日期的字段增加为 3 个，同时使用了更高级的属性。

代码清单6-5　使用高级属性的例子

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="horizontal">
   <mx:DateValidator
    monthSource="{month}" monthProperty="value"
    daySource="{day}" dayProperty="value"
    yearSource="{year}" yearProperty="text"
    trigger="{submitButton}" triggerEvent="click"/>
   <mx:Text text="Month:"/>
   <mx:NumericStepper id="month" />
   <mx:Text text="Day:"/>
   <mx:NumericStepper id="day" />l
   <mx:Text text="Year:"/>
   <mx:TextInput id="year"/>
   <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

标识分布于多个输入字段中的日期数据

运行这个示例，然后试着在表示日、月、年的 3 个日期区域分别输入日期值。图 6-7 显示在输入错误的日期值时，系统会弹出一个消息框，提示你修改问题。

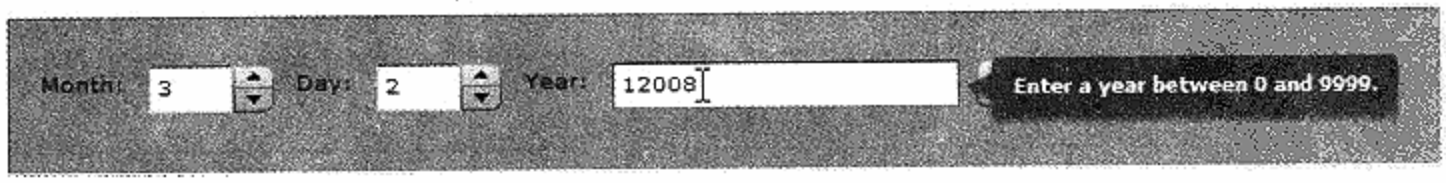


图 6-7　DateValidator 可以验证构成完整日期的 3 个独立输入字段

可见，使用 DateValidator 无论验证单个文本日期字段，还是验证多字段日期，都是非常方便的。

提示：

在实际应用程序中，应该使用 DateField 或 DateChooser 控件来捕获用户提供的日期数据。

以上介绍的是几个基本的验证器，接下来我们再看其他几个常见的验证器。首先从验证电子邮件地址的验证器开始。

6.2.5 EmailValidator

检查电子邮件地址的格式是任何应用程序注册功能中的重要一环，而且通常是在服务器端进行后期验证。EmailValidator 提供了十分彻底的客户端验证功能，它不仅能够验证电子邮件地址中是否包含@，同样能检查域名后缀是否合理。

使用 EmailValidator 时，唯一可以配置的属性是错误消息。原因很简单：电子邮件具有标准的命名格式和有限的结构变化形式，因此不需要开发人员为捕获地址做过多的准备。表 6-6 所示为 EmailValidator 的专有属性。

表 6-6 EmailValidator 的属性

属　性	类　型	说　明
invalidCharError	String	设置当电子邮件地址中存在无效字符(如空格)时，显示给用户的提示消息。如果没有设置这个属性，则显示默认的标准消息
invalidDomainError	String	设置当电子邮件地址中存在无效域名时，显示给用户的提示消息。如果没有设置这个属性，则显示默认的标准消息
invalidIPDomainError	String	与 invalidDomainError 类似，但用于设置当电子邮件地址中存在无效 IP 地址时，显示给用户的提示消息。如果没有设置这个属性，则显示默认的标准消息
invalidPeriodsInDomainError	String	设置当电子邮件地址中的句点字符出现在错误位置上时(通常是连着两个句点)，显示给用户的提示消息。如果没有设置这个属性，则显示默认的标准消息
missingAtSignsError	String	设置当电子邮件地址中缺少@符号时，显示给用户的提示消息。如果没有设置这个属性，则显示默认的标准消息
missingPeriodsInDomainError	String	设置当电子邮件地址域名部分缺少句点时，显示给用户的提示消息。如果没有设置这个属性，则显示默认的标准消息
missingUserName	String	设置当电子邮件地址中缺少用户名(位于@符号左侧)时，显示给用户的提示消息。如果没有设置这个属性，则显示默认的标准消息
tooManyAtSignsError	String	设置当电子邮件地址中存在多个@符号时，显示给用户的提示消息。如果没有设置这个属性，则显示默认的标准消息

代码清单 6-6 所示为一个使用 EmailValidator 检查电子邮件地址格式是否有效(不包含空格和特殊字符，不缺少字符，等等)的例子。

代码清单6-6　验证电子邮件地址的结构是否有效

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="horizontal">
  <mx:EmailValidator source="{email}" property="text"
   invalidCharError="You've got some funky characters in that Email"
   trigger="{submitButton}" triggerEvent="click"/>
  <mx:Text text="Email:"/>
  <mx:TextInput id="email"/>
  <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

自定义提示消息

编译并运行这个应用程序，然后故意输入一个包含一些无效字符的电子邮件地址，将看到与图 6-8 所示的错误消息。

图 6-8　EmailValidator 对这个错误的电子邮件给出提示，因为其中包含一个空格

在 Web 开发中，以实时方式来预先验证电子邮件地址的情况并不多见。验证过程越复杂，开发人员就要运用越多的后端服务器端技术。

在 HTML 加 JavaScript 的开发中，预先验证电子邮件不容易做到；但更不常见的是动态验证信用卡号码。

6.2.6　CreditCardValidator

下面我们将介绍在许多编程语言中都见不到的验证器，这个验证器就是 CreditCardValidator；它使用 Luhn mod10 算法检查用户提供的号码和信用卡类型是否匹配。

这个验证器需要两个输入源。

- 信用卡类型(Flex 支持 American Express、Diners Club、Discover、MasterCard 和 Visa)。
- 信用卡号码。

表 6-7 所示为 CreditCardValidator 的属性。

表 6-7　CreditCardValidator 的属性

属　性	类　型	说　明
allowedFormatChars	String	设置允许作为信用卡号码分隔符的字符。默认为空格和短划线(-)
cardNumberSource	Object	包含信用卡号码的对象(即 TextInput 的 id 属性)
cardNumberProperty	Object	包含信用卡号码的实际值的 cardNumberSource 属性(TextInput 的 text 属性)
cardNumberListener	Object	接收验证结果的对象。默认是 cardNumberSource 引用的对象

(续表)

属　性	类　型	说　明
cardTypeSource	Object	包含信用卡类型的对象(TextInput 的 id 属性)
cardTypeProperty	Object	实际包含信用卡类型的实际值的 cardTypeSource 属性(TextInput 的 text 属性)
cardTypeListener	Object	接收验证结果的对象。默认是 cardTypeSource 引用的对象
invalidCharError	String	设置当信用卡号码中包含无效字符时，显示给用户的消息。如果没有设置这个属性，则显示默认的标准消息
invalidNumberError	String	最可能出现的消息。当信用卡号码不匹配信用卡类型时显示。如果没有设置这个属性，则显示默认的标准消息
noNumError	String	设置当信用卡号码为空时，显示给用户的消息。如果没有设置这个属性，则显示默认的标准消息
noTypeError	String	设置当未指定信用卡类型时，显示给用户的消息。如果没有设置这个属性，则显示默认的标准消息
wrongLengthError	String	设置当输入的数字数量与给定信用卡类型不匹配时，显示给用户的消息。如果没有设置这个属性，则显示默认的标准消息
wrongTypeError	String	设置当信用卡类型未知时，显示给用户的消息。如果没有设置这个属性，则显示默认的标准消息

代码清单 6-7 中没有硬性编码某些属性，而是允许用户选择信用卡类型。然后，通过数据绑定来更新 CreditCardValidator 的适当属性。

代码清单6-7　通过数据绑定来了解要验证的信用卡类型

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="horizontal">
  <mx:CreditCardValidator
    cardNumberSource="{cardNumber}" cardNumberProperty="text"
    cardTypeSource="{cardType}"  cardTypeProperty="selectedItem"
    trigger="{submitButton}" triggerEvent="click"/>
    <mx:ComboBox id="cardType">
      <mx:dataProvider>
      <mx:String>American Express</mx:String>
      <mx:String>Visa</mx:String>
  <mx:String>Diners Club</mx:String>
  <mx:String>Discover</mx:String>
  <mx:String>MasterCard</mx:String>
    </mx:dataProvider>
    </mx:ComboBox>
    <mx:Text text="Card Number:"/>
    <mx:TextInput id="cardNumber"/>
    <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

更改被传递给验证器

使用绑定动态设置卡类型

编译并运行这个应用程序，然后输入 4 组、每组 4 个数字作为要验证的信用卡号码。

如图 6-9 所示，在输入错误的 Visa 卡号后会看到一条提示消息。

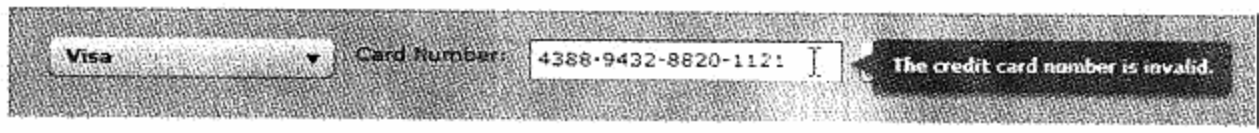


图 6-9　CreditCardValidator 用于验证信用卡类型与信用卡号码是否匹配

虽然本书讨论的都是与联机事务有关的主题，但有时候还得考虑怎么验证货币。如果我们开发的系统接收世界各地的订单，就必须知道怎样验证货币兑换的值。

6.2.7　CurrencyValidator

如果面向国际市场开发应用程序，必须理解需要多付出一些努力，来考虑各国的国民就同一事物的不同表达方式。

在验证包含货币的表达式时，CurrencyValidator 会帮上我们的忙。CurrencyValidator 与 NumberValidator 很相似，因为它也能够检查小数点精度、最小值和最大值以及小数点分隔符等。表 6-8 所示为 CurrencyValidator 的属性。

表 6-8　CurrencyValidator 的可用属性

属　性	类　型	说　明
allowNegative	Boolean	是否允许负值？默认值为 true
negativeError	String	设置当用户输入的值中包含负数，而 allowNegative 被设置为 false 的情况下，显示的自定义消息。如果不设置这个属性，则显示默认的标准错误消息
decimalSeparator	String	表示小数点的非数字字符。如果未设置，则使用默认的句点(.)
decimalPointCountError	String	设置当输入的数字中包含一个以上小数点的情况下，显示的自定义消息。如果不设置这个字段，则显示默认的标准错误消息
minValue	Number	最小值。如果不设置这个字段，则忽略此字段
lowerThanMinError	String	设置当输入的值小于 minValue 值时，显示的自定义消息。这个属性是可选的。如果不设置这个属性，则显示默认的标准错误消息
maxValue	Number	允许的最大值。如果不设置这个字段，则忽略此字段
exceedsMaxError	String	设置当用户输入的值大于 maxValue 值时，显示的自定义消息。这个属性是可选的。如果不设置这个属性，则显示默认的标准错误消息
precision	Number	设置允许小数点右侧出现几位数字。默认为-1，即忽略此参数
precisionError	String	设置当用户输入值的小数点右侧数字位数多于 precision 参数允许的位数时，显示的自定义消息。这个属性是可选的。如果不设置这个属性，则显示默认的标准错误消息
thousandsSeparator	String	作为千分位分隔符的字符。默认为逗号(,)

(续表)

属　性	类　型	说　明
separationError	String	设置当用户输入的值中包含无效的千分位分隔符时，显示的自定义消息。这个属性是可选的。如果不设置这个属性，则显示默认的标准错误消息
alignSymbol	String	用于指定货币符号的位置(left 或 right)。默认为 left
currencySymbol	String	用于指定货币符号。默认为$
currencySymbolError	String	设置当输入的货币符号与 CurrencySymbol 的定义不匹配时，显示的自定义消息
invalidCharError	String	设置当用户输入的值中包含不在预期范围内的字符时，显示的自定义消息。该属性是可选的。如果不设置这个属性，则显示默认的标准错误消息
invalidFormatCharsError	String	设置当用户输入的货币格式与货币定义不匹配时(如无效的千分位分隔符)，显示的自定义消息。该属性是可选的。如果不设置这个属性，则显示默认的标准错误消息

代码清单 6-8 使用 CurrencyValidator 验证只带两位小数的货币。

代码清单6-8　使用CurrencyValidator确保货币值无误

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:CurrencyValidator
   source="{income}" property="text" allowNegative="false"
   negativeError="You pay your employer?"
   precision="2" precisionError="Just 2 decimals please."
   trigger="{submitButton}" triggerEvent="click"  />
  <mx:Text text="How much do you make?"/>
  <mx:TextInput id="income" />
  <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

只允许有两位小数

如图 6-10 所示，为了响应提示而输入的钱数出现了问题。问题出现在小数点右侧的数字位数上，三位数字是不正确的(虽然你可能觉得是自己太抠门造成的)，是它触发了弹出式的自定义错误消息窗口。

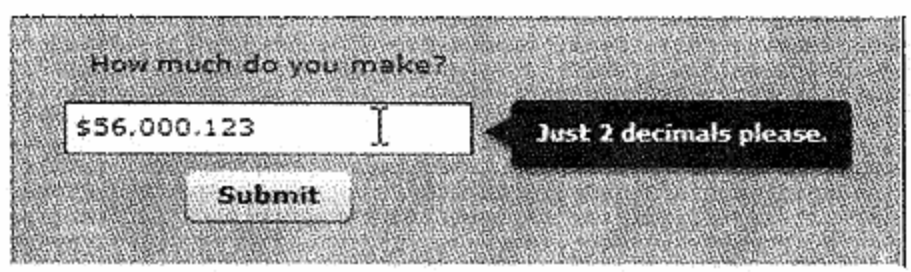


图 6-10　验证失败的原因是已将 CurrencyValidator 配置为只接受两位小数

与 CurrencyValidator 在国际贸易中的作用类似，能够验证世界各地电话号码的 Phone-

NumberValidator 显然促进了国际交流。

6.2.8 PhoneNumberValidator

PhoneNumberValidator 确实比较简单——电话号码要么正确，要么错误。这个验证器不仅能够识别国际电话号码，也能够识别北美各地电话号码。

使用 PhoneNumberValidator 只需关注两件事：电话号码必须至少包含 10 位数，电话号码中包含的格式化字符必须有效。表 6-9 所示为 PhoneNumberValidator 支持的属性。

表 6-9　PhoneNumberValidator 的属性

属　性	类　型	说　明
allowedFormatChars	String	指定允许电话号码中出现的格式化号码的字符。默认为()- .+
invalidCharError	String	设置当用户输入的值中包含 allowedFormatChars 没有指定的字符时显示的自定义消息。如果不设置这个字段，则默认显示标准的错误消息
wrongLengthError	String	设置当用户输入值的位数小于 10 时，显示的自定义消息。如果不设置这个值，则显示默认的错误消息

代码清单 6-9 所示为一个验证电话号码的简单例子。

代码清单6-9　检查用户输入的电话号码格式是否正确

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:PhoneNumberValidator
     source="{phone}" property="text"
     trigger="{submitButton}" triggerEvent="click" />
  <mx:Text text="What number can we reach you at?"/>
  <mx:TextInput id="phone" />
  <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

编译并运行这个应用程序，分别输入有效和无效的电话号码，看看验证器给出的验证结果，如图 6-11 所示。

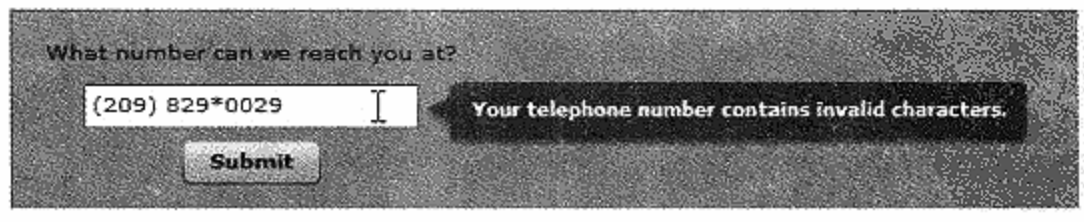


图 6-11　只有格式正确的电话号码才能通过 PhoneNumberValidator 验证

编写自定义验证器来验证格式不一的电话号码会十分繁琐。不过，即便是在上述所有验证器都无法胜任的情况下，可以使用 Flex 为我们提供的一个通用、万能的正则表达式验证器。

6.2.9 RegExpValidator

几十年来，RegEx(Regular Expression，正则表达式)始终都是一种简单、强大的语言，它几乎没有任何限制。RegExpValidator 的作用就是将用户输入的值与 RegEx 表达式做比较，以便确定是否匹配(找到了就说明值是有效的)。

如表 6-10 所示，RegExpValidator 的几个属性涉及正则表达式本身，以及发现不匹配项时显示的自定义消息。

表 6-10 RegExpValidator 的属性

属 性	类 型	说 明
expression	String	用来查找匹配项的 RegEx 模式
flags	String	对应的可选 RegEx 标志。g 表示全局搜索，即查找所有匹配项；i 表示忽略大小写；m 用于多行的情况，告诉验证器使用^和$ RegEx 表达式测试每一行(相对于测试一个连贯的字符串，此时要测试的是一行字符)
noExpressionError	String	设置当 expression 属性为空时显示的消息
noMatchError	String	设置当用户输入的值与 RegEx 表达式不匹配时显示的消息

下面我们以验证用户输入的 SSN(Social Security Number，美国社会保障号码)为例，代码清单 6-10 展示如何测试用户输入的文本是否匹配 SSN 模式(使用 RegEx 模式)。

代码清单6-10 RegExpValidator根据RegEx模式匹配用户输入的文本

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:RegExpValidator source="{ssn}" property="text" flags="gmi"
    expression="\d\{3\}.\d\{2\}.\d\{4\}"
    noMatchError="Your SSN is unrecognized."
    trigger="{submitButton}" triggerEvent="click"/>
  <mx:Text text="Social Security Number:"/>
  <mx:TextInput id="ssn" />
  <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

SSN RegEx模式

如图 6-12 所示，可以看到与其他验证器示例类似的结果，但 RegExpValidator 后台的工作原理稍有不同。毕竟正则表达式的工作原理是模式匹配，而非验证。因此，可以利用此功能查找与模式成功匹配的匹配项数，甚至匹配项在字符串中的位置。

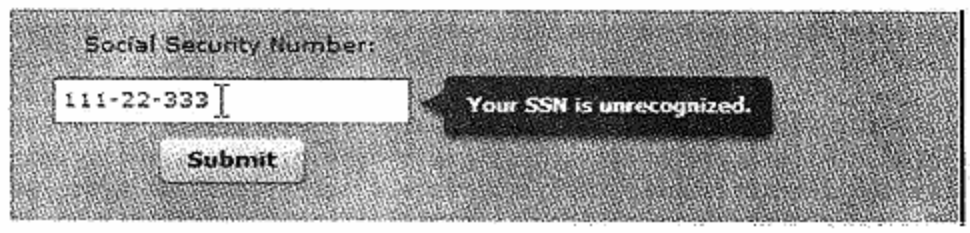


图 6-12 使用正则表达式验证 SSN

为此需要如代码清单 6-11 所示编写一些 ActionScript 脚本来创建验证事件处理程序。

代码清单6-11　使用RegExpValidator查找与模式匹配的所有匹配项

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
   <![CDATA[
     import mx.events.ValidationResultEvent;
     import mx.validators.RegExpValidationResult;
     import mx.controls.Alert;
     private function handleValidation(event:ValidationResultEvent):void
     {
      var oneResult:RegExpValidationResult;
      for (var i:int = 0; i < event.results.length; i++)
      {
       oneResult = event.results[i];
       Alert.show("Found a match at Index:" + oneResult.matchedIndex +
               "\nOn the characters of:" + oneResult.matchedString
               ,"RegEx Results",Alert.NONMODAL);
      }
     }
      ]]>
  </mx:Script>
  <mx:RegExpValidator source="{test}" property="text" flags="gmi" 1
                valid="handleValidation(event)"
                expression="m[ai]n" noMatchError="I don't like it!"
                trigger="{submitButton}" triggerEvent="click"/>
    <mx:Text text="Try me:"/>
  <mx:TextInput id="test" />
  <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

导入必需的类

评估每个匹配项

循环遍历每个模式匹配项

为方便而声明的临时变量

显示匹配项的信息

在这个示例中，返回两个有效的模式匹配项(图 6-13)。然后使用 ActionScript 循环遍历这两个匹配项。

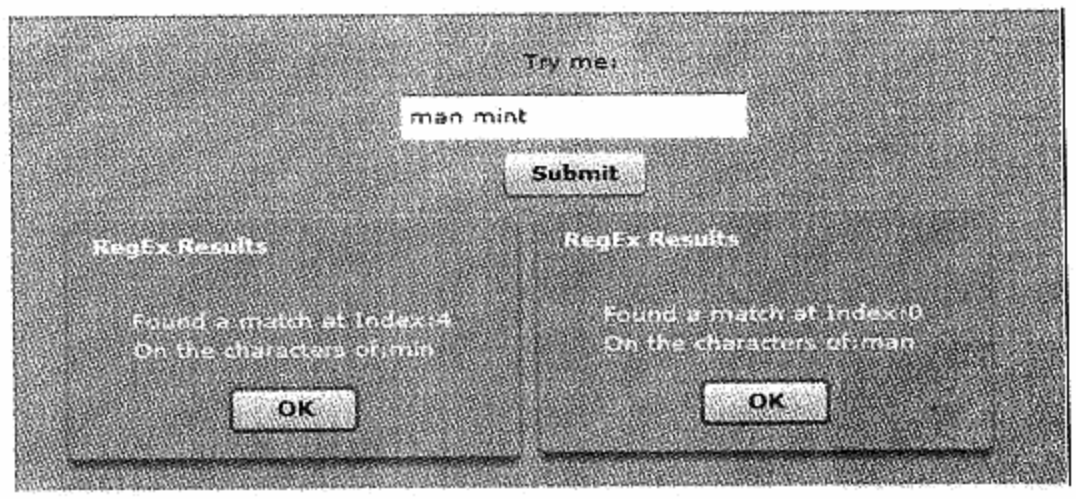


图 6-13　提交的文本中包含两个匹配正则表达式的项

正则表达式的确非常强大。事实上，RegExpValidator 几乎可以用来替代 Flex 提供的其他所有验证器。只不过这样一来，就要求我们每个人都必须是正则表达式专家(但很多人恐怕并未将成为 RegEx 专家定为职业目标)。

6.2.10 SocialSecurityValidator

上一个示例使用 RegExpValidator 来验证 SSN，但 Flex 还特地针对验证这个数字标识符设计了一个验证器——SocialSecurityValidator。

SocialSecurityValidator 使用起来很直观，它验证的是格式为“xxx-xx-xxxx”的一组数字和分隔符串。此外，SocialSecurityValidator 还允许我们指定分隔符(默认为短划线和空格)，并且坚守 SSN 前 3 位不能是“000”的规则(表 6-11)。

表 6-11 SocialSecurityValidator 的属性

属　性	类　型	说　明
allowedFormatChars	String	可以用来分隔每个数字块的有效字符。默认为空格或短划线(-)
invalidCharError	String	设置当用户输入的值中包含 allowedFormatChars 未指定的分隔符时，显示的自定义消息。如果不设置这个属性，则显示默认的标准错误消息
wrongFormatError	String	设置当用户输入值的格式与 9 位或 xxx-xx-xxxx 格式不匹配时，显示的自定义消息。如果不设置这个属性，则显示默认的标准错误消息
zeroStartError	String	设置当用户输入的值以 3 个“0”开头时，显示的自定义消息。如果不设置这个属性，则显示默认的标准错误消息

代码清单 6-11 使用正则表达式验证 SSN，而代码清单 6-12 使用 SocialSecurityValidator。

代码清单6-12 SocialSecurityValidator会在SSN格式错误时提醒用户

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:SocialSecurityValidator
    source="{ssn}" property="text"
    trigger="{submitButton}" triggerEvent="click" />
  <mx:Text text="Try me:"/>
  <mx:TextInput id="ssn" />
  <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

编译并运行这个应用程序，输入与既定 SSN 模式不匹配的号码，会触发如图 6-14 所示的错误消息。

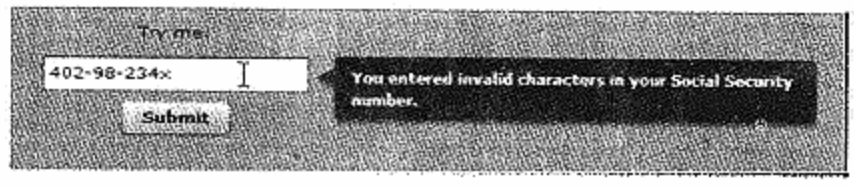


图 6-14 失败！SocialSecurityValidator 检测到输入值不是有效的 SSN

6.2.11　ZipCodeValidator

最后(但并非最不重要)，我们介绍 ZipCodeValidator。这个简便的验证器用来验证用户是否正确输入美国邮政编码(U.S. ZIP Code)或加拿大邮递区号(postal code)。美国邮政编码的格式要么是 5 位数字，要么是 5+4 的格式(表 6-12)。

表 6-12　ZipCodeValidator 的属性

属　性	类　型	说　明
domain	String	设置只验证美国邮政编码(US Only)，还是验证美国邮政编码或加拿大邮递区号(US or Canada)。默认值为 US Only
allowedFormatChars	String	用来分隔邮政编码中的每个数字块的有效字符。默认为空格或短划线(-)
invalidCharError	String	设置当用户输入的值中包含 allowedFormatChars 未指定的分隔符时，显示的自定义消息。如果不设置这个属性，则显示默认的标准错误消息
invalidDomainError	String	设置当 domain 包含的值无效时。显示的自定义消息。如果不设置这个属性，则显示默认的标准错误消息
wrongCAFormatError	String	设置当用户输入的值不符合加拿大邮递区号格式时显示的自定义消息。如果不设置这个属性，则显示默认的错误消息
wrongLengthError	String	设置当用户输入值的位数与相应范围(domain)要求不匹配时，显示的自定义消息
wrongUSFormatError	String	设置当用户输入的值不符合美国邮政编码格式时，显示的自定义消息。如果不设置这个属性，则显示默认的错误消息

代码清单 6-13 所示为使用 ZipCodeValidator 验证用户的输入是否符合美国邮政编码格式的例子。

代码清单6-13　使用ZipCodeValidator测试美国邮政编码

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:ZipCodeValidator
    source="{zipcode}" property="text" domain="US Only"
    trigger="{submitButton}" triggerEvent="click"  />
  <mx:Text text="What's your Zip Code (U.S Only)?"/>
  <mx:TextInput id="zipcode" />
  <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

读者可以试验一下这个验证器。输入无效的美国邮政编码会导致如图 6-15 所示的警告消息。图中显示，输入的是加拿大邮递区号，而程序只接受美国邮政编码。

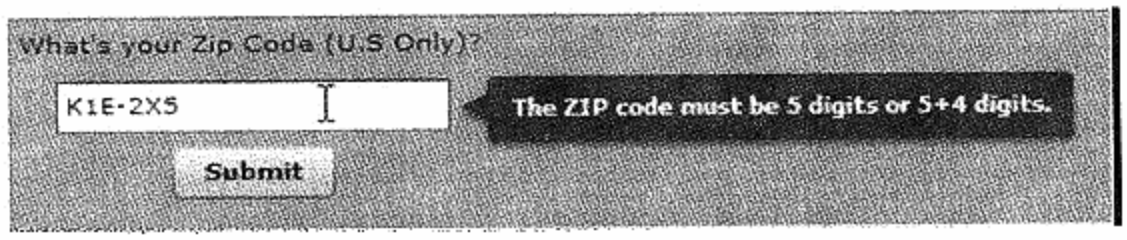


图 6-15 已将 ZipCodeValidator 配置为只检查美国邮政编码，因此加拿大邮递区号无法通过验证

由于把 ZipCodeValidator 配置成只允许用户输入美国邮政编码，因此输入加拿大邮递区号无法通过验证(当然，你可以使用 domain 属性，使 ZipCodeValidator 同时接受二者)。

至此，Flex 内置的所有验证就都介绍完毕。接下来我们讨论几种不同的验证方式。

6.3 实时验证

要进行实时验证，或者，在错误发生时就捕获输入错误，就必须“侦听”用户输入数据时发生的每个变化。

为此必须监视 change 触发器事件(triggerEvent)，如代码清单 6-14 所示。

代码清单6-14 使用change事件进行实时验证

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
 <mx:StringValidator source="{address}" minLength="5" property="text"
                trigger="{address}"
                triggerEvent="change"/>
 <mx:Text text="What's your address?"/>
 <mx:TextInput id="address"  />
</mx:Application>
```

通过change事件触发验证

在代码清单 6-14 中，从用户开始输入时起，文本字段就会变红，直到输入要求的最少字符数(5 个)为止。虽然在用户输入达到最低要求之前保持对用户的提示很有效，但也很容易引起用户的反感。为避免用户对这种小事的非议，更好的办法是在用户提交值时进行验证。

6.4 提交值验证

提交值验证与使用 change 事件的实时验证稍有不同，它不基于用户每次敲击键盘执行验证，而是用户提交输入完的值时验证。用户提交值的标志包括按下 Tab 键、回车键、方向键或鼠标单击其他组件。

为此，需要把触发器事件由 change 改为 valueCommit。

```
<mx:StringValidator source="{address}" minLength="5" property="text"
          trigger="{address}" triggerEvent="valueCommit"/>
```

这样的交互效果非常不错；然而也存在把验证操作推迟到用户填写完多个字段后再进行的情况。此时需要的是通过性验证。

6.5　通过性验证

由于验证操作被推迟的缘故，通过性验证更为被动一些。不过，让用户先填写完整个表单，然后把表单值提交到数据库之前验证，也是许多 Web 应用程序常见的做法。

通过性验证的主角是提交按钮，为此需要同时修改触发器(trigger)和触发器事件(triggerEvent)属性。也就是说，要把触发验证操作的对象由默认的 source 对象改为提交表单的按钮，使用按钮实例的 click 事件。

在实际表单中，可能很多输入字段的验证器都需要用到提交按钮的 click 事件。

代码清单 6-15 所示为如何通过多个验证器检查不同的 TextInput 字段，但是它们都被配置为侦听提交按钮的 click 事件。

代码清单6-15　使用提交按钮来验证多个表单字段

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:StringValidator source="{username}" property="text"
      minLength="6" trigger="{submitButton}" triggerEvent="click"/>

  <mx:EmailValidator source="{email}" property="text"
      trigger="{submitButton}" triggerEvent="click"/>

  <mx:Text text="Email:"/>
  <mx:TextInput id="email"/>

  <mx:Text text="Enter your username:"/>
  <mx:TextInput id="username" />

  <mx:Button label="Submit" id="submitButton"/>
</mx:Application>
```

为多个表单字段设置验证器

基于单击提交按钮验证

输入表单字段有验证器

这个示例及前面的示例中使用的都是自动事件，这意味着在触发事件时无需人工帮助即可做出响应。事实上，利用脚本式验证可以在任何时候验证用户输入的值。

6.6　脚本式验证

到此为止，对于我们介绍的所有验证器而言，如果使用相应的 ActionScript 脚本，将可以获得极大的便利和控制能力。

重用同一个验证器验证多个值时，可以使用相应的 ActionScript 版本。要验证的值不一定来自用户输入控件，你可以使用它们验证任意值。

代码清单6-16 使用ActionScript验证任意值

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      import mx.validators.EmailValidator;
      import mx.validators.StringValidator;
      import mx.controls.Alert;
      import mx.events.ValidationResultEvent;

      public var emailVal:EmailValidator = new EmailValidator();
      public var stringVal:StringValidator = new StringValidator();
      public function validateForm():void
      {
        var valResult:ValidationResultEvent;
        stringVal.source = username;
        stringVal.property = "text";
        stringVal.minLength=6;

        emailVal.source = email;
        emailVal.property = "text";

     valResult = emailVal.validate();
     if(valResult.type == "invalid")
     {
      Alert.show("Please fix your Email address.");
        }
     else
     {
      valResult = stringVal.validate();
      if(valResult.type == "invalid")
      {
       Alert.show("Please fix your Username.");
      }
     }

    }
   ]]>
  </mx:Script>

  <mx:Text text="Email:"/>
  <mx:TextInput id="email"/>

  <mx:Text text="Enter your username:"/>
  <mx:TextInput id="username" />

  <mx:Button label="Submit" id="submitButton" click="validateForm()"/>
</mx:Application>
```

导入必要的ActionScript类

代码清单 6-16 创建了两个验证器❶，分别用于验证两个 TextInput❸。当用户单击提交按钮时，会调用 validateForm()❷。首先验证电子邮件地址❹；如果验证通过，validateForm() 会继续验证下一个字段，检查用户名❺。

前面学习了各种类型的验证器及其用法。本章最后我们介绍几个与验证有关的小技巧。

6.7 验证技巧

虽然读者对验证大体上清楚了，但要彻底理解验证器的工作原理，还必须明确几个小问题。

6.7.1　验证器是否总是检查所有条件

如果用户输入的值与多个条件不相符，会不会显示所有错误消息呢？不会。Flex 会按照预订安排行事，一个条件一个条件地检查，遇到一个条件验证失败，会立即停止验证。

6.7.2　验证输入的值与条件匹配

要彻底理解验证器，必须知道不仅可以使用它们验证某种类型的值(如日期、信用卡号码或电话号码)，而且可以使用它们执行更重要的任务，即确定输入的值是否匹配某个条件。例如，检查用户输入的日期是否在一定时间段内。

6.7.3　控制由什么来触发验证操作

为了简明起见，我们描述了一些高级验证方式(如实时验证)，但使用 trigger 和 triggerEvent 属性就可以控制验证操作由什么事件来触发。

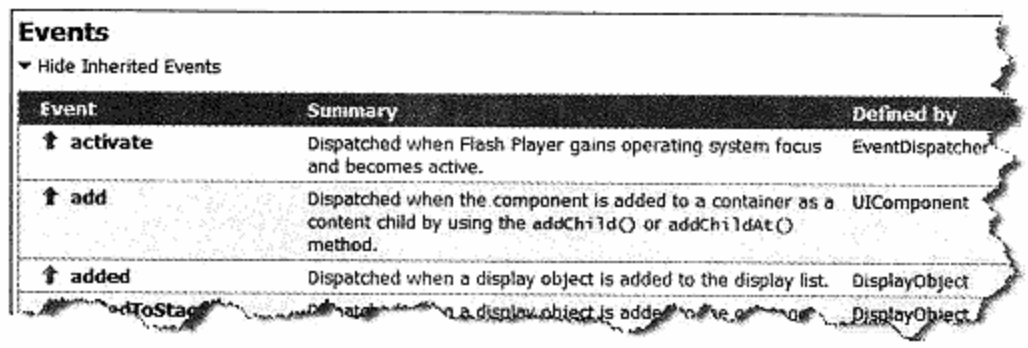
在前面的示例中，我们将按钮的 id 赋值给 trigger，将其 click 事件赋值给 triggerEvent。

那么，怎么知道可以使用哪些事件呢？在 Flex Builder 中，通过单击 Help | Search；或者单击相应组件，然后按 Shift+F2，可以打开相关组件的 API 参考。

提示：

另一种方式是在 Flex Builder 工作区中为动态帮助视图分配一个区域(参见第 2 章)，以便自动显示相应参考信息。

在 API 参考页面中，向下滚动到 Events 部分。其中列出的任何事件都是可以使用的(图 6-16)。例如，可以使用按钮的 mouseOver 事件作为触发器，这样就不必用户单击了。



Events

Hide Inherited Events

Event	Summary	Defined by
activate	Dispatched when Flash Player gains operating system focus and becomes active.	EventDispatcher
add	Dispatched when the component is added to a container as a content child by using the addChild() or addChildAt() method.	UIComponent
added	Dispatched when a display object is added to the display list.	DisplayObject
[illegible]	[illegible]	DisplayObject

图 6-16　通过 API 参考来了解可以用作 triggerEvent 的事件

6.8　小结

验证是增强用户的积极体验的宝贵机制，因为它可以把客户端设备与服务器之间往返交互的次数降至最低。通过实现客户端验证，可以减少用户填写表单的时间，从而提高可用性。此外，客户端验证对开发人员及后端系统同样是有好处的，毕竟如果提交给服务器的数据已在客户端预先经过验证，那么服务器的负载也可以在一定程度上降低。

Flex 内置了丰富实用的验证器，从简单的文本和数值验证器，到能够验证 SSN 及电话号码的目标更明确的验证器。如果这些验证器不能满足需要，还可以使用正则表达式验证器。正则表达式验证器不仅可以匹配模式，而且可以用来从文本中找出与模式匹配的所有项。

使用验证器的方式也不止一种。可以在用户每次敲击键盘时执行实时验证，也可以在用户提交值时验证，还可以等到用户填写完所有字段并单击提交按钮时执行一次性验证。

实际上，也不仅限于使用自动事件(如 click)；还可以使用 ActionScript 以编程方式动态创建验证器，并对某些既定的数据加以验证。

在所有字段都通过验证后，接下来考虑数据格式的规范化问题。第 7 章介绍如何使用 Flex 的格式化程序格式化数据。

第 7 章 格式化数据

本章内容：

- 通过 Flex 内置的格式化程序格式化原始数据
- 格式化程序的类型
- 实时格式化与脚本式格式化
- 处理格式化错误

格式化程序(Formatter)是一类对象，它们可以接收原始数据并将原始数据转换为适当的格式。从用法的角度讲，格式化程序与验证器可以相提并论，因为它们的实现机制比较相似。格式化程序的应用模式有两种。

- **实时格式化**：格式化是通过数据绑定自动完成的。
- **脚本式格式化**：在必要时，可以使用 ActionScript 明确创建格式化程序实例，然后再对应地处理数据。

最关键是要了解格式化程序的用法非常简单。输入原始数据，就会输出结构清晰、容易认读的格式化数据。本章主要讨论 Flex 内置格式化程序的使用方法及它们各自的适用范围。

注意：

格式化程序只适用于原始、未经格式化的数据。如果针对用户可以编辑的字段使用格式化程序，必须先删除输入数据中包含的所有格式化代码。

格式化程序可以独立使用，而且可以解决一定范围内的问题。格式化程序最常用的情形是从服务器(如数据库)中取得数据，然后以结构化、易认读的方式表现出来。当然，这种应用一般通过 Internet 实现，但本章并不想就连接问题浪费大量时间。为了简单起见，我们使用静态的 XML 文件模拟远程服务器中的数据。

下面从内置格式化程序开始介绍。

7.1 内置格式化程序

Flex 提供了很多内置的、可以拿来就用的格式化程序。这些格式化程序都派生自分类 Formatter，因而在默认情况下都支持一组公共的属性、函数和事件。

与验证器一样，Adobe 的 Flex 团队以扩展 Formatter 类的方式创建了一组比较常用，但具有针对性的专门工具(通过各自的与上下文有关的属性实现功能)。

下面将展示所有内置的格式化程序及它们的独到之处。别忘了父格式化程序(Formatter 类)能够做到的，每个专用的格式化程序同样能做到。

7.1.1 Formatter

Formatter 是所有专用格式化程序的父类，它作为其他格式化程序的模板存在。作为基类，它的功能有一定的局限性，如表 7-1 所示，它定义了一个重要的函数。

表 7-1 所有格式化程序共有的重要函数

函 数	说 明
format	接收一个需要被格式化的对象，返回 String 类型的结果

Formatter 也定义了一个重要的属性(表 7-2)，该属性用于确定格式化问题的来源。

表 7-2 所有格式化程序的共有属性

属 性	类 型	说 明
error	string	在想要隔离格式化问题时使用这个属性。在 format()函数返回空 String 时，意味着有错误发生

其他格式化程序拥有附加的配置属性，但用法是相同的。

7.1.2 NumberFormatter

勿庸置疑，最常用的格式化程序是 NumberFormatter。NumberFormatter 用于处理数值表现的细节，如小数精度和用作千分位分隔符的字符。

提示：

ColdFusion 提供了与 NumberFormatter 类似的 NumberFormat()函数。

表 7-3 所示为 NumberFormatter 拥有的额外属性。

表 7-3　NumberFormatter 的额外属性

属　性	类　型	说　明
decimalSeparatorFrom	String	指定输入数据中使用的小数点。默认为句点(.)
decimalSeparatorTo	String	指定格式化结果中使用的小数点。默认为句点(.)
precision	Number	设置格式化结果的精度级别(保留几位小数)。默认值为-1，表示忽略这个属性而采用与输入相同的精度
rounding	String	设置对小数舍入的方式。可用选项有 up、down 和 nearest。默认值为 none。nearest 表示让格式化程序把输入值向上或向下舍入为最接近的整数值
thousandsSeparatorFrom	String	指定输入数据中使用的千分位分隔符。默认为逗号(,)
thousandsSeparatorTo	String	指定格式化结果中使用的千分位分隔符。默认为逗号(,)
useNegativeSign	Boolean	true 或 false。默认为 true，即使用负号表示负值。false 表示把数值显示在一对圆括号中(账目风格)
useThousandsSeparator	Boolean	true 或 false。指定是否使用千分位分隔符。如果值为 true，输出中将包含 thousandsSeparatorTo 属性中指定的千分位分隔符

下面我们列举使用 NumberFormatter 的第一个例子。这个例子接收带 4 位小数的原始数值，并将其精度降低为两位显示(图 7-1)，如代码清单 7-1 所示。

代码清单7-1　使用NumberFormatter把数值格式化为包含两位小数

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:XML id="myData">
    <root>
      <forsale>
        <item name="weight" value="32.5698" />
      </forsale>
    </root>
  </mx:XML>
  <mx:NumberFormatter id="fmtNumber" precision="2"/>
  <mx:Label
     text="Weight {fmtNumber.format(myData.forsale.item.@value)}lbs"/>
</mx:Application>
```

原始数据

定义NumberFormatter

对原始数据进行格式化

Weight 32.56lbs

图 7-1　使用 NumberFormatter 格式化重量值

结果只是把原始数值(32.5698)截短了，但实际上我们可以使用表 7-3 所示的 rounding 属性舍入返回的值。由于代码清单 7-1 没有设置 rounding 属性，默认没有舍入。要执行舍入，需要如下重写该属性的默认值。

```
<mx:NumberFormatter id="fmtNumber" precision="2" rounding="up"/>
```

NumberFormatter 还有另外两个专有的因上下文而异的错误消息(比通用的 Formatter 多一些)，如表 7-4 所示。

表 7-4 NumberFormatter 的错误消息

错误消息	说 明
Invalid Value	输入值与格式化程序的期望值不相符。例如，提供了非数字字符
Invalid Format	提供的参数组合没有意义。原因可能是属性设置有误，例如为舍入指定一个无效的值

这些错误消息出现在 NumberFormatter 的 error 属性中，在这种情况下应该是 fmtNumber.error。

了解了 NumberFormatter 的基本用法后，接下来读者可能想知道怎样用它处理美元(或其他任何货币单位)。虽然表面上看可以使用 NumberFormatter 处理货币单位，但实际上有一个专门的格式化程序完成这一任务，这就是 NumberFormatter 的同代近亲 CurrencyFormatter。

7.1.3 CurrencyFormatter

对于给定的原始数值，CurrencyFormatter 会插入预定义的货币符号，并在适当位置插入千分位分隔符，将数据组织的公认的货币表示形式。

提示：

ColdFusion 中提供了类似的 DollarFormat()方法。

CurrencyFormatter 支持 NumberFormatter 支持的所有属性，除此之外，还包括另外两个特定于货币的属性，如表 7-5 所示。

表 7-5 CurrencyFormatter 的两个附加属性

属 性	类 型	说 明
alignSymbol	String	指定在输出字符串的什么位置放置货币符号。有效值为 left 和 right。默认值为 left
currencySymbol	String	指定输出字符串中使用的货币符号。默认为美元符号($)

CurrencyFormatter 的用法与 NumberFormatter 基本相同(代码清单 7-2)，只不过它能够自动为输出结果添加货币符号，如图 7-2 所示。

代码清单7-2 使用CurrencyFormatter显示价格

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:XML id="myData">
```

```
    <root>
      <forsale>
        <item name="Laptop" price="599.99" />
      </forsale>
    </root>
  </mx:XML>
  <mx:CurrencyFormatter id="fmtCurrency" precision="2"/>
  <mx:Label text="Laptop Price
    {fmtCurrency.format(myData.forsale.item.@price)}"/>
</mx:Application>
```

CurrencyFormatter 还与 NumberFormatter 具有相同的错误消息，如表 7-4 所示。

除了数值，格式化日期也是几乎所有应用程序都存在的需求。接下来我们介绍使用 DateFormatter 格式化日期。

7.1.4 DateFormatter

DateFormatter 大概是使用最频繁的一种格式化程序。顾名思义，它的作用是控制输出中日期的显示方式。表 7-6 所示为这个格式化程序支持的重要属性(其他属性继承自 Formatter 类)。

表 7-6　DateFormatter 的附加属性

属　性	类　型	说　明
formatString	String	应用于日期的模式掩码(如 YYYY/MM/DD)

仅此而已。这个唯一的属性使用模式掩码(*pattern mask*)，模式掩码是一系列字符，用于告知格式化程序输出日期各个部分的次序，以及各个部分间使用的分隔符。模式掩码(formatString)有很多种组合，表 7-7 所示为可以在模式掩码中使用的字符。

表 7-7　在 DateFormatter 的 formatString 属性中使用的格式化字符

字　符	说明/示例
Y	年： YY——两位数的年。如 08。 YYYY——四位数的年。如 2008
M	月： M——一位最小数的月。如 8 和 12。 MM——两位最小数的月。如 08 和 12。 MMM——三位数的月。如 Aug 和 Dec。 MMMM——完整的月名。如 August 和 December
D	日： D——一位最小数的日。如 6 和 22。 DD——两位最小数的日。如 06 和 22

(续表)

字　符	说明/示例
E	周几(周日为 0)： E——一位数。如 0。 EE：填充 0。如 00。 EEE：三位字符的名称。如 Sun。 EEEE：完整的名称。如 Sunday
A	AM 或 PM(上午或下午)
J	24 小时格式中的小时，0 表示第一小时。 J：一位最小数。如 4 和 18。 JJ：两位最小数。如 04 和 18
H	24 小时格式中的小时，1 表示第一小时。 H：一位最小数。如 4 和 12。 HH：两位最小数。如 04 和 12
K	12 小时格式中的小时，0 表示第一小时。 K：一位最小数。如 0 和 11。 KK：两位最小数。如 00 和 11
L	12 小时格式中的小时，1 表示第一小时。 L：一位最小数。如 1 和 12。 LL：两位最小数。如 01 和 12
N	分： N：一位最小数。如 5 和 55。 NN：两位最小数。例如：05 和 55
S	秒。 S：一位最小数。如 2 和 56。 SS：两位最小数。如 02 和 56
其他格式化字符	其他任何 DateFormatter 属性中嵌入的字符，你可以指定其他任何要在显示结果中使用的字符。常见的字符有逗号、短划线、斜杠和空格

与其他格式化程序一样，DateFormatter 也定义了错误消息，如表 7-8 所示。

表 7-8　DateFormatter 的错误消息

错误消息	说　明
Invalid Value	输入值与 DateFormatter 的期望值不相符。DateFormatter 接收 Date 对象或包含日期的 String 对象
Invalid Format	formatString 中包含不兼容的格式

DateFormatter 的 format()函数能够接收字符串格式的日期值，也能够接收实际日期对象，如代码清单 7-3 所示。

代码清单7-3　DateFormatter可以格式化日期字符串和日期对象

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      [Bindable]
      public var sDate:String = "12/01/08 12:42";
      [Bindable]
      public var dDate:Date = new Date("12/01/08 12:42");
    ]]>
  </mx:Script>
  <mx:DateFormatter id="fmtDate" formatString="MM.DD.YY"/>

  <mx:Label text="Formatting the Date as a String:
          {fmtDate.format(sDate)}" />
  <mx:Label text="Formatting the Date as a Date object:
          {fmtDate.format(dDate)}" />
</mx:Application>
```

要进行格式化的字符串格式的日期

要进行格式化的Date对象

在两种情况中使用的DateFormatter

在这个示例中，格式化程序并没有与任何特定变量绑定。创建一个实例，并根据需要重用它，以便既可以格式化 String 对象，也可以格式化 Date 对象。如图 7-3 所示。

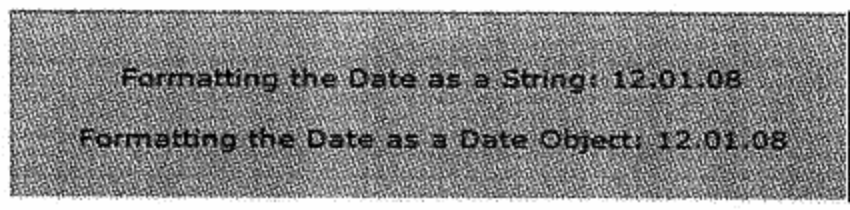


图 7-3　重用单个 DateFormatter 格式化 String 和 Date 对象

结合使用 DateFormatter 和 XML

代码清单 7-3 将变量用作要进行格式化的数据，但将相同的方法应用于 XML 数据并不会返回预期的结果，如代码清单 7-4 所示。

代码清单7-4　读者可能认为这样做没问题，实则不然

```
<mx:XML id="myData">
  <root>
    <info>
```

```
    <item lastvisit="12/01/08 12:42"/>
   </info>
  </root>
</mx:XML>

<mx:DateFormatter id="fmtDate" formatString="MM.DD.YY"/>
<mx:Label text="{fmtDate.format(new String(myData.info.item.@lastvisit))}"
   />
```

这个示例无效的原因是，Flex 会在内部把 XML 转换为一组高级对象——既不是 Date 也不是 String(而 format()函数只接收这两种对象)。

不过，解决这个问题的方法也很简单，就是把 XML 变量放到 String 中。

```
<mx:Label text="Last {fmtDate.format(String(myData.info.item.@lastvisit))}"
   />
```

这种技术叫做强制类型转换(*casting*)，即把一种数据类型转换为另一种数据类型。

DateFormatter 在 Flex 应用程序中的使用频率是很高的。使用 DateFormatter 要记住的关键一点是它与其他格式程序稍有不同，它不仅接收文本形式的 String 对象，也接收 Date 对象。

DateFormatter 的 formatString 属性也有特殊之处，通过该属性可以有效控制输出日期的格式。而这个属性也被 PhoneFormatter 所共享，下面我们介绍 PhoneFormatter。

7.1.5 PhoneFormatter

PhoneFormatter 特别适用于以普通数字形式保存(例如在数据库中)，但要以常规可识别格式显示的电话号码。

PhoneFormatter 的行为与 DateFormatter 非常相似，主要表现为可以通过 formatString 属性指定显示电话号码的格式，而它的 format()函数也接收两种类型的对象——String 或 Number。

表 7-9 所示为 PhoneFormatter 支持一组属性。

表 7-9 PhoneFormatter 的属性

属　性	类　型	说　明
formatString	String	应用于电话号码的格式
areaCode	Number	在输入 10 位电话号码的情况下，用来指定一个区号。如果把这个属性设置为-1，表示忽略此项。默认值为-1
areaCodeFormat	String	定义如何表示区号。默认值为(###)
validPatternChars	String	可以在 formatString 中使用的一组字符。磅字符(#)用于表示一个数字位。默认字符为+()#-

PhoneFormatter 属性为格式化电话号码提供了明显的灵活性。使用其默认字符及明确自定义的字符，几乎可以创造出任何输出模式。表 7-10 所示为一些示例。

表 7-10　可以在 formatString 属性中使用的模式示例

模　式	输入数据	输出结果
###-###-####	2099109872	209-910-9872
(###)-###-####	2099109872	(209)-910-9872
###.###.####	2099109872	209.910.9872
#-###-###-####	12099109872	1-209-910-9872
+##-########	6569362267	+65-69362267

代码清单 7-5 所示为一个使用 PhoneFormatter 把原始电话号码格式化为常规格式的例子。图 7-4 所示为格式化后的结果。

代码清单7-5　使用PhoneFormatter格式化原始、无格式的电话号码

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:XML id="myData">
    <root>
      <contactlist>
        <item name="contact" phone="2016679872" />
      </contactlist>
    </root>
  </mx:XML>
  <mx:PhoneFormatter id="fmtNumber" formatString="(###) ###-####"/>
  <mx:Label text="Contact Phone:
          {fmtNumber.format(myData.contactlist.item.@phone)}"/>
</mx:Application>
```

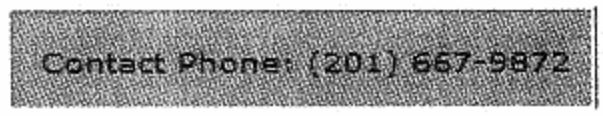


图 7-4　PhoneFormatter 帮助我们生成一个直观的电话号码

如果格式化期间出现了问题，别忘了通过 PhoneFormatter 的 error 属性来查看是否出现了错误。如果出现了错误，则错误将为表 7-11 中的一种。

表 7-11　PhoneFormatter 的错误消息

错误消息	说　明
Invalid Value	PhoneFormatter 接收 String 或 Number，其他类型的数据输入都将导致此错误。在输入 String 的情况下，输入字符数必须与 formatString 中的字符数相符，否则会导致此错误
Invalid Format	表示 formatString 中的错误。可能的原因是输入字符无效，或者区号的位数不是 3 位

使用 PhoneFormatter 最关键的一条规则是，输入数据中的数字个数必须与 formatString 中的磅字符(#)数量一致。

7.1.6 ZipCodeFormatter

ZipCodeFormatter 主要用于格式化美国邮政编码和加拿大邮递区号。虽然格式化邮政编码的机会不多，但在处理 5+4 格式的邮政编码以及要按 3 段形式输出加拿大邮递区号时，使用 ZipCodeFormatter 会带来一些便利。

表 7-12 所示为 ZipCodeFormatter 的属性。

表 7-12 ZipCodeFormatter 的主要属性

属 性	类 型	说 明
formatString	String	应用于美国邮政编码及加拿大邮递区号的模式掩码。此掩码必须是美国邮政编码及加拿大邮递区号的已知模式。默认为#####

对于美国邮政编码，必须使用 5 或 9 个磅字符(#)。对于加拿大邮递区号，则必须使用 6 个磅字符。

如果定义的格式是 9 位，但输入只有 5 位，则 5+4 格式邮政编码的其余 4 位会以“0000”显示。相反，如果输入包含 9 位，而 ZipCodeFormatter 只想要 5 位，则 5+4 格式邮政编码的其余 4 位被忽略。

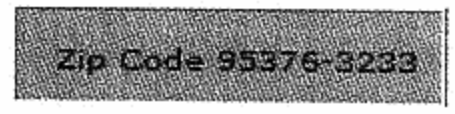


图 7-5 使用 ZipCodeFormatter 格式化生成的美国邮政编码

代码清单 7-6 使用我们熟悉的 formatString，以 5+4 格式输出美国邮政编码。运行该示例后将看到与图 7-5 所示类似的结果。

代码清单7-6 使用ZipCodeFormatter输出5+4格式的美国邮政编码

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:XML id="myData">
    <root>
      <contacts>
        <item name="John Doe" zipcode="953763233"/>
      </contacts>
    </root>
  </mx:XML>
  <mx:ZipCodeFormatter id="fmtZip" formatString="#####-####"/>
  <mx:Label text="Zip Code
          {fmtZip.format(myData.contacts.item.@zipcode)}"/>
</mx:Application>
```

格式化期间出现的错误基本上来源于输入的字符数量与 ZipCodeFormatter 要求的数量不一致。表 7-13 所示为可能的错误及消息说明。

表 7-13 ZipCodeFormatter 的错误消息

错误消息	说 明
Invalid Value	在输入的字符数量与 ZipCodeFormatter 要求的数量不相符时，会导致此错误。美国邮政编码中只允许使用数字，而加拿大邮递区号只能是 6 位字符
Invalid Format	表示 formatString 中的错误。可能的原因是输入字符无效，或者模式本身有问题

在 ZipCodeFormatter 及前面介绍的格式化程序都不能满足需要的情况下，可以考虑另

一个通用的 SwitchSymbolFormatter。

7.1.7 SwitchSymbolFormatter

SwitchSymbolFormatter 是一个通用的、包罗一切的格式化程序，用于处理 Flex 内置格式化程序不适用的数据。SwitchSymbolFormatter 与其他格式化程序的区别在于，它对要处理的数据类型的上下文信息缺乏了解，因此其用途有一定局限。

在高级 Flex 开发中，可以使用 SwitchSymbolFormatter 创建自定义格式化程序(通过扩展它并添加具体的逻辑，有关自定义组件的详细信息参见第 17 章)。因此，SwitchSymbolFormatter 可以说是自定义格式化程序的基础。

由于这个原因，SwitchSymbolFormatter 没有相应的 MXML 组件；但为了说明问题，可以使用其 ActionScript 版本。表 7-14 所示为可用函数的列表。

表 7-14　SwitchSymbolFormatter 的函数

函　　数	说　　明
SwitchSymbolFormatter	构造函数(默认函数)。它接收一个表示用作数字占位符的字符值。默认为磅字符(#)
formatValue	接收一个格式化字符串和一个源对象的引用(该对象包含要格式化的文本)的组合。其中，格式化字符串可以是需要的任何内容，但源对象中的每个数字都必须有对应的磅字符(#)(除非你指定不同的数字标识符)

如果没有示例，很难理解 SwitchSymbolFormatter 的工作原理。下面我们通过代码清单 7-7 展示一个使用 SwitchSymbolFormatter 处理并格式化原始数据的例子。

代码清单7-7　格式化并输出由短划线分隔的8位数

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
  <![CDATA[
    import mx.formatters.SwitchSymbolFormatter;
    public var fmtSymbol:SwitchSymbolFormatter =
                              new SwitchSymbolFormatter("#");
    public function formatMe(rawData:String):String
    {
      return fmtSymbol.formatValue("####-####",rawData);
    }
  ]]>
  </mx:Script>
  <mx:XML id="myData">
    <root>
      <workorders>
          <item name="Fix something" id="99818382" />
      </workorders>
    </root>
  </mx:XML>
  <mx:Label text="Work Order: {formatMe(myData.workorders.item.@id)}"/>
</mx:Application>
```

需要导入相关类

创建格式化程序的实例

根据需要格式化原始数据

原始数据

调用格式化函数来格式化数据

作为一个高级格式化程序，SwitchSymbolFormatter 没有特定的错误消息。程序清单 7-7 的结果如图 7-6 所示。

图 7-6 使用 SwitchSymbolFormatter 格式化生成的 8 位数

下面我们介绍利用前述格式化程序的不同方式。

7.2 实时格式化

实时格式化并不是一种官方说法，但是本章到目前为止所有示例中采用的一种方式(Switch SymbolFormatter 除外)，即格式化程序的 format()函数都是被动态调用的。

例如，以下代码以实时方式格式化电话号码。

```
<mx:PhoneFormatter id="fmtPhone" formatString="###-###-####"/>
<mx:Label text="Contact Phone: {fmtNumber.format("1112223333")}"/>
```

这没有什么特别，只不过易于使用而且能够完成工作；但是，局限性在于其中未嵌入任何种类的具体业务逻辑。如果想在格式化过程中加入自定义逻辑，恐怕就得求助于脚本式格式化了。

7.3 脚本式格式化

无论把这种方式称为脚本式格式化，还是叫做动态格式化，指的都是使用 ActionScript 更精细化地处理输入。

虽然实现这种格式化的手段有很多，但它们共同的特点是，都要通过函数来处理格式化。

7.3.1 结合使用函数与格式化程序组件

这种方式的特点是传入要格式化的值，然后从函数中接收格式化后的结果。而函数内部要使用前面已经创建的 MXML 格式化程序组件。

由于格式化程序并没有与信息源紧密耦合，也不了解信息源，因此会更容易重用函数，更加实用；同时使用起来也很简单。

代码清单 7-8 显示为需要进行格式化时调用了函数。函数中包含动态修改 formatString 的一些自定义逻辑。

代码清单7-8　使用ActionScript和静态格式化程序组件在需要时进行格式化

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      [Bindable]
      public var rawPhone:String = "2223333";

      public function formatThis(plainText:String):String
      {
//We can add some extra business logic if we want.
//For example, changing the formatString on the fly
//depending on the size of the text.
        if(plainText.length == 7)
          fmtPhone.formatString = "###-####";
        else
          fmtPhone.formatString = "###-###-####";

        return(fmtPhone.format(plainText));
      }
    ]]>
  </mx:Script>
  <mx:PhoneFormatter id="fmtPhone"/>
  <mx:Label text="{formatThis(rawPhone)}"/>
</mx:Application>
```

原始数据

添加附加的业务逻辑

❶

从需要格式化内容的代码角度看，这种方法与实时格式化方法几乎同样简单。但它在必要时调用 formatThis()函数❶。你可以将可重用的逻辑隔离到函数中。

这种方法的一个局限性在于，必须从一开始就知道应该使用哪种格式化程序。虽然可以创建每类格式化程序的 MXML 实例并进行相应的访问，但更简便的方法是使用 ActionScript 按照需要创建格式化程序的实例。

7.3.2　结合使用函数与格式化程序类

每个格式化程序都有一个对应的 ActionScript 类。这意味着要使用格式化程序不一定依赖于 MXML 组件。这样一来，创建完全独立的、不依赖于用户定义 MXML 格式化程序组件的函数变得更方便(代码清单 7-9)。

代码清单7-9　使用ActionScript动态创建PhoneFormatter实例

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
        //import the formatters we want to use
      import mx.formatters.*

      [Bindable]
      public var rawPhone:String = "2223333";
```

```
    public function formatThis(plainText:String):String
    {
      var fmtPhone:PhoneFormatter = new PhoneFormatter();
      if(plainText.length == 7)
        fmtPhone.formatString = "###-####";
      else
        fmtPhone.formatString = "###-###-####";

      return(fmtPhone.format(plainText));
    }
]]>
  </mx:Script>
  <mx:Label text="{formatThis(rawPhone)}"/>
</mx:Application>
```

在前面的代码清单中，每次函数调用都将动态创建一个 PhoneFormatter 对象。这个新的格式化程序会在函数执行完毕后被删除。显然，这种方法的额外开销会大一些。因此，如果知道要经常用到哪种格式化程序，那么最好在函数外部将其声明一次，如下所示：

```
import mx.formatters.*
public var fmtPhone:PhoneFormatter = new PhoneFormatter();
```

这种做法的缺点是无论是否用到格式化程序，它都会存在。

为提高模块化程度，可以继续增加可重用性。例如，可以在传入格式化 String 的同时，将 formatString 作为参数传入(代码清单 7-10)。

代码清单7-10 接收格式化模式的自定义格式函数

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      import mx.formatters.*
      public var fmtZip:ZipCodeFormatter = new ZipCodeFormatter();

      public function formatThis(plainText:String,formatString:String):String
      {

        fmtZip.formatString = formatString;
        return(fmtZip.format(plainText));
      }
    ]]>
  </mx:Script>
  <mx:Label text="U.S Zip Code: {formatThis('95376','#####')}"/>
  <mx:Label text="Canadian Postal Code: {formatThis('K1E2X5','### ###')}"/>
</mx:Application>
```

如图 7-7 所示，使用 ActionScript 具有更大的灵活性。而且还能够在此基础上继续改进函数。例如，可以根据数据类型决定调用哪种格式化程序——相信读者应该明白怎么做了。

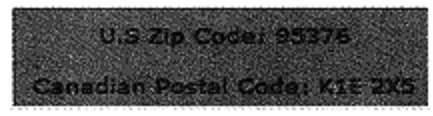


图 7-7 通过 ActionScript，使用单个函数来创建 ZipCodeFormatter，其中，格式化 String 作为参数传递

本章要讨论的最后一个问题是如何处理使用格式化程序时遇到的错误。

7.4 处理格式化错误

使用脚本式方法还有另外一个好处，就是可以处理格式化过程中发生的错误。前面曾经提到，格式化程序都有一个 error 属性。可以将这个属性用作指示发生了问题的提示。

如果格式化过程顺利，error 属性的值为空；如果格式化遇到问题，error 属性中会保存有相应的错误代码(Invalid value 或 Invalid format)。通过代码清单 7-11 可知，在发生格式化错误时，格式化的值也将被设置为空。

代码清单7-11　使用error属性确定格式化过程中出现的问题

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
 <mx:Script>
  <![CDATA[
   import mx.formatters.*
   public function formatThis(plainText:String):String
   {
    var fmtPhone:PhoneFormatter = new PhoneFormatter();
    var formattedString = fmtPhone.format(plainText);
    if(fmtPhone.error == "Invalid value")
    {
     mx.controls.Alert.show('The value you entered is invalid.');
    }
    else if(fmtPhone.error == "Invalid format")
    {
     mx.controls.Alert.show('The value you entered is invalid.');
    }
    return(formattedString);
   }
  ]]>
 </mx:Script>
 <mx:Label text="{formatThis('222')}"/>
</mx:Application>
```

检查错误

如果发生问题则返回空字符串

在使用格式化程序时，最好在调用 format()函数后立即检查 error 属性。这样，就有机会及时处理格式化过程中出现的问题。

7.5 小结

格式化程序与验证器类似，而且常常会同时使用它们先验证数据，再针对显示需要格式化输出。

Flex 内置了许多可以简化格式化任务的格式化程序。这些格式化程序专门用于处理通常在后端系统(如数据库)中保存的不方便用户认读的数据。

使用格式化程序的方式有很多种，包含依赖静态格式化程序的实时格式化，乃至基于 ActionScript、能够灵活加入自定义逻辑的脚本式格式化。

别忘了检查 error 属性，特别是在不能保证数据格式化一定顺利的情况下。

第 8 章介绍列表。列表是 Flex 应用程序中一个主要数据类型。一般来说，使用列表是为了显示取自后端系统的数据。而且在使用列表显示数据的过程中，还会用到格式化程序。

第 8 章 DataGrid、列表和树

本章内容：

- 使用基于列表的组件显示普通数据
- 使用基于列表的组件显示层次化数据
- 通过基于列表的组件处理用户交互
- 从基于列表的组件中检索选定项

基于列表(List)的组件是 RIA 的重要组成部分，有人认为它们是 Flex 应用程序中广为使用的组件。

List 不仅能够胜任接收和显示数据的大批任务，而且还能为用户提供诸多便利，如列排序和自动换行。

8.1 基于列表的一系列组件

第 2 章曾经介绍过，Flex 是一种面向对象的语言和框架。至少要理解对象间的关系，这一点非常重要。因为派生自另一个元素的元素会继承父元素的所有特性。

提到 Flex 中的列表(即 List 组件，而非泛泛的列表)，ListBase 与 Advanced- ListBase 是两个重要对象。虽然不会直接使用这两个对象，但它们是所有子 List 对象的基础。

ListBase 和 AdvancedListBase 为宽度、高度、要显示的行数以及基本信息显示等属性提供支持。而且它们也支持事件触发器的操作(例如选定某个项)。图 8-1 所示为所有列表组件之间的关系。

如图 8-1 所示，可以看出 List 组件能够做什么，当然也包括继承自 List 的 Tree 组件能够做什么。它们都派生自公共的父对象，因此共享一组核心属性。

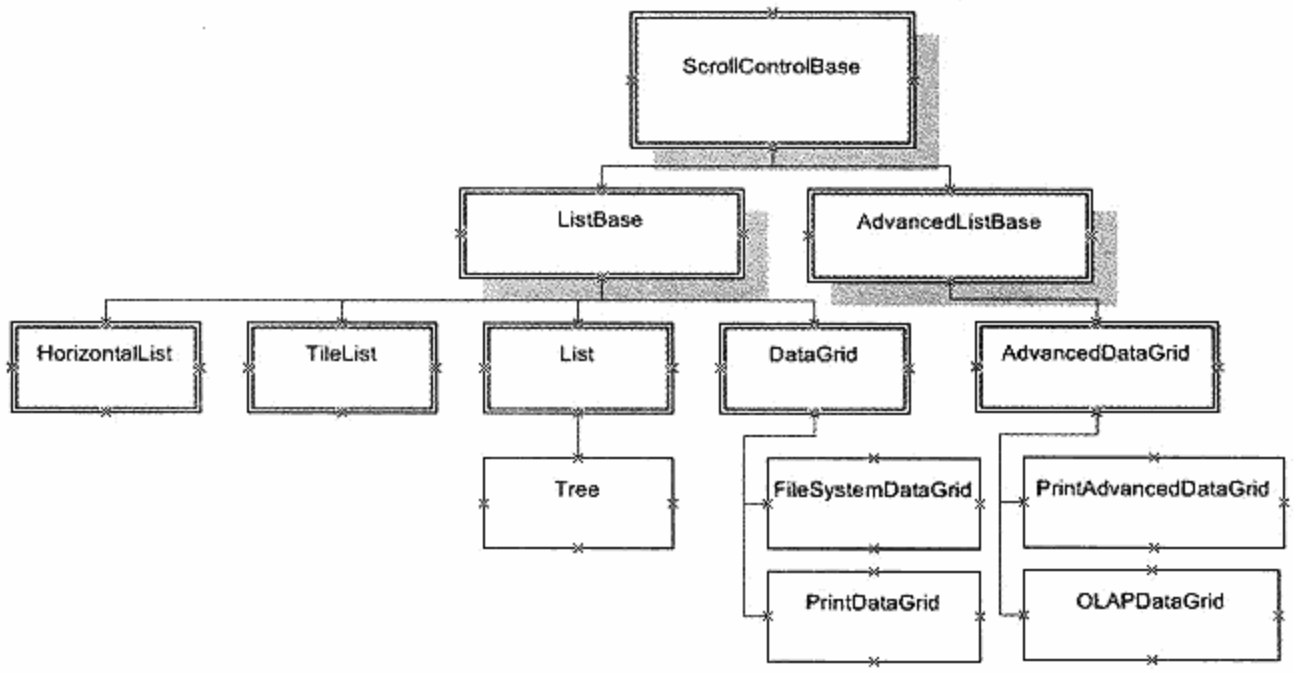


图 8-1 基于列表的一系列组件及相互关系

8.1.1 ListBase 和 AdvancedListBase 的属性

由于所有 List 或 Tree 组件都以某种形式或构成方式派生自 ListBase 和 AdvancedListBase，因此它们至少都支持这两个对象中定义的属性。这些属性的数量非常多，我们把其中一部分放在后面介绍。表 8-1 所示为列表的基本属性。

表 8-1 所有基于列表的组件都支持的公共属性

属 性	类 型	说 明
columnCount	Number	要显示的列数
columnWidth	Number	列宽
dataProvider	Object	包含要显示数据的对象
iconField	String	可选的字段名称，包含指向图标的引用。默认情况下，列表会查找名为 icon 的字段，如果可以找到就会使用它
iconFunction	Function	可选的函数名称，可以针对每个图标运行。用于设置图标的格式
labelField	String	字段名称，包含应该被显示在列中的值
labelFunction	Function	可选的函数名称，可以针对每条记录调用。用于确定标签的格式及显示方式
lockedColumnCount	Number	用户水平滚动视图时，始终在视图中可见的列数。由 DataGrid 和 AdvancedDataGrid 使用
lockedRowCount	Number	与 lockedColumnCount 相似，但作用于行。用于指定在用户垂直滚动视图时，始终显示在表格顶部的行数
rowCount	Number	要显示的行数
rowHeight	Number	行高(像素)
selectable	Boolean	表示是否允许用户选择项的值(true 或 false)。默认值为 true
selectedIndex	Number	如果组件允许选择，则为已经选中的行(如果有)

(续表)

属　性	类　型	说　明
selectedIndices	Array	在允许多选的情况下，用于保存所有选定索引的数组
selectedItem	Object	当前选定项的链接。通过它可以访问与选定行有关的所有值
selectedItems	Array	在允许多选的情况下，用于保存使用的 selectedItem 的数组
variableRowHeight	Boolean	此值表示是否允许每一行自动调整行高以适应其中内容的值(true 或 false)。如果允许自动换行，则需设置此属性为 true。默认值为 false
wordWrap	Boolean	表示是否允许自动换行的值(true 或 false)。默认值为 false

属性虽然不少，但考虑到基于列表的组件是 Flex 应用程序中广泛使用的可视组件，因此这些组件必须拥有足够多、可配置及灵活的选项，才能有效完成肩负的任务。

我们知道，Flex 是事件驱动的开发环境，因此除了属性还必须了解可以使用哪些事件。

8.1.2　ListBase 事件

ListBase 在为其子对象提供一组属性的同时，也提供了应该支持的事件(见表 8-2)。你可以指定这些事件发生时要运行的函数名称或者任意 ActionScript 代码。

表 8-2　基于列表的组件公用的事件

属　性	类　型	说　明
change	Event	在用户选择另一行时触发
dataChange	Event	在数据被修改时触发
itemClick	Event	在用户单击一列时触发
itemDoubleClick	Event	在用户双击一列时触发
itemRollOut	Event	在用户鼠标移出一项时触发
itemRollOver	Event	在用户鼠标悬停在一项上时触发

本章后面将解释如何使用这些事件；本质上，这些事件的作用机制仍然是标准的事件触发器和事件处理程序的模式。

只有基于列表的组件而没有数据实在太空洞。下面我们看一看数据都有哪些来源，以及将数据填充到组件中的方式。

8.2　dataProvider 与集合

我们曾在第 5 章接触过 dataProvider。这里对它做深入介绍。

大多数组件显示的数据项都来自 dataProvider，这些组件也称为*数据驱动控件*。而组件使用数据的实质方式，无非就是将一个保存有要显示的数据的变量名指定给组件。

有许多方式可以实现这一点。先来看如何填充 dataProvider 信息。

8.2.1 填充 dataProvider

变量可能是低级(lower-level)对象，包括数组、XML 对象或 XMLList 对象。但用于填充 dataProvider 的对象是一个集合。如果不选用集合，组件会自动将该低级对象包装为集合。

新手可以把集合想象成一种高性能数组，它具有内置的排序及筛选能力。此外，集合还为具有 dataProvider 的对象提供通用且即插即用的接口(方法是抽象出数据交互过程的复杂性)。

集合还有另一个重要优势，即在它的状态变化时，会自动向使用它的组件发送变化通知，以便这些组件刷新显示的数据。为此，集合会以广播方式发布表示变化的事件。任何侦听该事件的组件都可以据此做出响应。

低级对象不支持这种自动变化通知特性；如果被修改的是低级对象，则使用它们的任何组件(例如，显示它们数据的组件)都不会得知变化的发生。

8.2.2 集合的类型

不同类型的集合具有一组相同的基本功能，而且每种类型都是在这些基本功能之上，加入特定环境下专有功能扩展而成。

- ArrayCollection：一种主要的集合类型。ArrayCollection 基于 Array 构建。
- XMLListCollection：在包装 XML 及 XMLList 对象的基础上添加标准集合功能而成。
- GroupingCollection：由 AdvancedDataGrid 显式使用，用于分组数据。

这些集合可以用来驱动支持 dataProvider 的任何组件，而基于列表的组件是主要的使用者。

8.2.3 集合的使用者

基于列表的组件是集合的主要使用者，但集合的使用者并不限于列表。表 8-3 所示为可以通过 dataProvider 驱动的其他组件。

表 8-3 可以针对 dataProvider 使用集合的组件

AdvancedDataGrid	Menu
ButtonBar	MenuBar
Charting components including	OLAPDataGrid
Legends	PopUpMenuButton
ColorPicker	Repeater
ComboBox	TabBar
DataGrid	TileList
Datefield	ToggleButtonBar
HorizontalList	Tree
LinkBar	
List	

现在，我们已经了解了集合的优势，以及这些高级的类数组(array-like)对象可以用作许多数据驱动的组件的 dataProvider。接下来介绍如何使用集合。

8.3 集合的初始化

有两种初始化集合的方法。第一种是使用关联标签的 MXML 方法，如代码清单 8-1 所示。

代码清单8-1　通过在MXML标签中嵌入数据的方式初始化ArrayCollection

```
<mx:ArrayCollection id="myAC">
  <mx:Object label="Jon Hirschi" data="jhirschi"/>
  <mx:Object label="Tariq Ahmed" data="tahmed"/>
  <mx:Object label="Frank Krul" data="fkrul"/>
</mx:ArrayCollection>
```

第二种方法是完全使用 ActionScript 初始化集合，首先导入集合类，然后声明实例变量，如代码清单 8-2 所示。

代码清单8-2　完全使用ActionScript脚本初始化ArrayCollection

```
<mx:Script>
  <![CDATA[
    import mx.collections.ArrayCollection;
    public var myAC:ArrayCollection  = new ArrayCollection([
      {label:"Jon Hirschi", data:"jhirschi"},
      {label:"Tariq Ahmed", data:"tahmed"},
      {label:"Frank Krul", data:"fkrul"}
    ]);
  ]]>
</mx:Script>
```

导入类，以便Flex了解引用的ArrayCollection

注意：

也许有读者会问，为什么 ActionScript 方法要导入类，而 MXML 方法不需要导入类呢？请注意<mx:Application>标签并注意 xmlns:mx="..."语法。这是 MXML 实现与在 ActionScript 中导入类同样功能的方式。有关这个问题的详细内容将在第 17 章介绍。

下面是对 ActionScript 方法的简单说明。

- 首先将变量(myAC)分配给 ArrayCollection 的新实例。
- ArrayCollection 的构造函数(即创建对象时执行的函数)它接收一个用于初始填充的信息数组。
- 其中的方括号[]是创建数组的简化表示方式(参见第 3 章)，也是构造函数允许的语法。
- 花括号{}是创建对象的简化表示方式。如代码清单 8-2 所示，创建的是对象数组，而且每个对象都包含两个字段(label 和 data)。
- 可以包含任意数量的字段。

注意：

可以为字段指定任何名称，但默认情况下，使用 dataProvider 的组件假定 label 字段中包含要显示的值，而 data 字段中包含数据。为了方便起见，可以利用这种默认约定；但是，如果读者为字段定义了其他名称，那么让组件识别应该使用哪些字段名称是小事一桩。

在这个示例中，数据被硬编码进初始化过程。而在大多数实际情况下，都需要先初始化集合，然后再动态向其中填充数据。

8.4 集合的填充

一般来说，需要使用从某些中间层应用服务器中取得的数据(例如，从 ColdFusion 返回的查询数据)来动态填充基于列表的组件。

相应过程的详细内容将在第 14 章讨论。简单地讲，可以通过 HTTP 连接检索 XML 数据，或者通过调用 WebService 函数，使用 RemoteObject(Flex 的二进制数据传输机制)加载数据。例如，从 ColdFusion、LCDS 和 AMFPHP(在 PHP 中)等。在数据返回后，需要将数据转换为基于列表的组件正在侦听的集合，进而通过组件把数据显示出来。

为了简单起见，我们在此使用静态方式填充集合。接下来的几节将展示几种相关技术。

8.4.1 List

List 是用于显示列表信息的一个轻量级组件。它能够以单列方式显示信息。

1. 调用列表

创建及填充 List 组件的方式有很多种。我们先来看代码清单 8-3 所示的最简单方式。

代码清单8-3 使用List组件显示一列姓名

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:List id="myFriends">
    <mx:String>Tom Ortega</mx:String>
    <mx:String>Ryan Stewart</mx:String>
    <mx:String>Abdul Qabiz</mx:String>
  </mx:List>
</mx:Application>
```

编译并运行这个应用程序后，将看到与图 8-2 所示类似的列表组件。图中包含以列表形式显示的姓名。注意这个列表没有列标题，但鼠标移动到任何项上都会导致背景颜色自动变化。

非常简单，不是吗？这个示例使用一种简便方式填充 List 的 dataProvider；应该只在硬编码显示效果的情况下使用这种方法(对于验证概念还是非常合适的)。

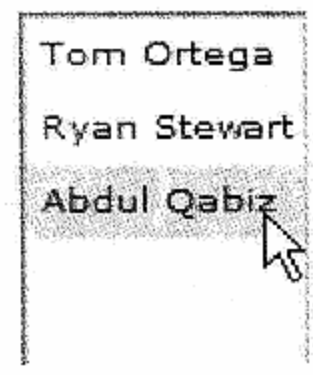


图 8-2 通过 List 组件显示人名

实际上，不应该让数据与代码耦合得如此紧密。要将它们分离开，必须告诉 List 组件哪里能找到信息，而不是了解填充信息的方式。这样一来，如果需要更改集合的管理方式，List 不会受影响。

在只使用 MXML 的情况下，可以将数据分离到一个 ArrayCollection 中，然后使用它填充 List 的 dataProvider，如代码清单 8-4 所示。

代码清单8-4　通过ArrayCollection驱动List组件的显示

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:ArrayCollection id="myAC">
    <mx:Object label="Tom Ortega"/>
    <mx:Object label="Ryan Stewart"/>
    <mx:Object label="Abdul Qabiz"/>
  </mx:ArrayCollection>
  <mx:List id="myFriends" dataProvider="{myAC}"/>
</mx:Application>
```

通过绑定把dataProvider链接到ArrayCollection

为什么这个示例使用了一个<mx:Object>数组，而前面的 List 组件中使用的是<mx:String>呢？可以使用下面的方式。

```
<mx:String>Tom Ortega</mx:String>
```

但 String 只能保存一个值，而实际的应用程序经常需要使用多个字段。

另一种方式是如代码清单 8-5 所示的那样，使用 ActionScript 类 ArrayCollection，用一组对象进行填充。

代码清单8-5　使用ActionScript填充List

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
  <![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var myAC:ArrayCollection  = new ArrayCollection([
                              {label:"Jon Hirschi"},
                              {label:"Tariq Ahmed"},
                              {label:"Frank Krul"}
                              ]);
  ]]>
  </mx:Script>
  <mx:List id="myFriends" dataProvider="{myAC}"/>
</mx:Application>
```

这种方式也很直观。但是，除非数据始终包含 label 字段，将需要一种更特别的方式来识别 label。

2. 指定标签

在实际的项目开发中，数据可能代表所在数据库中的列。对于返回的所有列，必须告诉 List 应该显示哪个字段；这需要通过指定 labelField 实现。

代码清单 8-6 在前一个示例的基础上，通过设置 labelField 属性告诉 List 应该显示的列。

代码清单8-6 使用labelField告诉List显示的列

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
 <![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var myAC:ArrayCollection  = new ArrayCollection([
       {name:"Jon Hirschi", email:"j_hirschi@domain.com",
                 url:"http://www.flexablecoder.com"},
       {name:"Tariq Ahmed", email:"t_ahmed@domain.com",
                 url:"http://www.dopejam.com"},
       {name:"Frank Krul", email:"f_krul@domain.com", url:""}
  ]);
 ]]>
</mx:Script>
<mx:List id="myFriends" dataProvider="{myAC}" labelField="name"/>
</mx:Application>
```

至此，读者应该掌握如何创建 List 组件，如何使用数据集合对其进行填充，以及如何指定要显示集合中的哪一列。不过，List 组件是在垂直方向上显示列表数据，如果想要水平地显示列表数据，该怎么办呢？

8.4.2 HorizontalList

List 组件默认的布局方向是垂直的(内容从上至下显示)。如果读者是喜欢表达精确的人，可以把这种列表称为*垂直列表*。幸运的是，与垂直列表相对，还有一个水平列表，即 HorizontalList。HorizontalList 组件与默认的 List 组件的功能相似，只不过它的布局方向是水平的。

以代码清单 8-6 为基础，把 List 修改为 HorizontalList(代码清单 8-7)，然后看一看有什么效果。

代码清单8-7 将List切换为HorizontalList

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
 <![CDATA[
  import mx.collections.ArrayCollection;
  public var myAC:ArrayCollection  = new ArrayCollection([
       {name:"Jon Hirschi", email:"j_hirschi@domain.com",
                 url:"http://www.flexablecoder.com"},
       {name:"Tariq Ahmed", email:"t_ahmed@domain.com",

                 url:"http://www.dopejam.com"},
       {name:"Frank Krul", email:"f_krul@domain.com", url:""}
  ]);
 ]]>
</mx:Script>
<mx:HorizontalList id="myFriends" dataProvider="{myAC}" labelField="name"/>
x:Application>
```

编译并运行这个应用程序(图 8-3)，可以看到内容与 List 相同的 HorizontalList 的外观——只是水平显示每个数据项而已。

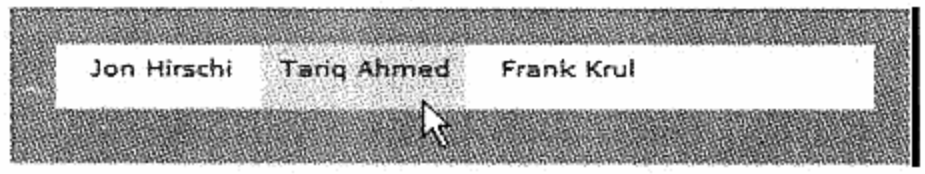


图 8-3　HorizontalList 除水平显示数据项之外，其他方面与 List 相同

这个组件的一个常见用法是在应用程序中并排显示图像，如产品选择表或照片查看器中的缩略图。

8.4.3　TileList

TileList 与同级的 List 的概念类似，但与 List 以单列形式显示数据不一样的是，TileList 会创建包含显示项的大小相同的拼贴块(tile)的格子。当然，也没有列标题，不能排序。TileList 非常适合显示目录。

1. 方向

TileList 可以沿水平(默认)或垂直方向显示数据项。这两种布局方式的区别主要体现在滚动条的用法上。

- **水平方向**：水平方向从左向右依次显示数据项，按照需要添加新行。如果行数太多，超过视图显示的范围，则会显示垂直滚动条。
- **垂直方向**：垂直方向从上至下依次显示数据项，按照需要添加新列。如果列数太多，超过视图显示的范围，则会显示水平滚动条。

要控制每个视图显示的行数或列数，可以分别检查 TileList 组件的 columnCount 和 rowCount 属性。

2. 调用 TileList

下面我们把前面验证无误的 List 示例修改一下，将 List 修改为 TileList(代码清单 8-8)，看一看现在会是什么数据显示效果。

代码清单8-8　使用TileList

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
 <![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var myAC:ArrayCollection  = new ArrayCollection([
       {name:"Jon Hirschi", email:"j_hirschi@domain.com",
```

```
                    url:"http://www.flexablecoder.com"},
        {name:"Tariq Ahmed", email:"t_ahmed@domain.com",
                    url:"http://www.dopejam.com"},
        {name:"Frank Krul", email:"f_krul@domain.com", url:""}
 ]);
]]>
</mx:Script>
<mx:TileList id="myFriends" dataProvider="{myAC}" labelField="name"/>
</mx:Application>
```

编译并运行这个应用程序，应该看到 ArrayCollection 按照如图 8-4 所示的布局被显示出来。

图 8-4 TileList 默认显示水平布局，它会从左至右依次排布每个数据项，直至一行结尾，然后另起一行

如图 8-4 所示，TileList 默认从左至右排列数据项。要修改这个行为，可以添加 direction 属性，将它的值设置为 vertical(效果如图 8-5 所示)。

```
<mx:TileList id="myFriends"
   direction="vertical" dataProvider="{myAC}"
   labelField="name"/>
```

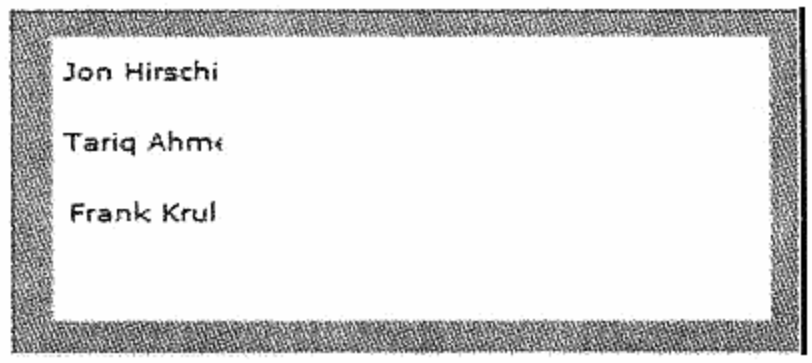


图 8-5 将 direction 属性设置为 vertical，可以让 TileList 逐列显示数据项

我们注意到，无论是图 8-4 还是图 8-5，好像都有一些文本被切掉了。这是因为列的默认宽度不足以容纳要显示的数据项。要解决这个问题很简单，只要为 columnWidth 属性设置一个等于容纳最长数据项的值即可。

3. TileList 与 Tile

稍等一下！这不是跟第 4 章中介绍的 Tile 组件很像吗？没错，是很相似；但也存在一些基本的性能差异。

使用 TileList 占用内存较少且初始响应较快，但它只会渲染数据的可见部分。随着用户拖动滚动条，更多数据项进入视图，TileList 必须动态渲染进入的数据项。因此，会增加滚动与重绘窗口期间的延迟时间。TileList 的优势主要体现在内存占用上，它会根据屏幕显示的需要生成最少量的对象(而非不管屏幕区域大小，一概生成全部对象)。TileList 的缺点是，当 TileList 的内

容复杂且数量庞大时，会导致可用性问题——用户会感觉应用程序反应迟缓。

相反，Tile 会一次性渲染所有对象，无论是否需要立即把这些对象显示给用户。如果渲染的对象很多，用户就要等待较长时间才能看到初始窗口。从有利的方面看，当渲染处理结束后，滚动窗口时的延迟时间就会很短甚至没有延迟。但是，这种可用性的提升是以内存占用的明显增多为代价的。

一般来说，TileList 应该是更好的选择。

8.4.4 DataGrid

勿庸置疑，DataGrid 是任何人都喜欢使用的显示组件。DataGrid 最适合显示表格(tabular)信息，这种信息一般来自数据库查询的结果。大多数 Web 开发人员都在使用这个组件替代类似的 HTML 表格。从技术上讲，Grid 组件更接近 HTML 表格，但 DataGrid 使用起来更方便，让人着迷。

提示：

在 Flex 社区中，DataGrid 组件经常被人称作 DG。

DataGrid 支持列排序和用户交换列(用户可以调整列的次序)等特性。DataGrid 与 List 有几分相似，但 DataGrid 具有多列格式布局和列标题。

提示：

ColdFusion 开发人员熟悉的 CFGRID 标签(使用 Flash 格式)就是基于 Flex 的 DataGrid 构建的。

前面我们针对 List 讨论的内容同样适用于 DataGrid。表 8-4 所示为 DataGrid 支持的另外两个属性，这两个属性与多列有关。

表 8-4　DataGrid 组件的附加属性

属　性	类　型	说　明
resizeableColumns	Boolean	确定用户是否可以调整列的大小。适用于所有列，有效值为 true 和 false，默认值为 true。
sortableColumns	Boolean	设置用户是否可以对列进行排序，适用于所有列。有效值为 true 和 false，默认值为 true

1. 调用 DataGrid

在基于列表的组件家族中，DataGrid 与 List 可谓是同代的近亲。如代码清单 8-9 所示，调用 DataGrid 的方式与调用 List 非常相似。

代码清单8-9 调用DataGrid的方式与调用List相似

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:DataGrid id="dg" width="500" height="200" >
    <mx:Object name="Tariq Ahmed" email="t_ahmed@domain.com"/>
    <mx:Object name="Jon Hirschi" email="j_hirschi@domain.com"/>
    <mx:Object name="Frank Krul" email="f_krul@domain.com"/>
  </mx:DataGrid>
</mx:Application>
```

代码清单 8-9 的输出结果(见图 8-6)与前面的示例有明显不同。首先，DataGrid 有标题行，而且它由多列构成。

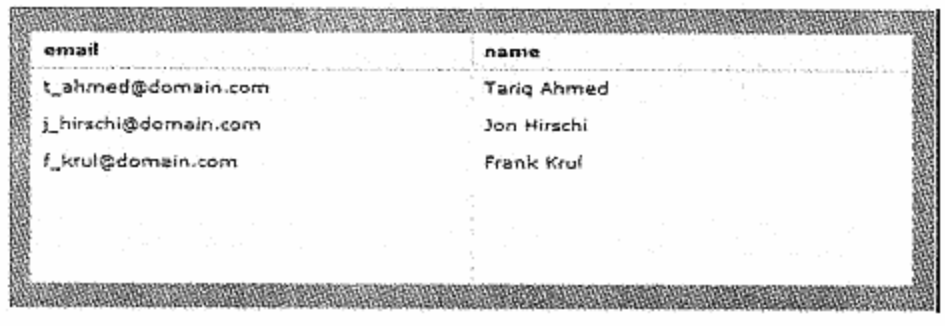


图 8-6 DataGrid 组件，拥有可排序的标题行及可调整的列

接下来试一试 DataGrid 提供的友好功能。单击列标题可以对相应列进行排序。单击并拖动列间的分隔条，可以左右调整列的宽度。最后，再试试单击拖动列标题来更改其位置。

与 List 一样，dataProvider 也可以由外部数据填充(参见 8.2 节)。在前面 ActionScript 示例的基础上，可以将 dataProvider 与一个包含数据的变量绑定(代码清单 8-10)。

代码清单8-10 使用ArrayCollection填充DataGrid的dataProvider

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
 <![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var MyAC:ArrayCollection = new ArrayCollection([
  {name:"Jon Hirschi", email:"j_hirschi@domain.com"},
  {name:"Tariq Ahmed", email:"t_ahmed@domain.com"},
  {name:"Frank Krul", email:"f_krul@domain.com"}
  ]);
 ]]>
</mx:Script>
<mx:DataGrid id="dg" width="500" height="150" dataProvider="{myAC}" >
  <mx:columns>
    <mx:DataGridColumn dataField="name"
                headerText="Contact Name" width="300" />
    <mx:DataGridColumn dataField="email"
                headerText="E-Mail" width="200"/>
  </mx:columns>
</mx:DataGrid>
</mx:Application>
```

为支持和控制 DataGrid 的其他显示功能，可以使用另一组属性。尤其可以控制其排序能力。

2. 控制排序能力

sortableColumns 是控制能否对整个 DataGrid 排序的总开关。要禁用排序，把 sortable Columns 设置为 false 即可。如果要有选择地禁用或启用某一列的排序功能，可以把相应 DataGridColumn 的 sortable 属性设置为 false 或 true。

3. 通过 DataGridColumn 指定列标题

DataGrid 是个聪明伶俐的家伙；无需多少信息，它就能找到作为列标题的字段名。

但是作为苛刻的编码人员，我们总是要求更高——这里，我们想实现更精确的控制。为了让开发人员更好地控制 DataGrid，Flex 提供了一个名叫 DataGridColumn 的标签，它可以与 DataGrid 配合使用。

可以使用 DataGridColumn 标签控制下列属性。

- 列宽
- 列标头(或标题)
- 列中的自动换行
- 启用行内编辑
- 指定鼠标在列中单击时的处理方式

下面我们试一试 DataGridColumn 标签，并通过它为每一列添加更符合人们阅读习惯的标题文本，如代码清单 8-11 所示。

代码清单8-11　通过DataGridColumn控制包括列标题在内的各种属性

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:DataGrid id="dg" width="500" height="150" >
    <mx:columns>
      <mx:DataGridColumn dataField="name"
                         headerText="Contact Name" width="300"/>
      <mx:DataGridColumn dataField="email"
                         headerText="E-Mail" width="200"/>
    </mx:columns>
    <mx:Object name="Tariq Ahmed" email="t_ahmed@domain.com"/>
    <mx:Object name="Jon Hirschi" email="j_hirschi@domain.com"/>
    <mx:Object name="Frank Krul" email="f_krul@domain.com"/>
  </mx:DataGrid>
</mx:Application>
```

定义每一列的标题和宽度

图 8-7 所示为结果。任务完成，而且，这仅仅使用了 DataGridColumn 支持的多个属性中的两个。要查看 DataGridColumn 支持的所有属性，请查看表 8-5。

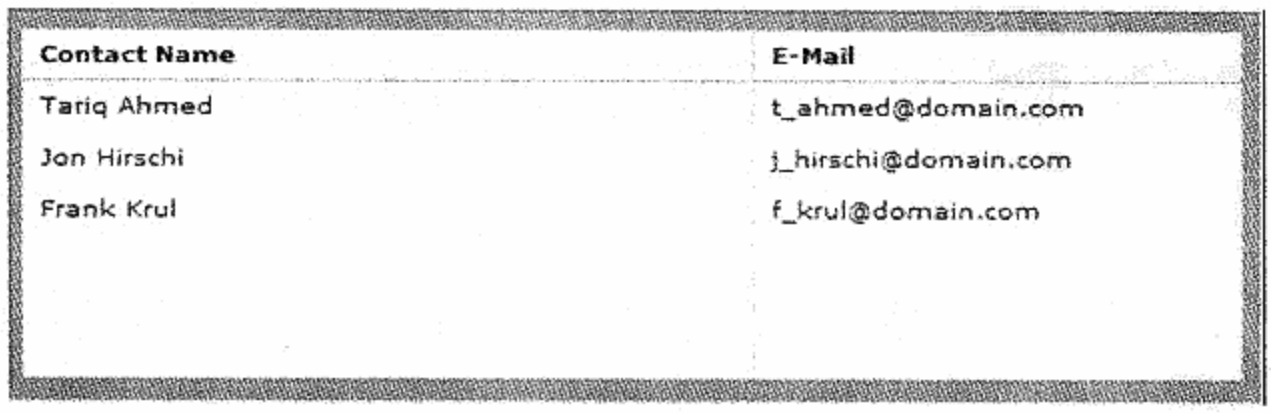


图 8-7　通过 DataGridColumn 为每一列分别指定标题和宽度

表 8-5 DataGridColumn 的属性

属 性	类 型	说 明
dataField	String	设置所选列应该显示数据集中的哪个字段
headerText	String	列标题
headerWordWrap	Special	设置是否允许列标题自动换行。可取值为 true(启用自动换行)和 false(禁用自动换行)。如果未输入值，默认取得 DataGrid 组件的 wordWrap 属性的值
labelFunction	Function	如果指定了函数名称，则将原始数据传给这个函数并显示返回的文本。用于格式化原始数据
minWidth	Number	最小列宽
resizeable	Boolean	用于指定用户是否可以调整列宽的值。可取值为 true(默认值)和 false
sortable	Boolean	用于指定用户是否可以对列进行排序。可取值为 true(默认值)和 false
sortCompareFunction	Function	添加必需的自定义逻辑，用于让 DataGrid 根据特殊的条件(如数值、日期等)对列进行排序。默认情况下，DataGrid 使用基本的字符串比较方式
sortDescending	Boolean	默认排序顺序。可取值为 true(降序)和 false(升序，默认值)
visible	Boolean	该值指定是否显示当前列。可取值为 true(默认值)和 false
width	Number	设置列宽。可以是一个固定值(单位为像素)，或相对于列所在窗口的百分比值。如果所有列都没有设置宽度，DataGrid 会以等宽方式显示各列；如果有的列设置了宽度而有的列没有设置，则剩余未分配的空间用于以等宽方式显示未指定宽度的列
wordWrap	Special	与 headerWordWrap 相似，但作用于列的内容。如果提供任何值，内容自动换行；否则，默认取得 DataGrid 组件的 wordWrap 属性的值。在启用 wordWrap 时，要确保将 DataGrid 的 variableRowHeight 属性设置为 true

看起来属性很多，但 DataGrid 毕竟是使用最频繁且功能最强大的一款组件，只有通过这些属性才能方便地控制其各方面特性。

DataGrid 和 List 虽然大受欢迎，但只能用于显示普通数据。如果要表现层次化的数据，则需要使用 Tree。

8.4.5 Tree

树(tree)的概念源自图形界面的操作系统，而且每个人都比较熟悉。作为 List 的后代，Tree 是显示某些层次化信息(如文件夹结构)的重要组件。

1. 层次化数据

由于要使用嵌套方式显示信息，因此以结构化方式组织数据是明智的做法。XML 是一种按照嵌套结构组织数据的格式。

第 15 章将深入探讨 XML，现在我们仅对它做一简单介绍。下面的代码创建了一个 XML 对象，其中只包含一个根节点(即一个主外部标签)。

```
<mx:XML id="myXML">
  <friends>
    <friend name="Jon Hirschi"/>
    <friend name="Frank Krul"/>
    <friend name="Tariq Ahmed"/>
  </friends>
</mx:XML>
```

而通过 source 属性还可以从独立文件导入数据：

```
<mx:XML source="my.xml" id="myXML"/>
```

这适用于很多场合；但是，XMLList(用于保存 XML 片段)的优势表现在允许拥有多个根节点。

```
<mx:XMLList id="myXML">
  <friends label="Friends">
    <friend label="Jon Hirschi"/>
  </friends>
  <families label="Family">
    <family label="Sabina Ahmed"/>
  </families>
</mx:XMLList>
```

无论 XML 还是 XMLList，都可以与 Tree 一起使用。正如 8.2 节所提到的，一切都要围绕集合进行。换句话说，XMLListCollection 可以包装(直观的说法)XML 或 XMLList 对象，并为它们提供标准的集合功能。以下是包装过程。

```
<mx:XMLListCollection id="myXMLCollection">
  <mx:XMLList id="myXML">
    <friends label="Friends">
      <friend label="Jon Hirschi"/>
    </friends>
    <families label="Family">
      <family label="Sabina Ahmed"/>
    </families>
  </mx:XMLList>
</mx:XMLListCollection>
```

在数据可能发生变化的情况下，应该使用集合，以便在数据变化时通知那些 dataProvider 属性与相应集合绑定的组件刷新其显示。

2. 调用 Tree

在有数据的情况下创建 Tree 非常简单。首先从调用 Tree 的基本步骤开始。代码清单 8-12 创建了一个联系人文件夹结构。

代码清单8-12 提供Tree的XML数据

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:XMLListCollection id="myXMLCollection">
  <mx:XMLList id="myXML">
    <friends label="Friends">
      <friend label="Jon Hirschi"/>
      <friend label="Frank Krul"/>
      <friend label="Ryan Stewart"/>
    </friends>
    <families label="Family">
      <family label="Sabina Ahmed"/>
      <family label="Shiraz Ahmed"/>
    </families>
  </mx:XMLList>
</mx:XMLListCollection>

<mx:Tree dataProvider="{myXMLCollection}" labelField="@label"
       width="300" height="200"/>

</mx:Application>
```

顶级节点将转换为顶级文件夹

告知Tree查找称为label的属性用于显示

编译并运行这个应用程序后的结果如图 8-8 所示。其中，顶级节点由文件夹代表，而末级节点则显示为文件图标。任何包含子节点的项也带有扩展/折叠图标，用于切换数据层次的显示效果。

图 8-8 Tree 组件的输出布局会让人联想到操作系统中的文件—文件夹结构

Tree 不知道要显示哪个字段或属性。由于其与 XML 数据相关，必须通过指定@label(或任何适当的属性)告诉 Tree，XML 节点的哪个属性包含显示字段。

3. 使用包含一个根节点的 XML 文档

如果将外部 XML 文档用作源，该文档只能包含一个根节点。有读者可能会认为初始的文件夹会浪费显示空间 (见图 8-9)。

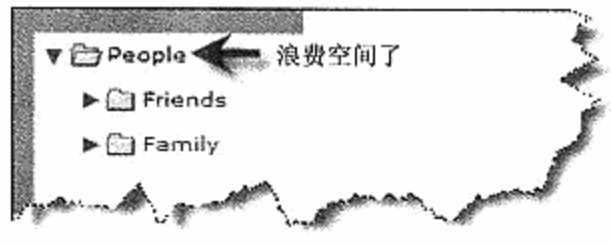


图 8-9 如果为 Tree 提供 XML 文档作为数据源，则会在顶部显示单个根文件夹，因为此 XML 文档中只有单个根节点

不过，通过把 Tree 的 showRoot 属性设置为 false(见代码清单 8-13)，可以实现与使用 XMLList 多根节点文档相同的效果。

代码清单8-13　使用showRoot属性实现从第二级节点开始显示

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:XML id="myXML">
    <people label="People">
      <friends label="Friends">
        <friend label="Jon Hirschi"/>
      </friends>
      <families label="Family">
        <family label="Shiraz Ahmed"/>
      </families>
    </people>
  </mx:XML>
  <mx:Tree dataProvider="{myXML}" showRoot="false" labelField="@label"/>
</mx:Application>
```

与 List 类似，Tree 不难实现，尤其是在灵活、恰当地使用 showRoot 和 labelField 等具有帮助作用的属性来配置显示的情况下更是如此。

Tree 的局限性在于它只能显示一个标签。如果每个层次上需要显示多个字段该怎么办呢？那就需要把DataGrid和Tree的优点结合起来，Flex提供了兼具它们二者之长的AdvancedDataGrid组件。

8.4.6　AdvancedDataGrid

下面介绍 DataGrid 的老大哥——AdvancedDataGrid(ADG)。ADG 是 Flex 3 新引入的组件；在 Flex 2 中，要实现相同的功能必须由水平极高的开发人员在 DataGridr 基础上大量扩展才能做到(有读者可能会问，为什么不在介绍完 DataGrid 之后马上介绍 ADG 呢？原因上面已经提到过，因为 ADG 兼具 DataGrid 与 Tree 的长处)。

ADG 的主要功能包括如下。

- **多列排序**：DataGrid 只能实现单列排序，而 ADG 支持基于任何数量的列排序。
- **行列样式**：DataGrid 只支持将整个组件作为一个整体添加样式。
- **列组**：利用这个功能可以让多列共用一个标题。
- **增强的项渲染器能力**：我们把详细介绍放在第 9 章，本章只涉及在单元格中添加自定义显示逻辑的内容。在 ADG 中，项渲染器可以作用于整行(而非一个单元格)，而一个单元格也可以使用多个项渲染器(在 DataGrid 中则只能使用一个)。

可想而知，ADG 既然有这么多功能，支持这些功能的属性也肯定不会少。表 8-6 所示为这些属性及其说明。

表 8-6 AdvancedDataGrid 支持的专有属性

属 性	类 型	说 明
resizeableColumns	Boolean	设置是否允许用户调整列的宽度。应用于所有列。可取值为 true(默认值)和 false
sortableColumns	Boolean	设置是否允许用户排序列。应用于所有列。可取值为 true(默认值)和 false
displayDisclosureIcon	Boolean	用于扩展/折叠与之关联的树节点的层次切换图标。如果关闭此属性，意味着用户不能扩展/折叠树节点。可取值为 true(默认值)和 false
displayItemsExpanded	Boolean	设置为 true 时表示显示扩展后的数据。默认值为 false
groupedColumns	Array	共用一个列标题的列组
groupIconFunction	Function	函数名称，用于确定在 ADG 的 Tree 中显示什么图标。如果不指定，则使用默认图标
groupLabelFunction	Function	与其他任何 List 的 labelFunction 类似。可调用此函数，来自定义 ADG 中 Tree 的基本显示方式
groupRowHeight	Number	设置 Tree 组中行的高度
itemIcons	Object	指定网格中的项使用什么图标的对象
selectedCells	Array	一些信息，说明当前选中了哪些单元格(行、列位置)
sortExpertMode	Boolean	设置用户如何进行多列排序。如果为 true，用户可以使用 Ctrl 键排序；否则，会使用每列顶部右侧的排序控件框。可取值为 true 和 false(默认值)

使用这些属性可以对允许用户完成的交互类型施加大量控制。下面我们创建一个 Advanced DataGrid 组件的实例，看一看它的工作原理。

1. 调用 AdvancedDataGrid

ADG 除了具有 DataGrid 的功能外，还有其他功能。基本调用只需在出现的每个 DataGrid 前面加上 Advanced 一词。

如代码清单 8-14 所示，通过方便地修改前面的一个 DataGrid 示例即可使用 AdvancedDataGrid。

代码清单8-14　创建AdvancedDataGrid

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
 <![CDATA[
  import mx.collections.ArrayCollection;
  public var myAC:ArrayCollection  = new ArrayCollection([
   {category:"Fruits", food:"Apple"},
   {category:"Fruits", food:"Avocado"},
   {category:"Vegetables",  food:"Carrot"}]);
 ]]>
</mx:Script>
<mx:AdvancedDataGrid id="adg" width="500" dataProvider="{myAC}" >
 <mx:columns>
  <mx:AdvancedDataGridColumn dataField="category"
                             headerText="Category" width="300" />
  <mx:AdvancedDataGridColumn dataField="food"
                             headerText="Food" width="200"/>
 </mx:columns>
</mx:AdvancedDataGrid>
</mx:Application>
```

图 8-10 所示为编译运行这个应用程序的结果。其中列标题稍有不同，即标题区右侧多了一个小图标。这个图标表示如何对每一行进行排序。

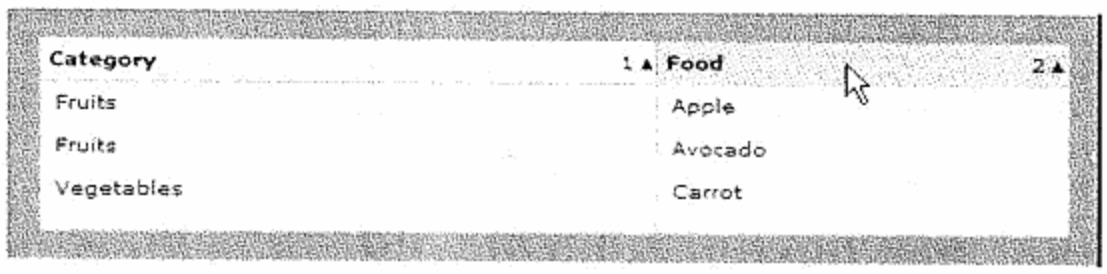


图 8-10　AdvancedDataGrid 与 DataGrid 外观相似，但有一处明确不同——可以同时对多列进行排序

2. 多列排序

实现多列排序的方式有两种，而且都通过 sortExpertMode 属性控制。这个属性接收布尔参数 true 或 false 以决定使用哪种排序方式：

- **false(默认值)**：单击想要据以排序的第一列，然后单击每个后续列标题右上角的框，即可添加作为排序依据的其他列。这种方式的好处是，用户使用鼠标即可实现多列排序；缺点是用户必须把鼠标指针移到一个小框上。
- **true**：单击想要据以排序的第一列，然后按住 Ctrl 键单击另一列标题的任意位置，即可将相应列添加为排序依据。这种方式的好处是，单击后续列标题的任意位置都可以实现多列排序；缺点是必须同时按住 Ctrl 键。

通过将 sortableColumns 属性设置为 true 或 false，可以启用或禁用整个 ADG 的列排序功能。如果想对每一列进行个别控制，可以根据需要将 AdvancedDataGridColumn 的 sortable 属性设置为 true 或 false。

3. 显示层次化数据

使用这种方法的基本前提是，要显示的数据必须具有某种层次化的结构。XML 天然的结构化形式很适合作为这种数据源使用。

步骤如下。

- 以 XMLList 形式创建数据。
- 默认情况下，ADG 不知道数据是否具有某种结构(即使对 XML 也是如此)。这时需要以 HierarchicalData 组件作为媒介，让它帮助我们转换 XMLList。
- 开始使用这种方法。代码清单 8-15 实现一个 ADG 以显示层次化数据。

代码清单8-15 使用AdvancedDataGrid显示层次化数据

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:XMLList id="myXML">
  <friends name="Friends">
    <friend name="Jon Hirschi" email="j_hirschi@domain.com"/>
    <friend name="Frank Krul" email="f_krul@domain.com"/>
    <friend name="Ryan Stewart" email="r_stewart@domain.com"/>
  </friends>
  <families name="Family">
    <family name="Zafira Ahmed" email="z_ahmed@domain.com"/>
    <family name="Shiraz Ahmed" email="s_ahmed@domain.com"/>
  </families>
</mx:XMLList>

<mx:HierarchicalData source="{myXML}" id="myHD"/>

<mx:AdvancedDataGrid dataProvider="{myHD}" width="300">
  <mx:columns>
    <mx:AdvancedDataGridColumn dataField="@name" headerText="Contacts"/>
    <mx:AdvancedDataGridColumn dataField="@email" headerText="Email"/>
  </mx:columns>
</mx:AdvancedDataGrid>

</mx:Application>
```

原始XML数据

将XML数据转换为HierarchicalData

使用HierarchicalData创建显示效果

注意，HierarchicalData 组件先绑定 XMLList，AdvancedDataGrid 继而又绑定 HierarchicalData。

图 8-11 所示为 ADG 显示层次化数据的结果——既类似于 Tree，又具有 DataGrid 的多列特色。

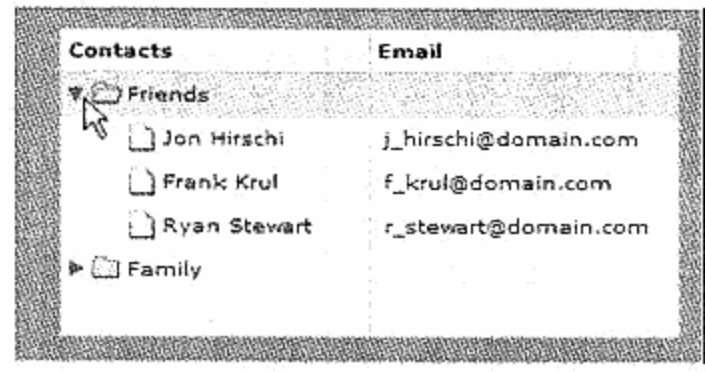


图 8-11 使用 AdvancedDataGrid 显示表列及层次化数据

4. 使用层次化数组集合

前面示例中使用 XML 虽然也很合适(只要以这种方式嵌套数据)，但通常可以对 XML 进行转换，以便用于层次化环境。例如，ArrayCollection 几乎可以用来保存任何数据——数值、文本甚至其他 ArrayCollection。

如代码清单 8-16 所示，可以使用 ActionScript 中的 ArrayCollection 实现与前面示例相同的效果。

代码清单8-16　基于ArrayCollections的嵌套数据，用于实现层次化显示

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

<mx:Script>
 <![CDATA[
 import mx.collections.ArrayCollection;
 [Bindable]
 public var myAC:ArrayCollection = new ArrayCollection([
 {
  name:'Friends',
   children:[
    {name:'Jon Hirschi',  email:'j_hirschi@domain.com'},
    {name:'Frank Krul',   email:'f_krul@domain.com'},
    {name:'Ryan Stewart', email:'r_stewart'}]
 },
 {
  name:'Family',
   children:[
    {name:'Zafira Ahmed', email:'z_ahmed@domain.com'},
    {name:'Shiraz Ahmed', email:'s_ahmed@domain.com'}]
 }]);
 ]]>
</mx:Script>

<mx:HierarchicalData source="{myAC}" id="myHD"/>

<mx:AdvancedDataGrid dataProvider="{myHD}" width="300">
 <mx:columns>
  <mx:AdvancedDataGridColumn dataField="name" headerText="Contacts"/>
  <mx:AdvancedDataGridColumn dataField="email" headerText="Email"/>
 </mx:columns>
</mx:AdvancedDataGrid>

</mx:Application>
```

包含嵌套数据的ArrayCollection

使用children关键字指出新数据层次

在 ActionScript 中，通过构造称为 children 的节点来新建一个嵌套数据级别。基于层次化的数据实现层次化显示是件自然而然的事，但在 Flex 中，也可以将普通数据转换为层次化数据。

5. 以层次化的方式使用普通数据

在需要与后端系统(如 Web 服务)通信的情况下，由于无法控制，取得的数据可能不像前面示例中的数据那么便于使用。编写一套将普通数据转换为新结构格式的代码是一种解决方案，但 Flex 可以帮助我们自动解决这个问题。

Flex 3 中的 GroupingCollection 组件是一个新功能，该组件可以将普通数据转换为结构化数据。代码清单 8-17 所示为将普通的表数据集合转换为层次化格式的例子，做法是将 category 字段组合成文件夹。

代码清单8-17 使用GroupingCollection组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

<mx:Script>
 <![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var myAC:ArrayCollection = new ArrayCollection([
 {category:'Friends', name:'Jon Hirschi', email:'j_hirschi@domain.com'},
 {category:'Friends', name:'Frank Krul',  email:'f_krul@domain.com'},
 {category:'Friends', name:'Ryan Stewart', email:'r_stewart'},
 {category:'Family', name:'Zafira Ahmed', email:'z_ahmed@domain.com'},
 {category:'Family', name:'Shiraz Ahmed', email:'s_ahmed@domain.com'}]);
 ]]>
</mx:Script>

<mx:GroupingCollection id="myGC" source="{myAC}">
 <mx:Grouping>
  <mx:GroupingField name="category"/>
 </mx:Grouping>
</mx:GroupingCollection>

<mx:AdvancedDataGrid dataProvider="{myGC}" initialize="myGC.refresh()"
  width="300">
 <mx:columns>
  <mx:AdvancedDataGridColumn dataField="name" headerText="Contacts"/>
  <mx:AdvancedDataGridColumn dataField="email" headerText="Email"/>
 </mx:columns>
</mx:AdvancedDataGrid>

</mx:Application>
```

普通数据格式

通过组合category列创建层次

利用 Flex 的这个新功能，可以在不修改 ADG 代码的情况下实现基于普通数据的层次化显示；无论数据源的格式是层次化的还是普通格式，都无所谓(如本例所示)。图 8-12 显示如何转换表格数据，以便显示组合列上的文件夹。

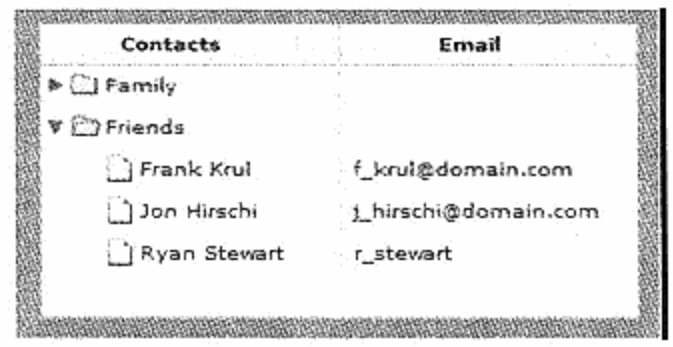


图 8-12 通过分组联系人类别，将普通表列数据转换为层次化数据

这里还有一个小技巧。读者大概注意到代码中的 initialize="myGC.refresh()"；请记住，在初始设置或更新 GroupingCollection 的属性以触发它处理数据时，必须调用 GroupingCollection 的 refresh 函数——它不会自动执行。因此，应该在初始化 ADG 的同时调用 refresh 函数(这样更方便)。

6. 对列分组

AdvancedDataGrid 支持的另一个功能是可以让多个列共享一个标题，即对列进行分组。此时，应该以<mx:groupedColumns>取代<mx:columns>来指派列，然后通过<mx:AdvancedDataGrid-ColumnGroup>标签将要分组的列包装起来。

代码清单 8-18 的设计意图是：客户想拥有一个工具基于目的地跟踪每个人的上下班时间。这里可以把最小和最大通勤时间分组为一列。

代码清单8-18　使用groupedColumns组件将列组合在一起

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

<mx:Script>
  <![CDATA[
    import mx.collections.ArrayCollection;
    public var myAC:ArrayCollection = new ArrayCollection([
      {dest:'San Jose, CA',     min:'45 mins', max:'90 mins'},
      {dest:'San Francisco, CA', min:'60 mins', max:'120 mins'},
      {dest:'San Ramon, CA',    min:'25 mins', max:'60 mins'}]);
    ]]>
</mx:Script>

<mx:AdvancedDataGrid dataProvider="{myAC}" width="300">
  <mx:groupedColumns>
    <mx:AdvancedDataGridColumn dataField="dest" headerText="Destination"/>
    <mx:AdvancedDataGridColumnGroup headerText="Commute Time">
      <mx:AdvancedDataGridColumn dataField="min" headerText="Min"/>
      <mx:AdvancedDataGridColumn dataField="max" headerText="Max"/>
    </mx:AdvancedDataGridColumnGroup>
  </mx:groupedColumns>
</mx:AdvancedDataGrid>

</mx:Application>
```

要分组的列

如图8-13所示，AdvancedDataGridColumnGroup组件将Min和Max列分为一组，即让两个常规列共用了一个主标题。

至此我们已经介绍了所有基于列表的组件及其显示数据的方式。下面要讨论的是如何处理用户对这些组件的交互。

Destination	Commute Time	
	Min	Max
San Jose, CA	45 mins	90 mins
San Francisco, CA	60 mins	120 mins
San Ramon, CA	25 mins	60 mins

图 8-13　AdvancedDataGrid中的Min和Max列被分在一组

8.5　与列表交互

在 Flex 中，列表(有序的数据集合，而非 List 组件)的用途多种多样，应用程序很少将列表仅仅用于显示目的。列表扮演的另外一个角色是充当用户与应用程序交互的手段。

默认情况下，单击一个单元格，意味着单击了一整行。这种情况在 List 组件中不是很明显，因为每一项只有一个单元格。但在 DataGrid 中，当鼠标悬停在一行上时，整行都会突出显示。这种突出显示就是事件导致的结果；事实上，事件是与列表交互的核心。

8.5.1　列表事件

事件是 Flex 交互背后的驱动力量；操作触发事件，事件处理程序进而执行响应相应操作的代码。并不是所有事件都与 List 有关，表 8-7 所示为必须关注的几个事件。

表 8-7 可以使用这些事件来处理与基于 List 的组件的交互

事 件	说 明
click	在用户单击组件时发生。这是在许多 Flex 应用程序中常见的高级事件
doubleClick	在用户双击鼠标时发生
itemClick	表示单击的行或列
change	当用户在选定项之外的其他行中单击时发生

使用与前几章介绍的技术类似的技术可以实现这些事件。利用事件的方式有很多种。有些方式能较快地投入使用，而有些方式通过解耦依赖的逻辑提高了代码的可重用性。

8.5.2 向函数传递事件

代码清单 8-19 指示 DataGrid 调用一些 ActionScript 代码(这里是一个函数)并传递此过程中创建的事件对象。

代码清单8-19 通过向函数传递事件对象处理用户交互

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

<mx:Script>
 <![CDATA[
  import mx.collections.ArrayCollection;
  import mx.events.ListEvent;
  import mx.controls.Alert;

  public var myAC:ArrayCollection  = new ArrayCollection([
  {name:"Jon Hirschi", email:"j_hirschi@domain.com"},
  {name:"Frank Krul",  email:"f_krul@domain.com"}]);

  public function handleClick(evt:ListEvent):void
  {
   Alert.show("You clicked on row:" + evt.rowIndex + " and col:" +
            evt.columnIndex + "." +
             "Which is for " + evt.currentTarget.selectedItem.name);
  }
 ]]>
</mx:Script>

<mx:DataGrid id="dg" width="500" height="150" dataProvider="{myAC}"
         itemClick="handleClick(event)">
  <mx:columns>
   <mx:DataGridColumn dataField="name"
                headerText="Contact Name" width="300" />
   <mx:DataGridColumn dataField="email" headerText="E-Mail" width="200"/:
  </mx:columns>
</mx:DataGrid>

</mx:Application>
```

图 8-14 所示为单击 DataGrid 中一行时的情况。按照代码指示，函数显示与所选行相关的信息，但最值得一提的是，handleClick 函数并不知道是哪个 DataGrid 发送的事件。这是可重用函数的一个重要特征。

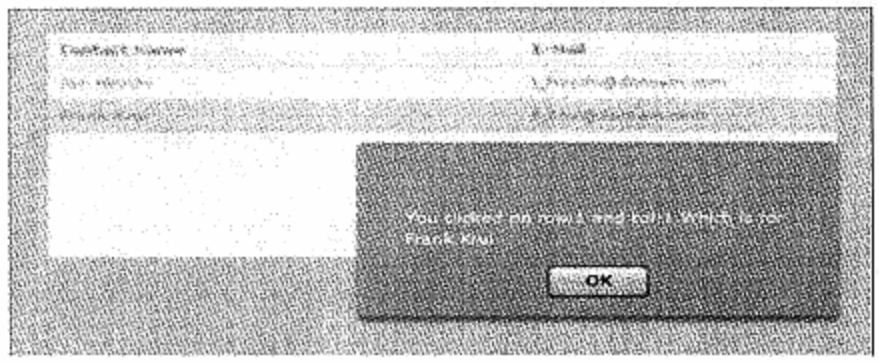


图 8-14　通过函数显示被单击行的详细信息

虽然这个示例使用了 DataGrid，但事实上可以方便地使用其他组件，如下列代码所示。

```
<mx:List dataProvider="{myAC}" labelField="name"
   itemClick="handleClick(event)"/>
```

更具体一点，我们可以关注下面这段。

```
itemClick="handleClick(event)"
```

这里，我们告诉 DataGrid 在用户触发 itemClick 事件时运行 handleClick 函数，同时将在此期间创建的事件对象传递给该函数。

然后 handleClick 函数以参数形式接收事件对象并访问与其相关的各种属性。在使用这种方法时，我们关心的是 currentTarget.selectedItem。

currentTarget 属性是一个引用，或者是一个指针，指向与事件相关的项。从较高的层次来看，事实还要更复杂一些；但从新开发人员的角度讲，它就是用户单击的组件。在此可以认为 currentTarget 指向 DataGrid。

进一步，既然有了对 DataGrid 的引用，就可以访问 selectedItem 属性，而 selectedItem 中包含相关行的数据的引用。再向下，就可以访问到该行的具体数据字段。

```
evt.currentTarget.selectedItem.name
```

提醒一下，并非只能访问显示数据，也可以访问提供数据的任何字段。即使有多个字段，也可以这么做：

```
{name:"Tariq Ahmed", email:"tariq@domain.com", domain:"www.dopejam.com"}
```

但这里只显示 name 和 email 字段。此时，通过 selectedItem 仍然可以访问到 domain 字段。

```
evt.currentTarget.selectedItem.domain
```

与 selectedItem 类似的属性是 selectedIndex。这是一个数值，指出单击的是哪一行(第一行为 0)。

8.5.3　向函数传递数据

另一种方式是只传递你感兴趣的事件，而非传递整个事件对象。这种技术可以进一步提升接收函数的可重用性，因为函数无需知道数据是事件负载的一部分；否则，就要事先确定事件的来源。

某些情况下，的确需要让函数了解有关事件的具体信息。但是，我们在此假设需要进一步提升抽象程度。代码清单 8-20 显示了做法。

代码清单8-20 在处理用户交互时只向函数传递数据

```
<mx:DataGrid id="dg" width="500" height="150" dataProvider="{myAC}"
         itemClick="handleClick(event.currentTarget.selectedItem)">
  <mx:columns>
    <mx:DataGridColumn dataField="name"
                   headerText="Contact Name" width="300" />
    <mx:DataGridColumn dataField="email" headerText="E-Mail" width="200"/>
  </mx:columns>
</mx:DataGrid>
```

这样，当用户单击一项时，会在调用函数时传入被单击的项的引用。这允许访问对象的各个属性。

```
public function handleClick(data:Object):void
{
  Alert.show("Name:" + data.name + ",Email:" + data.email);
}
```

这种方式的缺点是，函数不引用引发事件的组件。而没有此信息，函数就无法实现很多操作，例如不能在出现问题时将组件的颜色改为红色。

8.5.4 直接访问被选定的行

更直观的方式是直接访问组件。这种方式的好处(除了实现简单外)是，可以方便地访问相应组件类型(在此是 DataGrid)特有的任何属性。

```
<mx:DataGrid id="dg" dataProvider="{myAC}" itemClick="handleClick()">
```

注意，我们没有向 itemClick 处理程序传递任何信息。然后这个函数会直接访问相应的值。

```
public function handleClick():void
{

  Alert.show("Name:" + dg.selectedItem.name + ", Email:" +
          dg.selectedItem.email);
}
```

这是一种简单的技术，但它同样可以达到目的，而且函数的维护较为简单——毕竟，函数不用关心哪个组件调用了它，或者为什么要调用它。

8.5.5 绑定到被选定的行

这是最简单的一种方式。通过使用 Flex 的绑定功能，甚至不必借助事件处理程序就可以显示或存储被选定的项。代码清单 8-21 所示为当前选项绑定到调用相应值的项的过程，其中没有用到函数。

代码清单8-21 使用绑定捕获基于列表组件的选定项

```
<mx:DataGrid id="dg" width="500" height="150" dataProvider="{myAC}">
  <mx:columns>
```

```
    <mx:DataGridColumn dataField="name"
                       headerText="Contact Name" width="300" />
    <mx:DataGridColumn dataField="email"
                       headerText="E-Mail" width="200"/>
    </mx:columns>
</mx:DataGrid>
<mx:Label text="The selected person is {dg.selectedItem.name}
    ({dg.selectedItem.email})" fontSize="16"/>
```

图 8-15 所示为以上代码的输出结果。被选人的信息显示在窗口下方。

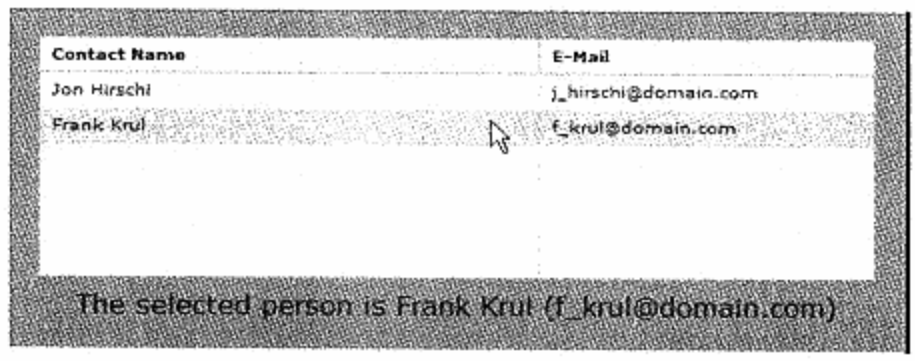


图 8-15　通过 Flex 的绑定机制将被选人信息显示在窗口下方

如果需要考虑的只有值，那么绑定就是最方便的方式。这种方式的缺点是，不能添加任何逻辑或选择之后的处理方式。

8.5.6　用其他事件作为触发器

关于处理用户交互，我们最后想说的是，虽然示例中使用的是 itemClick 事件，但使用前面提到的其他事件都同样简单。例如，要使用 change 事件，只需用 change 代替 itemClick 即可，就这么简单。当然，也可以同时使用两种事件，即在每次单击时执行某些逻辑，而在焦点从选定行移开时执行另外一些逻辑。

8.6　小结

勿庸置疑，基于列表的组件是 Flex 应用程序中的绝对主力。一方面，因为列表是一种常用的 UI 模式；另一方面，许多 Flex 应用程序都会以这样或那样的方式与后端数据库交互，而显示数据行正是列表组件的拿手好戏。

如果要显示数据，选择 List 是没错的。List 支持从简单数组到用于层次化数据的 XML 的信息。由于集合组件/类的超前优势，使用其中之一是最佳选择。

大多数基于列表的组件本质上是平面化的，也就是说，它们只能按顺序逐一显示内容项。不过，对于结构化的数据，也可以使用层次化组件。AdvancedDataGrid 就是一个既能显示普通数据，又能显示结构化数据的组件。

显示数据之后，最后要考虑的是处理用户交互。处理用户交互的方式也有很多——可以向事件处理程序传递整个事件对象，可以只传递数据，或者通过函数直接取得数据。在简单情况下，可以利用数组绑定。

第 9 章将继续关注列表，讨论如何自定义列表。

第 9 章 自定义列表

9

本章内容：

- 标签函数
- 项渲染器和编辑器
- 组合项渲染器和项编辑器
- 高级项渲染器
- 筛选函数

List(组件类)是 Flex 应用程序中的绝对主力。因此，Flex 开发过程的很大部分是处理 List 的外观和交互。

除美化页面格式(前几章已经讨论过)之外，应该对基于 List 的组件的行与单元格的内容进行精细的控制。这涉及增加交互功能，以便用户处理内容。这个过程称为自定义列表的过程。为此，本章从标签函数开始谈起，因为标签函数是自定义列表的最简便方式。

9.1 标签函数

首先介绍一下标签函数(label function)的应用背景。我们从可能的数据源(如数据库)取得数据，此时要做的不仅仅是按照原样显示这些数据。必须把原始数据格式化为某种对用户更有吸引力的形式，例如把布尔值转换为人们看来清晰易懂的格式，或者为清晰起见把多个数据字段联合起来放到一列中。

这些都是可以使用标签函数的情形。标签函数是由 List(或与 List 有关的其他组件，如 DataGrid 和 TileList)的某一列基于每个数据行调用的函数，该函数返回的值就是要显示的值。

9.1.1 标签函数的类型

有两种标签函数：一种用于只包含单列的 List，如 List、HorizontalList、TileList 和 Tree；另一种用于包含多列的 List，如 DataGrid、AdvancedDataGrid、PrintDataGrid、FileSystemDataGrid 和 OLAPDataGrid。

这两种标签函数的区别主要体现在参数的形式上。

1. 单列标签函数

单列标签函数接收一个参数——对当前数据记录(或数据行)的引用。通过这个参数可以访问任何数据字段并返回一个字符串值，如代码清单 9-1 所示。

代码清单9-1 在只包含一列的列表组件中使用的标签函数

```
public function fullName(rowItem:Object):String
{
   return rowItem.firstName + ' ' + rowItem.lastName;
}
```

进而，如果要展示由多列构成的组件，需要使用多列标签函数。

2. 多列标签函数

多列标签函数除了像单列标签函数一样接收对当前行的引用之外，还接收另一个参数——对当前列的引用(以确定针对哪一列)。代码清单 9-2 所示为一个多列标签函数的示例。

代码清单9-2 在包含多列的列表组件中使用的标签函数

```
public function formatDate(rowItem:Object,column:DataGridColumn):String
{
   var retVal:String = "";
   if(column.dataField == "dtJoined")
     retVal = dFmt.format(rowItem.dtJoined);
   else if(column.dataField == "dtLogin")
     retVal = dFmt.format(rowItem.dtLogin);
   return retVal;
}
```

DataGrid 等类似的多列组件可以使用这种标签函数。

3. 并非列表组件才能使用标签函数

图表组件和表单控件(如 ComboBox 和 DateField 等)等很多元素都支持标签函数。请注意，即使本章的主题是自定义列表(由此涉及了标签函数)，但有关标签函数的原理同样也适用于上述非列表组件。

前面简单介绍了两种类型的标签函数，接下来我们看几个如何使用它们的示例。

9.1.2 使用单列标签函数

假设有一个保存着业务伙伴联系人信息的数据文件，但有一个问题：每个联系人的姓和名分别保存在各自的字段中，而我们想在一个 List 组件中显示每个人的全名。

这里所说的 List 组件只包含一列；但如前所述，数据分散在两个字段中。通过使用标签函数可以将两个字段显示在一列中。

代码清单 9-3 中定义的标签函数从 firstName 和 lastName 字段提取数据，并将它们连接成一个新字符串。然后，将这个新字符串返回给 List 组件以便显示。

代码清单9-3　在单列List组件中使用标签函数

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Script>
   <![CDATA[
      import mx.collections.ArrayCollection;
      [Bindable]
      public var myAC:ArrayCollection  = new ArrayCollection([
      {firstName:"Jon",lastName:"Hirschi"},
      {firstName:"Tariq",lastName:"Ahmed"}
      ]);
      public function fullName(rowItem:Object):String
      {
         return rowItem.firstName + ' ' + rowItem.lastName;
      }
   ]]>
</mx:Script>
<mx:List dataProvider="{myAC}" labelFunction="fullName"/>
</mx:Application>
```

用于填充列表的数据

为处理数据项由列表调用的标签函数

合并两个字段并返回到列表

列表请求的标签函数

相信读者已经意识到使用标签函数有多简单了。整个过程非常清楚，而我们想要的结果如图 9-1 所示。

图 9-1　标签函数把原先存储在不同数据字段的姓和名合并到了一列中

由于 List 组件只包含一列，因此只能使用单列标签函数。如果使用的是基于列表的多列组件(例如 DataGrid 可能由多个列构成)，就需要使用多列标签函数。

9.1.3　使用多列标签函数

在接下来的示例中，有一个数据文件，包含一个虚拟组织(也许是在线论坛或博客)的成员信息。文件中的日期和时间字段包含着每位成员加入组织的时间，还有一个字段包含它们的最近登录时间。现在，我们并不关心字段中的时间部分，只想显示日期。因此，需要通过一个标签函数将原始数据传递给格式化程序(代码清单 9-4)，然后显示格式化之后的结果。

代码清单9-4　在多列DataGrid中使用标签函数

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Script>
   <![CDATA[
      import mx.collections.ArrayCollection;
      [Bindable]
      public var myAC:ArrayCollection  =
         new ArrayCollection([
      {name:"Jon Hirschi", username:"jhirschi",dtJoined:"01/02/2005
         12:32:55",dtLogin:"05/20/2008 13:33:22" },
      {name:"Tariq Ahmed", username:"tahmed",dtJoined:"03/12/2005
         05:44:32",dtLogin:"07/09/2008 14:44:23"},
      ]);
```

包含日期和时间字段的数据

```
        public function
    formatDate(rowItem:Object,column:DataGridColumn):String
        {
            var retVal:String = "";
            if(column.dataField == "dtJoined")
                retVal = dFmt.format(rowItem.dtJoined);
            else if(column.dataField == "dtLogin")
                retVal = dFmt.format(rowItem.dtLogin);
            return retVal;
        }
    ]]>
</mx:Script>

<mx:DateFormatter id="dFmt" formatString="MM/DD/YY"/>

<mx:DataGrid id="dg" width="500" height="100" dataProvider="{myAC}">
    <mx:columns>
        <mx:DataGridColumn dataField="name"  headerText="Name"/>
        <mx:DataGridColumn dataField="username" headerText="Username"/>
        <mx:DataGridColumn dataField="dtJoined" headerText="Joined"
                           labelFunction="formatDate"/>
        <mx:DataGridColumn dataField="dtLogin"  headerText="Last Login"
                           labelFunction="formatDate"/>
    </mx:columns>
</mx:DataGrid>

</mx:Application>
```

如果是加入日期字段，则只返回日期部分

如果是登录日期字段，则只返回日期部分

只针对日期的格式化程序

使用标签函数的两列

在运行这个应用程序后，可以看到如图 9-2 所示的结果。其中，原先包含日期和时间的字段经过格式化，只显示日期部分而忽略时间部分。

Name	Username	Joined	Last Login
Jon Hirschi	jhirschi	01/02/05	05/20/08
Tariq Ahmed	tahmed	03/12/05	07/09/08

图 9-2 由于使用了标签函数，Joined 和 Last Login 字段只显示了日期

关于这个示例还有两点说明。

- 不能向标签函数传递参数，Flex 会自动传递行和列的引用。
- 函数根据列数据决定使用哪一列，以确保格式化适当的值。

与不那么复杂的单列标签函数相比，这个示例中的标签函数稍微有些不好理解。

9.1.4 标签函数的用途

以下是标签函数的常见用途。

- 连接多个数据列创建一个字符串，以便在一列中显示。
- 格式化原始数据。
- 将原始数据转换为易于阅读的格式。

还有另外一种可能的用途：针对行中的数据运行条件逻辑，但不显示原始数据，而是显示自定义信息。例如，可以在付款截止日之后显示一条“逾期”信息。

下一节，我们介绍能够突破标签函数局限性的项渲染器。

9.2　项渲染器

虽然标签函数简单易用，但除前两个例子展示的用途之外，它的用处有限。如果想要的不仅仅是返回并显示格式化文本，该怎么办呢？

这时就要用到项渲染器。项渲染器可以精细地控制某个单元格的外观。例如，可以通过项渲染器定义单元格背景的颜色，以及添加图像或交互式 UI 组件。

默认情况下，Flex 以<mx:Text>组件作为项渲染器，但<mx:Text>只能呈现提供给它的文本内容。我们所说的指定项渲染器，通常都意味着要覆盖这个默认组件。

9.2.1　渲染器类型

Flex 提供了 3 种项渲染器。

- **常规渲染器(regular)**：就是通常所说的项渲染器。相关逻辑被包含在一个可以引用的独立文件中。
- **行内渲染器(inline)**：作用与常规渲染器一样，但为了方便起见，相关逻辑包含在代码中。
- **混入渲染器(drop-in)**：指的是作为项渲染器使用的通用组件(如 Button 或 DateField)。

下面分别介绍如何创建这 3 种项渲染器。然后读者可以尝试添加自定义逻辑。

9.2.2　创建(常规)项渲染器

可将创建项渲染器的基本步骤归纳如下。

(1) 创建一个独立的 MXML 文件，将其命名为 myRenderer.mxml。这就是我们的项渲染器。

(2) 使用 dataProvider 的相关行记录提取所需的字段。

(3) 将数据值封装在一个容器(如 HBox)中以便显示。

代码清单 9-5 和代码清单 9-6 是下面所示的两个源文件。其中，代码清单 9-5 中是项渲染器，代码清单 9-6 中是主应用程序文件。

代码清单9-5　项渲染器：myRenderer.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="100%"
   height="110">
   <mx:Image source="{data.thumbnail}"/>
   <mx:Text text="{data.firstName} {data.lastName}" fontSize="16"
      fontWeight="bold"/>
</mx:HBox>
```

代码清单9-6 使用项渲染器的主应用程序：HelloWorld.mxml

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Script>
   <![CDATA[
      import mx.collections.ArrayCollection;
      [Bindable]
      public var myAC:ArrayCollection  = new ArrayCollection([
      {firstName:"Jon",lastName:"Hirschi",thumbnail:"assets/jon.jpg"},
      {firstName:"Tariq",lastName:"Ahmed",thumbnail:"assets/tariq.jpg"}
      ]);
   ]]>
</mx:Script>

<mx:List dataProvider="{myAC}" itemRenderer="myRenderer" height="230"
   width="220"/>

</mx:Application>
```

图 9-3 所示为令人惊讶的结果(当然，还有代码——读者应该知道我们想要表达什么意思吧？)。使用项渲染器不仅能够显示图像，而且能够为字体添加样式；这些都是使用标签函数做不到的。

图 9-3 使用项渲染器在一列中显示较为复杂的内容

有关这个示例，读者要记住以下两点。

- MXML 文件遵循 XML 标准；也就是说，项渲染器文件必须包含如下声明：<?xml version="1.0" encoding="utf-8"?>。
- HBox 包含的 xmlns 是一个名称空间声明，该声明告诉 Flex 编译器我们使用的 HBox 来自 Adobe Flex，不是同名的第三方组件。

虽然这个结果还算令人满意，但要知道包含条件逻辑的项渲染器更具有吸引力。

9.2.3 向项渲染器中添加逻辑

前面的示例只是取得目标信息并原样显示出来。但是，在开发 Flex 应用程序的过程中，总会遇到需要通过添加业务逻辑决定在单元格中显示什么的情况。

在下面的示例中(代码清单 9-7 和代码清单 9-8)，我们将构建一个包含 3 列的联系人表。其中，第三列包含一个按钮，用户单击该按钮可以向相应联系人发送一封电子邮件(但只有在该联系人提供电子邮件的情况下才会显示这个按钮)。如果没有电子邮件信息，则不会显示该按钮。

代码清单9-7　条件式项渲染器：myRenderer.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="100%"
   creationComplete="checkEmail()">
   <mx:Script>
      <![CDATA[
      import flash.net.*;
      private function checkEmail():void
      {
         if(data.email.length > 0)
         {
            emailButton.visible = true;
         }
      }

      public function sendMail():void
      {
         var u:URLRequest = new URLRequest("mailto:" + data.email);
         navigateToURL(u,"_self");
      }
      ]]>
   </mx:Script>
   <mx:Button id="emailButton" visible="false" label="Send Email"
   click="sendMail()"/>
</mx:HBox>
```

如果存在电子邮件信息，则显示按钮

按下按钮可以打开邮件客户端

代码清单9-8　使用项渲染器的主应用程序文件：HelloWorld.mxml

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Script>
   <![CDATA[
      import mx.collections.ArrayCollection;
      [Bindable]
      public var myAC:ArrayCollection = new ArrayCollection([
      {firstName:"Jon",lastName:"Hirschi",email:""},
      {firstName:"Tariq",lastName:"Ahmed",email:"tariq@dopejam.com"},
      {firstName:"Faisal",lastName:"Abid",email:""},
      {firstName:"Tom",lastName:"Ortega",email:""},
      {firstName:"Ryan",lastName:"Stewart",email:"ryan@isreallycool.com"}
      ]);
   ]]>
</mx:Script>

<mx:DataGrid id="dg" width="500" height="100" dataProvider="{myAC}">
   <mx:columns>
      <mx:DataGridColumn dataField="firstName"  headerText="First Name"/>
      <mx:DataGridColumn dataField="lastName"  headerText="Last Name"/>
      <mx:DataGridColumn headerText="Email" itemRenderer="myRenderer"/>
   </mx:columns>
</mx:DataGrid>

</mx:Application>
```

如图 9-4 所示，添加的逻辑代码实现了期望的效果。

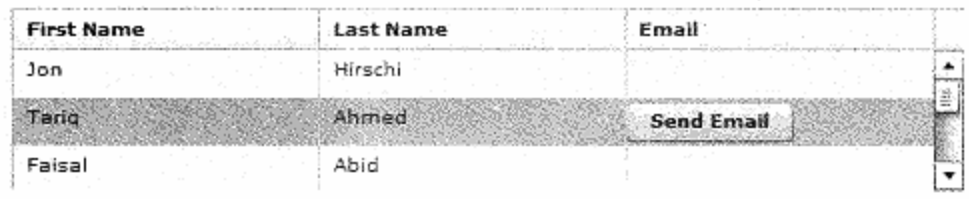


图 9-4 项渲染器只在有电子邮件信息时显示按钮

这个示例的运行过程如下。

- Flex 根据需要创建 myRenderer 实例。
- 随着每个 myRenderer 实例的创建，会触发项渲染器的 creationComplete 事件，因而运行 checkEmail 函数。
- 如果 email 字段中有信息(只要有字符即可，这里没有检查电子邮件格式)，checkEmail 会将按钮的 visible 属性由默认值 false 修改为 true，这样，按钮就会显示出来(按钮的实例是早已创建的)。

没错，就是这么简单——尽管有一个小小的陷阱。

小陷阱

出于性能方面的考虑，Flex 会重用已有的行——如果需要显示 1 000 个对象，它不会生成 1 000 行；相反，它会生成比屏幕可显示的行数稍多一些的行。换句话说，如果在这个示例中有大约 10 行，那么在上下滚动时，很可能看到按钮随机无规律地出现。原因是 Flex 在重用先前已经设置为显示按钮的行。

为了避免这个问题，需要按照下列步骤操作。

- 继续使用 creationComplete 事件初始化项渲染器，以防 ArrayCollection(myAC)还没有数据。
- 使用 dataChange 事件检查是否需要显示按钮。
- 检查显示效果时，根据 email 字段是否为空，配置不可见或可见设置。

代码清单 9-9 展示了改进后的项渲染器。

代码清单9-9 在数据变化时重新设置自身的更完善的项渲染器

```
<?xml version="1.0" encoding="utf-8"?>
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="100%"
   creationComplete="initDisplay()"
   dataChange="checkEmail()">
   <mx:Script>
      <![CDATA[
      import flash.net.*;
      private function initDisplay():void
      {
         emailButton.visible = false;
      }

      private function checkEmail():void
      {
         if(data.email.length > 0)
            emailButton.visible = true;
```

在创建项渲染器时，调用函数进行初始化

在数据变化时，调用重置显示效果的函数

将按钮初始化为不可见

```
            else
                emailButton.visible = false;
        }
        public function sendMail():void
        {
            var u:URLRequest = new URLRequest("mailto:" + data.email);
            navigateToURL(u,"_self");
        }
        ]]>
    </mx:Script>
    <mx:Button id="emailButton" visible="false" label="Send Email"
    click="sendMail()"/>
</mx:HBox>
```

随着数据变化调整的可见性

常规项渲染器最为常用，原因是可以在应用程序的其他位置方便地重用它。但为了方便起见，下面介绍另外两种项渲染器，先从行内项渲染器开始。

9.2.4　创建行内项渲染器

行内项渲染器是把逻辑从独立文件转移到需要它的代码中。在应用程序的概念验证阶段，行内项渲染器绝对是一种方便的手段，但它最大的缺点是，不能在应用程序的其他地方重用相应逻辑。而常规项渲染器可以在多个列表组件及多个文件中使用。

代码清单 9-10 所示为创建行内项渲染器的过程，无非是把逻辑直接复制到代码中，并嵌套在适当的标签中。

代码清单9-10　在DataGrid列中使用行内项渲染器

```
<mx:DataGrid id="dg" width="500" height="100" dataProvider="{myAC}">
    <mx:columns>
        <mx:DataGridColumn headerText="Name">
            <mx:itemRenderer>
                <mx:Component>
                    <mx:Text text="{data.firstName} {data.lastName}"/>
                </mx:Component>
            </mx:itemRenderer>
        </mx:DataGridColumn>
        <mx:DataGridColumn headerText="Email" itemRenderer="myRenderer"/>
    </mx:columns>
</mx:DataGrid>
```

如前所述，这种方法很方便。如果想更简单，混入项渲染器会更省事。

9.2.5　使用混入项渲染器

Flex 提供了很多可以作为混入项渲染器(drop-in renderer)的组件。下面我们举例说明怎样使用这些组件中的一个组件(Image 组件)，执行与项渲染器相同的任务。这样会更省事。

以前面通过自定义项渲染器显示图像的示例为样本，下面我们创建另一个显示图像和文本的应用程序——但这次使用 Image 组件(代码清单 9-11)。

代码清单9-11 用于在一列中渲染图像的混入项渲染器

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Script>
   <![CDATA[
      import mx.collections.ArrayCollection;
      [Bindable]
      public var myAC:ArrayCollection  = new ArrayCollection([
      {name:"Jon Hirschi",thumbnail:"assets/jon.jpg"},
      {name:"Tariq Ahmed",thumbnail:"assets/tariq.jpg"}
      ]);

   ]]>
</mx:Script>

<mx:DataGrid id="dg" width="250" height="100" dataProvider="{myAC}">
   <mx:columns>
      <mx:DataGridColumn headerText="Name" dataField="name"/>
      <mx:DataGridColumn headerText="Picture" dataField="thumbnail"
                    itemRenderer="mx.controls.Image"/>
   </mx:columns>
</mx:DataGrid>
</mx:Application>
```

调用混入项渲染器

图 9-5 所示为如何以最省事的方式，通过 Image 组件在 DataGrid 中将图像显示为缩略图。

提示：

如果想让图像更大些，需要把 DataGrid 的 rowHeight 属性设置为指定大小。如果需要将图像扩大成最大尺寸，则需要设置 variableRowHeight="true"。

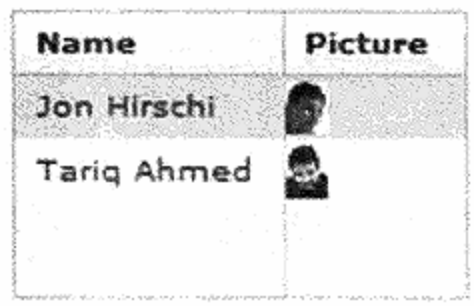


图 9-5 将 Image 组件作为混入项渲染器使用

表 9-1 所示为可以作为混入项渲染器使用的所有组件。

表 9-1 支持混入项渲染器的组件

组 件	默 认 属 性	说 明
bButton	selected	显示一个按钮。遗憾的是，无法控制按钮的相关标签
CheckBox	selected	
DateField	selectedDate	
Image	source	如果想让图像显示得比默认行高更大一些，可以使用基于 List 的组件的 rowHeight 和 variableRowHeight 属性
Label	text	

(续表)

组　件	默 认 属 性	说　明
NumericStepper	value	
Text	text	默认情况下，已经使用这个组件了
TextArea	text	
TextInput	text	

要特别注意这些组件的默认属性，因为这是组件在 dataProvider 中查找的字段。举例来说，如果有以下的 DataGrid：

```
<mx:DataGridColumn headerText="Picture" itemRenderer="mx.controls.Image"/>
```

Image 组件会假定数据中有一个名为 source 的字段。代码清单 9-12 所示的数据可以满足这个要求。

代码清单9-12　Image组件假设数据中包含source字段

```
public var myAC:ArrayCollection = new ArrayCollection([
      {name:"Jon Hirschi",source:"assets/jon.jpg"},
      {name:"Tariq Ahmed",source:"assets/tariq.jpg"}]);
```

如代码清单 9-11 所示，其中的字段名是 thumbnail。这意味着已经为组件明确指定了作为其目标的 dataField。dataField 会告诉组件使用设置的值，而不是默认值。

项渲染器针对单元格中的内容提供高度的控制和逻辑处理能力。如果想为用户提供编辑能力，可以使用另一种类似的功能，即项编辑器(item editor)。

9.3　项编辑器

项编辑器可以临时将基于 List 的组件中的单元格转换为一个表单组件，从而允许用户编辑其中的值。从技术角度看，表单其实也是项渲染器，只不过看待它们的角度不同而已。

- **项渲染器**：用于生成默认显示，通常从阅读角度来看。
- **项编辑器**：用于临时调用，通常从编辑角度来看。

下面我们先介绍怎样启用编辑过程，以便使项编辑器及其相关事件捕获用户提供的数据。

9.3.1　启用项编辑功能

支持项渲染器的组件同样也支持项编辑器，但是默认情况下，会禁用项渲染器的编辑功能。为支持编辑处理，必须启用编辑功能。启用编辑功能后，用户通过单击单元格可以编辑其中的数据。

以下是启用项编辑器的过程。

(1) 将组件设置为可编辑(即设置 editable 属性为 true)。这个设置可以应用于整个组件、

一个单元格或一列。

(2) 当焦点转移到可以编辑单元格中时(通过鼠标单击，或者按 Tab 键或回车键)，项编辑器临时取代项渲染器。

(3) 项编辑器与用户交互以便捕获某些数据，然后基于用户操作(按回车键或 Tab 键，或者单击单元格外部区域)自行终止。

(4) 在项编辑器终止时，会把从已编辑的单元格捕获的值传递给基于 List 的组件以便显示。更重要的是，提供数据的原始变量(即 dataProvider 属性指向的属性)也会被新值更新。

代码清单 9-13 所示为一个最基本的示例。

代码清单9-13 使用项编辑器的简单示例

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Script>
   <![CDATA[
      import mx.collections.ArrayCollection;
      [Bindable]
      public var myAC:ArrayCollection  = new ArrayCollection([
      {name:"Jon Hirschi", email:"jon@iscool.com"},
      {name:"Tariq Ahmed", email:"tariq@iscool.com"}]);
   ]]>
</mx:Script>

<mx:DataGrid id="dg" width="150" height="100" dataProvider="{myAC}"
   editable="true">
   <mx:columns>
      <mx:DataGridColumn headerText="Name" dataField="name"/>
      <mx:DataGridColumn headerText="EMail" dataField="email"
   editable="false"/>
   </mx:columns>
</mx:DataGrid>

</mx:Application>
```

默认设置所有列都可以编辑

覆盖DataGrid的设置；
强制将此列设置为只读

当用户单击某个姓名时，相应的单元格切换为可以编辑的状态，用户即可就地编辑其中的值(图 9-6)。

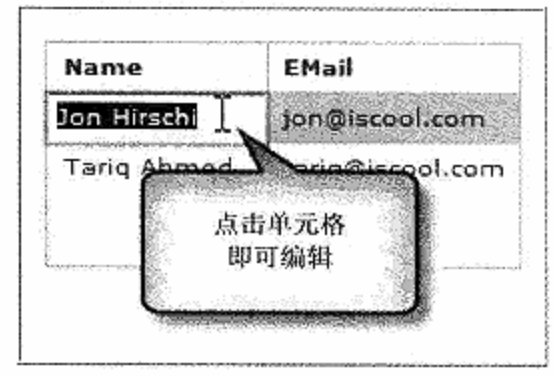


图 9-6 通过设置可编辑字段允许用户在单元格中编辑数据

在前面的示例中，我们使用 Text 组件作为默认的项渲染器。那么，读者是否能猜到默认的项编辑器是什么呢？如果你回答 TextInput，那么完全正确。不过，并非只能使用 TextInput，可以随意修改并创建自定义的项编辑器。

9.3.2 创建项编辑器

创建项编辑器与创建项渲染器类似，不过要配置另外一些特性。

- 创建一个单独的 MXML 文件，以便不同应用程序调用。

- 创建一个公共变量，以便基于 List 的组件知道新值保存在哪里。一定要保证这个变量的类型与原始数据类型兼容(数值对应数值，字符串对应字符串，日期对应日期，等等)。
- 在项编辑器被初始化之后，应该让表单输入默认使用当前值(及公共变量)。

代码清单 9-14 创建了一个自定义的项编辑器，代码清单 9-15 调用了这个项编辑器。

代码清单9-14　自定义的项编辑器：myEditor.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml"
   creationComplete="initMe()">
   <mx:Script>
      <![CDATA[
      import mx.collections.ArrayCollection;

      Bindable]
      public var newEmployeeType:String = "";

      [Bindable]
      private var typeAC:ArrayCollection =
            new ArrayCollection([{type:"Employee"},
               {type:"Customer"}]);

      private function initMe():void
      {
         newEmployeeType = data.type;

         for(var i:int = 0; i<typeAC.length;i++)
            if(typeAC[i].type == data.type)
               myCB.selectedIndex = i;
      }

      private function selectionChange():void
      {
         newEmployeeType = myCB.selectedItem.type;
      }

      ]]>
   </mx:Script>
   <mx:ComboBox id="myCB" dataProvider="{typeAC}"
            labelField="type" change="selectionChange()"/>
</mx:HBox>
```

基于List的组件读取的变量

将返回变量初始化为当前值

循环遍历ComboBox的值并默认使用当前值

用户改变选项，将返回变量更新为新值

代码清单9-15　为列使用自定义项编辑器的Helloworld.mxml

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Script>
   <![CDATA[
      import mx.collections.ArrayCollection;
      [Bindable]
      public var myAC:ArrayCollection = new ArrayCollection([
      {name:"Jon Hirschi", email:"jon@iscool.com", type:"Employee"},
```

```
        {name:"Tariq Ahmed", email:"tariq@iscool.com", type:"Customer"}]);
    ]]>
</mx:Script>

<mx:DataGrid id="dg" width="350" height="200" dataProvider="{myAC}"
    editable="true">
    <mx:columns>
        <mx:DataGridColumn headerText="Name" dataField="name"/>
        <mx:DataGridColumn headerText="EMail" dataField="email"
    editable="false"/>
        <mx:DataGridColumn headerText="Type" dataField="type"
                    editorDataField="newEmployeeType"
                    itemEditor="myEditor"/>
    </mx:columns>
</mx:DataGrid>

</mx:Application>
```

指示当前列使用自定义项编辑器

确定项编辑器中的哪个变量包含新值

以上代码归根结底做了两件事：首先，初始化项编辑器以确保显示正确的数据类型；其次，传递用户选定的结果值(图 9-7)。注意，代码中使用 editorDataField 属性确定项编辑器中的变量。当编辑终止后，会使用这个变量的值填充 DataGrid。

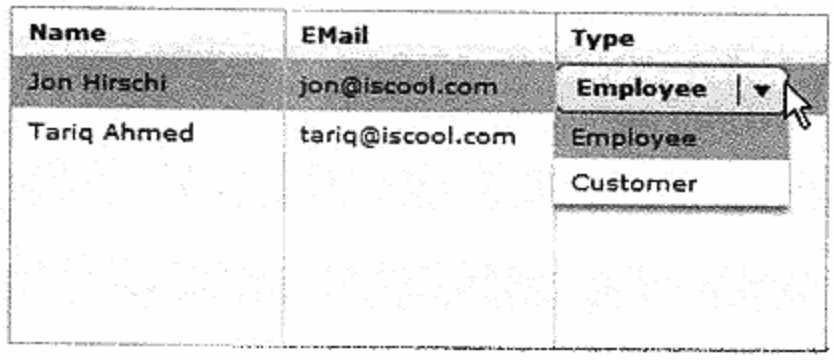


图 9-7 自定义的项编辑器，用于在编辑状态下创建 ComboBox

虽然创建和使用项编辑器的过程可以归结为两步，但这些过程中也有许多可利用的事件。

9.3.3 项编辑事件

在编辑期间，Flex 会触发很多事件。通过截获这些事件，可以添加业务逻辑或参数、执行清理例程或添加检查点。这些事件包括如下。

- itemEditBeginning：在项编辑过程快要开始时分派(dispatch)。由用户鼠标单击单元格、在单元格上按回车键，或者按 Tab 键从一个单元格移动到另一个单元格时触发。通常，可以利用这个事件添加业务逻辑，以决定是否真正启用编辑。
- itemEditBegin：在 itemEditBeginning 之后，数据即将被传到项编辑器时立即触发。因此，利用这个事件可以对数据执行某些操作。
- itemEditEnd：当用户执行结束编辑的操作时发生。此时项编辑器仍然存在，但即将被销毁。利用这个事件可以比较新值和原始值，也可以执行额外的验证。

本章稍后将通过示例说明如何在基于 List 的组件中处理这些事件。下面先来了解这些事件发生时生成的事件对象。

基于 List 的组件不一样，相关的事件对象也不相同。

- ListEvent：由比较简单的列表组件使用，如 List 和 Tree。
- DataGridEvent：由 DataGrid 使用。

- AdvancedDataGridEvent：由 AdvancedDataGrid 使用。

在编写处理这些事件的函数时，使用代码清单 9-16 所示的语法。

代码清单9-16　基于组件处理项编辑事件的3个函数示例

```
import mx.events.ListEvent;
public function handleListEvent(event:ListEvent):void
{
   //logic...
}

import mx.events.DataGridEvent;
public function handleDGEvent(event:DataGridEvent):void
{
   //logic...
}

import mx.events.AdvancedDataGridEvent;
public function handleADGEvent(event:AdvancedDataGridEvent):void
{
   //logic...
}
```

每个事件对象都有自己独特的属性；但它们也有一些共有属性，特别值得关注的是表 9-2 所示的属性。

表 9-2　在项编辑期间生成的事件对象的属性

属　性	类　型	说　　明
columnIndex	Number	列索引，从 0 开始计
currentTarget	Object	引用生成事件的组件。例如，有一个名为 myHList 的 HorizontalList，从中可初始化单元格编辑过程。此时的 currentTarget 会链接回 myHList
itemRenderer	Object	引用当前被编辑单元格中使用的项渲染器。可以通过这个属性确定单元格的当前值(即编辑开始前的值)
reason	String	只读。导致 itemEditEnd 发生的操作(在发生此类操作的情况下)。可能的值有 cancelled(用户取消编辑)、newRow(用户把焦点转移到新行)、newColumn(用户把焦点转移到同一行的另一列)及 other
rowIndex	Number	被编辑单元格所在的行
type	String	发生的事件类型
dataField	String	与被编辑列相关的数据字段。专门针对 DataGrid 和 AdvancedDataGrid
preventDefault	Function	调用它可以中断事件链的函数

请读者注意，不理会这些事件，让它们自生自灭也没有任何问题。不过，必要时也可以从中挑选要处理的事件。下面我们分别介绍如何利用这些事件。

1. 利用 itemEditBeginning 事件

当用户鼠标直接在可编辑单元格上单击，当鼠标停在单元格上时按回车键，或按 Tab

键从一个单元格切换到另一个单元格时，会触发 itemEditBeginning 事件。此时单元格编辑功能尚未启用，因此为确认是否允许编辑的一个好时机。

代码清单 9-17 检查焦点所在记录是否为用户记录。如果是还要检查当前记录是否处于活动状态。如果并非如此，则阻止编辑过程。

代码清单9-17 截获itemEditBeginning事件并添加自定义逻辑

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  backgroundColor="white">
<mx:Script>
   <![CDATA[
      import mx.collections.ArrayCollection;
      import mx.events.DataGridEvent;
      [Bindable]
      public var myAC:ArrayCollection  = new ArrayCollection([
      {name:"Jon Hirschi", email:"jon@iscool.com", type:"Employee",
         isactive:true},
      {name:"Tariq Ahmed", email:"tariq@iscool.com", type:"Customer",
         isactive:false}]);

      private function checkIfAllowed(event:DataGridEvent):void
      {
            var dataItem:Object = event.itemRenderer.data;
         if(event.dataField == "type" && dataItem.isactive == false)
            event.preventDefault();
      }
   ]]>
</mx:Script>

<mx:DataGrid id="dg" width="350" height="150" dataProvider="{myAC}"
   itemEditBeginning="checkIfAllowed(event)" editable="true">
   <mx:columns>
      <mx:DataGridColumn headerText="Name" dataField="name"/>
      <mx:DataGridColumn headerText="EMail" dataField="email"
         editable="false"/>
      <mx:DataGridColumn headerText="Type" dataField="type"
                  editorDataField="newEmployeeType"
                     itemEditor="myEditor"/>
   </mx:columns>
</mx:DataGrid>

</mx:Application>
```

处理itemEditBeginning事件 ❶

检查type字段是否存在以及记录是否处于活动状态 ❷

❸ 停止项编辑

为 itemEditBeginning 使用事件处理程序

代码清单 9-17 中的 DataGrid 会在编辑过程即将开始时调用一个函数❹。checkIfAllowed 函数❶首先检查是否有一个名为 type 的字段❷，如果是，则继续检查当前记录是否处于活动状态。如果任一项测试失败，会调用事件的 preventDefault 函数中断过程❸。

如果 checkIfAllowed 函数确定条件得到满足，则允许过程继续前进，继续处理队列中的下一个事件——itemEditBegin。

2. 利用 itemEditBegin 事件

如果事件经过 itemEditBeginning 阶段，会继续处理 itemEditBegin 事件。此时，单元格编辑功能已经启用，而且即将创建项编辑器。但在此之前，可以操作数据或视情况动态调

用其他项编辑器。

事实上，itemEditBegin 并不常用，属于高级主题。为利用它，必须重新生成默认行为，然后再添加自定义逻辑。在此，我们略过细节不谈，只介绍整个过程中涉及的几个要点。

- 调用事件对象的 preventDefault()函数，以免 Flex 执行默认事件链。
- 在基于 List 的组件上调用 createItemEditor 函数，以便在目标单元格中创建一个项编辑器。
- 将项渲染器的 data 属性指向项编辑器。
- 然后根据需要操作数据。

在代码清单 9-18 中，在用户尝试编辑 type 字段时操纵此字段(在 isactive 返回 false 时将 type 字段设置为 Deactivated)。

代码清单9-18　利用itemEditBegin事件

```
<mx:Script>
   <![CDATA[
      import mx.collections.ArrayCollection;
      import mx.events.DataGridEvent;
      [Bindable]
      public var myAC:ArrayCollection  = new ArrayCollection([
      {name:"Jon Hirschi", email:"jon@iscool.com", type:"Employee",
         isactive:true},
      {name:"Tariq Ahmed", email:"tariq@iscool.com", type:"Customer",
         isactive:false}]);

      private function modifyIfNecessary(event:DataGridEvent):void
      {
         if(dg.columns[event.columnIndex].dataField=="type")
         {
            event.preventDefault();

            dg.createItemEditor(event.columnIndex,event.rowIndex);

            dg.itemEditorInstance.data = dg.editedItemRenderer.data;

            if(dg.editedItemRenderer.data.isactive == false)
               dg.itemEditorInstance.data.type = "Deactivated";
         }
      }
   ]]>
</mx:Script>

<mx:DataGrid id="dg" width="350" height="150" dataProvider="{myAC}"
   itemEditBegin="modifyIfNecessary(event)" editable="true">
   <mx:columns>
      <mx:DataGridColumn headerText="Name" dataField="name"/>
      <mx:DataGridColumn headerText="EMail" dataField="email"
         editable="false"/>
      <mx:DataGridColumn headerText="Type" dataField="type"
                  editorDataField="newEmployeeType"
                     itemEditor="myEditor"/>
   </mx:columns>
</mx:DataGrid>
```

当前列是不是type?

阻止默认的itemEditBegin功能

在单元格中创建项编辑器

取得单元格的值并赋给项

如果账户处于不活动状态则强制设置type字段的值

itemEditBegin 事件完成后，用户开始编辑值；直至用户提交编辑后的值，才会触发 itemEditEnd 事件。

3. 利用 itemEditEnd

itemEditEnd 事件是此过程的最后一个阶段，它在用户编辑完数据时发生；而利用这个机会可以做一些编辑后的处理工作。利用 itemEditEnd 事件的常见场景有很多种，其中包括：

- 验证输入的数据与任何已定义的参数是否兼容。
- 格式化数据。
- 将编辑结果传递给应用程序的其他部分。
- 把修改后的数据永久写入后端数据库中。
- 强制单元格继续处于被编辑过状态。
- 编辑完当前单元格后，阻止用户通过按回车键或 Tab 键移动到下一个单元格。

利用 itemEditEnd 事件的第一个问题是搞清楚终止编辑过程的操作是什么。如前所述，可以通过查看 event.reason 属性了解原因。如果原因是 cancelled，则无需后续处理。

此时，如果需要将焦点保持在当前单元格内，要使用一个名为 callLater 的函数，通过它可以指定另一个函数，以便在当前函数退出后调用。

代码清单 9-19 利用 itemEditEnd 事件来检查 email 字段，如果其中的数据无法通过有效性验证，则继续保持单元格的编辑模式，直至用户输入正确的数据为止。

代码清单9-19 利用itemEditEnd事件验证编辑后的值

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Script>
<![CDATA[
   import mx.collections.ArrayCollection;
   import mx.events.DataGridEvent;
   import mx.validators.EmailValidator;
   import mx.controls.TextInput;
   [Bindable]
   public var myAC:ArrayCollection = new ArrayCollection([
   {name:"Jon Hirschi", email:"jon@iscool.com"},
   {name:"Tariq Ahmed", email:"tariq@iscool.com"}]);

   private function onEditEnd(event:DataGridEvent):void
   {
      //if you needed to validate different cells differently...
      if(event.dataField == 'email')      ❶ 是电子邮件字段吗？
      {
         var fCell:Array=[event.columnIndex,event.rowIndex];
         var newData:String =      ❷ 将新值告知用户
           TextInput(event.currentTarget.itemEditorInstance).text;

         var emailVal:EmailValidator =                    ❸ 创建 EmailValidator
            new EmailValidator();                          实例并验证新值
         var valResult:* = emailVal.validate(newData);
         if(valResult.type == "invalid")      ❹ 检查验证是否失败
         {
            event.preventDefault();      ❺ 阻止事件传播
            callLater(maintainEdit,fCell);
         }                                          在当前事件结束后
      }                                          ❻ 字段继续保持编辑模式
   }
```

```
        private function maintainEdit(colIndex:int,rowIndex:int):void
        {
            var editCell:Object = {columnIndex:colIndex, rowIndex:
                rowIndex};
            //This will invoke the datagrid's itemEditBegin
              dg.editedItemPosition = editCell;
        }

    ]]>
</mx:Script>

<mx:DataGrid id="dg" width="350" height="150" dataProvider="{myAC}"
    itemEditEnd="onEditEnd(event)" editable="true">
    <mx:columns>
        <mx:DataGridColumn headerText="Name" dataField="name"/>
        <mx:DataGridColumn headerText="EMail" dataField="email"/>
    </mx:columns>
</mx:DataGrid>

</mx:Application>
```

❼ 让单元格保持在编辑模式

注意：

CFLEX.Net 上有一个与此类似的更复杂例子，地址为 http://www.cflex.net/showFileDetails. cfm?ObjectID=723。

在这个示例中，我们定义了一个函数来处理 itemEditEnd 事件。这个函数首先会检查被编辑的是不是 email 字段❶；如果不是，则切换为默认行为并继续。

如果是电子邮件字段，则捕获并比较用户编辑后的新值❷。注意，此时用户还没有提交值，因此容易使用 EmailValidator 验证其是不是有效的电子邮件地址❸。如果不是❹，则需要阻止事件传播❺。在结束当前事件后，还要让 Flex 继续调用 maintainEdit 函数❻，该函数会重新启用同一单元格中的编辑过程❼，并要求用户输入有效的电子邮件地址。

到目前为止，我们一直将项渲染器和项编辑器视为两个独立组件。但是，它们之间有许多共通之处，利用这一点可以创建出既是项渲染器又是项编辑器的组件。

9.4　组合项渲染器和项编辑器

也可以创建兼具项渲染器和项编辑器功能的混合型组件。这种组件可以作为将阅读视图切换为编辑视图的开关，或者可以让某个字段保持编辑模式。

例如，很多基于 Web 的电子邮件程序都会提供这样一种功能：在每个电子邮件条目旁边显示一个复选框，让用户可以选择要进行批量处理的全部电子邮件。这是一种使用户快速执行多个更新的不错方式。

下面我们以代码清单 9-14 中的示例为原型，介绍如何创建这种混合型组件。代码清单 9-14 允许用户通过 ComboBox 修改用户类型，而且创建了一个随时可编辑的字段。将代码清单 9-14 作为起点，请查看以下代码段：

```
<mx:DataGridColumn headerText="Type" dataField="type"
            editorDataField="newEmployeeType" itemEditor="myEditor"/>
```

我们把这行代码修改为如下所示。

```
<mx:DataGridColumn headerText="Type" dataField="type"
   rendererIsEditor="true" editorDataField="newEmployeeType"
   itemRenderer="myEditor"/>
```

修改的目的是激活 ComboBox，并让它在每一行都显示出来，如图 9-8 所示。

图 9-8 ComboBox 同时作为项编辑器和项渲染器

使用这种技术，可以让应用程序的条目始终处于编辑模式下，它消除了渲染器与编辑器之间的差别(更确切地讲，用户随时可以编辑相应字段)。

这些项渲染器和编辑器能够适应一般的需要。下面我们介绍专门为 AdvancedDataGrid 设计的高级项渲染器。

9.5 高级项渲染器

AdvancedDataGrid 与其他基于 List 的组件一样也支持项渲染器，但它支持的是更高级的渲染。AdvancedDataGrid 支持项渲染器跨列使用或在同一列中使用多个项渲染器，还支持可以插入行的项渲染器(后者可用于实现数据汇总或摘要行)。

总之，您可以使用这些功能完成很多操作。接下来简单地介绍用法。

9.5.1 AdvancedDataGridRendererProvider

AdvancedDataGridRendererProvider 也许是 Flex 中名字最长的组件了！它使在 Advanced DataGrid 中使用自定义项渲染器成为可能。表 9-3 所示为这个组件的属性。

表 9-3 AdvancedDataGridRendererProvider 的属性

属　性	类　型	说　明
column	Object	引用 AdvancedDataGridColumn。通过绑定，将指定高级渲染器引用的列的 id。例如，可以使用字段{myCol}代替 columnIndex。即可以利用这个属性通过列名引用列
columnIndex	Number	与 column 类似，只不过是通过数字引用列。第一列的索引值为 0
columnSpan	Number	类似于 HTML 表格中的 colspan 属性；用于指定要跨过的列数。这个属性是可选的，其默认值为 1；设置为 0 表示跨过所有列
dataField	String	用于确定将数据提供者中的哪个字段提供给项渲染器(如果有)。是一个可选属性

(续表)

属　性	类　型	说　明
renderer	Component	要调用的项渲染器的名称。是一个必需的属性
rowSpan	Number	类似于 HTML 表格中的 rowspan 属性，用于指定要跨过的行数。这个属性是可选的，其默认值为 1

带有这些属性的 AdvancedDataGridRendererProvider 要嵌套在<rendererProviders>标签中，而后者则嵌套在 AdvancedDataGrid 中。代码清单 9-20 所示为这一结构。

代码清单9-20　高级渲染器与列嵌套在同一层次

```
<mx:AdvancedDataGrid>
   <mx:columns>
      <mx:AdvancedDataGridColumn dataField="col1"/>
      <mx:AdvancedDataGridColumn dataField="col2"/>
     Etc…
   </mx:columns>

   <mx:rendererProviders>
      <mx:AdvancedDataGridRendererProvider [properties]/>
      <mx:AdvancedDataGridRendererProvider [properties]/>
     Etc…
   </mx:rendererProviders>
</mx:AdvancedDataGrid>
```

代码清单 9-20 是一个整体性概念。从实现角度说，也不过就是引用高级项渲染器应用到的列。

9.5.2　引用列

引用列的方式有两种：使用列的 id 属性和指定列的位置。

1. 通过 ID

如果想使用名称方法来指定应用渲染器列，可以使用 column 属性。代码清单 9-21 所示为如何绑定名为 nameCol 的列。

代码清单9-21　通过列名应用高级项渲染器

```
<mx:AdvancedDataGrid dataProvider="{myAC}">
   <mx:columns>
      <mx:DataGridColumn id="nameCol" headerText="Name"
         dataField="name"/>
   </mx:columns>

   <mx:rendererProviders>

      <mx:AdvancedDataGridRendererProvider column="{nameCol}"
        renderer="myRenderer"/>
   </mx:rendererProviders>
</mx:AdvancedDataGrid>
```

这种方式适用于列的位置有可能改变的情况：无论目标列移至何处，项渲染器都可以有效工作。

2. 通过索引

如果想通过列的位置来引用列，需要使用 columnIndex 属性。代码清单 9-22 指示 AdvancedDataGridRendererProvider 将自身应用到第一列。

代码清单9-22 为第一列应用高级项渲染器

```
<mx:AdvancedDataGrid dataProvider="{myAC}">
   <mx:columns>
      <mx:DataGridColumn id="nameCol" headerText="Name"
         dataField="name"/>
   </mx:columns>

   <mx:rendererProviders>
      <mx:AdvancedDataGridRendererProvider columnIndex="0"
         renderer="myRenderer"/>
   </mx:rendererProviders>
</mx:AdvancedDataGrid>
```

这种方式适合必须按照列的位置应用项渲染器的情形。例如，有的项渲染器只适用于第二列，但它不关心该列的内容。或者也可以动态设置一个变量，并通过该变量设置 columnIndex。

确定高级项渲染器与哪个列关联是最基本的用法，还可以跨列使用它。

9.5.3 跨列使用

要跨列使用，需要使用 columnSpan 属性。columnSpan 属性与 HTML 表格<td>标签中的 colspan 属性作用类似。代码清单 9-23 显示如何使用 columnSpan，方法是告知高级项渲染器跨越两列(从第二列开始)。

代码清单9-23 跨越两列使用高级项渲染器

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Script>
   <![CDATA[
      import mx.collections.ArrayCollection;
      [Bindable]
      public var myAC:ArrayCollection  = new ArrayCollection([
      {name:"Jon Hirschi", email:"jon@iscool.com", type:"Employee"},
      {name:"Tariq Ahmed", email:"tariq@iscool.com", type:"Customer"}]);
   ]]>
</mx:Script>

<mx:AdvancedDataGrid width="350" height="120" dataProvider="{myAC}">
```

```
    <mx:columns>
        <mx:AdvancedDataGridColumn headerText="Name" dataField="name"/>
        <mx:AdvancedDataGridColumn headerText="Email" dataField="email"/>
        <mx:AdvancedDataGridColumn headerText="Type" dataField="type"/>
    </mx:columns>
    <mx:rendererProviders>
        <mx:AdvancedDataGridRendererProvider columnIndex="1"
            columnSpan="2" renderer="myRenderer"/>
    </mx:rendererProviders>
</mx:AdvancedDataGrid>
</mx:Application>
```

运行这个应用程序后，会发现每个人的电子邮件会出现在相邻的列，如图 9-9 所示。

Name	Email	Type
Jon Hirschi	jon@iscool.com	
Tariq Ahmed	tariq@iscool.com	

图 9-9　使用高级项渲染器让电子邮件数据跨越两列显示

接下来，我们要解决的问题是怎样实现通行显示数据。

9.5.4　实现通行显示

要实现通行显示，可以将 columnIndex 属性设置为 0。在显示普通数据时，这种技术没有太大用处，因为实际上可以创建一列。但是在显示层次化数据时，如果想添加摘要行，这种技术就非常方便了，如代码清单 9-24 和代码清单 9-25 所示。

代码清单9-24　通行显示层次化数据：HelloWorld.mxml

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

<mx:XMLList id="myXML">
    <friends name="Friends">
        <friend name="Jon Hirschi" email="j_hirschi@domain.com">
            <info>
                <summary>
                    47 emails sent in the last 24hrs
                </summary>
            </info>
        </friend>
        <friend name="Tariq Ahmed" email="t_ahmed@domain.com">
            <info>
                <summary>
                    12 emails sent in the last 24hrs
                </summary>
            </info>
        </friend>
    </friends>
</mx:XMLList>
```

```
<mx:HierarchicalData source="{myXML}" id="myHD"/>

<mx:AdvancedDataGrid dataProvider="{myHD}" width="300">
   <mx:columns>
      <mx:AdvancedDataGridColumn dataField="@name"
         headerText="Contacts"/>
      <mx:AdvancedDataGridColumn dataField="@email" headerText="Email"/>
   </mx:columns>
   <mx:rendererProviders>
      <mx:AdvancedDataGridRendererProvider dataField="summary"
         columnIndex="0" columnSpan="0" renderer="myRenderer"/>
   </mx:rendererProviders>
</mx:AdvancedDataGrid>

</mx:Application>
```

代码清单9-25 用于显示摘要信息的项渲染器：myRenderer.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Label xmlns:mx="http://www.adobe.com/2006/mxml" width="100%"
   text="{data.summary}"/>
```

如图 9-10 所示，在显示层次化数据的情况下，这种技术正好派上用场。

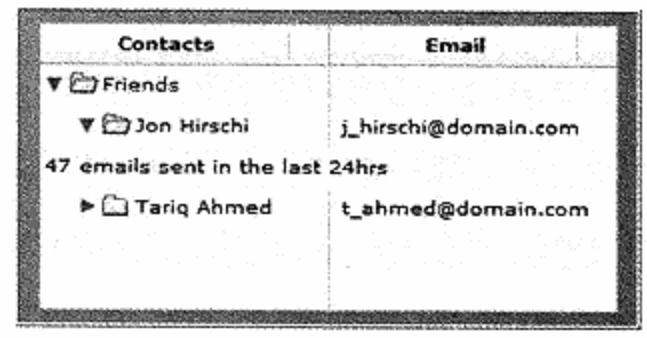


图 9-10 通行显示适用于层次化数据

请读者记住，无论作为普通的项渲染器，还是高级项渲染器使用，渲染器本身没有什么不同。

对于 Flex 开发新手来说，在使用基于 List 的组件时，早晚还会碰到这类问题：想要依据某个条件来显示数据中的一个子集。这时候，就该用到筛选函数了。

9.6 筛选函数

筛选函数并不是特定于列表的函数，它们是集合特性。筛选函数可以从集合中筛选出数据子集。筛选函数的一个常见用法是与基于 List 的组件一起使用。与其耗费精力创建临时数组来保存数据子集，不如让集合替我们完成同样的工作。

假设要创建一个汇总报表，让用户可以按照某个条件显示数据。代码清单 9-26 通过筛选函数实现了这样的功能。

代码清单9-26 使用筛选函数控制报表显示

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
   <mx:Script>
      <![CDATA[
         import mx.collections.ArrayCollection;
         [Bindable]
         public var salesAC:ArrayCollection = new ArrayCollection([
         {name:"Canada",revenue:295323},
         {name:"United States",revenue:982832},
         {name:"England",revenue:109283},
```

```
        {name:"Brazil",revenue:12495},
        {name:"India",revenue:597232},
        {name:"China",revenue:682011}
        ]);

        public function filterFunc(item:Object):Boolean
        {
            if(item.revenue >= salesRange.values[0] && item.revenue <=
                salesRange.values[1])
                return true;
            else
                return false;
        }

        public function filterSales():void
        {
            salesAC.filterFunction=filterFunc;
            salesAC.refresh();
        }
    ]]>
</mx:Script>

<mx:Panel width="400" height="300" title="Sales Dashboard">
    <mx:HSlider change="filterSales()" id="salesRange" width="100%"
        thumbCount="2"
            labels="['0','250000','500000','750000','1000000']"
            tickInterval="50000" height="50"
                maximum="1000000"/>
    <mx:DataGrid id="dg" width="100%" height="100%"
        dataProvider="{salesAC}">
        <mx:columns>
            <mx:DataGridColumn headerText="Country" dataField="name"/>
            <mx:DataGridColumn headerText="Revenue" dataField="revenue"/>
        </mx:columns>
    </mx:DataGrid>
  </mx:Panel>
</mx:Application>
```

❶ 检查记录与条件是否匹配

记录与条件匹配

记录与条件不匹配

❷ 应用筛选函数

❸ 设置ArrayCollection的filterFunction属性

❹ 应用与筛选函数匹配的记录

❺ 让用户筛选值的范围

以下是代码清单 9-26 实现的主要功能。

- 通过 HSlider❺捕获用户设置的筛选值范围。
- 在 HSlider 的值发生变化时，调用 filterSales 函数❷。
- filterSales 告诉 ArrayCollection(salesAC)将 filterFunc❸作为筛选函数。
- filterSales 调用 salesAC.refresh❹，要求 salesAC 使用 filterFunc 筛选每条记录。
- filterFunc 检查❶当前评估的记录是否介于 HSlider 的值范围之内。

图 9-11 所示为这个示例的结果，其中只有各国销售收入介于指定滑块范围内时才会显示出来。

也可以为值使用 ComboBox 组件甚至普通的 TextInput 控件。只要你认为方便即可；所有操作完全发生在为 ArrayCollection 的 filterFunction 属性指定的函数中。

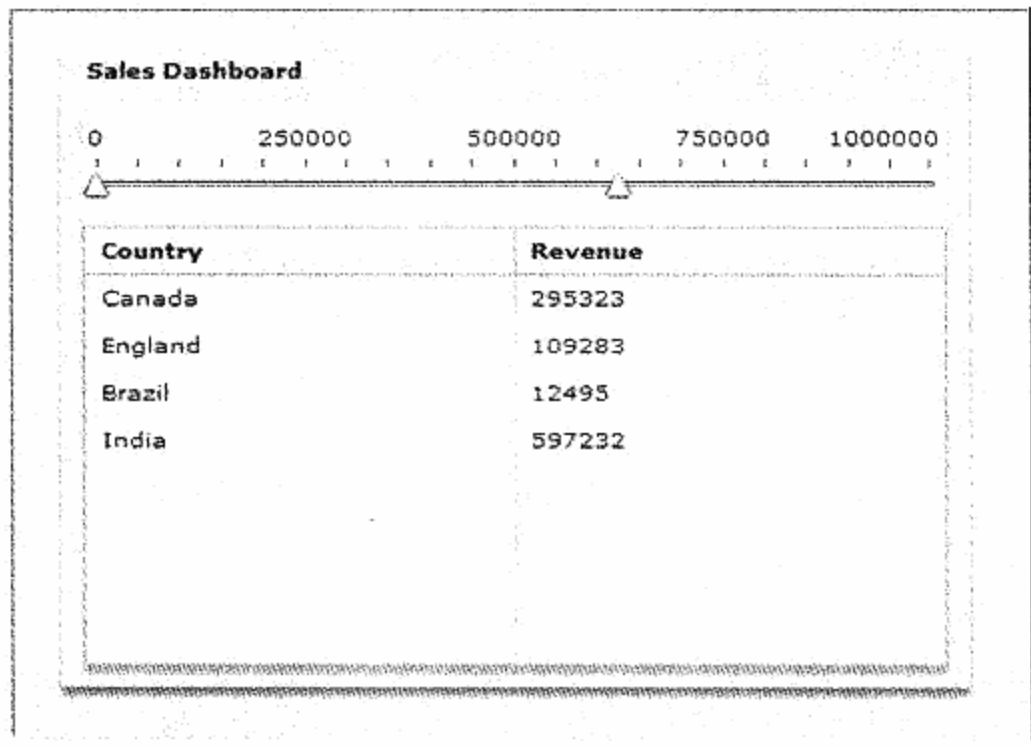


图 9-11 随着用户移动滑块，DataGrid 中的内容会通过筛选函数发生相应变化

9.7 小结

基于 List 的组件可以显示大量数据，而且无需开发人员太多的干预。但是，增强功能的需求会使问题也会变得十分复杂。本章全面介绍与自定义列表相关的技术，通过这些技术能够有效解决开发中遇到的实际问题。

本章首先介绍使用标签函数自定义基于列表的组件的显示。标签函数易于初始化，而且常常与格式化程序结合使用。但是标签函数的功能具有一定的局限性。

自定义列表组件的主要工具是项渲染器。项渲染器为控制显示逻辑赋予最大的灵活性。与项渲染器互为补充的是项编辑器，项编辑器可以让用户在行内完成编辑。项渲染器和项编辑器都有一些事件，可以利用这些事件介入编辑过程，并注入自定义逻辑和检查点。

Flex 3 引入基于列表的高级组件(如 AdvancedDataGrid)，它们以更高级的方式(例如跨列)支持同样的项渲染器。

在传统 Web 应用程序领域中，用户改变选择标准(如数据范围、销售数字或地理区域)会导致应用程序重新查询后台数据库，进而更新相应数据的子集。而在 Flex 开发中，则无需这种往返数据库的多余过程；只要把数据全部加载到前端(在逻辑上合理的情况下)，用户就可以在客户端通过筛选函数实现数据分析。

前几章介绍了很多与事件有关的交互操作，本章我们重点讨论与项编辑器相关的事件。而且希望读者时刻铭记，Flex 是事件驱动的应用程序框架。既然读者已经了解在各种情况下使用事件的方法，下一步将深入了解事件的工作原理。

第II部分

应用程序流与结构

本书第I部分关注的是构成应用程序的基本部件。第II部分将把关注焦点转移到应用程序的内部流程与结构。

Flex是事件驱动的开发环境。第I部分介绍使用事件的基本机制，本部分中的9章将深入探讨事件模型的工作原理和自定义事件的方式。充分利用事件的方式，可以为Flex应用程序增加导航与交互功能。

如果没有后端系统支持数据交换，Flex应用程序不会有多大用处。第II部分将介绍作为应用程序流程中一个部分的来回传输数据的机制。

Flex最擅长处理的数据格式是XML。有了Flex对ECMAScript for XML(E4X)的支持，开发人员就可以使用ActionScript像操作本地数据类型一样操作XML数据。

理解了应用程序的构件和流程之后，我们将关注应用程序的结构。如何组织应用程序将决定代码是否容易管理、维护和重用。Flex支持许多组织方式，本部分最后讨论的内容是通过模块封装逻辑代码。

第 10 章 事　件

10

本章内容：

- 深入理解事件
- 侦听事件
- 分派事件
- 创建自定义事件

理解事件是做好 Flex 开发，或者说是设计出引人注目的 Flex 应用程序的关键所在。事件是 Flex 框架中的强大工具，也是它的一个主要特性。

事件表面看起来简单，却能为应用程序提供极为强大的沟通机制。我们反复强调 Flex 是事件驱动的框架这一事实已经好几次了。打个比方，事件就像是应用程序的中枢神经系统，而在这个四通八达的系统中，随时随地都会产生和传递着各种信息。可以毫不夸张地说，理解了事件就理解了 Flex。

本章首先介绍事件系统及它在 Flex 应用程序中扮演的极其重要的角色。

10.1 事件系统

读者如果使用过 ColdFusion、.Net、ASP、PHP、Perl 或 Python 等 Web 技术，那么一定知道信息是基于请求和响应模型传输的。在这个模型中，远程服务器等待客户端(用户)的请求，然后收集并处理与请求相关的信息。处理完毕后，服务器再以响应方式将数据发送回客户端。这样，一个事务就结束了(直至客户端请求下一个事务)。

Flex 框架的数据传输模式有所不同。没有发送请求和接收响应，Flex 组件会按照事先的设计，耐心等待并侦听事件。当某个组件侦听到与它有关的事件时，该组件将执行设计好的任务，然后等待下一次事件发生。

在 Flex 框架中，触发事件的目的是为了响应各种用户输入。而所有事件的主要发起人也是用户。单击按钮、移动鼠标或选择下拉菜单中的一项，都会在不经意间触发事件。当事件发生时，应用程序会做出适当的响应。

对用户而言，Flex 的这个模型与标准的请求/响应模型没有什么区别(当他们单击网站中的链接时，Web 服务器也会提供一个新页面作为响应)。而对 Flex 来说，区别在于用户操作在后台透明化了。这意味着开发人员可以根据用户操作并行执行任务，这些任务之间

相互没有依赖关系。也就是说，一方面后台活动在透明地进行；另一方面应用程序仍然可以实时地与用户交互并向用户提供反馈(在顺序执行代码的情况下，用户则必须等待每个响应依次完成)。

此外，在发生请求新信息的事件时，Flex 应用程序会向目标服务器发送异步请求(发送请求之后，应用程序仍然继续处理其他任务)。至于响应，什么时候返回，什么时候再处理(无论是 5 秒钟，还是 5 分钟)。

与此形成对照的是，在向 Web 服务器发送请求后，查看 HTML 页面的传统客户端应用程序(如 IE 或 Firefox)会一直等在那里，直到接收完响应之后，才将结果显示出来或允许用户继续其他交互操作。

在 Flex 应用程序内部，执行过程更像是对客户端应用程序进行程序设计。有关细节稍后再做介绍；在此之前，我们先来看与邮局非常类似的事件系统。

10.1.1 事件系统——类似于邮政服务

函数、组件和类之间都需要借助事件系统通信和传递数据。为了表达得更直观一些，我们可以把事件系统比作邮政服务。邮局也是在百姓、公司和组织间传递数据(以书信和包裹的方式)。

邮递员会按照既定路线往返于邮局和我们的信箱之间取送信件和包裹。而在应用程序中，事件就是被取送或传递的信件或包裹，它们往返于构成程序的元素之间。与信件类似，事件会通知接收者当前发生了什么事情。

继续这个类比，当我们想通过邮局寄送包裹时，要将物品装箱并贴上标签，写明这个包裹的寄件人及地址、收件人和地址，有时候还要注明包裹里装了什么物品。

Flex 中的事件与包裹非常类似，而事件传送系统就像是邮递员。当事件经过正关注或侦听某些事件的组件时，组件就有可能检查“包裹”的内容和来源，看它是不是自己正在侦听的事件。

拿邮局做类比有助于我们理解事件系统，但是它们之间也有不同之处。

10.1.2 事件发送系统

事件发送系统与邮局的区别在于它是把事件发送给每一个组件，而且掌握 Flex 应用程序中事件的发送路径非常重要。事件的发送路径可以决定哪个组件能够接收到事件，而哪个组件不能。

但这次的情况并不像我们想象的那样。如果严格按照邮局的类比，读者可能会认为如果某个组件触发并分派(dispatch)了一个事件，该组件一定是事件的发源地。但是，这个逻辑反过来也成立——发源于应用程序根级组件(Flex 应用程序的最上层)的事件被发送到要交付的组件，然后又被发送回应用程序根级组件，如图 10-1 所示。

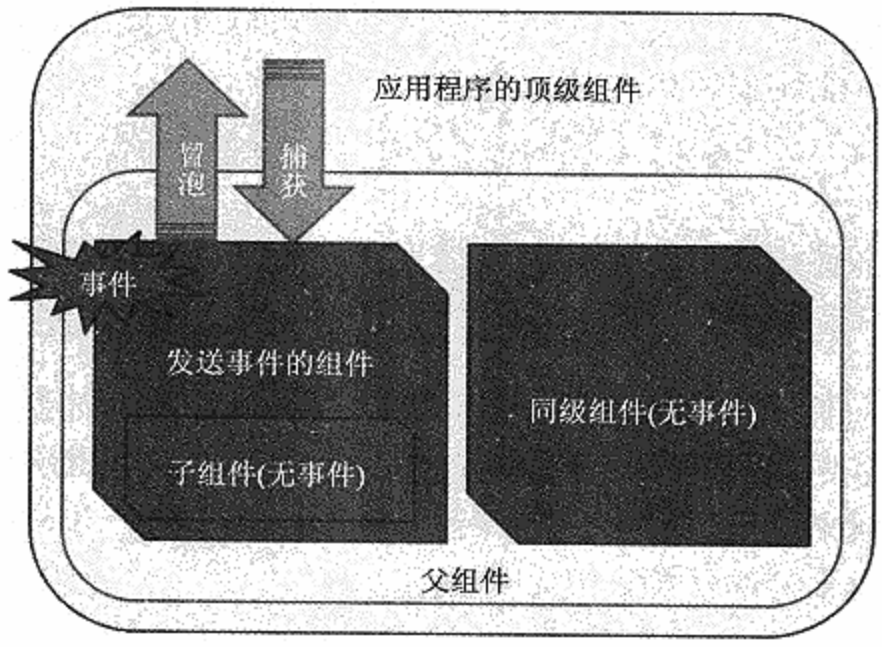


图 10-1　Flex 应用程序事件的父/子关系

在需要广播警告事件时，事件会沿父组件(自最外层开始)向下传播到发出警告的组件。事件只会经过父树(及其中专门侦听该事件的组件)。如图 10-1 所示，组件的父组件会接收到事件通知，而子组件及同级组件都不会接到通知。

从应用程序的角度讲，当组件分派事件时，该事件可能会冒泡或不冒泡。如果冒泡，事件就会经过父组件链，一直传送到应用程序的根级组件。每个父组件都可以在自己的层次上侦听应用程序事件并按照需要转发这些事件；或者调用其子组件的方法以通知它们采取某些行动。

这是 Flex 事件系统真正的强大之处——开发人员可以创建和传播自定义事件。充分利用自定义事件系统，可以最大限度地解耦逻辑并优化使用该逻辑的组件，进而可以实现最大程度的代码重用。

在初步了解应用程序如何传送事件之后，我们把事件从开始到结束的整个行程分成几个阶段。

10.1.3　设置事件传播的阶段

可以将事件行程(从应用程序根级组件到分派组件，再到根级组件)分为几个阶段。根据事件在传播过程中的位置，可以分为如下 3 个阶段。

- **捕获(Capture)**：应用程序将事件发送到分派组件(dispatcher)。
- **冒泡(Bubbling)**：事件从分派组件发送回应用程序。
- **目标(target)**：即事件已经到达其目标(即分派组件)。

在这 3 个阶段中，冒泡阶段用得最多。捕获阶段使用较少，除非需要检查并事先截获事件。

通过事件对象的 Event.eventPhase 属性可以确定事件所处的阶段。这个属性可能包含代表下列 3 个对象常量的整数值。

- `Event.CAPTURING_PHASE:uint = 1`
- `Event.AT_TARGET:uint = 2`
- `Event.BUBBLING_PHASE:uint = 3`

在引用或监视这些阶段时，可以使用整数值，也可以使用常量。不过，为保证代码容易被人看懂，只要有可能还是使用对象常量为好。

注意：

在捕获阶段侦听事件比在冒泡阶段的速度慢一些。如果在捕获阶段侦听的事件过多，应用程序很可能因此而运行缓慢。通常，创建的大多数事件侦听器都在冒泡阶段使用。

可以把这几个事件阶段想象为在某个点之前、期间和之后。捕获阶段在事件被送达分派组件之前。由于大多数组件在冒泡期间(或之后)检查事件，因此大体上可以通过在捕获阶段侦听并停止其传播来停止事件。这个问题将在 10.3.3 节进一步解释。

接下来，我们先看事件的结构以及如何发送事件。

10.2 发送和接收事件

Flex 事件由下列信息段构成。

- Event.target：分派组件。
- Event.currentTarget：当前包含并检查事件的组件。
- Event.type：识别事件类型的字符串，即事件是单击事件(单击按钮)、鼠标事件(移动鼠标)还是选择事件(选择某个项)。事件有很多类型，每种类型都有自己独特的信息；但是，每个事件都包含这里提到的共性信息。
- Event.eventPhase：事件当前所处的阶段。

“包裹”中也可能包含其他信息，但至少包含上述信息。

下面我们来体验由单击鼠标触发的单击事件，如代码清单 10-1 所示。

代码清单10-1 单击事件示例(MXML)

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
 layout="absolute">

 <mx:Button label="Click button!"
  click="clickFunction(event)"/>          | 单击事件
 <mx:Script>
    <![CDATA[
      import mx.controls.Alert;

      public function clickFunction(event:Event):void
      {
        var message:String = "you clicked me! ";
        message += event.target.label;          | 调用clickFunction()
        Alert.show(message,"Event Test");       | 函数
      }
    ]]>
  </mx:Script>
</mx:Application>
```

用户单击按钮时，会生成一个单击事件，该事件随即调用 clickFunction()。每种对象都包含自己独有的一组事件，而且会在适当的时候分派事件。如代码清单 10-1 所示，用户单

击按钮会导致分派单击事件。

虽然这个示例展示的是在 MXML 中侦听事件，但在 ActionScript 中通过使用 addEventListener()函数也能达到侦听事件的目的。

10.2.1　添加事件侦听器

使用 ActionScript 的 addEventListener()函数不仅可以对事件进行更精细地控制，而且也是在 ActionScript 代码中侦听组件分派的事件的唯一途径。使用 ActionScript 为组件添加的事件侦听器可以被移除，但不能在 MXML 中移除。

在某些环境中，需要使用 addEventListener()函数。例如，对于通过 ActionScript 在运行时创建的对象，也必须其后在运行时为它们添加事件侦听器。

如果想在冒泡阶段之外的其他阶段侦听事件，也必须使用这个 ActionScript 方法来添加侦听器。为便于参考，表 10-1 所示为事件侦听器的主要属性。

表 10-1　事件侦听器属性及方法参数

参　数	类　型	说　明
event_type	String	(必需)组件侦听的事件类型。可以直接定义事件类型或采用更实用的方式，使用在每个事件对象上定义的事件类型常量
event_listener	Function	(必需)将要响应事件的函数引用或名称
use_capture	Boolean	(可选)组件侦听阶段。如果值为 true，组件在捕获阶段侦听事件。默认值为 false，即在冒泡阶段侦听事件
priority	Integer	(可选)调用侦听器的时间。数值越大，调用时间越早。可以是负值，默认值为 1
weakRef	Boolean	(可选)垃圾回收程序收集和销毁事件侦听器对象的迅捷程度。如果值为 true，表示尽快销毁。默认值为 false，意味着会阻止垃圾回收程序销毁侦听器(具有性能优势，但需要付出占用内存空间的代价)

我们来看代码清单 10-2 中的简单 eventlistener，这个示例展示了通过 ActionScript 在已经实例化的显示对象中添加事件侦听器的过程。

代码清单10-2　使用ActionScript的单击事件示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute" creationComplete="initFunc()">

  <mx:Button id="thisButton" label="Click button!"/>
  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;

      public function initFunc():void
      {
        thisButton.addEventListener("click",clickFunction);
      }

      public function clickFunction(event:Event):void
```

设置在单击事件发生时运行的clickFunction()函数

```
        {
          var message:String = "you clicked me! " +
             event.target.label;
          Alert.show(message,"Event Test");
        }
      ]]>
    </mx:Script>
  </mx:Application>
```

如代码清单 10-1 和代码清单 10-2 所示，通过 MXML 和 ActionScript 添加简单事件侦听器的主要区别在于，是否使用 addEventListener()。

当按钮被单击时，它会分派一个单击事件，进而激活并向 clickFunction()中传递单击事件。在使用 ActionScript 的情况下，需要一个中间函数为按钮添加事件侦听器。

代码清单 10-2 中的 initFunc()就是这个中间函数，它会在应用程序分派 creationComplete 事件时被调用。creationComplete 事件会在应用程序中的每一个 MXML 项都创建完成并即将显示时分派。

注意：

事件侦听器只能在一个阶段侦听事件。例如，把事件侦听器添加到按钮单击事件的捕获阶段后，该侦听器就只在捕获阶段侦听事件。如果需要同时侦听捕获和冒泡阶段，必须专门为对应的阶段添加第二个事件侦听器。

读者或许对此已经有了自己的经验，而且还知道 Flex 中的大多数操作都有对应的事件；可以通过事件侦听器来侦听这些事件，然后再根据情况做出响应。这就是 Flex 应用程序中的通信和神经系统。有时候，即使设置变量也会导致广播事件；而这种分派事件的方式就是绑定。

10.2.2 绑定事件

Flex 中的绑定是在事件系统中实现的。在绑定变量时，实际上会创建一个专用的侦听器，该侦听器侦听的是绑定变量或对象的数据变化事件(要了解有关绑定的更多内容，请参见第 3 章)。

无论何时创建对某个变量的绑定，都会注册一个事件侦听器，这个侦听器会在变量被修改时做出响应。在 MXML 中绑定时，这些更新操作都在后台完成，如代码清单 10-3 所示。

代码清单10-3 MXML绑定

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="vertical">
  <mx:Script>
    <![CDATA[
     [Bindable]
     public var labelText:String = "Label before event";
    ]]>
  </mx:Script>

  <mx:Button id="myButton" label="Change Label!"
     click="labelText='my label has changed';"/>
  <mx:Label id="myLabel" text="{labelText}"/>
</mx:Application>
```

[Bindable]元数据的用意是监视这个变量

读者可以比较一下代码清单 10-3 与实现同样目的的 ActionScript 代码(代码清单 10-4)。ActionScript 代码需要使用 ChangeWatcher 对象，该对象监视被绑定属性值的任何变化。如果有变化发生，ChangeWatcher 会触发必要的事件来监视值。这和事件侦听器对象十分类似，因为它侦听属性的特定事件。

代码清单10-4　ActionScript绑定

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="vertical" creationComplete="initFunc()">
  <mx:Script>
    <![CDATA[
      import mx.Events.PropertyChangeEvent;
      import mx.Binding.utils.*;
      import mx.Events.FlexEvent;

      public var thisWatcher:ChangeWatcher;

      public function initFunc():void
      {
        watchIt();
      }

      public function watchIt():void
      {
        thisWatcher = ChangeWatcher.watch(
              someTextInput,"text",doTheBind);
      }

      public function stopIt():void
      {
       if (thisWatcher.isWatching())
       {
         thisWatcher.unwatch();
       }
      }

      public function doTheBind(e:Event):void
      {
        myLabel.text = someTextInput.text;
      }

    ]]>
  </mx:Script>
  <mx:Button id="watchButton" label="Watch Text" click="watchIt()"/>
  <mx:Button id="stopButton" label="Stop Watching"
     click="stopIt()"/>
  <mx:TextInput id="someTextInput" text="start text"/>
  <mx:Label id="myLabel" text=""/>
</mx:Application>
```

要动态绑定变量，必须使用import语句

告诉ChangeWatcher检测变化

检查监视变量列表

取消监视行为

这种方式并没有那么简单，但却更灵活。在接收到应用程序分派的 creationComplete 事件后，在 someTextInput 对象上设置绑定，从而监视或绑定其文本值。这是通过调用 ChangeWatcher.watch()方法完成的，该方法作为 ChangeWatcher 对象的构造方法使用；调用 watch()时，会返回一个已经实例化的 ChangeWatcher 对象。

ChangeWatcher.watch()方法接收多个参数，第一个是想要监视的对象，第二个是想要监视的对象属性(这里是 someTextInput 对象的 text 属性)。第三个属性指定当触发事件时调用的函数，在这里是 doTheBind()。

与 addEventListener()对比一下就会发现，这两个方法的行为类似。调用 addEventListener()方法时侦听的是整个对象，而代码清单 10-4 中的方法侦听的是具体的对象属性。

当用户向文本框中输入值时，ChangeWatcher 会侦听到值的变化并执行事件。用户每按一下键盘，ChangeWatcher 都会接收到一个事件。如代码清单 10-4 所示，绑定事件的处理函数可以仅仅复制用户输入的值，也可以执行任何必要的复杂操作。

这种实现方式的另一个好处是，可以从对象中移除绑定。在 MXML 中则无法移除绑定。如代码清单 10-4 所示，用户单击 Stop Watching 按钮，就可以通过 ActionScript 把绑定的事件彻底移除。

查看 stopIt()函数可了解到，isWatching()函数检查特定的监视程序是否在执行监视任务。如果是，调用 unwatch()方法可以移除监视程序并断开绑定。

10.2.3 移除事件侦听器

我们刚刚介绍了 unwatch()的用法，该方法可以取消监视变量变化的行为。在使用事件侦听器的情况下，同样可以做到这一点。如果事件侦听器是在运行时通过 ActionScript 添加的，则只能使用 removeEventListener()方法来移除。

代码清单 10-5 所示为如何测试一个对象，以便确定是否有特定的事件侦听器在监视它(由事件类型、阶段及响应函数共同定义)。

代码清单10-5 单击事件的示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  width="100%" height="100%" layout="vertical">
  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;

      [Bindable]
      public var labelText:String = "Label before event";
      public var myCounter:Number = 0;

      public function clickFunction(event:Event):void
      {
        var message:String = "you clicked me! " +
           event.target.label;
        Alert.show(message,"Event Test");
      }

      public function addMyEvent():void
      {
        if (myButton.hasEventListener(MouseEvent.MOUSE_OVER))
        {
           myButton.removeEventListener(
              MouseEvent.MOUSE_OVER,changeLabel);
           myInput.text = "Event was Removed";
           addEventButton.label =
              "Click this button to add event";
```

MXML绑定监视程序(不能移除)

移除事件侦听器

```
        myLabel.text = "Mouse over Event listener removed";
        myCounter = 0;
      }
      else
      {
        myButton.addEventListener(
          MouseEvent.MOUSE_OVER,changeLabel);
        myInput.text = "Event was added";
        addEventButton.label =
           "Click this button to remove event";
      }
    }

    public function changeLabel(event:MouseEvent):void
    {
       myCounter++;
       myLabel.text = "Mouse over Event listener listened! - "
       myLabel.text += myCounter + " times";
     }

  ]]>
</mx:Script>

<mx:Button id="addEventButton"
    label="click this button to add listener"
    click="addMyEvent()"/>
<mx:Button id="myButton" label="Click button!"
     click="clickFunction(event)"/>
<mx:TextInput id="myInput" />
  <mx:Label id="myLabel" text="{labelText}"/>
</mx:Application>
```

添加事件侦听器

代码清单 10-5 全面展示了与事件有关的操作。编译并运行这些代码，会发现处理事件原来如此简单。

单击第一个按钮(addEventButton)，会为 myButton 添加 MouseOver 事件。检查此按钮，看其是否列出了已经注册的 MouseOver 事件，如代码清单 10-6 所示，如果还没有就创建一个。

代码清单10-6 如果不存在，就创建一个事件侦听器

```
public function addMyEvent():void   {
     if (myButton.hasEventListener(MouseEvent.MOUSE_OVER))   {
      ...
     } else {
      myButton.addEventListener(MouseEvent.MOUSE_OVER,changeLabel);
      ...
    }
  }
```

把鼠标移到第 myButton 按钮上，会调用 changeLabel()函数，鼠标每次经过该按钮标签中的数值都会递增 1。再单击第一个按钮则移除事件侦听器，如代码清单 10-7 所示。

代码清单10-7 如果存在，则移除事件侦听器

```
public function addMyEvent():void   {
     if (myButton.hasEventListener(MouseEvent.MOUSE_OVER))   {
      myButton.removeEventListener(MouseEvent.MOUSE_OVER,changeLabel);
      ...
```

同样，移除操作只对使用 ActionScript 添加的事件侦听器有效，通过 MXML 添加的侦听器会一直附着在相应对象上。例如，无法移除通过以下 MXML 脚本添加的事件侦听器。

```
<mx:Button id="myButton" label="Click button!"
 click="clickFunction(event)"/>
```

因此，如果将来有可能需要移除某个事件侦听器，一开始就要使用 addEventListener()方法添加这个事件侦听器。

在单击 Click Button！按钮时，会调用 clickFunction(event)。熟悉 HTML 与 JavaScript 的读者一定对这个功能不陌生。如果想把事件传递给函数，就要在函数语法中专门输入事件；这么做会将事件作为参数传递。以这种方式定义事件侦听器，不仅可以向函数传递事件，还可以传递要响应的自定义参数。

```
<mx:Button id="myButton" label="Click button!"
 click="clickFunction(event, "newparam1 ", someVar) "/>
```

这样一来，定义事件的灵活性就大大增加，因为可以让创建的处理程序按照我们的意愿做出反应。

下一节，我们介绍分派和创建自定义事件。自定义事件有助于增进对事件的控制，也能够让开发人员动态确定何时发送事件以及事件应该包含哪些信息。

10.3 自定义事件

Flex 事件如此强大的原因之一，是允许开发人员创建并使用自定义警报与应用程序通信。通过在数据变化或用户执行某些操作时发送事件，可以实现应用程序逻辑与使用这些逻辑的对象的分离。这种分离有助于应用程序结构的模块化，即一个组件的改变不会影响其他部分。

10.3.1 使用分派程序发送事件

在发送自定义事件时，要使用对象的 dispatchEvent()方法。只有显示列表中的对象才能把事件分派到事件流中。

通常，这意味着分派事件的对象必须在某个时刻显示在应用程序的某个区域上。可以通过 MXML 创建的绝大多数对象都具有分派事件的能力。

请看下面这行代码。

```
myButton.dispatchEvent( new Event("newCustomEvent"));
```

这行代码执行两项不同的操作。

(1) 实例化一个新事件类，以便处理和保存信息。

(2) 调用 dispatchEvent()函数把事件注入到事件流中。

代码清单 10-8 所示为分派自定义事件的简单方法。

代码清单10-8　自定义事件

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="vertical" creationComplete="initFunc()">

  <mx:Script>
    <![CDATA[
      import flash.events.Event;
      import mx.controls.Alert;

      public function initFunc():void
      {
        myButton.addEventListener("newCustomEvent",
           respondToEvent);
      }
      public function doEvent():void
      {
       myButton.dispatchEvent( new Event("newCustomEvent"));
      }

      public function respondToEvent(e:Event):void
      {
       Alert.show("got the Event and it is of type " + e.type,
          "got the event");
      }
    ]]>

  </mx:Script>
  <mx:Button id="myButton" label="Fire off Event" click="doEvent()"/>
</mx:Application>
```

分派 newCustomEvent 类型的自定义事件

如代码清单 10-8 所示，使用自定义事件的最简单方式就是分派一个新事件并修改其事件类型，然后将其发送出去。事件类型是事件的一个文本属性，由事件侦听器监视。

要侦听自定义事件，必须添加事件侦听器并让它关注分派对象的特定事件类型。在触发 creationComplete 事件后调用的 initFunc()函数中完成此操作。在应用程序设置中，可以侦听来自组件的事件，然后在事件发生时以自定义操作作为响应。在这种模式下，组件之间可以相互不知道对方是否存在或位于何处，但仍然能够基于其他组件的事件做出反应。

事件实例化与分派也可以分开进行。

```
Var myEvent:Event = new Event("eventname",false);

 myButton.dispatchEvent(myEvent);
```

采用这种方式可以更充分地定制事件和添加信息。例如，可以创建一个自定义事件类，让这个类能接收额外的信息或负载，以便转发给任何事件侦听器。

除了发送自定义事件之外，还可以伪造(fake)事件。换句话说，可以创建一个事件，用它模拟 Flex 系统正常生成的真正事件。可以在任何时候动态分派伪造事件。

例如，可以像下面这样伪造事件，从而冒充用户单击按钮所引发的事件。

```
myButton.dispatchEvent(new MouseEvent(MouseEvent.Click,false));
```

这行代码实例化一个 MouseEvent 新对象，然后把它插入到事件流中，就像该事件是由按钮生成的一样。任何侦听这个鼠标事件的侦听器都会像响应真正的事件一样响应这个事件。

在了解了如何发送事件之后，接下来我们讨论以子类化事件对象的方式创建自定义事件。

10.3.2 创建自定义事件

动态创建某种自定义事件很容易，但有时却不能满足需要。随着开发经验的增长，读者会发现生成简单自定义事件的一些缺点。例如，无论事件分派代码还是事件侦听器代码中存在语法错误，都必须在运行时隔离和修复这个错误。

在开发大型应用程序时，可能很多宝贵时间和精力都要花在追踪错误上。如果选择使用自定义事件，可以在编译时检查出这些错误，以防它们被留在应用程序中。

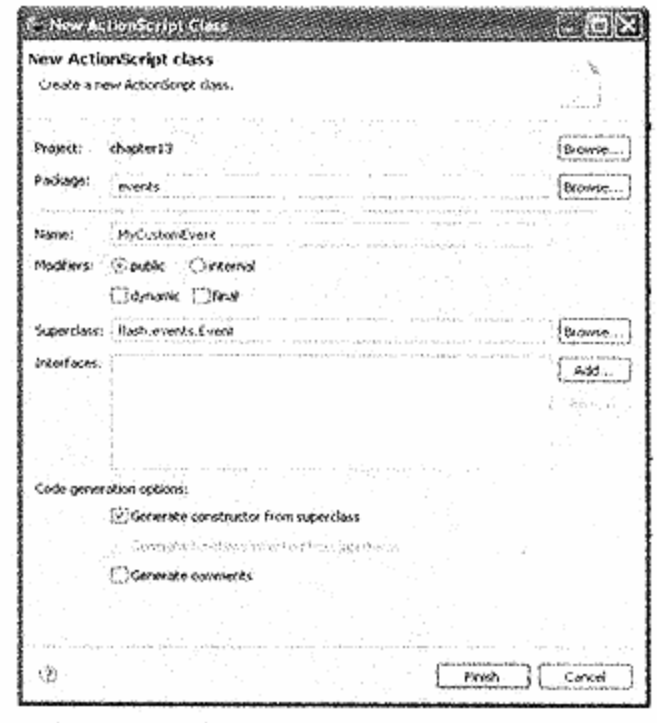


图 10-2 使用 New ActionScript Class 向导创建自定义事件

使用 New ActionScript Class 向导，几乎可以创建自定义事件过程中所需的一切。图 10-2 所示为如何创建自定义事件类 MyCustomEvent，该类继承了 flash.events.Event 基类(要了解与扩展类相关的内容，请参见第 16 章和第 17 章)。

该向导能够自动为我们生成自定义事件所需的大部分代码及用于填充该类的大多数信息。在这个示例中，我们只做了几处修改来创建 MyCustomEvent 类。

代码清单 10-9 所示为我们在向导中创建和修改的自定义事件的代码。由于在向导中选中了 Generate constructor from superclass 复选框，结果向导帮我们生成了大部分代码。由于基类拥有一组创建事件必需的，因此使用超类可以满足应用程序的绝大部分需要。

代码清单10-9 创建自定义事件

```
package Events
{
  import flash.Events.Event;

  public class MyCustomEvent extends Event
  {
    public static
    var CUSTOMEVENTTYPE:String = "CustomEventType";

    public var info:String;

     public function MyCustomEvent(type:String,
       info:String, bubbles:Boolean=false,
       cancelable:Boolean=false)
    {
      super(type, bubbles, cancelable);
      this.info = info;
    }
  }

}
```

扩展Event对象

定义自定义事件类型

可以使用多个输入参数，也可以不用

调用Event对象的构造函数

读者如果使用向导创建自定义事件，就会看到与代码清单 10-9 中相似的代码；这里添加的代码很少。

为事件添加了两个变量。一个是公共静态变量 CUSTOMEVENTTYPE，用于指定事件类型。这个变量不会改变，可以通过它在代码中确定事件类型。

另一个变量 info 是字符串，用于保存与用户沟通的消息。当监视对象接收到这个事件时，它不仅拥有可以访问这个基本事件类中的常见事件属性，还拥有 info 属性。

代码清单 10-10 所示为一个简单示例，该示例使用这个自定义事件类在应用程序中实现通信。虽然示例中没有比较复杂的自定义组件或其他对象，但其中展示的交互过程同样适用于宏观层面的组件间通信。

代码清单10-10　使用自定义事件

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="vertical" creationComplete="initFunc()">

   <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
      import Events.MyCustomEvent;

      public function initFunc():void
      {
        myButton.addEventListener( MyCustomEvent.CUSTOMEVENTTYPE,
             respondToEvent);
      }

      public function doEvent(info:String):void
      {
        myButton.dispatchEvent( new
         MyCustomEvent(MyCustomEvent.CUSTOMEVENTTYPE,info));
      }

      public function respondToEvent(e:MyCustomEvent):void
      {
        var msg:String = "got the Event and it is of type  " +
           e.type;

        msg += " and here is the message  "+ e.info;
        msg += " and here is some info from the button  "

        msg += e.target.label;
       Alert.show(msg,"got the event");
     }
    ]]>
   </mx:Script>
   <mx:Button id="myButton" label="Fire off Event"
     click="doEvent('buttons are really great!')"/>
</mx:Application>
```

初始化函数

触发自定义事件

使用自定义事件

响应 MyCustomEvent

info 是 MyCustomEvent 中的变量

target是myButton

如代码清单 10-10 所示，在应用程序创建阶段结束后，会调用 initFunc()函数。initFunc()函数为按钮添加事件侦听器，以侦听自定义事件。当按钮被单击时，会调用 doEvent()函数，该函数负责创建并分派自定义事件；而这个事件将被侦听它的对象接收到。

最后，respondToEvent()函数使用事件中的所有信息，并以警告形式将创建的消息展示给用户。

示例中的代码在分派自定义事件时，会将该事件注入到应用程序的事件流中。在那里，

自定义事件将得到与其他事件相同的处理方式。这个示例展示的自定义内容是事件能够携带什么信息，以及如何以子类化的方式创建相应类型的事件对象。

在了解了如何发送自定义事件后，我们来看如何停止事件传播。

10.3.3 停止事件传播

在任何事件阶段(捕获、目标或冒泡)都可以使用 stopPropagation()和 stopImmediatePropagation()方法停止事件向其他组件传播。这两个方法本质上是相同的，区别仅在于是否允许同一组件上的其他事件侦听器接收相应的事件。

换句话说，如果在事件上调用 event.stopPropagation()方法，它会在给定组件上的所有其他事件侦听器完成对这个事件的响应之后停止事件传播。如果使用 event.stopImmediatePropagation()方法，事件会在传播到其他任何事件侦听器之前被停止，即使这些事件侦听器侦听的都是同一个组件。

与事件侦听器的 priority 属性结合使用时(在添加事件侦听器时设置)，可以将某个函数设置为对事件的第一个响应者。这样，如果需要评估事件，并阻止传播到给定组件的其他任何事件侦听器，能够形成有效的闸门机制(gating mechanism)。请看代码清单 10-11 所示的例子。

代码清单10-11 停止传播

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
   creationComplete="initFunc()">

  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;

      public function initFunc():void
      {
        myButton.addEventListener("click",          | 添加带有高优先
         respondToEvent,false,100);                 | 级的事件侦听器

        myParentHBox.addEventListener("click",      | 在父组件中侦
          parentEventResponse, false,0);            | 听冒泡事件
      }

     public function respondToEvent(e:Event):void

     {
       var msg:String = "This is the first responder, "
       msg += " and now the Event will be stopped ";
       Alert.show(msg,"First Event Listener");

         e.stopPropagation();      <---- 停止传播——第二个
      }                                 事件触发

      public function respondToEventClick(e:Event):void
      {
        Alert.show("this is the mxml click Event
             and this will fire second ",
             "Second Event Listener");
      }
```

```
        public function parentEventResponse(e:Event):void
        {
          Alert.show("You should never see this alert",
          "Parent Event Response");
        }

    ]]>
  </mx:Script>
    <mx:HBox id="myParentHBox">
    <mx:Button id="myButton" label="Fire off Event"
     click="respondToEventClick(event)"/>
  </mx:HBox>

</mx:Application>
```

事件永远不会传播到父组件

这里，我们是在按钮的层次上中断事件传播。正常情况下，事件将流经所有函数，触发所有(3 个)警告。但在第一个事件处理函数执行后，我们调用 stopPropagation()方法。运行这个示例，前两个事件(它们直接侦听按钮)都会运行，但侦听 myParentHBox 的父事件不会接收到此事件，因而也不会执行。

如果把 stopPropagation()方法修改为 stopImmediatePropagation()，结果只能看到第一条警告。stopImmediatePropagation()会在第一个事件侦听器执行完毕后，立即终止事件传播。可见，中断事件传播是一种控制事件流的有效方式(应该视环境而定)。特别是在自定义组件发送自定义事件的情况下。

在了解了停止事件传播和创建自定义事件之后，接下来讨论向组件中添加事件元数据。

10.3.4　向组件添加事件元数据

在完整的事件分派过程中(特别是在使用自定义组件和类时)，一个非常重要的环节是向与分派事件有关的组件中添加元数据，以便利用 Flex Builder 的代码提示功能。

向组件中添加元数据不是绝对必需的，但这样做通常会更方便一些，因为可以把事件看成组件的属性。代码清单 10-12 展示了为组件添加元数据的过程。

代码清单10-12　MXML组件中的事件元数据

```
<mx:Metadata>
      [Event(name="customButtonClick", type="flash.Events.Event")]
</mx:Metadata>
```

前面的代码展示了与来自组件的事件通信时如何使用元数据。在下面的代码中，组件通过执行事件分派代码发送 customButtonClick 事件。

```
this.dispatchEvent(new Event("customButtonClick",true));
```

这行分派命令会向父应用程序发送一个类型为 customButtonClick 的事件。而组件中的 Metadata 标签会提示上层父组件及编译器该组件将发送自定义事件，而父组件可以侦听这个自定义事件。在父组件中，下面的代码会接收到自定义事件并将其定向到相应函数。

```
<comp:CustomEventMeta customButtonClick="doAlert(event)"/>
```

按照本章介绍的方式在自定义组件中使用事件，有助于保持代码的松散耦合。应用程序中的组件不必相互了解，而父窗口仍然可以控制它们的行为和交互。

有关自定义组件的更多内容将在第 16 章和第 17 章介绍。

10.4 小结

本章重点介绍事件系统及其在 Flex 中的用法。勿庸置疑，事件系统是 Flex 框架中最重要的组成部分，因为事件渗透到应用程序中的每一个组件。

读者如果具有 Web 开发背景，可能需要转换一下创建 Web 应用程序时标准的请求/响应思维方式。事件系统是理解这种转换的关键所在。通过 Flex 实现的一切功能至少在某种程度上都会涉及事件系统。如果读者能够在应用程序中游刃有余地使用事件，那么所有对象的行为都是相互独立的。甚至，这些对象都不必知道是在哪个应用程序中运行。

在解决了事件系统的问题之后，第 11 章我们将讨论应用程序导航。

第 11 章 应用程序导航

11

本章内容：

- 促进导航的组件
- 为导航组件集成数据
- 与导航组件交互

深入开发过程的一个重要标志，是考虑如何为应用程序划分不同的功能区域，同时让用户为实现不同目的而在这些区域间往返移动。这个开发阶段叫做*导航设计*。本章我们主要介绍下列适合为应用程序添加导航功能的组件。

- Menu
- MenuBar
- TabBar
- TabNavigator
- Accordion

在详细介绍如何使用这些组件之前，必须准备相关数据。与很多任务类似，准备数据的方式也有很多种，我们将在介绍这些方式的同时分析各自的优缺点。

11.1 为菜单和菜单栏提供数据

自从 GUI 进入人们的视野以来，下拉菜单就成为界面导航的一种基本组件。下拉菜单既可以方便地显示应用程序特性及选项，又不会占用宝贵的屏幕资源——你不用它们，它们就会隐藏起来。而且用户单击按钮、标签页或标题时，菜单就在近旁出现，同样可以减少用户找来找去的麻烦。

使用 HTML 从头创建下拉菜单是一项棘手的任务，因为要处理分层、Z 索引、重叠表单元素和跨浏览器兼容性等诸多问题，以及相当多的 JavaScript 和 DHTML 代码(尤其是在创建上下文菜单时)。虽然有很多第三方产品抽象了创建 HTML 下拉菜单的复杂性，但后台的复杂性依旧存在。所幸的是，Flex 提供了更简便的解决方案。

先抛开菜单的可用性不谈，首先必须解决的问题是为菜单提供数据。用于生成级联菜单的信息本身通常也会表现为结构化，因此我们的数据必须符合这个条件。

为菜单提供数据的方式很多，且各有利弊。下面我们从手边的解决方案说起。

11.1.1 嵌套的数组

嵌套的数组由一层或多层数组构成，用于模仿类似树的数据结构，可以在 ActionScript 中编写它们。

嵌套的数组有利于节省内存空间。但是它们的语法相对复杂，底层数据的修改也不能自动反映到视觉组件中。嵌套的数据毕竟不是基于集合的对象。

代码清单 11-1 所示为如何组织和使用嵌套的数组。

代码清单11-1 使用嵌套数组为菜单提供数据

```
public var myMenuData:Array =
  [
    {
     label:'New',
       children: [{label:'Task'},{label:'Request'},{label:'Person'}]
    },
    {
     label:'Import',
       children: [{label:'Image'},{label:'Document'},{label:'Project'}]
    }
  ];
```

实例化数组

嵌套的数组 ❶

代码清单 11-1 中包含两行声明为 children:的代码❶，这两行代码在 myMenuData 数组中定义了两个嵌套的数组。而 children:声明会告诉 Menu 对象将包含在这两个子数组中的数据当作嵌套信息。

提示：

默认情况下，Menu 组件会查找名为 children 的嵌套数组。

使用数组的问题主要在于，它们是一种简单的对象类型；由于它们不是集合，因此底层数据的改变不会自动传播到有侦听任务的其他对象。这也是为什么升级使用基于集合的对象更方便的原因，对于 ArrayCollection，尤其如此。

11.1.2 嵌套的数组集合

嵌套的数组集合与嵌套的数组相似，但它使用 ArrayCollection 类。数组集合能够自动广播数据变化；但与数组类似，它们的语法相对复杂。

如代码清单 11-2 所示，嵌套的数组集合与稍微简单一些的嵌套数组没有多大区别。

代码清单 11-1 和代码清单 11-2 十分相似，只不过代码清单 11-2 中导入了 ArrayCollection 类❶，并且调用该类的构造函数初始化一个实例。与数组类似，在此也要指定 children:声明来创建新级别❷。

ActionScript 虽然能够处理数组和数组集合，但对于多层嵌套的数据来说，其语法的确是对人类视觉分辨力的一种挑战。说到这里，也许我们可以考虑语法更简单的基于 Model

或基于 XML 的数据。

代码清单11-2　使用嵌套的数组集合为菜单提供数据

```
import mx.collections.ArrayCollection;

public var myMenuData:ArrayCollection = new ArrayCollection(
  [
    {
      label:'New',
      children: [{label:'Task'},{label:'Request'},{label:'Person'}]
    },
    {
      label:'Import',
      children: [{label:'Image'},{label:'Document'},{label:'Project'}]
    }
  ]);
```

❶ 导入ArrayCollection类

实例化ArrayCollection

❷ 嵌套的数组数据

11.1.3　Model

Model 是与 XML 极为相似的一种数据类型，但它实际上是嵌套的低层(low-level)对象。Model 可以用来存储未来会被处理的数据。例如，可以使用 Model 临时保存表单的值。

Model 具有如下优点。

- 虽然看起来与 XML 相似，但 Flex 会在后台将数据转换为嵌套的低层对象。这会在语法方面提供某些便利，毕竟要操作的是一些本地 ActionScript 对象。
- 由于类似 XML 结构，因此可以方便地编写和查看代码结构。
- 可以方便地实现绑定。
- 它带给应用程序的开销非常小(因为它们是低层对象)。
- 可以使用 source 属性从独立的文件中导入数据。

从不足的方面讲，Model 必须遵从一种特定的语法，以便与显示嵌套结构中层次化数据的组件(如菜单)兼容。

代码清单 11-3 所示为一个使用 Model 的例子。

代码清单11-3　通过创建Model为菜单提供数据

```
<mx:Model id="myMenuData">
  <menuinfo>
    <menuitem label="Task">
      <children label="Request"/>
      <children label="Person"/>
    </menuitem>
    <menuitem label="Import">
      <children label="Image"/>
      <children label="Document"/>
      <children label="Project"/>
    </menuitem>
  </menuinfo>
</mx:Model>
```

注意，对于 menuinfo 和 menuitem 这两个节点名称，我们可以随便命名；但是，如果要创建更深层的节点级别，必须将它们声明为 children。

Model 具有类似 XML 的便利，但它实际上并不是 XML。在实际开发中，经常会遇到必须

使用 XML 的情形。例如，如果应用程序要求执行服务器端数据存储，就该考虑使用 XML 了。

11.1.4 XML 组件及类

Flex 对 XML 的支持源自 XML 组件及类；第 15 章将会介绍，XML 是 Flex 中非常强大的工具。以类似树的嵌套结构表达数据是 XML 与生俱来的本能。

使用 XML 的优点如下。

- 从易读易写的方面讲，XML 具有与 Model 同样的特点。
- XML 既可以作为组件(<mx:XML>)，也可以作为 ActionScript 类。
- 与 Model 不同，可以随意命名 XML 节点，即不限于顶级节点。
- XML 组件是一种更加智能化的对象，支持操作 XML 的功能。
- 可以使用 source 属性从独立的文件中导入数据。

与嵌套数组一样，XML 组件也不是基于集合的对象，因此底层数据的变化不会自动通过事件系统传播。

代码清单 11-4 将代码清单 11-3 中的 Model 数据转换为 XML 组件。

代码清单11-4 使用XML组件为菜单提供数据

```
<mx:XML id="myMenuData">
 <menuinfo>
  <menuitem label="Task">
   <submenu label="Request"/>
   <submenu label="Person"/>
 </menuitem>
 <menuitem label="Import">
  <submenu label="Image"/>
  <submenu label="Document"/>
  <submenu label="Project"/>
  </menuitem>
 </menuinfo>
</mx:XML>
```

在此，对于其他数据类型而言必需的 children 节点，被任意命名的 submenu 所取代。类似地，也可以在 ActionScript 中创建相同数据，如代码清单 11-5 所示。

代码清单11-5 使用XML ActionScript类创建菜单数据

```
<mx:Script>
<![CDATA[
 public var myMenuData:XML=
  <menuinfo>
   <menuitem label="Task">
    <submenu label="Request"/>
    <submenu label="Person"/>
   </menuitem>
   <menuitem label="Import">
    <submenu label="Image"/>
    <submenu label="Document"/>
    <submenu label="Project"/>
   </menuitem>
  </menuinfo>
]]>
</mx:Script>
```

XML 是为菜单提供树状结构数据的好搭档。但是必须要明确的一个前提条件是，XML 规则要求所有 XML 文档必须有一个根节点。

好在我们有现成的办法解决这个问题。可以在 Menu 组件中标出一个起点位置，或者干脆直接使用 XMLList 组件。

11.1.5　XMLList 组件

XMLList 组件与 XML 组件非常相似，但它的主要用途是保存从 XML 文档中提取出来的数据片段。使用 XMLList 为菜单提供数据不仅可以获得 XML 所能提供的全部优点，而且还能定义多个根节点。XMLList 还支持更多操作数据的功能。

另一方面，由于 XMLList 组件不是基于集合的对象，因此其底层数据变化不会自动传播。

可以在代码清单 11-5 的基础上，通过修改菜单数据来尝试使用 XMLList 组件，如代码清单 11-6 所示。

代码清单11-6　使用XMLList来配置菜单数据

```
<mx:XMLList id="myMenuData">
  <menuitem label="Task">
    <submenu label="Request"/>
    <submenu label="Person"/>
  </menuitem>
  <menuitem label="Import">
    <submenu label="Image"/>
    <submenu label="Document"/>
    <submenu label="Project"/>
  </menuitem>
</mx:XMLList>
```

代码清单 11-6 所示为多个根节点。虽然这个区别不算太大，却可以因此多一种选择。

前面曾多次提到非集合数据类型的缺点。数组就是一个例子，但可以使用数组集合代替它。与此类似，XML 组件/类也有相应的替代物，即 XMLListCollection 组件/类。

11.1.6　XMLListCollection 组件及类

XMLListCollection 就像一个包装程序，它包装现有的 XML 并添加了集合功能。XMLListCollection 既拥有 XML 的所有优点，又是集合数据类型，因此其底层数据的变化能够自动传送到负责侦听任务的组件，即 Menu 组件。XMLListCollection 既可以用于组件和 ActionScript 类，而且也能用于包装已有的 XML 数据。由于 XMLListCollection 的确强大，因此它比前面提到的轻量级对象要复杂和庞大一些——不过仍然“物有所值”。

使用 XMLListCollection 的方式有很多种，但基本上都是通过它包装现有的 XML 数据。下面我们就介绍使用 XMLListCollection 的各种方式。代码清单 11-7 所示为如何使用 XMLListCollection 包装现有的 XMLList，以便自身获得相应数据。

代码清单11-7　使用XMLListCollection组件包装XMLList

```
<mx:XMLListCollection id="myMenuData">
  <mx:XMLList>
    <menuitem label="Task">
```

```
    <submenu label="Request"/>
    <submenu label="Person"/>
  </menuitem>
 </mx:XMLList>
</mx:XMLListCollection>
```

代码清单 11-8 所示为另一种方式，其中通过 source 属性间接地包装现有 XML 组件。

代码清单11-8 以XML组件作为数据源的XMLListCollection组件

```
<mx:XML id="config">
 <configData>
  <appData>
   <appName name="My Application"/>
  </appData>
  <menuData>
   <menuitem label="Task">
    <submenu label="Request"/>
    <submenu label="Person"/>
   </menuitem>
  </menuData>
 </configData>
</mx:XML>

<mx:XMLListCollection id="myMenuData"
                      source="{config.menuData.menuitem}"/>
```

代码清单 11-8 的优点是，可以把 XML 保存在一个独立的 XML 配置文件中，从而能够把 XMLListCollection 的数据减少到驱动菜单所必需的程度。

```
<mx:XML id="config" source="config.xml">
<mx:XMLListCollection id="myMenuData" source="{config.menuData.menuitem}"/>
```

代码清单 11-9 所示为使用 XMLListCollection 类(而非组件)的方式。

代码清单11-9 以XML组件作为数据源的XMLListCollection类

```
<mx:XML id="config" source="config.xml">
<mx:Script>
<![CDATA[
 import mx.collections.XMLListCollection;
 public var myMenuData:XMLListCollection = new XMLListCollection();
 public function setup():void
 {
  myMenuData.source = config.menuData.menuitem;
 }
]]>
</mx:Script>
```

使用 ActionScript 可以声明 XMLListCollection 的一个实例，然后将其 source 属性指向 XML 信息源。这里的信息源可以是 XML 和 XMLList 组件和类的实例。

11.1.7 选定数据提供者数据类型

如前所述，为菜单和菜单栏配置数据的方式有很多。但是，如果只能选择一种方式，应该选择 XML ListCollection。毕竟，XMLListCollection 具备快速开发应用程序所需的一切优点。鉴于它无可争议的优势，我们将在本章后面展示 Menu 组件用法的示例中使用它。

11.2　使用菜单

菜单创建一个包含用户可选项的单列窗口。菜单中可以包含子菜单(可以嵌套至任意级别)，每个子菜单可以展开或级联更多可选项。

接下来我们探讨有关菜单的如下内容：

- 创建菜单
- 定位菜单
- 自定义菜单
- 与菜单交互

下面就从最基本的内容开始——创建一个基本菜单。

11.2.1　创建菜单

Flex 中的组件通常都有 MXML 和 ActionScipt 两个版本，但菜单却没有 MXML 组件，只能使用 ActionScript 设置和显示菜单(见代码清单 11-10)。

代码清单11-10　从头创建菜单

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="#FFFFFF" >

<mx:XMLListCollection id="myMenuData">
<mx:XMLList>
  <menuitem label="Tasks">
    <submenu label="Add Request"/>
    <submenu label="Add Person">
    <submenu label="Customer"/>
      <submenu label="Employee"/>
    </submenu>
  </menuitem>
</mx:XMLList>
</mx:XMLListCollection>

<mx:Script>
  <![CDATA[
    import mx.controls.Menu;
    public var myMenu:Menu;

    private function InitializeMenu():void
    {
      myMenu = Menu.createMenu(null, myMenuData,true);
      myMenu.labelField="@label";
    }
  ]]>
</mx:Script>

<mx:Button id="myButton" creationComplete="InitializeMenu()"
          label="Display Menu" click="myMenu.show()"/>
</mx:Application>
```

用于驱动菜单的数据

导入ActionScript中的菜单类

❶ 菜单实例

❷ 初始化菜单

❸ @表示XML节点的属性

在Flex创建按钮时运行 ❹

Flex 在创建了按钮之后，会运行 InitializeMenu()函数(通过 creationComplete 事件❹)。

InitializeMenu()❷调用 Menu 类的 createMenu()函数创建一个 Menu 实例，同时为该实例指定关联数据。

你通知 Menu 实例(myMenu)显示字段的名称为 label❸。由于是 XML 节点的属性，所以指定为@lable。如果我们在这里使用数组集合，则无需使用@声明。

InitializeMenu()函数也可以在其他任何位置运行。例如，可以运行以下脚本：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   creationComplete="InitializeMenu()">
```

如果读者想把编写的代码量降到最低，可以使用代码清单 11-11 中展示的方法，毕其功于一役。

代码清单11-11 一次性初始化并显示菜单的简捷方式

```
<mx:Script>
<![CDATA[
 import mx.controls.Menu;
 private function showMenu():void
 {
  var myMenu:Menu = Menu.createMenu(null, myMenuData,true);
  myMenu.labelField="@label";
  myMenu.show();
 }
]]>
</mx:Script>
<mx:Button id="myButton" label="Display Menu" click="showMenu()"/>
```

这种方式唯一的缺点是，会频繁地创建新菜单。实际上，这也不算什么大问题，因为 Flex 能够销毁旧菜单。不过，还是应该尽力避免过多占用资源。

单击 Display Menu 按钮，屏幕左上角会显示一个菜单(见图 11-1)。

图 11-1 单击 Display Menu 按钮会出现一个多级菜单

虽然生成这个简单菜单需要编写相当多的代码，但关键只在于一个函数——createMenu()。下面我们重新审视一下这个函数，调用它的脚本如下：

```
var myMenu:Menu = Menu.createMenu(null, myMenuData,true);
```

createMenu()函数接收 3 个参数，如表 11-1 所示。

表 11-1 Menu.createMenu()函数的参数

参 数	类 型	说 明
parent	DisplayObjectContainer	(可选)指定与菜单相关的容器。如果将 parent 设置为 null，菜单将与应用程序相关联
mdp	Object	包含数据的变量

(续表)

参　数	类　型	说　明
showRoot	Boolean	如果使用 XMLList 并希望显示 XML 变量中的每个根值，应该选择 true。相反，如果使用单根节点的数据(即目标信息包含在根节点的下一级中)，应将值设置为 false。默认值为 true

代码清单 11-11 默认把菜单放在窗口的左上角位置。在知道如何创建基本菜单之后，我们来看如何定位菜单。

11.2.2　定位菜单

要明确控制菜单的显示位置，可以在调用 show()函数时为其传递 *x* 和 *y* 坐标值，例如：

```
myMenu.show(10,20);
```

第 4 章曾经介绍过，这里的坐标值是相对于当前窗口左上角而言的。因此，上面这行代码指定的 10 像素的 *x* 坐标和 20 像素的 *y* 坐标，会把菜单左上角定位在窗口左边缘右侧 10 像素、上边缘下方 20 像素的点上。

11.2.3　自定义菜单项

驱动菜单的数据中也可以包含额外的菜单显示信息。表 11-2 所示为这些属性及其说明。

表 11-2　用于设置菜单如何显示数据的属性

属　性	类　型	说　明
enabled	Boolean	用于设置相应的菜单项是否可以选定。如果值为 true，用户可以选择该项。如果值为 false，则相应菜单项会以半灰色显示且不能选择。默认值为 true
type	String	用于指定选项装置的类型(如果设置的话)，有效值如下 • normal：用户选择菜单项(默认值) • separator：位于菜单项上下的黑色实线 • check：菜单项旁边会显示一个复选框，以便用户选择 • radio：菜单项旁边会显示一个单选按钮，以便用户选择
groupName	String	用于分组单选按钮。当 type 属性被设置为 radio 时，必须同时设置这个属性

(续表)

属　性	类　型	说　明
toggled	Boolean	当 type 属性被设置为 check 或 radio 时，按照所选项设置菜单项
icon	Class	用作图标的图像变量。有关图标的详细介绍参见第 19 章
label	String	菜单项中显示的值

代码清单 11-12 所示为这些属性的实际用法。

代码清单11-12　自定义菜单的显示细节

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="#FFFFFF">

  <mx:XMLListCollection id="myMenuData">
  <mx:XMLList>
    <menuitem label="Tasks">
      <submenu label="Add Request" enabled="false"/>
      <submenu label="" type="separator"/>

      <submenu label="Add Person" icon="addPerson">

      <submenu label="Customer" type="radio" groupName="personGroup"/>

      <submenu label="Employee" type="radio" groupName="personGroup"
            toggled="true"/>
      </submenu>
      <submenu label="Auto Update" type="check" toggled="true"/>
    </menuitem>
  </mx:XMLList>
  </mx:XMLListCollection>

  <mx:Script>
    <![CDATA[
      [Bindable]
      [Embed(source="addPerson.gif")]
      public var addPerson:Class;

      import mx.controls.Menu;
      private function showMenu():void
      {
        var myMenu:Menu = Menu.createMenu(myButton, myMenuData);
        myMenu.labelField="@label";
        myMenu.iconField="@icon";
        myMenu.show();
      }
    ]]>
  </mx:Script>
  <mx:Button id="myButton" label="Display Menu" click="showMenu()"/>
</mx:Application>
```

菜单数据

在菜单项旁边显示图标

添加菜单分隔符

显示单选按钮

显示可切换的单选按钮

显示复选框

显示为菜单图标的图像

@label属性表示包含标签

指示哪个属性具有可选图标

图 11-2 所示为代码清单 11-12 的结果。

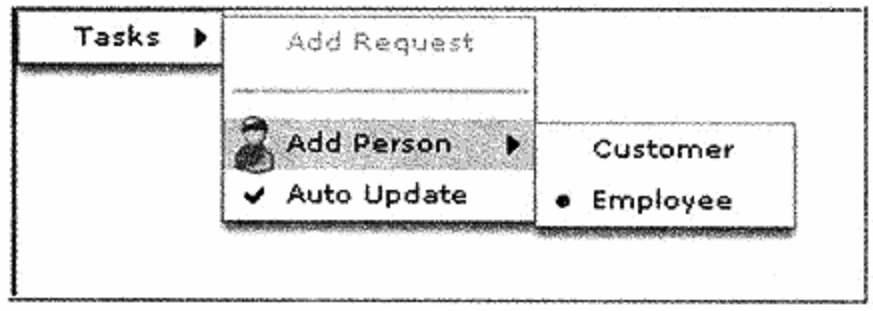


图 11-2　使用其他控制显示的属性，可以自定义菜单的外观

由此可见，自定义菜单的外观并非难事。接下来要考虑的是，如何处理用户与菜单的交互。

显示菜单只完成了导航的一半功能；另一半同样重要的工作，是处理用户与菜单交互时生成的所有事件。表 11-3 所示为用户在打开菜单和选择菜单项时可能触发的事件。

表 11-3　用户使用菜单时可能会触发的事件

事　件	说　明
change	用户改变当前选项
itemClick	用户单击一个已经启用的菜单项
itemRollOut	鼠标指针从一个菜单项上移出
itemRollOver	鼠标指针移入到一个菜单项上
menuHide	菜单或子菜单关闭
menuShow	菜单或子菜单打开

无论表 11-3 中的任何事件发生，都将创建一个 MenuEvent 类型的事件对象，如同使用 ActionScript 创建 MenuEvent 类的实例一样。MenuEvent 实例也会带有一组属性及信息。表 11-4 列出了这些属性。

表 11-4　MenuEvent 的属性

属　性	类　型	说　明
item	Object	菜单项关联的数据行
index	Number	菜单或子菜单的索引
label	String	包含显示标签的数据字段。如果这个字段是 XML 节点的属性，应前置@声明
menu	Object	引用创建事件的菜单。不适用于菜单栏
menuBar	Object	引用创建事件的菜单栏。不适用于菜单

代码清单 11-13 组合使用这些属性，并将事件传递给函数(此后便可以使用相应的数据)。

代码清单11-13 处理用户与菜单的操作

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="#FFFFFF">

  <mx:XMLListCollection id="myMenuData">
    <mx:XMLList>
      <menuitem label="Tasks">
        <submenu label="Add Request"/>
        <submenu label="Add Person">
        <submenu label="Customer" personType="32"/>
        <submenu label="Employee" personType="57" toggled="true"/>
        </submenu>
      </menuitem>
    </mx:XMLList>
  </mx:XMLListCollection>

  <mx:Script>
    <![CDATA[
    import mx.controls.Alert;
    import mx.controls.Menu;
    import mx.events.MenuEvent;

    private function showMenu():void
    {
      var myMenu:Menu = Menu.createMenu(myButton, myMenuData);
      myMenu.labelField="@label";
      myMenu.addEventListener("itemClick",handleMenuClick);
      myMenu.show();
    }
    private function handleMenuClick(evt:MenuEvent):void
    {
      mx.controls.Alert.show("The user just clicked on " +
                        evt.item.@label);
      lastEvent.text = "Selection:" + evt.item.@label +
                    ", Position: " + evt.index +
                    " Type:" + evt.item.@personType;
    }
    ]]>

  </mx:Script>

  <mx:Button id="myButton" label="Display Menu" click="showMenu()"/>
  <mx:Label id="lastEvent"/>
</mx:Application>
```

❶ 当用户单击菜单项时调用handlemenuClick告诉Flex

❷ 通过警告窗口显示菜单事件信息

❸ 在菜单中的位置

❹ 访问与菜单项相关的数据

单击某个菜单项，会触发 itemClick 事件，进而执行 handleMenuClick()函数❶。根据事件负载(evt:MenuEvent)，可以确定用户单击了哪个菜单项、该菜单项在菜单中的位置❸以及该菜单项包含的所有数据❹。屏幕上会显示一个警告窗口❷，指出菜单项的标签。已经设置的 Label 字段会进一步显示这个菜单项的其他信息。

编译并运行这个应用程序，将看到类似于图 11-3 所示的结果。

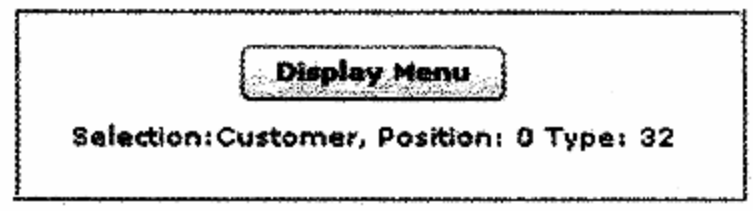


图 11-3 单击按钮并选择一个菜单项，将看到从 MenuEvent 对象中提取的信息

注意：

由于提取的是 XML 数据，因此访问其中的任何属性都要使用“@属性”语法(如 evt.item.@personType)。如果使用嵌套的对象、数组或数组集合，则不必使用这种语法。

利用 MenuEvent 对象能够创建出可重用的函数，该函数可以在处理应用程序中其他菜单时使用。

Menu 组件本身存在局限性——当处理 XML 数据时，它只能处理一个根节点。不过，读者不必担心，菜单的老大哥菜单栏，可以轻松解决这个问题。

11.3　使用菜单栏

菜单栏在菜单的基础上扩展而成，可以容纳多个顶级级联菜单。菜单并排排列，是大多数应用程序中常见的布局形式。菜单栏作为一种非干扰性机制，可以让用户在应用程序的特性与功能中导航时，仍然保持在现有页面上。

除了外观上明显不同之外，菜单与菜单栏的主要区别是，菜单栏不是一种弹出式组件。菜单栏的用途只是在 Flex 应用程序中提供与桌面应用程序相同的菜单功能。

MenuBar 可以作为组件使用(这一点与 Menu 不同，Menu 只是 ActionScript 类)。表 11-5 所示为 MenuBar 的属性。

表 11-5　MenuBar 的属性

属　性	类　型	说　　明
dataProvider	Object	用于填充菜单栏的数据。有关提供数据的内容请参见 11.1 节
iconField	String	字段名，表示哪个图像变量将作为图标使用。默认情况下，会查找名为 icon 的字段。如果该字段存在，则使用该字段
labelField	String	字段名，指示菜单项的显示值。默认值为 label。如果是 XML 数据，务必使用@声明指定要使用的 XML 节点属性(如@label)
labelFunction	String	函数，调用它以便自定义标签的显示(有关标签函数的更多信息请参见第 9 章)
selectedIndex	Number (int)	表示被选中菜单项的位置
showRoot	Boolean	如果设置为 true，则将 XML 的顶级节点显示在菜单中。如果设置为 false，则从顶级节点的下一级开始显示。默认值为 true

下面我们创建一个菜单栏。

11.3.1 创建菜单栏

代码清单 11-14 创建了一个简单的菜单栏。注意，每个顶级节点都位于最上层位置。

代码清单11-14 创建简单的菜单栏

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:XMLListCollection id="myMenuData">
  <mx:XMLList>
    <menuitem label="File">
      <submenu label="New">
        <submenu label="Project"/>
        <submenu label="Request"/>
      </submenu>
      <submenu label="Print"/>
    </menuitem>
    <menuitem label="View">
      <submenu label="Users"/>
      <submenu label="Reports"/>
    </menuitem>
  </mx:XMLList>
  </mx:XMLListCollection>
  <mx:MenuBar id="myMBar" dataProvider="{myMenuData}"
           labelField="@label"/>
</mx:Application>
```

编译并运行代码清单 11-14，将看到类似于图 11-4 所示的基本菜单栏。

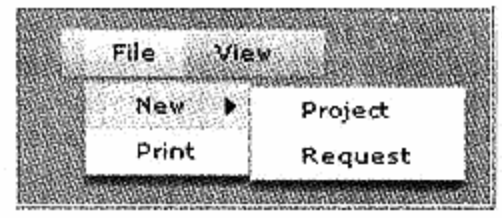


图 11-4 基本菜单栏

可以像定位菜单一样细致地调整菜单栏的位置。

11.3.2 定位菜单栏

因为菜单栏是位于一个容器(即这里的<mx:Application>)中的组件，因此定位菜单栏要遵循容器的规则。默认情况下，Application 容器使用自动布局，因此菜单栏将出现在应用程序的中间位置。

如果想把菜单栏定位在浏览器窗口左上角，需要把 Application 容器的 layout 属性设置为 absolute，然后把菜单栏的 x 和 y 属性都设置为 0。以下代码把 layout 属性设置为 absolute：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute">
```

类似地，如果菜单栏位于 Panel 容器中，则默认行为是沿垂直方向布局子组件。对于这个示例而言，无需调整菜单栏的坐标。但是，如果需要进行更为细致的控制，必须将 Panel 容器的 layout 属性设置为 absolute。

```
<mx:Panel title="My Application" layout="absolute">
  <mx:MenuBar id="myMBar" dataProvider="{myMenuData}"
           labelField="@label" />
</mx:Panel>
```

使用 Panel 容器，菜单栏如图 11-5 所示。

如图 11-5 所示，菜单栏定位在 Panel 容器的左上角，因为菜单栏默认的 x 和 y 坐标是(0,0)。如果想让菜单栏再偏右 5 像素，将它的 x 属性设置为 5 即可。

和自定义菜单一样，也可以自定义菜单栏。

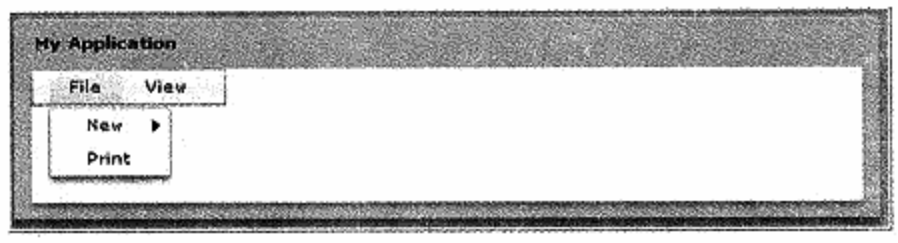


图 11-5　要控制菜单栏的准确位置，将容器的 layout 属性设置为 absolute

11.3.3　自定义菜单栏中的项

菜单和菜单栏中的项都遵循同样的规则显示，并且访问相同的属性。也就是说，可以像自定义菜单项一样，为菜单栏添加分隔符、单选按钮、复选框和图标。

代码清单 11-15 通过 XML 数据自定义菜单栏，与前面菜单的示例相似。

代码清单11-15　在XML数据中通过特殊属性自定义菜单栏

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:XMLListCollection id="myMenuData">
    <mx:XMLList>
      <menuitem label="Edit">
        <submenu label="Users" icon="addPerson" />
        <submenu type="separator"/>
        <submenu label="Reports" type="check"/>
      </menuitem>
      <menuitem label="Mode">
        <submenu label="Basic"   type="radio" groupName="grpMode"
              toggled="true"/>
        <submenu label="Advanced" type="radio" groupName="grpMode"/>
      </menuitem>
    </mx:XMLList>
  </mx:XMLListCollection>

  <mx:Script>
    <![CDATA[
      [Bindable]
      [Embed(source="addPerson.gif")]
      public var addPerson:Class;
    ]]>
  </mx:Script>

  <mx:MenuBar id="myMBar" dataProvider="{myMenuData}" labelField="@label"
          iconField="@icon"/>
</mx:Application>
```

编译并运行这个应用程序后，将看到如图 11-6 所示的自定义菜单栏。

跟前面介绍菜单示例时一样，接下来我们需要解决菜单栏与用户交互的问题。

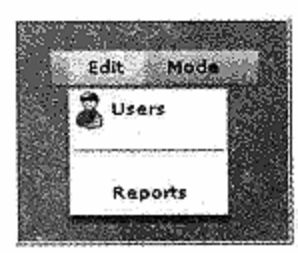


图 11-6　通过 XML 数据中的属性控制菜单栏的外观

11.3.4 处理用户与菜单栏的交互

适用于菜单的事件同样适用于菜单栏。为了方便起见，表 11-6 再次列出这些事件。

表 11-6 菜单栏与菜单拥有相同的事件

事 件	说 明
change	用户改变当前选项
itemClick	用户单击一个已经启用的菜单项
itemRollOut	鼠标指针从一个菜单项上移出
itemRollOver	鼠标指针移入到一个菜单项上
menuHide	菜单或子菜单关闭
menuShow	菜单或子菜单打开

表 11-6 所示的 ActionScript 中 MenuEvent 类的事件与相应的菜单事件具有相同的属性和值，表 11-7 所示为这些属性。

表 11-7 菜单栏中的 MenuEvent 属性

属 性	类 型	说 明
item	Object	菜单项关联的数据行
index	Number	菜单或子菜单的索引
label	String	包含作显示标签的数据字段。如果这个字段是 XML 节点的属性，应前置@声明
menu	Object	引用创建事件的菜单。不适用于菜单栏
menuBar	Object	引用创建事件的菜单栏。不适用于菜单

代码清单 11-16 所示为选择和显示菜单项的例子。

代码清单11-16 处理用户选择菜单项的操作

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white">

  <mx:XMLListCollection id="myMenuData">
    <mx:XMLList>
      <menuitem label="File">
        <submenu label="New">
          <submenu label="Project"/>
          <submenu label="Request"/>
        </submenu>
        <submenu label="Print"/>
      </menuitem>
      <menuitem label="View">
        <submenu label="Users"/>
        <submenu label="Reports"/>
      </menuitem>
```

```
    </mx:XMLList>
  </mx:XMLListCollection>
  <mx:Script>
    <![CDATA[
      import mx.events.MenuEvent;
      private function handleMenuClick(evt:MenuEvent):void
      {
        mx.controls.Alert.show("The user just clicked on " +
                               evt.item.@label);
        lastEvent.text = "Selection:" + evt.item.@label +
                       ", Position: " + evt.index;
      }
    ]]>
  </mx:Script>

  <mx:MenuBar id="myMBar" itemClick="handleMenuClick(event)"
          dataProvider="{myMenuData}" labelField="@label"/>
  <mx:Text id="lastEvent"/>
</mx:Application>
```

编译并运行这个应用程序，将看到类似于图 11-7 所示的结果。单击菜单栏中的项，会看到被单击项的信息已得到更新。

菜单栏与菜单很相似，但菜单栏是一个 MXML 组件，它接受多个顶级项。

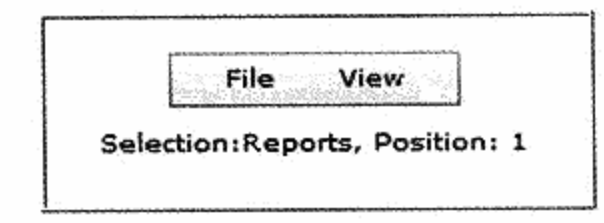


图 11-7　通过菜单栏中的 MenuEvent 对象可以确定用户选择了哪个菜单项，并显示与相应位置有关的信息

11.4　使用 ViewStack

ViewStack 是一种非常有趣的应用程序导航机制，它将每个子容器一层一层地相互摞在一起，就像是一副扑克牌，用户随时可以从这副扑克中抽出一张。

与 Flex 其他的导航容器相比，ViewStack 相对更简单。如果要实现多个容器在同一位置上互换，采用 ViewStack 可以方便地创造出新颖的导航界面。

ViewStack 的缺点是，它没有任何内置的 UI 控件可以用来辅助导航。而这些控件正是我们要在代码中添加的。

11.4.1　创建 ViewStack

ViewStack 是一个包含其他容器的容器。调用某个 ViewStack 时，要指定其中的所有容器，如代码清单 11-17 所示。

代码清单11-17　创建ViewStack，为每一层使用Canvas

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:ViewStack id="myViewStack" width="100%" height="100%">
    <mx:Canvas id="TVs" width="100%" height="100%"/>
    <mx:Canvas id="Cameras" width="100%" height="100%"/>
    <mx:Canvas id="Computers" width="100%" height="100%"/>
  </mx:ViewStack>
</mx:Application>
```

容器的类型不一定相同，代码清单 11-18 所示为根据应用程序需要混合使用几种不同类型的容器。

代码清单11-18 在一个ViewStack中混合使用多类容器

```
<mx:ViewStack id="myViewStack" width="100%" height="100%">
  <mx:Canvas id="TVs" width="100%" height="100%"/>
  <mx:HBox id="Cameras" width="100%" height="100%"/>
  <mx:VBox id="Computers" width="100%" height="100%"/>
</mx:ViewStack>
```

为了演示方便，代码清单 11-17 和代码清单 11-18 的内部容器没有包含内容；但在真正的应用程序中，这些容器都是要显示信息的。接下来我们为它们填充内容。

可以将应用程序分割为较小的片段，这种片段叫做自定义组件。也就是说，ViewStack 的每个子容器都可以另存为一个独立的 MXML 组件文件。代码清单 11-19 所示为配置方式(第 17 章将详细介绍这种方法)。

代码清单11-19 用作ViewStack容器的组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
        width="100%" height="100%">
  <mx:Panel title="TVs" width="100%" height="100%">
    <mx:TextInput id="myTI"/>
    <mx:Button label="Search for TVs"/>
  </mx:Panel>
</mx:Canvas>
```

如代码清单 11-20 所示，在应用程序中为包含自定义组件的子目录指定一个别名(正规称呼应该是名称空间)，然后就万事大吉了。

代码清单11-20 将tvs.mxml用作ViewStack的一个容器

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            xmlns:stax="stacks.*">
  <mx:ViewStack id="myViewStack" width="100%" height="100%">
    <stax:tvs id="tvs"/>
    <stax:computers id="computers"/>
  </mx:ViewStack>
</mx:Application>
```

将stacks子目录映射为别名

调用自定义组件

回顾一下，我们在此总共创建了 3 个文件，其中两个是自定义组件，它们位于子目录 stacks 中，结果如图 11-8 所示。

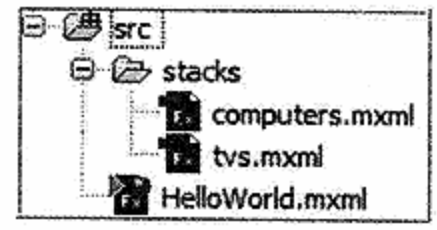


图 11-8 HelloWorld.mxml 包含一个 ViewStack，而 ViewStack 调用了 stacks 子目录中的自定义组件

最后一种方法不仅容易理解，更重要的是，可以按照模块化方式将不同功能的组件分离到一组独立的文件中。同前面的紧密耦合做法相比，显然后者更胜一筹。

接下来考虑如何在 ViewStack 容器间切换。

11.4.2　为 ViewStack 添加导航功能

导航不是自动实现的，需要明确指定 ViewStack 中的哪个容器在当前状态下可见。下列组件内置了足够多的导航功能。

- LinkBar
- ButtonBar
- TabBar

要正确使用这些组件，必须将它们的 dataProvider 属性设置为 ViewStack 的 id 值，如代码清单 11-21 所示。

代码清单11-21　为ViewStack添加导航功能

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white" xmlns:stax="stacks.*">
 <mx:ButtonBar id="myButtonBar" dataProvider="{myViewStack}"/>

 <mx:ViewStack id="myViewStack" width="100%" height="100%">
  <stax:tvs id="tvs" label="TVs"/>
  <stax:computers id="computers" label="Computers"/>
 </mx:ViewStack>

</mx:Application>
```

DataProvider是ViewStack的名称

由ButtonBar提供导航功能

添加了导航功能后，可以通过单击鼠标实现栈(stack)之间的切换，如图 11-9 所示。

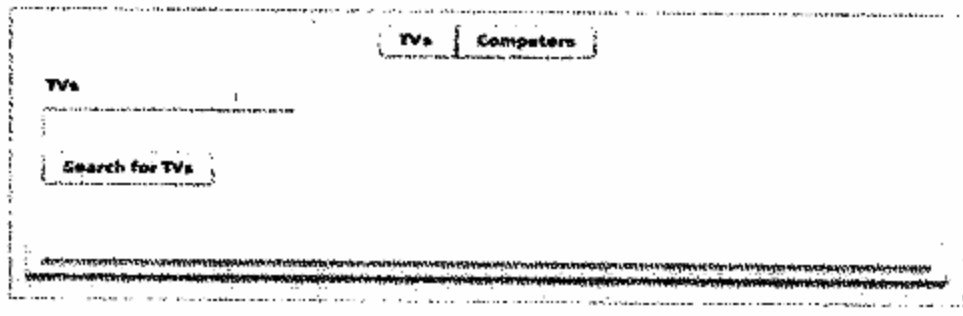


图 11-9　使用 ButtonBar 切换 ViewStack 中的栈

注意：

LinkBar、ButtonBar 和 TabBar 会根据每个栈的 label 属性显示按钮名称。

使用显示组件的默认属性，可以方便地组合栈。实际上，也可以按照需要的方式控制组件的属性。

手工控制

添加自定义导航功能并不比使用其默认属性更复杂。实现控制的两个关键属性如下。

- selectedIndex：ViewStack 中容器的位置编号，第一个位置的索引为 0。
- selectedChild：ViewStack 中给定容器的 id 属性。

代码清单 11-22 创建了两个用于切换视图的按钮。

代码清单11-22 以编程方式控制ViewStack的导航

```
<mx:Button label="Look at TVs"     click="myViewStack.selectedIndex=0"/>
<mx:Button label="Look at Computers" click="myViewStack.selectedIndex=1"/>

 <mx:ViewStack id="myViewStack" width="100%" height="100%">
  <stax:tvs id="tvs" label="TVs"/>
  <stax:computers id="computers" label="Computers"/>
</mx:ViewStack>
```

类似地，要想采用“指名道姓”的方式，可以换成以下代码。

```
<mx:Button label="Look at TVs" click="myViewStack.selectedChild=tvs"/>
<mx:Button label="Look at Computers"
        click="myViewStack.selectedChild=computers"/>
```

请记住，与 selectedChild 属性关联的是容器的 id 属性，而不是 label 属性。

提示：

在自定义导航功能时，读者可以发挥自己的创意，不必拘泥于按钮。例如，可以使用图像、单选按钮、组合框和文本链接等。

Flex 为控制 ViewStack 的导航和自定义行为提供极大的便利。在为应用程序集成自定义逻辑以便创建与上下文相关的栈时，这一点表现得尤为突出。

控制选择的栈是一种控制方式。另一种方式则是对选择做出响应，或者确定当前选定的栈。

11.4.3 确定当前选定的栈

通过 selectedIndex 和 selectedChild 属性也可以确定当前选定的容器。代码清单 11-23 所示为如何显示当前选定容器的索引及其 label 属性。

代码清单11-23 使用selectedIndex确定当前选定的栈

```
<mx:Script>
 <![CDATA[
  import mx.controls.Alert;
  private function checkSelection():void
  {
   Alert.show("Current Index:" +
            myViewStack.selectedIndex.toString() +
            " Which has a label of " +
            myViewStack.selectedChild.label);
  }
 ]]>
</mx:Script>

<mx:Button label="Check which Container is Selected"
   click="checkSelection()"/>
```

取得当前选定栈的信息

显示所选的栈

显示被选栈的标题

调用使用所选栈的函数

运行这个示例将得到如图 11-10 所示的结果。

示例中使用的是设置当前选择时使用的属性，但这里使用它取得当前选择的信息。将它们组合在一起的最后一个步骤是处理用户的选定操作。

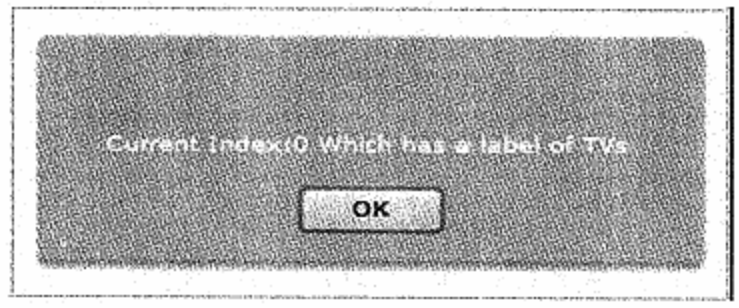


图 11-10　通过 selectedIndex 和 selectedChild 取得当前选定栈的信息

11.4.4　处理用户与 ViewStack 的交互

ViewStack 是容器类的一个子组件，因而继承了容器的所有属性。但是，由于不能与 ViewStack 直接交互，因此有一个事件值得关注——change；change 事件在 ViewStack 的选项变化时发生。

change 事件会生成 IndexChangedEvent 类的一个事件，其属性如表 11-8 所示。

表 11-8　IndexChangedEvent 类的属性

属　性	类　型	说　明
newIndex	Number	指定新 selectedIndex 的索引号
oldIndex	Number	指定 selectedIndex 的前一个索引号
relatedObject	Object	引用 newIndex 指向的容器

代码清单 11-24 所示为旧索引和新索引以及新索引的容器标签。

代码清单11-24　使用View Stack的索引属性

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white" xmlns:stax="stacks.*">

 <mx:TabBar id="myBar" dataProvider="{myViewStack}"/>

 <mx:ViewStack id="myViewStack" change="checkSelection(event)">
  <stax:tvs id="tvs" label="TVs"/>
  <stax:computers id="computers" label="Computers"/>
 </mx:ViewStack>
 <mx:Script>
  <![CDATA[
   import mx.events.IndexChangedEvent;

   private function checkSelection(evt:IndexChangedEvent):void
   {
    var newObj:Object = evt.relatedObject;
    mx.controls.Alert.show(
                 " Old Index:" + evt.oldIndex.toString() +
                 " New Index:" + evt.newIndex.toString() +
                 " Label of new Selection:" + newObj.label) ;
   }
  ]]>
 </mx:Script>

</mx:Application>
```

选项变化时调用 checkSelection

导入事件类

接收 IndexChangedEvent 对象

这个示例中使用选项卡条(TabBar)来控制 ViewStack。无论使用的是 TabBar、ButtonBar，还是 LinkBar(统称为导航组件)，它们都支持 itemClick 事件，相应事件包含的属性如表 11-9 所示。

表 11-9 itemClick 事件的属性

属 性	类 型	说 明
currentTarget	Object	引用与事件相关的对象。在代码清单 11-24 所示的例子中，引用的是导航组件。如果把事件传递给函数，函数事件无需知道组件的 id，因此可以创建出更具重用性的函数
index	Number	当前选项的位置(从 0 开始)。例如，ButtonBar 第二个按钮的索引为 1。
label	String	相关导航组件按钮的 label 属性。在这些例子中，与 ViewStack 中相应容器的 label 属性相同，因为导航组件按钮的标签默认取自该容器的 label 属性
item	Object	引用与导航组件选项相关的数据项(来自 dataProvider)
relatedObject	Object	类似于 currentTarget；但引用的不是整个导航组件，而是导航组件的给定子组件的引用(按钮或选项卡)

代码清单 11-25 所示为一个利用 itemClick 事件的例子。在这个例子中，每当发生 itemClick 事件时，就会调用一次函数。该函数会根据事件确定被选项的索引及标签，同时将被选项的颜色修改为红色。

代码清单11-25 响应对导航组件的交互操作

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white" xmlns:stax="stacks.*">
  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
      import mx.events.ItemClickEvent;
      private function handleClick(evt:ItemClickEvent):void
      {
        Alert.show(" Tab Index:" + evt.index +
                 " Label:" + evt.label);
        var myObj:Object = evt.relatedObject;
        myObj.setStyle("backgroundColor","red");
          }
        ]]>
    </mx:Script>

  <mx:TabBar id="myBar" dataProvider="{myViewStack}"
          itemClick="handleClick(event)"/>

  <mx:ViewStack id="myViewStack" width="100%" height="150">
    <stax:tvs id="tvs" label="TVs"/>
    <stax:computers id="computers" label="Computers"/>
  </mx:ViewStack>

</mx:Application>
```

指出被单击的对象

显示被单击对象

引用被单击对象

修改被单击按钮的颜色

图 11-11 所示为运行这个示例的结果。在单击一个选项卡时，弹出式窗口会显示该选项卡的索引及标签。

通过 ViewStack 的 change 事件可以操作 ViewStack；而直接使用导航组件的 itemClick 事件可以进行精细的控制。

ViewStack 与选项卡条结合，构成选项卡式导航。而 Flex 也为选项卡式导航默认提供了相应的组件。

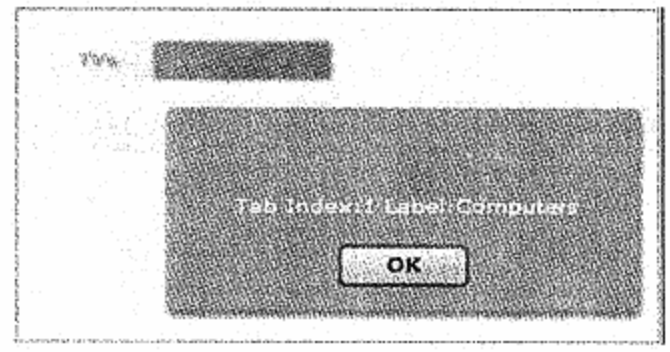


图 11-11　通过导航组件(不是 ViewStack)的 itemClick 事件可以轻易地直接与导航组件交互

11.5　TabNavigator

选项卡式导航始终是 GUI 导航的一个重要组成部分，长期以来，它被广泛应用于各种类型的应用程序。选项卡式导航可以充分利用有限的屏幕空间，让用户选择该空间中当前显示什么内容。

在代码清单 11-25 中，我们使用选项卡条控制 ViewStack，来实现选项卡式导航。但是使用 TabNavigator 组件也可以达到相同的目的。TabNavigator 派生于 ViewStack，但直接将选项卡条整合进来。

11.5.1　创建 TabNavigator

在使用 TabNavigator 时，会为其中每个容器创建一个选项卡。代码清单 11-26 展示了使用 TabNavigator 组件创建选项卡有多么简单。

代码清单11-26　使用TabNavigator创建选项卡式导航

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white">

  <mx:TabNavigator width="300" height="100">
    <mx:Canvas  label="Tab 1"/>
    <mx:VBox    label="Tab 2"/>
    <mx:HBox    label="Tab 3"/>
  </mx:TabNavigator>

</mx:Application>
```

选项卡标题

代码清单 11-26 中简洁的代码生成了图 11-12 所示的结果。

注意：

通过每个子容器的 label 属性指定相应选项卡的标题。

与 ViewStack 一样，可以混合搭配多个容器。TabNavigator 是 ViewStack 的子对象，因而继承了 ViewStack 的属性，但它提供了更多功能。

由于存在这种继承关系，处理用户与 TabNavigator 的交互也与处理用户与 ViewStack 的交互相似。

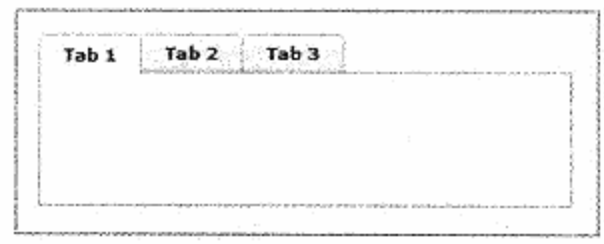


图 11-12 TabNavigator 会为其中的每个容器创建一个选项卡

11.5.2 处理用户与 TabNavigator 的交互

由于 TabNavigator 是 ViewStack 的子对象，因此可以借用 ViewStack 的属性。

- selectedIndex：当前选项卡的位置编号(从 0 开始)。
- selectedChild：当前选项卡的 id 属性。

同样，TabNavigator 的 change 事件也和 ViewStack 的相应事件一样(IndexChangedEvent 事件)，在用户切换选项卡时发生(见代码清单 11-27)。

代码清单11-27 响应TabNavigator的选项卡变化

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:Script>
   <![CDATA[
    import mx.controls.Alert;
    private function handleChange():void
    {
     Alert.show("Changed to Tab Index " + myTabNav.selectedIndex +
             "Label: " + myTabNav.selectedChild.label);
    }
   ]]>
    </mx:Script>
    <mx:TabNavigator width="200" height="100" id="myTabNav"
                change="handleChange()">
   <mx:Canvas  label="Tab 1"/>
   <mx:VBox   label="Tab 2"/>
   <mx:HBox   label="Tab 3"/>
  </mx:TabNavigator>
</mx:Application>
```

编译并运行这个应用程序，会看到一个警告窗口(见图 11-13)。警告窗口中显示了当前选项卡的索引值及其标签。

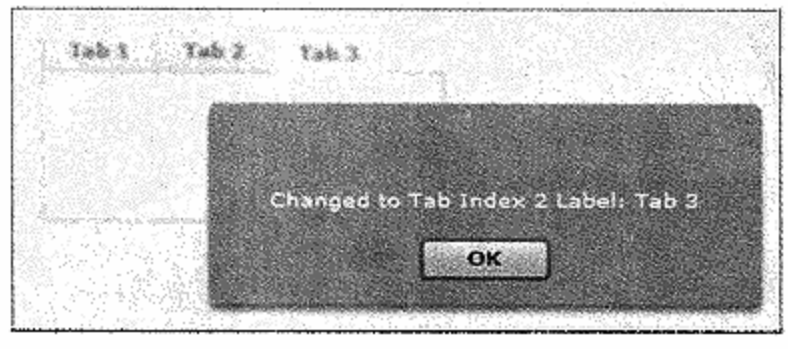


图 11-13 通过 change 事件可以确定用户选择了哪个选项卡

这个示例与前面 ViewStack 那个示例一样，都使用了 change 事件。问题在于，这个示例中的函数被明确告知使用哪个 TabNavigator。正如我们反复强调的，应该在这里采用更容易被重用的模块化技术。

向函数传递事件对象

前面的示例要求函数清晰地了解 TabNavigator 的名称(myTabNav)。但是，如果想让函数更灵活、更容易适应多选项卡导航下的需求，应该向函数传递导航组件事件，然后访问事件的 currentTarget 属性引用相关的 TabNavigator。代码清单 11-28 展示了配置过程。

代码清单11-28　通过事件对象传递当前选项卡的信息

```
<mx:Script>
 <![CDATA[
  import mx.controls.Alert;
  import mx.events.IndexChangedEvent;
  private function handleChange(evt:IndexChangedEvent):void
  {
   Alert.show("Changed to Tab Index "
            + evt.currentTarget.selectedIndex +
            " Label: "
            + evt.currentTarget.selectedChild.label);
  }
 ]]>
</mx:Script>
<mx:TabNavigator width="200" height="100" id="myTabNav"
            change="handleChange(event)">
  ...
</mx:TabNavigator>
```

接受来自任何TabNavigator的事件对象

将事件对象传递给函数

代码清单 11-28 与代码清单 11-27 的结果相同，但经过抽象，其中的函数不必事先知道 TabNavigator 的 id 属性。

回过头看表 11-9 列出的 IndexChangedEvent 属性，会发现事件对象中还包含 newIndex (及 oldIndex)属性和 relatedObject 属性。

```
private function handleChange(evt:IndexChangedEvent):void
{
 var selectedContainer:Object = evt.relatedObject;
 Alert.show("Changed to Tab Index " + evt.newIndex +
           " Label: " + selectedContainer.label);
}
```

虽然以这种方式配置 IndexChangedEvent 会带来更大的便利，但并不是唯一可行的方式；也可以根据个人偏好采取其他方式。

选项卡和菜单都有其适用的场合，而且已经成为 UI 中常用的组件。不过，接下来我们要介绍一种更有新意的导航组件——Accordion。

11.6 Accordion

当 Flex 及其实现的 Accordion 组件刚刚进入人们的视野时，Web 应用程序领域曾为之震惊，人们对如此新颖的应用程序导航方式感到惊喜。虽然 Flex 不是第一个引入这种导航组件的框架(Outlook 2003 在 Flex 之前就使用了这种导航)，但这在传统 Web 应用程序开发领域还是头一次。

今天，Accordion 在诸多 Ajax 工具包(如 Adobe 的 Spry)的推动下，已经成为人们公认的一种容易实现的导航组件，但 Flex 内置的 Accordion 组件仍然是其中的佼佼者。

11.6.1 创建 Accordion

创建 Accordion 与创建 TabNavigator 非常相似；但结果不是为每个子容器显示一个选项卡，而是显示一个分隔条(divider)。当然，如代码清单 11-29 所示，创建 Accordion 同样很简单。

代码清单11-29 创建Accordion导航器

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white">
  <mx:Accordion width="150" height="150" id="myAccord">
    <mx:Canvas  label="Section 1"/>
    <mx:VBox    label="Section 2"/>
    <mx:HBox    label="Section 3"/>
  </mx:Accordion>
</mx:Application>
```

编译并运行这个应用程序后，将看到与图 11-14 所示类似的由 3 个窗格组成的 Accordion。

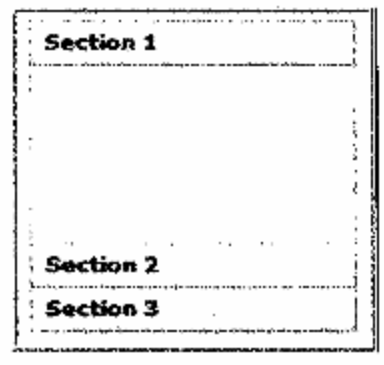


图 11-14 包含 3 个窗格的 Accordion

由此可见，创建 Accordion 的模式与创建 ViewStack 及 TabNavigator 相同，都是将子容器放到它们内部。不过，代码清单 11-29 的结果太简单了，因此还需要向其中添加一些交互性内容。

11.6.2 填充 Accordion

一般来说，可以放到容器中的内容，都可以用来填充 Accordion。例如，可以在影集应用程序的每个窗格中显示缩略图。

1. 分解表单

如果要合理地布局一个大型表单，可以使用 Accordion 把它分解成较小、较具体的部分。图 11-15 所示为一个贷款应用程序的例子。

为了保证代码便于管理，如图 11-16 所示，其中的代码使用 Accordion 将表单分解成几个不同部分。

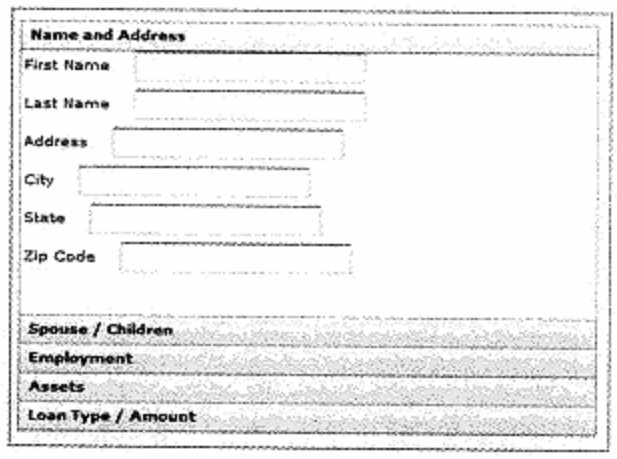


图 11-15　用 Accordion 将大型表单分解为几个不同部分

```
<mx:Accordion width="400" height="300" id="myAccord">
    <mx:VBox label="Name and Address">
        <mx:FormItem label="First Name">
            <mx:TextInput id="firstName"/>
        </mx:FormItem>
        <mx:FormItem label="Last Name">
            <mx:TextInput id="lastName"/>
        </mx:FormItem>
```

图 11-16　Accordion 中包含容器，每个容器包含表单的一部分

这里的 Accordion 使用 VBox 创建一个部分/窗格，而在其中可以填充任何内容——在此是表单。可以重复使用这种配置容器的模式，来创建每个 Accordion 窗格。

不过，这种做法不仅适用于表单，还可以适用于 Accordion 中的任何组件，包括列表。

2. 在 Accordion 中使用列表

列表是可以放在 Accordion 中的另一种常见组件。如图 11-17 所示，可以在每个部分放入 DataGrid、List 甚至 Tree 组件。

图 11-18 展示了图 11-17 中的可见 Accordion 部分的 VBox 和 List 组件。

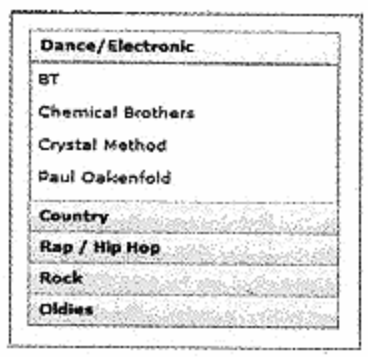


图 11-17　每个 Accordion 部分是一个包含列表的 VBox

```
<mx:Accordion width="200" height="200" id="myAccord">
    <mx:VBox label="Dance/Electronic" width="100%" height="100%">
        <mx:List width="100%" height="100%">
            <mx:Array>
                <mx:Object label="BT"/>
                <mx:Object label="Chemical Brothers"/>
                <mx:Object label="Crystal Method"/>
                <mx:Object label="Paul Oakenfold"/>
            </mx:Array>
        </mx:List>
    </mx:VBox>
```

图 11-18　Accordion 的每个部分都包含一个容器，中又包含一个 List 组件

表单和列表是用来填充 Accordion 的最常见组件；当然，也可以考虑放置其他任何组件。例如，对于一个复杂的目录结构而言，可以在每个 Accordion 部分中放置一个 Tree，以便将它分解为较小、较容易看清楚的小单位。

现在，唯一没有介绍的是处理用户与 Accordion 的交互。

11.6.3 处理用户与 Accordion 的交互

我们让读者的眼睛放松一下。处理用户与 TabNavigator 交互的内容完全适用于 Accordion。这么一说，相信读者肯定心领神会。不过，我们在此还想提醒读者，可以使用 selectedIndex 和 selectedChild 设置或确定当前的哪个 Accordion 部分被选中，然后使用 change 事件对用户操作做出响应。

大多数情况下，用户交互操作都发生在容器中的组件上，而非 Accordion(或 TabNavigator)上。

11.7 小结

为用户提供导航功能的方式很多，关键是这些方式都非常容易实现！没有复杂的 DHTML，也没有复杂的 JavaScript，轻轻松松就能够创建类似桌面应用程序的体验。而且这些都是 Flex 内置的功能。

Flex 提供了常见的、大多数用户都很熟悉的 UI 导航机制，尤其是下拉菜单和选项卡式导航。如果想增加点创意，可以使用 TabNavigator 的父类 ViewStack，并且使用按钮(或其他任何图像)切换不同的栈。如果想让用户体验一下独特的导航机制，可以试试 Accordion 组件。

除了自成体系的菜单之外，其他所有导航组件都遵循相同的模式，即放置容器，从而显示和导航每一项。菜单适合表现类似树(层次化)的数据，而且可以采用很多组合方式为菜单提供数据。不过，由于 XML 本身具有层次化的数据结构，因此往往是为菜单提供数据的首选。

掌握以上内容后，读者可以在实际开发中充分发挥自己的想象力，交替使用不同的导航组件或者以各种图片来美化它们。

作为应用程序导航这一主题的延续，第 12 章将向读者介绍弹出式窗口。主要内容是与弹出式窗口交换数据和自定义弹出式窗口。

第 12 章 弹出式窗口简介

12

本章内容：

- 创建弹出式窗口
- 缩放和定位弹出式窗口
- 与弹出式窗口通信
- 关闭弹出式窗口
- 使用警告窗口

在构建应用程序时，弹出式窗口或编辑窗口经常会成为设计的一部分。Flex 为创建、删除、定位、关闭及销毁窗口提供了一种便捷的机制，即弹出式窗口管理程序。当选择在应用程序中使用弹出式窗口时，最好事先有一个规划，确定弹出式窗口的用途及任务。

12.1 创建首个弹出式窗口

所有弹出式窗口都是通过弹出式窗口管理程序创建和管理的。弹出式窗口管理程序是一个类，它负责初始化窗口并管理窗口在应用程序中的分层布局(layering)，目的是帮助开发人员避免布置和安排窗口过程中可能的冲突。弹出式窗口管理程序没有多少属性，使用起来很简单。下面我们就介绍一个创建弹出式窗口的示例。

12.1.1 首先创建标题窗口

创建弹出式窗口的方式很多，但每种方式至少都需要一个窗口文件，用以配置窗口的显示细节。弹出式窗口管理程序会调用这个窗口文件，然后在现有层的上方再渲染一层。其他新弹出式窗口也会如法炮制，并被放到其他打开的弹出式窗口或警告窗口上方。当顶层窗口被关闭后，下面的最高层窗口则获得焦点并突出显示。这一过程会持续到最后一个弹出式窗口被关闭，而此时只剩下主应用程序层。

代码清单 12-1 所示为一个 TitleWindow 组件，该组件具有上述全部特性。

代码清单12-1 SimplePopupWindow.mxml：TitleWindow组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
layout="vertical" width="400" height="100">
  <mx:Text text="Hello there!  I'm a simple popup window.
              I don't do much.">
  </mx:Text>
</mx:TitleWindow>
```

在创建弹出式窗口时，可以选择一个容器组件作为弹出式窗口的基础(也称为基础标签)。最常见也最容易使用的基础标签是 TitleWindow。只有少数标签能作为主基础标签，我们经常用到的<Application>、<VBox>和<HBox>都不能作为弹出式窗口的基础标签使用。

在将 TitleWindow 配置为弹出式窗口的基础标签后，就可以向其中添加其他任何 MXML 或 ActionScript 了。代码清单 12-1 添加了在屏幕上显示简单文本消息的<mx:Text>标签。它只是一个向用户显示交互消息的简单示例。本章后面还将逐步展示更多更有用的示例。

接下来，我们看一看如何通过 PopUpManager 把这个窗口展示给用户。

12.1.2 使用 PopUpManager 打开窗口

配置好弹出式窗口要显示的文本，紧接着编写使用它的代码。步骤如下。

(1) 在 Flex Builder 中创建一个项目，命名为 CH12。

(2) 在 CH12 的 src 文件夹中创建一个子文件夹，命名为 windows。

(3) 将代码清单 12-1 创建的 SimplePopupWindow.mxml 复制到 src/windows 文件夹。

(4) 在 src 文件夹中创建 testSimplePopup.mxml，然后将代码清单 12-2 的内容复制过来。

代码清单12-2 testSimplePopup.mxml：显示弹出式窗口

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   creationComplete="openSimpleWindow()">
<mx:Script>
  <![CDATA[
   import mx.managers.PopUpManager;
   import windows.SimplePopupWindow;

   private var simpleWindow:Object;
   private function openSimpleWindow():void
   {
       simpleWindow =
     PopUpManager.createPopUp(this,windows.SimplePopupWindow,false);
   }
  ]]>
</mx:Script>
</mx:Application>
```

查看 windows 文件夹

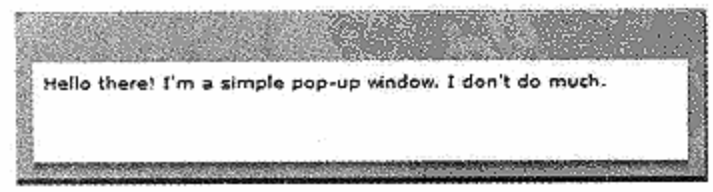


图 12-1 基本的弹出窗口

代码清单 12-2 将生成如图 12-1 所示的弹出

式窗口，该窗口随着应用程序加载完成立即打开。

这是调用新弹出式窗口最常用，也最简单的方式。其中，关键字 this——createPopUp()方法的第一个参数，表示弹出式窗口的父对象是启动该窗口的组件。

第二个参数是用来创建弹出式窗口的类或文件名。最后一个参数用于指定是否显示模态窗口(modal)。模态窗口会限制用户只能在弹出式窗口的范围内单击鼠标，即不允许在窗口的边框之外单击鼠标。在将弹出式窗口显示为模态窗口的情况下，发生在弹出式窗口外部的单击都会被忽略(不能因此返回主应用程序窗口)。

另外，代码清单 12-2 将调用 createPopUp()方法的结果赋值给一个变量。这是因为 createPopUp()返回一个初始化窗口对象，代表通过代码清单 12-1 创建的那个窗口组件。把这个返回的对象保存到变量中，就可以与其交互，具体内容将在本章后面介绍。

注意一下作用域的问题。本书前面曾经讨论过，像代码清单 12-2 这样把返回结果放在私有函数作用域中，意味着不允许函数外部代码访问该变量。如果想在函数外部访问弹出式窗口，必须将该变量声明为作用于类或组件范围。

知道了怎么创建弹出式窗口，接下来我们看一看怎么让用户关闭它。

12.1.3　关闭弹出式窗口

如图 12-2 所示，我们注意到弹出式窗口右上角有一个×图标。勿庸置疑，这个图标是各种 Web 及桌面应用程序中广为人知的关闭(close)按钮。

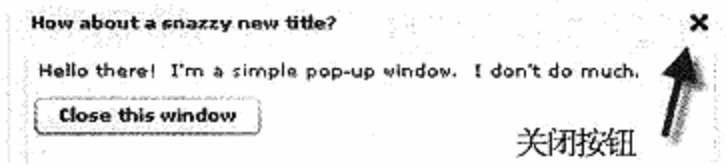


图 12-2　注意弹出式窗口右上角的关闭按钮

可以通过 TitleWindow 标签设置是否显示关闭按钮。将 TitleWindow 的 showCloseButton 属性设置为 true(默认值为 false)，弹出式窗口的右上角就会显示这个 ✖ 图标。当用户单击这个按钮时，TitleWindow 会分派一个 close 事件，用以触发侦听该事件的组件。

虽然单击关闭按钮只会发送一个事件，但通过该事件可以触发自定义的关闭逻辑。代码清单 12-3 所示为从视图中移除弹出窗口的 closeMe()函数。

代码清单12-3　关闭弹出式窗口

```
private function closeMe():void
{
  PopUpManager.removePopUp(this);
}
```

这就是关闭窗口的代码。下面我们看一看如何控制弹出式窗口的位置。

12.2　控制窗口位置

在第一次启动窗口时，我们注意到，前面示例中显示的弹出式窗口并没有自动在屏幕上居中显示。如果不明确为弹出式窗口指定位置，它们默认会出现在创建它们的父对象的左上角。

12.2.1 使用 centerPopUp()方法

PopUpManager 提供了一个居中显示(不出所料)弹出式窗口的方法——centerPopUp()。这个方法只接收一个 IFlexDisplayObject 类型的参数，即一个视觉组件。换句话说，也就是弹出式窗口的名字，例如：

```
PopUpManager.centerPopUp(IFlexDisplayObject);
```

其中的 IFlexDisplayObject 可以是任何已经定义的弹出式窗口变量。例如，在创建弹出式窗口时，如果将调用 createPopUp()返回的结果赋值给一个变量，那么传递该变量就可以在父应用程序中居中显示弹出式窗口。在前面示例的基础上，要居中窗口，可以使用以下代码。

```
simpleWindow =
   PopUpManager.createPopUp(this,windows.SimplePopupWindow,false);
PopUpManager.centerPopUp(simpleWindow as IFlexDisplayObject);
```

由于前面示例将 simpleWindow 声明为一个通用对象，Flex Builder 不知道 simpleWindow 是不是 IFlexDisplayObject。因此，需要在 centerPopUp()方法中进行转型。

注意：

centerPopUp()方法相对于父窗口居中显示弹出式窗口，而非相对于整个屏幕。

需要解释一下，centerPopUp()方法的作用恐怕与大家的直觉有些出入。不熟悉这个方法的 Flex 程序员一般都认为它会将弹出式窗口居中显示在浏览器窗口的中心位置。实际上并非如此。centerPopUp()方法会相对于创建弹出式窗口的父对象来确定中心点位置。如果这个父对象只占据了浏览器窗口右上角的一小块地方，那么 centerPopUp()方法会将弹出式窗口在那一小块地方上居中，不会让它在浏览器窗口居中。

解决这种定位问题的最简单办法，就是指定一个与应用程序根节点最接近的父对象，并将该父对象尽可能设置为与浏览器窗口相同大小。或者通过主应用程序来执行打开所有弹出式窗口的任务(例如，可以定义一个全局函数，相关组件调用它即可通过主应用程序创建弹出式窗口)。

在多数情况下，如果不是关键性的窗口定位问题，centerPopUp()方法就够用了。不过，在大多数环境中，需要管理窗口的位置。

12.2.2 计算窗口的位置

创建窗口虽然容易，但控制窗口的位置和大小是一件颇费周折的事。下面举一个例子，我们在 src/windows 目录中创建另一个名为 MoveWindow.mxml 的窗口文件，这个窗口文件对应的结果如图 12-3 所示。界面中弹出式窗口包含一些按钮，单击上面 4 个按钮可以将窗口移动到父窗口的每个角上，而单击下面的按钮则可以切换可见性属性。

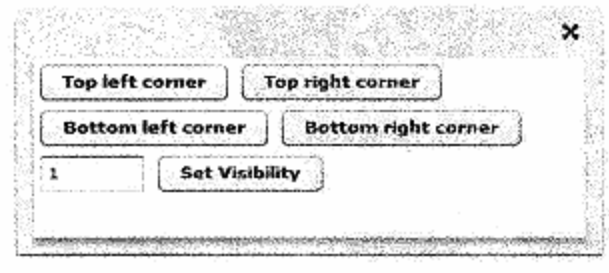


图 12-3 MoveWindow.mxml：单击按钮可以修改 x 和 y 坐标，从而重新定位弹出式窗口

虽然手动控制窗口在屏幕上的位置能够精确到像素级别，但考虑并计算窗口的位置是不可避免的。代码清单 12-4 所示为用以确定窗口当前位置及大小的代码(这些代码将在后面用来把弹出式窗口移动到父窗口的 4 个角上)。变量 this.x 和 this.y 分别对应着水平和垂直坐标。

代码清单12-4　取得弹出式窗口当前的位置和大小

```
var currentX:Number = this.x;
var currentY:Number = this.y;
var currentWidth:Number = this.width;
var currentHeight:Number = this.height;
```

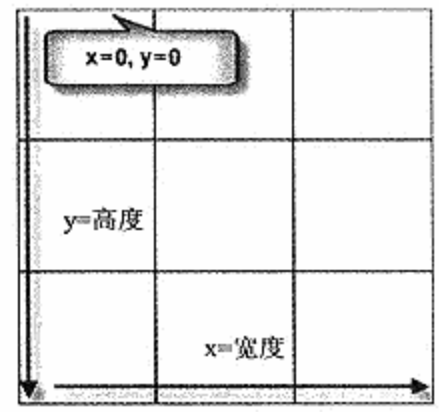


图 12-4　定位参数的形象化表示

如图 12-4 所示，可以把窗口想象成由横向纵向的直线交叉划分而成的网格。而 x 和 y 分别是用于确定窗口或任何对象在网格中位置的点。如代码清单 12-4 所示，this.x 和 this.y 就是这个网格中的点。

可以在屏幕上显示的任何对象都有 x 和 y 两个点，这两个点表示对象在屏幕上的当前位置。因此，通过 x 和 y 的值能够以像素精度在屏幕上定位对象。

对于新创建的弹出式窗口，在没有对它应用任何定位的情况下，其 x 和 y 的值都为 0(即与图 12-4 所示的左上角是对齐的)。在此基础上结合窗口的高度和宽度，就能够确定对象在屏幕上的实际位置。

以上面这些论述作为依据，代码清单 12-5 所示为如何按照弹出式窗口及父组件(在这个示例中是主应用程序)的大小计算窗口的新位置。为了看起来更自然一点，我们为每个边都添加 10 像素的边距。

代码清单12-5　MoveWindow.mxml：操作弹出式窗口的位置

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" width="340" height="150"
   showCloseButton="true" close="closeMe()">

  <mx:Script>
  <![CDATA[
    import mx.managers.PopUpManager;

    private function moveWindow(where:String):void
    {
      var newX:Number = 0;
      var newY:Number = 0;
      var buffer:Number = 10;
      switch (where)
      {
        case "toprightcorner" :
          newY = buffer;
          newX = (parent.width - this.width)-buffer;
```

```
        break;
      case "topleftcorner"  :
        newY = buffer;
        newX = buffer;
        break;
      case "bottomrightcorner" :
        newY = (parent.height - this.height)-buffer;
        newX = (parent.width - this.width)-buffer;
        break;
      case "bottomleftcorner"  :
        newY = (parent.height - this.height)-buffer;
          newX = buffer;
        break;
    }
    this.move(newX,newY);
  }

  private function closeMe():void
  {
    PopUpManager.removePopUp(this);
  }
]]>
  </mx:Script>

<mx:HBox>
  <mx:Button label="Top left corner"
          click="moveWindow('topleftcorner');"/>
  <mx:Button label="Top right corner"
          click="moveWindow('toprightcorner');"/>
 </mx:HBox>
 <mx:HBox>
    <mx:Button label="Bottom left corner"
            click="moveWindow('bottomleftcorner');"/>
    <mx:Button label="Bottom right corner"
           click="moveWindow('bottomrightcorner');"/>
  </mx:HBox>
</mx:TitleWindow>
```

根据窗口大小计算 x 和 y 坐标

前面提到过，在按照坐标计算并设置窗口位置(特别是在居中弹出式窗口)时，所有坐标的计算都必须相对于父对象的大小及位置，而不是主窗口的坐标。

在创建模态窗口时，窗口的位置非常重要。因为用户只能在模态窗口内部执行操作，在关闭窗口之前无法点击窗口边界外面的任何组件。在最极端的情况下，如果模态窗口被放到屏幕的可见空间之外，那么由于用户也无法关闭窗口，也无法通过单击屏幕上的其他任何位置来释放模态窗口对应用程序的控制，因此会导致应用程序无法使用。

指定弹出式窗口在应用程序中的位置具有功能和视觉上的双重意义。Flex 也为控制弹出式窗口的属性提供了更多手段，以便简化编程与视觉设计工作。下一节我们介绍通过 alpha 属性设置窗口的透明度。

12.3 设置窗口的透明度

可以设置弹出式窗口的不同级别的透明度，应该说这个特性能为应用程序增色不少。设置透明度是通过修改弹出式窗口的 alpha 属性实现的。虽然这个属性对所有视觉对象都

有效，但对于弹出式窗口而言，却显得格外有用和引人注目。

alpha 属性可以接受的值是数字 0～1，分别对应窗口的不透明度级别。值为 0 意味着窗口完全不可见，值为 1 意味着窗口完全可见。另外，透明设置只针对窗口本身，不针对窗口中包含的内容。

在窗口文件中，可以通过下面代码设置其透明度。

```
this.alpha = 0.4;
```

在窗口文件外部(如创建弹出式窗口的组件中)，代码是类似的，但要引用下面弹出式窗口实例。

```
moveWindow.alpha = 0.4;
```

为了控制透明度，我们添加一个接收 alpha 属性值的 TextInput 和一个设置该值的函数。首先，添加下列 MXML 组件。

```
<mx:HBox>
 <mx:TextInput id="visibilityText" text="{this.alpha}" restrict="0-9\."
           width="60" />
 <mx:Button label="Set Visibility" click="setVisibility();"/>
</mx:HBox>
```

然后再添加下列 ActionScript 函数，来提供逻辑：

```
private function setVisibility():void
{
 this.alpha = Number(visibilityText.text);
}
```

如图 12-5 和图 12-6 所示，改变弹出式窗口的透明度可以产生独特的视觉效果，也增加了控制应用程序外观的灵活性。

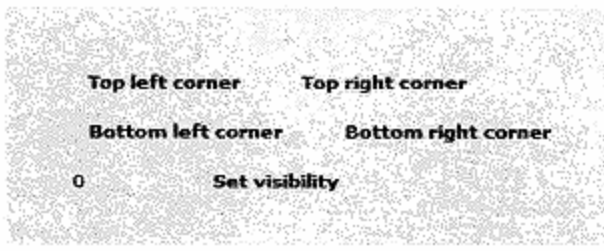


图 12-5　不透明度设置为 0 的弹出式窗口(完全看不见边框)

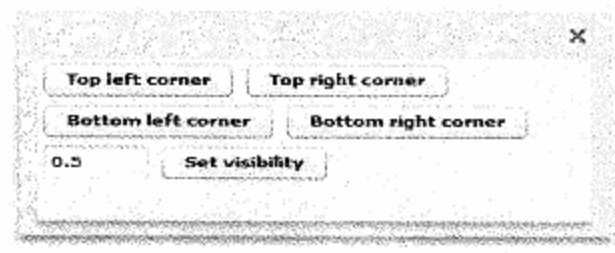


图 12-6　同一个弹出式窗口，不透明度设置为 0.5 后的结果

在掌握了创建及显示弹出式窗口的视觉方面后，下一节我们探讨一种常见的用例：弹出式表单。

12.4　弹出式窗口与数据的结合

本节我们介绍如何通过弹出式窗口检索数据(通常是用户在表单中输入的信息)或将信息发送给弹出窗口。如图 12-7 所示的一个例子，它在弹出式窗口中放入一个文本输入框，用户在其中输入的内容将被不断追加到一个变量中。

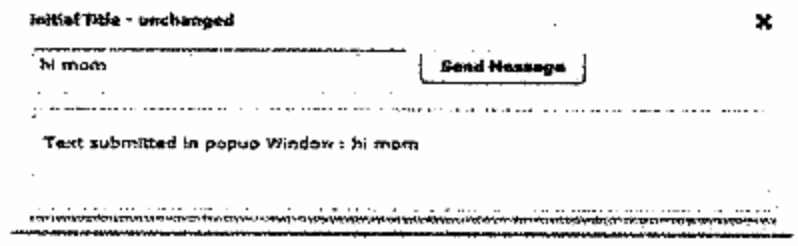


图 12-7 使用弹出式窗口接收用户输入并保存在变量中，以便将来使用

首先，创建 MXML 窗口文件，并将其命名为 AdvancedPopupWindow.mxml，保存到 src/windows 文件夹中。接着输入代码清单 12-6 所示的代码。

代码清单12-6 AdvancedPopupWindow.mxml：捕获用户输入的弹出式窗口

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
            width="500" height="300" showCloseButton="true"
            close="closeMe()" title="initial Title - unchanged"   >

 <mx:Script>
 <![CDATA[

  import mx.managers.PopUpManager;

  [Bindable] public var myPopUpVar:String = "";
  [Bindable] public var myPopupLog:String = "";

  private function closeMe():void
  {
   PopUpManager.removePopUp(this);
  }

  public function setTitle(title:String):void
  {
   this.title = title;
  }

  private function sendMessage():void
  {
   var outgoingEvent:Event = new Event("popupMessage",false);
   myPopUpVar = windowInput.text;
   myPopupLog += "\n Text submitted in popup Window : " +
              windowInput.text;
  }
 ]]>
 </mx:Script>
 <mx:HBox width="100%" height="40%">
  <mx:TextInput id="windowInput" name="windowInput" height="30"
             width="240" text=""/>
  <mx:Button id="sendMessageButton" label="Send Message"
           click="sendMessage()"/>
 </mx:HBox>
 <mx:TextArea id="pageDescription" name="pageDescription"
           text="{myPopupLog}" width="100%" height="60%"/>
</mx:TitleWindow>
```

使用组件对象打开弹出式窗口的一个最大好处是可以重用组件并保持所有状态值。

在代码清单 12-7 所示的例子中，会打开弹出式窗口，修改其中的内容，然后关闭它并重新打开；再次打开它的时候，我们会发现，窗口内容仍然保持修改后的状态。这是因为

对象既没有被销毁，也没有重新初始化。

代码清单12-7　以窗口方式使用组件对象

```
import windows.AdvancedPopupWindow;

[Bindable]
public var advancedWindow:AdvancedPopupWindow =
new AdvancedPopupWindow();
```

对于不是在组件对象中生成的窗口，每当窗口打开时，弹出式窗口都会重新创建并初始化。这种做法既有优点又有缺点，主要取决于实际需要。一方面，如果经常要显示同一个弹出式窗口，由于该窗口能够维护自身状态，因而可以减少应用程序保存相应信息的工作量。另一方面，保存窗口及其包含的数据可能造成内存资源的消耗，特别是在窗口只使用一次的情况下。

要启动并使用已有组件作为弹出式窗口，只需将弹出式窗口添加到管理程序管理的列表中即可：

```
PopUpManager.addPopUp(advancedWindow,this,false);
```

代码清单 12-7 中的脚本创建了一个组件对象变量，可以在组件、类或应用程序的作用域中访问该变量。也就是说，一旦在主应用程序或组件的作用域中创建这个对象，就可以在函数的外部访问这个对象。

如代码清单 12-7 所示，我们使用 PopUpManager.addPopUp()方法创建弹出式窗口。通过 PopUpManager.addPopUp()方法，可以将已有对象或组件添加到显示列表中。从技术角度看，createPopUp()方法创建和初始化对象，而 addPopUp()则取得先前已经初始化的对象并将其添加到显示列表中。

对于先前已经初始化的对象，重新初始化意味着销毁当前存储的所有数据。使用 addPopUp()方法，无论重复隐藏和显示多少次窗口，都不会影响窗口中的数据。这一区别非常重要，如果从窗口中取得用户修改的数据，尤为如此。

12.4.1　从弹出式窗口中取得数据

操作弹出式窗口的一半功夫会放在创建它们上；另一半功夫则用于取得它们所包含的信息。使用弹出式窗口的一种流行方式是在其中包含一个表单，以便用户填写相关信息。而构造表单和管理在其中输入的信息的方式也有很多种。

对于 Flex 新手而言，很多人都会想到在弹出式窗口中包含一种数据服务标签，如 WebService、RemoteObject 或 HTTPService，以便实现用户输入的表单数据与服务器直接通信。这种方式容易管理，但很难重用。一旦后端的 Web 服务发生变化，维护它们将成为一件令人头痛的事。

捕获表单数据的更好方式是先将它们呈现为公共变量，然后发送事件，通知父应用程序这些信息已经准备就绪，可以处理了。在接下来的几节中，我们探讨这种处理方式；不过需要先重温一下有关事件的知识。

12.4.2 发送事件

从弹出式窗口发送事件时，要记住弹出式窗口与其他组件分派事件的方式并不完全相同。

一般来说，所有事件的工作原理都一样，但弹出式窗口发送的事件有所不同。弹出式窗口的事件不会沿继承链传播到应用程序的根节点。相反，弹出式窗口分派的事件传播到窗口的根节点即停止。

这是因为弹出式窗口管理程序是弹出式窗口的父对象。结果，弹出式窗口发送的事件不会向上冒泡到主应用程序窗口，尽管该窗口才是主应用程序中所有对象的终极父对象。

弹出式窗口分派的任何事件都起始并终止于<mx:TitleWindow/>标签。从实践角度讲，这意味着为从弹出式窗口中接收到事件，需要注册要指定为侦听器的任何对象。

事件侦听器是响应窗口事件的对象(见代码清单 12-8)。这里的事件既可以是自定义事件，也可以是应用程序生成的系统事件。要进一步了解与事件有关的信息以及如何使用事件，参见第 10 章。代码清单 12-8 所示为在 advancedWindow 对象上注册事件侦听器的例子。

代码清单12-8 注册窗口事件

```
advancedWindow.addEventListener("popupMessage",getWindowData);
advancedWindow.addEventListener(CloseEvent.CLOSE,windowClose);
```

这里为 advancedWindow 对象配置了两个事件侦听器。这两个事件侦听器会时刻侦听指定的自定义事件，一旦相应事件发生就着手从窗口中收集用户提供的信息，从而做出响应(代码清单 12-9)。

代码清单12-9 分派窗口事件

```
private function closeMe():void
{
  PopUpManager.removePopUp(this);
  dispatchEvent(new Event("popupClose",false)
}
private function sendMessage():void
{
  var outgoingEvent:Event = new Event("popupMessage",false);
  myPopUpVar = windowInput.text;
  myPopupLog += "\n Text submitted in popup Window : " + windowInput.text;
  dispatchEvent( outgoingEvent);
}
```

分派 close 事件，模拟关闭窗口

通知消息已经更新

此时，当用户输入数据并单击按钮提交信息时，他们在文本框中输入的内容将被复制到一个公共变量，然后便会分派事件。这个自定义事件可能被其他组件监视，其他组件可能会继续分派该事件或者处理事件对象包含的数据。

代码清单 12-9 分派了一个自定义事件，并以该事件触发从窗口中取得数据的操作。下一节将详细解释这一过程。

12.4.3 取出数据

要从父对象中取出数据，可以访问父对象的公共属性或公共函数。正如 12.4.2 节所介

绍的，在接收到事件对象后，即可访问目标的属性及公共函数。代码清单 12-10 在主窗口中设置了事件侦听器，并在接收到 popupMessage 事件后调用相应函数。

代码清单12-10　响应窗口事件

```
advancedWindow.addEventListener("popupMessage",getWindowData);

public function getWindowData(event:Event):void
{
 mainWindowText.text += "\n Window submitted: " +
 advancedWindow.myPopUpVar;
}
```

在接收到事件对象后，可以读取窗口的公共属性并获取必需的数据。完成这个任务只需执行一个简单的函数，该函数会返回正确数据。取得数据之后，就可以进一步处理数据了。本章提供的这个完整示例以日志方式记录在窗口发生的一切。

在处理从弹出式窗口中取得数据的问题时，许多经验不太丰富的程序员总是忍不住直接访问源表单对象来获取数据。我们强烈反对这种做法，因为它违反了对象编码(object-coding)最佳实践。显然，如果将来有一天修改了表单，那么势必还要在多个位置重新编写调用代码才能确保功能正常。

从窗口中取得数据的更好方式是通过用户触发的事件，即当事件被触发时，把表单元素中的数据复制到一个窗口对象的属性中，然后再从这个属性中提取数据。另一种方式是在编辑完后，将数据保存到窗口分派的事件对象中。

前面最后提到的无论哪一种方式都有助于最大限度地重用窗口组件，同时可以保证尽可能少地修改应用程序。另外，同样的技术还会减少我们花在测试、调试及重新编码上的时间。当然，这些方式也可以用在向窗口发送数据上，只不过思路正好相反。

12.4.4　向窗口中发送数据

从窗口中取得数据或向窗口中发送数据的方式有很多种。首先，如果把窗口看成对象，那么可以从父对象中访问该窗口的所有公共属性。其次，窗口中定义的任何公共函数也可以在父对象中调用。代码清单 12-11 所示为主应用程序通过访问窗口，以便向窗口中发送数据，或从窗口中取得数据。

要向窗口中发送数据，最简单最方便的做法是调用窗口中定义的公共函数，通过该函数向窗口中传入数据。代码清单 12-11 所示为一个设置窗口标题的例子。

代码清单12-11　向弹出式窗口发送数据

```
public function launchSimplePopup():void
{
 var simpleWindow:Object =
 PopUpManager.createPopUp(this,windows.SimplePopupWindow,false);
 simpleWindow.setTitle("How about a snazzy new title?");
}

public function setTitle(newTitle:String):void
{
 pageTitle = newTitle;
}
```

在主应用程序中定义的函数

在窗口中定义的函数

这个示例很简单，但实际上可以传递更复杂的变量，如类、数组甚至对象。然后使用它们布置弹出式窗口。

代码清单 12-11 也反映出一个值得注意的问题，即如果以这种方式将窗口对象限制在函数中，则只能通过特定的函数来访问该对象。对这个示例而言，只能访问 launchSimplePopup()函数。

如果想在任何地方都能访问窗口(不仅限于在创建它的函数中)，可以以组件形式创建变量，或者在主组件体将其声明为一个应用程序级变量，如代码清单 12-12 所示。

代码清单12-12 创建一个能在函数外部访问的弹出式窗口对象

```
Private var simpleWindow:Object;

public function launchSimplePopup():void
{
 simpleWindow =
 PopUpManager.createPopUp(this,windows.SimplePopupWindow,false);
 simpleWindow.setTitle("How about a snazzy new title?");
}
```

代码清单 12-12 中既包含能在调用函数之外访问的对象，也具有根据需要动态创建该对象的灵活性。

前面探讨了如何创建弹出式窗口及如何使用 PopUpManager 管理弹出式窗口，接下来我们介绍另一种读者早就见过的弹出式窗口——警告窗口。

12.5 使用警告窗口

Alert 是专门设计用来快速创建特殊弹出式窗口的一个类。本书前面的示例中曾多次用到警告窗口。通过警告窗口可以方便地向用户显示有关重要信息(通常是错误)的忠告，也可以将它作为一种调试工具，显示代码中的返回值。如果读者熟悉 JavaScript，那么对这种调试方法一定不会陌生。

12.5.1 创建简单的警告窗口

创建警告窗口非常容易。代码清单 12-13 使用我们熟悉的代码创建了一个警告窗口。

代码清单12-13 创建简单的警告窗口

```
import mx.controls.Alert;

Alert.show("Hello World!");
```

显然，这样会创建一个简单的警告窗口。弹出式对话框唯一必需的参数是显示给用户的消息。图 12-8 所示为以上代码清单的输出结果。

图 12-8 显示“Hello World!”警告窗口，单击 OK 按钮可以关闭它

图 12-8 所示的(代码清单 12-13 创建的)警告窗口只能执行一项操作，即单击 OK 按钮关闭这个弹出式窗口。

虽然交互功能有限，但对于显示错误消息或指导用户发现问题已经足够了；只不过用户无法跟警告窗口交互而已。当然，警告窗口的功能并不止于此，下一节我们进行详细介绍。

12.5.2　挖掘警告窗口的潜力

警告窗口与一般的弹出式窗口很相似。针对它们的很多技巧也可以共享：侦听和移动事件以及改变透明度等。

但是警告窗口毕竟是一种特殊的弹出式窗口，只能对它添加或删除按钮。除了让警告窗口显示消息外，还可以为它指定下列属性。

- 标题
- 按钮(Yes、No、OK、Cancel)
- 所属对象
- 自定义处理方法

这些属性都是可选的。如果指定的按钮不止一个，就必须指定自定义处理方法。

12.5.3　高级警告窗口

一个按钮包打天下的警告窗口并不能适应所有情况。例如，想就某事迅速征询用户的意见，也许是想问一问用户在转移到下一个屏幕之前，要不要保存某个文件或当前配置。

类似地，假设我们希望向用户显示一个小小的确认警告窗口，询问用户是否在关闭窗口前保存修改。代码清单 12-14 为此创建了包含多个按钮的警告窗口。

代码清单12-14　创建高级警告窗口

```
import mx.controls.Alert;
import mx.events.CloseEvent;

private function simpleAlert(event:Event):void {
   Alert.show("Do you want to save your changes?", "Save Changes",
   Alert.YES|Alert.NO, this, simpleAlertHandler);
}
```

在这种情况下，Alert.show()方法用来创建这个弹出式警告窗口，我们为它添加了消息及标题信息。为了添加多个按钮，可以使用 Alert 静态变量指出要显示的按钮。代码清单 12-14 中的 Alert.YES 表示 Yes 按钮，Alert.NO 表示 No 按钮。

注意，指定按钮的顺序不会影响它们在警告窗口中显示的顺序。警告窗口按钮的显示顺序始终是 OK、Yes、No、Cancel。图 12-9 所示为代码清单 12-14 的输出结果。

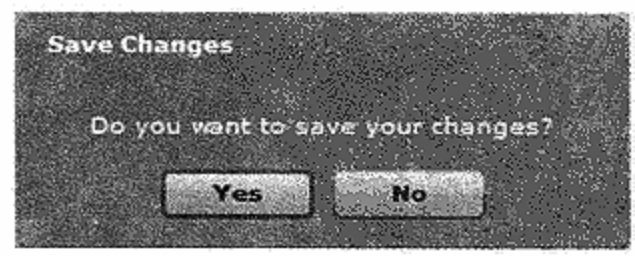


图12-9　显示Yes和No按钮的警告窗口。用户可以做出选择

代码清单 12-14 中还用到一个警告窗口处理方法，即 simpleAlertHandler。这个方法会在用户单击按钮时被调用。对于警告窗口而言，用户单击一个按钮就会自动关闭警告窗口并触发 CloseEvent 事件。

响应这个事件时，可以比较 event.detail 值与警告窗口对象的静态值。例如，代码清单 12-15 测试了 event.detail 是否等于 Alert.YES。这个代码清单也展示了图 12-9 中警告窗口的处理方法。

代码清单12-15 警告窗口处理方法

```
private function simpleAlertHandler(event:CloseEvent):void {
  if (event.detail==Alert.YES)
     status.text="You answered Yes";
  else
     status.text="You answered No";
}
```

注意：

尽可能使用对象静态变量是一种最佳实践，因为它会让代码容易阅读和理解。而且，无论什么时候需要修改这些值，只修改一个地方即可。想象一下，Alert.YES 与 1 哪一个更容易理解？虽然它们本质上没有区别，但 Alert.YES 可以让人一目了然。

代码清单 12-15 所示为响应警告窗口关闭事件的自定义处理方法。响应警告窗口的处理方法一般不会因其他事件发生而有什么不同。换句话说，在响应警告窗口时，总是要侦听 CloseEvent 事件，然后再通过这个事件对象获悉用户单击了哪个按钮。

可能经常会有所不同的是警告窗口的外观，为此需要做更多编码工作。

12.5.4 修饰警告窗口

对话框的外观可以有多种形式变化，例如改变大小、改变按钮标签以及为警告窗口添加图标等。代码清单 12-16 集这些可能的形式于一身，倾力衬托一条消息。

代码清单12-16 打扮一新的警告窗口

```
import mx.core.IFlexDisplayObject;
   import mx.controls.Alert;
   import mx.events.CloseEvent;
   import mx.managers.PopUpManager;

   [Embed(source="assets/warning.gif")]
   [Bindable]
   public var iconWarning:Class;

   private function customAlert(event:Event):void {
     var AlertObj:Alert;
     Alert.buttonWidth = 150;
     Alert.okLabel = "Disneyland";
     Alert.yesLabel = "Kennedy Space Port";
     Alert.noLabel = "Six Flags";
     Alert.cancelLabel = "Marine World";

     AlertObj = Alert.show("Where do you want to go today?", "Destination"
       ,Alert.OK|Alert.YES|Alert.NO|Alert.CANCEL,
```

分配自定义按钮标签

```
        this,customAlertHandler,iconWarning,Alert.YES);
        AlertObj.height = 150;
        AlertObj.width = 700;
        PopUpManager.centerPopUp(AlertObj as
           IFlexDisplayObject);
    }
```

使用导入的图像

居中显示警告窗口

使用 buttonLabel 属性(每个按钮都有一个)可以设置每个按钮的标签。设置 buttonWidth 属性可以指定按钮大小。但遗憾的是，buttonWidth 是一个共享属性，无论将 Alert.buttonWidth 设置为多少，这个值都会应用于每个按钮。

代码清单 12-16 还导入一幅图像并将它作为弹出式警告窗口的图标。Alert 类支持在警告消息前面显示图标。另外，还可以指定哪个按钮默认获得焦点。在这个示例中，我们将 Alert.YES(调用 Alert.show 时传递的最后一个参数)设置为默认按钮。图 12-10 所示为这个打扮一新的警告窗口。

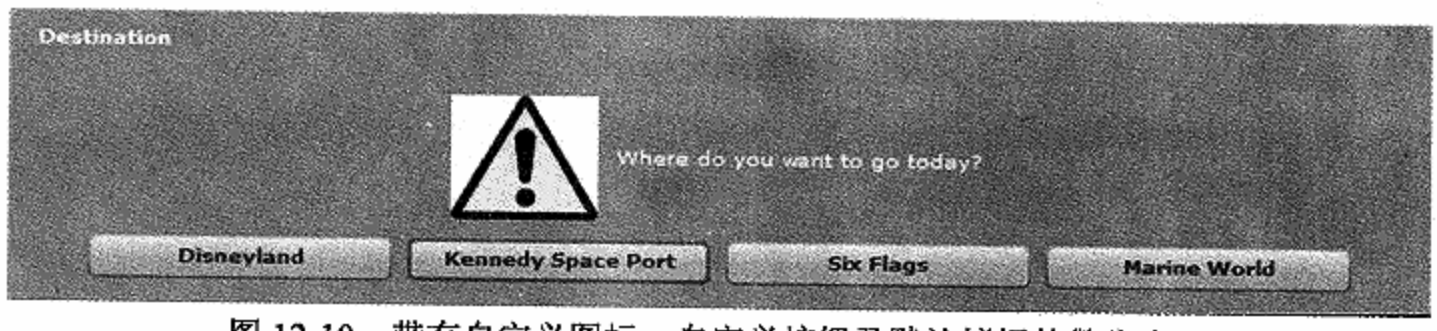


图 12-10　带有自定义图标、自定义按钮及默认按钮的警告窗口

代码清单 12-16 也将 Alert.show 函数的返回值赋给一个对象，该对象是一个初始化的 Alert 对象。当警告窗口打开后，这个对象就与标准的弹出式窗口一样。正常情况下，可以对标准弹出式窗口执行的操作，几乎都适用于这个警告窗口。例如，修改警告窗口的高度、宽度、位置及透明度。

要确定用户单击了哪个按钮，同代码清单 12-15 一样，侦听 CloseEvent 并通过 event.detail 取得相应值。

最后别忘了把修改过的所有按钮的标签改回原始值，如代码清单 12-17 所示。如果没有还原 buttonLabel 的值，后续创建的任何警告窗口都将显示相同的标签。

代码清单12-17　自定义的警告窗口处理方法

```
private function customAlertHandler(event:CloseEvent):void {
  var message:String = "Woohoo!  Looks like we're going to ";
  switch (event.detail)    {
      case Alert.YES :
           status.text = message + Alert.yesLabel;
              break;
           case Alert.NO :
             status.text = message + Alert.noLabel;
              break;
           case Alert.OK :
              status.text = message + Alert.okLabel;
```

```
                break;
            case Alert.CANCEL :
              status.text = message + Alert.cancelLabel;
        }
      Alert.okLabel = "OK";
      Alert.yesLabel = "Yes";
      Alert.noLabel = "No";
      Alert.cancelLabel = "Cancel";
    }
```

重置标签

改变警告窗口的外观是在应用程序中体现你经验丰富的一种途径。不过，改变警告窗口的功能严格来讲并不是必需的。在多数情况下，程序员都会使用 Yes/No 型警告窗口，因为它是与用户交互的一种简捷而又模态的方式。了解了如何自定义警告窗口消息后，读者又掌握了一种为自己的应用程序增添色彩的手段。

12.6 小结

很多应用程序都以这样或那样的方式在使用弹出式窗口。弹出式窗口是实现应用程序功能的好助手，但不要忘记下列几点。

- 创建弹出式窗口可以使用也可以不用 MXML 文件，但使用 MXML 组件通常更方便一些。
- 弹出式窗口传播的事件总是终止于弹出式窗口根。即窗口中的事件不会通过父对象传播到应用程序的根节点。
- 使用公共级别的对象是维护窗口状态的一种有效方式，即使窗口关闭也没有问题。而且，创建这种对象会简化向窗口中发送数据的操作。
- 弹出式窗口中可以定义公共属性和公共函数，通过它们既可以从窗口中取得数据，也可以向窗口中传递数据。
- 弹出式窗口在应用程序中的位置取决于生成该窗口的父对象的位置。在大多数情况下，应该尽可能使用最接近根节点的父对象生成弹出式窗口；至少是在屏幕中占据最大空间的组件。
- 可以修改警告窗口让它显示图标乃至 4 个按钮，按钮标签可以更改。通过 CloseEvent 及 event.detail 可以获悉用户单击了哪个按钮。
- 警告窗口与弹出式窗口的行为类似。能够对弹出式窗口执行的大多数操作，同样适用于警告窗口，包括移动位置、调整大小和改变透明度。

这一章的学习到此结束，我们很高兴读者能坚持到现在。第 13 章，将在本章展示的组件功能基础上，进一步扩展到视图状态这个主题。通过视图状态可以更改组件的外观(包括弹出式窗口)。

第 13 章 视图状态

13

本章内容：

- 理解和使用视图状态
- Flex 中的视图状态
- 视图状态过渡

前几章我们分别介绍了控件、容器、布局和各种 Adobe Flex 内置的功能，以及如何利用这些工具创建 RIA。

本章我们讨论另外一种重要功能——视图状态。借助视图状态，可以让应用程序处处显得匠心独运，为用户提供良好的体验。在探讨特定于 Flex 视图状态功能之前，我们先来介绍通常意义上的视图状态概念。

13.1 理解视图状态

RIA 的用户界面由一组不同的视图构成。传统的 Web 应用程序由无状态或静态页面(如 HTML 文件)构成。

视图可能具有不同的视觉外观，而且根据呈现它们的条件会有不同的表现。用户输入就是一种条件，例如从下拉菜单中选取一项或者单击按钮。再比如，应用程序响应发送给服务器的请求或某个算法返回的结果。简单的应用程序可以只有一个包含几种状态的视图，而复杂应用程序则包含多个视图，每个视图又包含不同的状态。

所谓视图状态(viewstate)，可以理解为某个视图(UI)的一种特定视觉外观、行为或表现。

每个视图至少有一种状态，称为该视图的*默认状态*或*基础状态*。可以根据需要为视图定义任意多种状态。

为进一步说明这个概念，我们举一个真实的例子。图 13-1 所示为 Buzzword(一种用 Adobe Flex 开发的在线字处理程序)登录表单。这幅屏幕截图展示了这个登录表单(视图)的默认(基础)状态。

图 13-2 所示为 Buzzword 登录表单的另一种状态，该状态会在用户单击图 13-1 所示界面中的 Forgot Your Password 链接后显示。

在其他很多情况下，也会用到视图状态，例如：

- 在搜索视图中，搜索表单是一种状态，搜索结果则是另一种状态。
- 在个性化布局视图中，用户通过选择不同的视图状态来自定义应用程序的外观。
- 某个视图以一种状态向用户显示数据，而以另一种状态允许用户编辑数据。

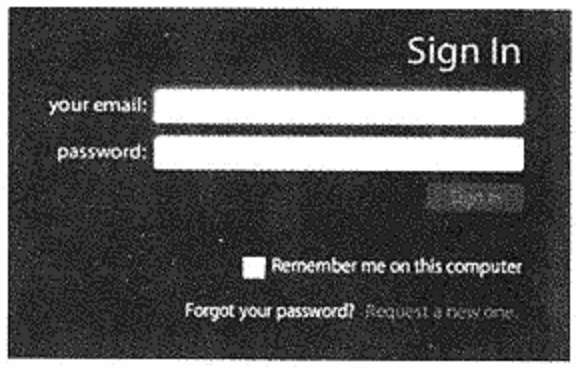


图13-1 在线文档编辑器Buzzword.com登录表单的默认视图状态

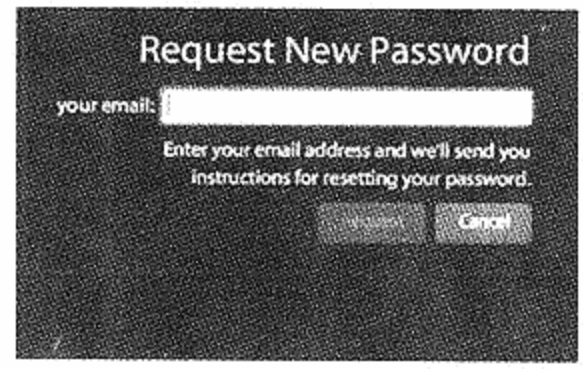


图13-2 Buzzword.com登录表单的另一种视图状态

以上是对视图状态的简单介绍。本章后面的内容将深入探讨 Adobe Flex 中的视图状态和相关功能。

13.2 Flex 中的视图状态

在 Flex 开发中，每个应用程序、自定义组件或视图都至少表现为一种状态，这种状态称为基础或默认视图状态。通过设置 states 属性的值可以定义更多状态。

基础或默认视图状态指的是在应用程序中编写组件的初始状态。其他状态都是在初始状态基础上，通过设置属性、移除或添加子对象定义而成的。代码清单 13-1 所示为了定义两个新状态的过程。

代码清单13-1 定义两个状态的视图

```
<mx:states>
   <mx:State name="ViewState1">
      <mx:SetProperty />
      <mx:SetProperty />
      <mx:RemoveChild />
      <mx:RemoveChild />
   </mx:State>
   <mx:State name="ViewState2">
   </mx:State>
</mx:states>
```

代码清单 13-1 定义了两个状态，分别名为 ViewState1 和 ViewState2。但每次只能显示一个状态，即只有一个活动状态。活动状态通过视图(这里的视图也就是应用程序或自定义组件)的 currentState 属性设置。currentState 属性的默认值是空字符串('')，这说明基础视图状态没有名称。

表 13-1 所示为 mx.core.UIComponent 定义的与视图状态相关的属性。

表 13-1 mx.core.UIComponent 类中与视图状态相关的属性

属 性	类 型	说 明
states	Array	状态对象的数组。填充这个数组以创建新视图状态，或者读取这个数组以查找可用状态
currentState	String	当前视图状态的名称。默认值为空字符串(即基础或默认视图状态)

更改视图状态时，要将应用程序的 currentState 属性的值设置为新状态的名称。代码清单 13-1 中定义了两个状态，名称分别为 ViewState1 和 ViewState2。

视图也会在状态更改前、后自动发送事件，以便通知侦听组件视图状态将要或已经更改。表 13-2 所示为与视图状态更改有关的事件。侦听这两个事件，可以分别让组件做出不同响应。要了解有关事件侦听的更多内容，参见第 10 章。

表 13-2 mx.core.UIComponent 类中与视图状态相关的事件

事 件	类 型	说 明
currentStateChanging	StateChangeEvent	这个事件会在 currentState 属性被修改后但视图状态更改前分派。 可以通过这个事件在状态更改前验证更改请求，或者执行某些操作；也可以用于阻止状态更改
currentStateChange	StateChangeEvent	这个事件在视图状态更改后分派

13.2.1 视图状态类

代码清单 13-1 展示了与视图状态相关的代码。表 13-3 列出了 mx.states 包中所有的类，并简单介绍了每个类和接口，帮助我们进一步理解代码清单 13-1 中的代码。mx.states 包中的类可以在 ActionScript 中使用，也可以声明为 MXML 组件。

表 13-3 mx.states 中定义的不同视图状态类

类/接口	说 明
State (<mx:State />)	定义视图状态。该类有一个 overrides 实例属性，值应该为实现 mx.states.IOverride 接口的对象数组
IOverride (接口)	由所有重写类实现。State 类实现了此接口。 Flex 框架内置实现了重写类，包括本表中解释的 AddChild、RemoveChild、SetProperty、SetStyle 和 SetEventHandler。 通过实现此接口来编写自定义的重写类
AddChild(<mx:AddChild />)	在视图状态改变时向视图中添加一个组件。 相关视图和组件都应该实现 mx.core.IUIComponent 接口，或者是作为 mx.core.IUIComponent 的子类。大多数 Flex 组件都是 mx.core.IUIComponent 的子类

(续表)

类/接口	说　明
RemoveChild(<mx:RemoveChild />)	在视图状态改变时从视图中移除一个组件。 相关视图和组件都应该实现 mx.core.IUIComponent 接口，或者是作为 mx.core.IUIComponent 的子类。大多数 Flex 组件都是 mx.core.IUIComponent 的子类
SetProperty(<mx:SetProperty />)	设置或重写对象的属性值，该值只在当前视图状态中有效
SetStyle(<mx:SetStyle />)	设置组件的样式值，该值只在当前视图中有效
SetEventHandler(mx:SetEventHandler />)	这个函数用于做出只对当前视图状态有效的事件处理程序更改。这样就可以只针对在具体视图状态中定义的元素设置自定义操作
Transition	用于定义从一个视图状态到另一个视图状态变化的过渡效果(参见 13.4 小节)

使用表 13-3 中列出的这些类，可以将 Flex 中的其他类和对象添加到某个状态中，或者将它们从某个状态中移除；可以控制应用程序或组件在状态变化时的过渡效果，乃至这些状态的外观。下一节将通过真实的例子具体说明这些类在创建视图状态时的用途。

13.2.2 视图状态实例

我们来看图 13-3 所示的登录表单的经典实例。这个表单是以 Buzzword 登录表单为原型构建的。其中，当用户忘记密码并索取新密码时，还会显示另一个视图状态。图 13-3 所示为基础状态，图 13-4 所示为请求新密码的另一个状态。

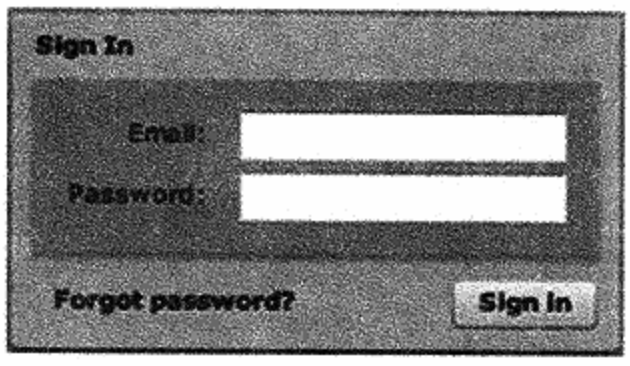


图13-3　登录表单的基础视图状态

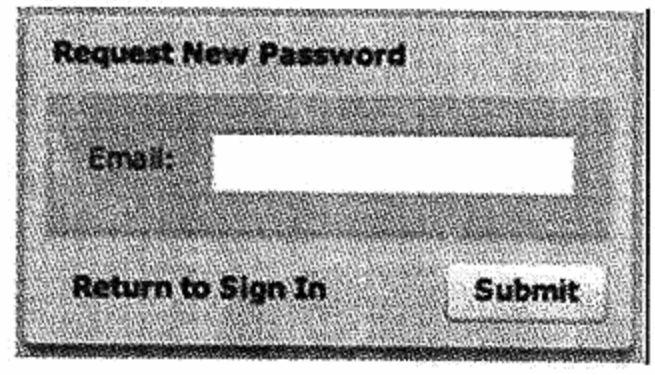


图13-4　请求新密码视图状态，有的控件已被移除，有的已经修改

生成图 13-3 的代码如代码清单 13-2 所示，从中可以清楚地看到定义每个新视图状态的过程。

代码清单13-2 登录表单视图状态的代码

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Panel title="Sign In" id="signinPanel" fontSize="12"
        backgroundAlpha="0.0">
    <mx:Form id="loginForm" backgroundAlpha="0.0">
      <mx:FormItem label="Email:"
              id="emailItem" backgroundAlpha="0.0">
        <mx:TextInput id="emailText"/>
      </mx:FormItem>
        <mx:FormItem label="Password:"
                id="passwordItem">
        <mx:TextInput id="passwordText" displayAsPassword="true"/>
      </mx:FormItem>
    </mx:Form>
    <mx:ControlBar>
      <mx:LinkButton id="forgotPasswordLink"
                label="Forgot password?"
                  click="currentState='RequestPassword'" />
      <mx:Spacer width="100%" id="spacer"/>
      <mx:Button label="Sign in" id="signinButton"/>
    </mx:ControlBar>
  </mx:Panel>

    <mx:states>
    <mx:State name="RequestPassword">
      <mx:SetProperty target="{signinPanel}" name="title"
                value="Request New Password"/>
      <mx:SetProperty target="{signinButton}" name="label"
                  value="Submit"/>
      <mx:RemoveChild target="{forgotPasswordLink}"/>
      <mx:AddChild relativeTo="{spacer}"
              position="before">
        <mx:target>
          <mx:LinkButton id="signInLink"
                    label="Return to Sign In"
                    click="currentState=''"/>
        </mx:target>
      </mx:AddChild>
      <mx:RemoveChild target="{passwordItem}"/>
    </mx:State>
  </mx:states>
</mx:Application>
```

基础状态下的界面元素

在视图中定义的 RequestPassword

新视图状态

如代码清单 13-2 所示，应用程序基础(默认)状态下的视图包含一些用户元素。而当用户单击 Forgot Password 按钮时，则会定义另一个视图状态，以便进入不同的用户界面。

上述应用程序视图状态的改变，是通过将应用程序的 currentState 属性的值修改为 RequestPassword 实现的。同样，当用户单击 Return to Sign In 按钮时，则将 currentState 属性设置为空字符串。如前所述，如果 currentState 属性的值是空字符串，会显示基础视图状态。

视图状态中的变化只对当前显示的状态有效。例如，在 RequestPassword 状态下，密码输入字段将被移除；而在默认视图状态下，该输入字段及标签会自动返回到表单中。

下面进一步分析代码清单 13-2。首先，应用程序的基础状态包含一个面板，面板中有两个表单项：一个带按钮的控件栏和一个链接按钮。然后定义另外一个视图状态，以便请求新密码。

RequestPassword 状态下的 UI 与基础状态不同。因此，更改 RequestPassword 状态时，需

要修改基础状态来满足需要。修改基础状态用到重写类：RemoveChild 用于移除表单项(密码输入字段)，SetProperty 用于分别修改面板及按钮的标题和标签，RemoveChild 用于移除 Forgot Password 按钮，而 AddChild 用于添加标签为 Return to Sign In 的按钮。

图 13-5 所示为这两个视图状态切换时用户界面中先后发生的变化。

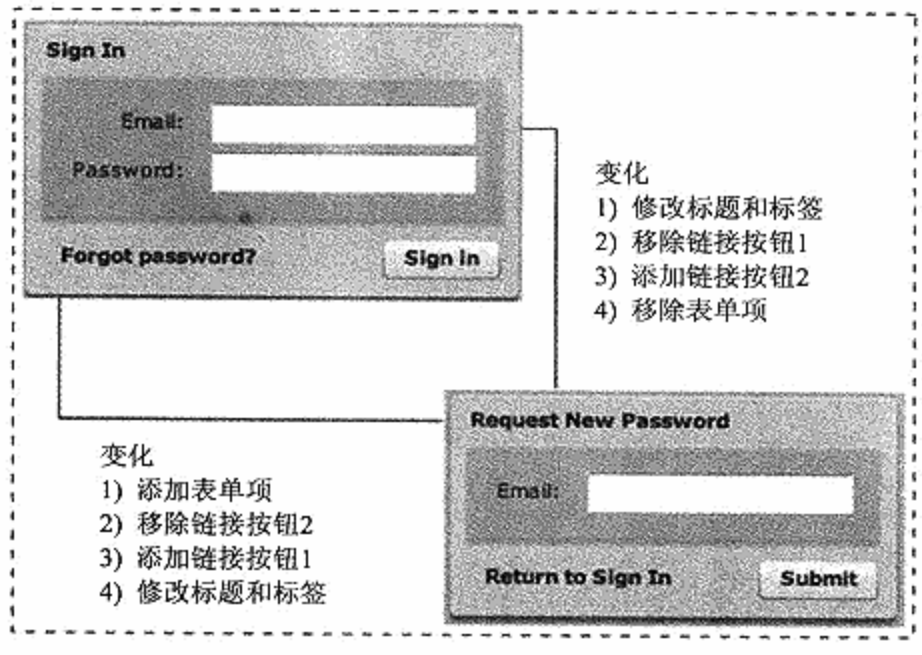


图 13-5 视图状态发生变化的顺序

当视图状态切换为 RequestPassword(currentState = 'RequestPassword')时，变化发生的顺序如下。

(1) 面板的标题被修改为 Request New Password。

(2) 按钮的标签被修改为 Submit。

(3) 移除与 forgotPasswordLink 关联的按钮。

(4) 在 Spacer 组件前添加与 signLink 关联的按钮。

(5) 移除与密码输入字段(passwordItem)相关的表单项。

相反，当视图状态从 RequestPassword 切换为基础状态(currentState='')时，变化发生的顺序如下。

(1) 还原与密码输入字段(passwordItem)相关的表单项。

(2) 移除与 signLink 关联的按钮。

(3) 在 Spacer 组件前添加与 forgotPasswordLink 关联的按钮。

(4) 按钮的标签被修改为 Sign In。

(5) 面板的标题被修改回 Sign In。

由此可见，视图状态重写时发生的变化只对当前视图状态有效。只要视图状态一变，就会以相反的顺序恢复到基础视图状态。

为了更好地理解视图状态，下面我们逐一介绍每个状态类。

13.2.3 定义视图状态(<mx:State />)

通过创建 mx.states.State 类的实例，可以为一个视图定义不同的状态。在 MXML 中，以<mx:State />标签来声明视图的不同状态。表 13-4 所示为 State 类中用于定义视图状态的

主要属性。

表 13-4 mx.states 包中的不同视图状态类

属 性	类 型	说 明
name	String	必需指定的属性。这个属性值(即视图状态的名称)，必须在视图作用域内独一无二。如前所述，可以在应用程序的根标签或任意组件中定义视图状态，但状态名称必须在定义它们的作用域中唯一
overrides	Array	实现了 IOverride 接口的对象数组。这些对象用于在视图状态切换时应用变化。实际上，它是一个包含所有变化的列表，这些变化会在一个视图切换为另一个视图时应用于基础视图
basedOn	String	指定构建当前视图状态的基准状态，必须是 State 类的实例。默认值为 null，意味着当前视图状态基于应用程序或默认组件的默认(或基础)状态创建

视图的状态(即 State 类的实例)应该有名称，该名称通过 State 类的 name 属性设置。视图状态的名称必须在定义它的作用域内独一无二。如图 13-6 所示，MyView 视图定义了另外 4 种状态；因此，这 4 种状态的名称必须在 MyView 中保持唯一。

视图状态可以指定怎样改变现有对象，包括其基准状态和重写类。使用重写类需要设置 overrides 属性，但在此之前必须明确 basedOn 属性的含义。

视图状态可以嵌套，换句话说，一个视图状态可以基于另一个视图状态构建。图 13-6 所示为一个视图(MyView)，将基础视图状态包含在内，该视图总共有 5 种状态。

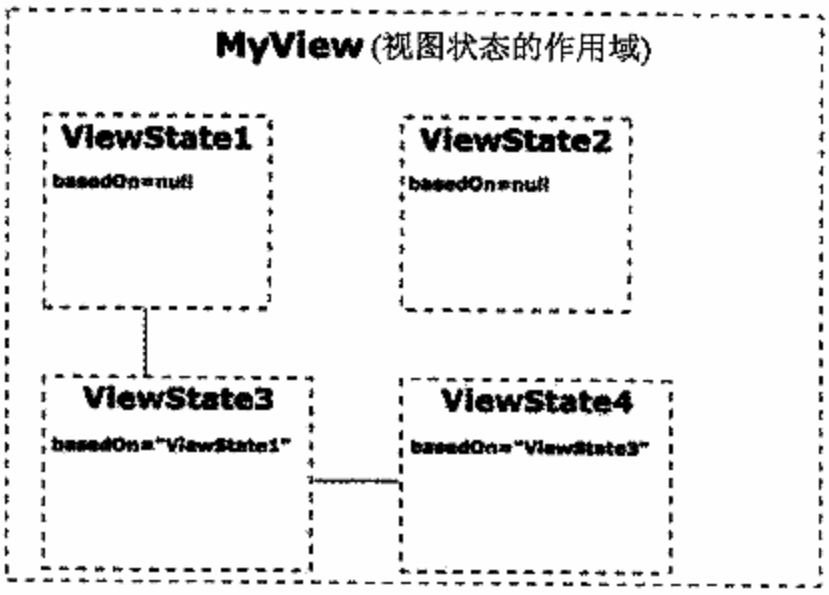


图 13-6 视图状态的作用域和 basedOn 属性

ViewState1 和 ViewState2 的基准状态是视图的基础状态，因为它们的 basedOn 属性值为默认的 null。ViewState3 的基准状态是 ViewState1，这意味着 ViewState1 中的所有 UI 对象同样存在于 ViewState3 中。

类似地，ViewState4 也有访问 ViewState3 中所有对象的权限。而通过指定 ViewState3 和 ViewState4 的 overrides 属性(或创建重写类的实例)，可以修改所有可用对象。

总而言之，当一个状态(状态 C)基于另一个状态(状态 B)构建，而另一个状态不是基础状态(状态 A)时，首先要应用第一个状态(状态 B)的变化，然后才会应用第二个状态(状态 C)的变化。这样，就可以基于其他状态构建新状态，从而保证把创建不一样但却类似状态的工作量降到最低。

下面我们通过一个例子进一步理解 basedOn 属性。代码清单 13-3 创建了两个视图状态，其中一个将另一个作为基准状态。

代码清单13-3 通过baseOn属性实现视图状态重写

```
<mx:states>
      <mx:State name="RequestPassword">
         <mx:SetProperty target="{signinPanel}" name="title"
            value="Request New Password"/>
         <mx:SetProperty target="{signinButton}" name="label"
             value="Submit"/>
         <mx:RemoveChild target="{forgotPasswordLink}"/>
         <mx:AddChild relativeTo="{spacer}" position="before">
            <mx:target>
               <mx:LinkButton id="signInLink"
            label="Return to Sign In"
         click="currentState=''"/>
            </mx:target>
         </mx:AddChild>
      </mx:State>
      <mx:State name="NewPassword" basedOn="RequestPassword" >
          <mx:RemoveChild target="{passwordItem}"/>
      </mx:State>
</mx:states>
```

创建视图状态最简单的方式是使用 MXML，不过也可以使用 ActionScript。而且使用 ActionScript 创建视图状态相对而言会更灵活一些。代码清单 13-4 所示为一个使用 ActionScript 创建视图状态的例子。

代码清单13-4 使用ActionScript创建视图状态

```
import mx.states.*;
import mx.controls.LinkButton;
private var signInLink:LinkButton = new LinkButton();

private function initFunc():void   {
  var thisState:State = new State();
  thisState.name = "RequestPassword";
  thisState.overrides.push(new SetProperty(signinPanel,
     "title","Request New Password"));
  thisState.overrides.push(new SetProperty(signinButton,"label","submit"));
  thisState.overrides.push(new RemoveChild(forgotPasswordLink) );
  thisState.overrides.push(new RemoveChild(passwordItem) );

     signInLink.label = "Return to Sign In";
     signInLink.addEventListener("click",resetCurrentState);
     thisState.overrides.push(new AddChild(spacer,signInLink,"before"));
     this.states.push(thisState);
}

private function resetCurrentState(e:Event):void   {
  this.currentState='';
}
```

如代码清单 13-3 所示，也可以借助组件的 creationComplete 事件通过 ActionScript 来创建视图状态。在创建完所有项之后，就可以为每个状态准备链接了。

视图栈中可见的组件或 UIComponent 的子类都有 states 属性。虽然大多数 UIComponent 都有 states 属性，但只有在创建自定义组件的情况下才能修改 states 属性。如果修改非自定义组件的 states 属性(或者向 states 中添加新状态)，Flex 编译器会抛出一个错误。

这个 states 属性是一个可以接收任意数量状态对象的数组。要向组件或应用程序中添加状态，只需将相应的状态对象推入(push)到 states 数组中即可。此外，每个状态对象都有

添加到 overrides 数组的任意数量的 overrides 对象。

13.2.4 视图状态过渡

视图状态主要用于改变应用程序的用户界面。在处理应用程序的界面问题时，应该记住 Flex 会将对 UI 的所有修改组织为队列，然后再一次性重绘整个界面。因此在状态切换时，用户会看到从一个状态跳到另一个状态。

仍然以前面移除和添加表单字段的登录表单为例。当用户切换状态时，他们会感觉到界面闪了一下，然后包含不同元素集的界面出现了。显然，这并不是切换视图状态的最理想方式。

通过添加过渡(transition)，可以使视图切换过程更加平滑流畅。这种技巧虽然简单，却能让应用程序更具专业水准。

过渡是组织起来的一系列 Flex 效果，当视图状态变化时这些效果会集中播放。有关效果的更多内容将在第 20 章讨论。

代码清单 13-5 所示为一个简单的过渡。

代码清单13-5 简单的过渡

```
<mx:Panel title="Sign In" id="signinPanel" fontSize="12"
      backgroundAlpha="0.0">
      <mx:Form id="loginForm" backgroundAlpha="0.0">
         <mx:FormItem label="Email:"
              id="emailItem"
              backgroundAlpha="0.0">
            <mx:TextInput id="emailText"/>
         </mx:FormItem>
         <mx:FormItem label="Password:"
                id="passwordItem">
            <mx:TextInput id="passwordText"
              displayAsPassword="true"/>
         </mx:FormItem>
      </mx:Form>
      <mx:ControlBar>
         <mx:LinkButton id="forgotPasswordLink"
           label="Forgot password?"
           click="currentState='RequestPassword'" />
         <mx:Spacer width="100%" id="spacer"/>
         <mx:Button label="Sign in" id="signinButton"/>
      </mx:ControlBar>
</mx:Panel>

<mx:states>
   <mx:State name="RequestPassword">
      <mx:SetProperty target="{signinPanel}" name="title"
          value="Request New Password"/>
      <mx:SetProperty target="{signinButton}" name="label"
        value="Submit"/>
       <mx:RemoveChild target="{forgotPasswordLink}"/>
       <mx:AddChild relativeTo="{spacer}" position="before">
         <mx:target>
          <mx:LinkButton id="signInLink"
             label="Return to Sign In" click="currentState=''"/>
         </mx:target>
       </mx:AddChild>
       <mx:RemoveChild target="{passwordItem}"/>
```

```
    </mx:State>
  </mx:states>

  <mx:transitions>
    <mx:Transition fromState="*" toState="*">
      <mx:Parallel id="t1" targets="{[signinPanel,loginForm,passwordItem]}">
        <mx:WipeLeft />
      </mx:Parallel>
    </mx:Transition>
  </mx:transitions>
```

过渡一般都要使用<mx:Parallel>或<mx:Sequence>标签来控制过渡的时间和方式。应用效果的目标通过 targets 数组指定，其中每个目标在状态切换时都会执行 wipeLeft 效果。另外，我们注意到 fromState 和 toState 的值都是星号(*)，星号的含义是所有状态切换都播放过渡。如果要指定或排除(筛选)某些状态，可以在此指定视图状态的名称。

通过筛选可以指定效果只针对特定组件播放。例如，以本章前面的登录表单为例，可以专门为某个表单元素(如密码输入字段)准备一种动画效果；或者可以基于具体的状态变化应用不同的过渡。如果应用程序中包含一些大型组件，且组件本身要处理自己的状态变化，这种技术会比较有用。

13.2.5 组件的状态

此前我们介绍的是如何处理主应用程序文件的视图状态。其实，也可以只针对某个组件定义状态。例如，可以为主应用程序中的某个子组件分别定义几种状态，而该组件可以独立于应用程序在自己的不同状态间切换。

这个过程与在主应用程序中切换和配置状态更改的过程相同。不过，通过设置组件的 currentState 属性还可以改变父应用程序的状态。

而这正是视图状态在 RIA 开发中至关重要的原因所在。只要改变组件的状态(可以在组件外部做到)，就可以改变组件的外观。这样一来，父组件不必知道自定义组件中都包含哪些项；它只要知道自定义组件有几个状态即可。因此，就可以简化监控和切换操作。代码清单 13-6 所示为在组件中添加视图状态的例子。

代码清单13-6 component.passwordBox：自定义视图状态的组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Panel title="Sign In" fontSize="12"
   backgroundAlpha="0.0" xmlns:mx="http://www.adobe.com/2006/mxml" >
   <mx:Form id="loginForm" backgroundAlpha="0.0">
      <mx:FormItem label="Email:" id="emailItem"
          backgroundAlpha="0.0">
         <mx:TextInput id="emailText"/>
      </mx:FormItem>
      <mx:FormItem label="Password:" id="passwordItem">
         <mx:TextInput id="passwordText"
           displayAsPassword="true"/>
      </mx:FormItem>
   </mx:Form>
```

```
<mx:ControlBar id="forgotControl">
   <mx:LinkButton id="forgotPasswordLink"
      label="Forgot password?"
      click="currentState='RequestPassword'" />
   <mx:Spacer width="100%" id="spacer"/>
   <mx:Button label="Sign in" id="signinButton"/>
</mx:ControlBar>

<mx:states>
   <mx:State name="RequestPassword">
      <mx:SetProperty target="{this}" name="title"
         value="Request New Password"/>
      <mx:SetProperty target="{signinButton}" name="label"
          value="Submit"/>
      <mx:RemoveChild target="{forgotPasswordLink}"/>
      <mx:AddChild relativeTo="{spacer}" position="before">
         <mx:target>
            <mx:LinkButton id="signInLink"
               label="Return to Sign In" click="currentState=''"/>
         </mx:target>
      </mx:AddChild>
      <mx:RemoveChild target="{passwordItem}"/>
   </mx:State>
</mx:states>

<mx:transitions>
    <mx:Transition fromState="*" toState="*">
        <mx:Parallel id="t1"
targets="{[this,loginForm,forgotControl,emailItem,passwordItem]}">
          <mx:WipeLeft />
        </mx:Parallel>
     </mx:Transition>
    </mx:transitions>
</mx:Panel>
```

在保存代码清单 13-6 所示的组件文件后，可以创建一个引用自定义组件的主组件(见代码清单 13-7)。

代码清单13-7 包含视图状态的组件

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute" xmlns:comp="component.*"
   creationComplete="initFunc()">
  <mx:Script>
    <![CDATA[
    import mx.collections.ArrayCollection;
    import mx.states.State;

    [bindable] private var possibleStates:Array= new Array();
     private function initFunc():void   {
      var i:Number;
      possibleStates.push('');
      for (i=0; i < myPasswordView.states.length; i++)   {
        possibleStates.push(myPasswordView.states[i].name);
      }
      viewSwitch.dataProvider = possibleStates;
    }
```

```
        private function resetState():void   {
        myPasswordView.currentState = '';
         }

        private function newPasswordState(e:Event):void   {
       myPasswordView.currentState = possibleStates[e.target.selectedIndex];
      }
     ]]>
     </mx:Script>
    <mx:ComboBox id="viewSwitch" change="newPasswordState(event)"/>

     <comp:passwordBox id="myPasswordView" x="20" y="40" />
  </mx:Application>
```

这样，只要在父应用程序中修改组件的 currentState 属性，就可以切换组件的视图状态。同时，组件负责管理和维护自己的状态列表。

代码清单 13-5 还循环遍历了 states 属性，以便为用户提供可选的状态。而组件状态则使用与本章登录窗口示例相同的名称。感兴趣的读者也可以访问 www.flexinaction.com，其中包含这个示例的代码文件。

13.3 小结

视图状态是改变应用程序外观的一种重要技术。在应用程序中使用视图状态，可以通过简单的操作实现视图之间的切换。以下是本章内容要点。

- 通过设置组件或应用程序的 currentState 属性改变视图状态。
- 使用 baseOn 属性，可以基于其他状态构建新状态，并且重用源状态的属性。
- 在状态改变时，可以针对特定对象应用过渡；而使用过渡来组织和排队效果，能够有效控制效果播放的时间和顺序。

本章介绍视图状态，以及如何以可配置的方式操作组件的视图状态。第 14 章的内容将脱离视觉主题，将讲述的重点转移到从 Web Service(在服务器端)中获取应用程序所需的数据。

第 14 章 使用数据服务

14

本章内容：

- WSDL 文档的主要元素及理解 WSDL 文档
- 连接到 Web 服务
- 通过 URLLoader 取得数据文件
- 处理 ResultEvent
- 处理 FaultEvent
- 使用 AsyncToken 来传递信息
- 使用新的 Flex Import Web Service 向导

对于许多 Flex 应用程序而言，大多数数据源都来自专为 Web 应用程序提供数据的 Web 服务。有些 Web 服务还提供了传输数据的多种不同方法。本章不会面面俱到地讨论 Web 服务的所有方法，也不会分析使用各种 Web 服务的得失。

但是，我们会全面介绍如何使用 Flex 提供的不同方法和对象(包括 RemoteObject、WebService、HTTPService 和 URLLoader 等)，把数据传送给应用程序。

理解这些方法的前提是能够看懂和理解描述 Web 服务的文档。而一种名叫 WSDL(稍后解释这个缩写词的含义)的文档是我们在使用 Web 服务时的主要信息来源。通常，我们使用的任何一种 Web 服务，都会维护并发布一份 WSDL 文档。如果有读者对 WSDL 及相关内容非常熟悉，可以跳过以下几节直接阅读数据对象部分。

14.1 什么是 WSDL 文档

首先，介绍几个 Web 服务领域的常见术语。SOAP(Simple Object Access Protocol，简单对象访问协议)是一种协议或方法，它解决将复杂对象从一台服务器传送到另一台服务器或客户端应用程序的问题。SOAP 以 XML 标签格式传送数据，客户端可以读取这些数据并据以重新创建对象。

而 WSDL(Web Services Definition Language，Web 服务定义语言)用于创建描述 Web 服务的文档。WSDL 文档中包含一组方法，而通过这些方法可以从服务器上获取文档。

对于我们而言，Flex 应用程序要读取 WSDL、解析其中的方法和 WSDL 返回的对象。然后再使用解析的信息连接到 Web 服务。在读取 Web 服务的数据和管理传输数据类型方面，Flex 负责完成许多基础性工作。

在研究 Flex 代码之前，必须先理解 WSDL 文档。WSDL 文档中包含要向服务器发送什么数据、发送该数据要使用什么方法以及服务器返回什么对象等信息。

14.2 理解 WSDL 文档

本节将以一个 WSDL 文档为例，示范如何从中找到重要信息。WSDL 文档包含几个不同的部分，分别描述与 Web 服务交互的方式。接下来我们要讨论以下主要部分。

- 可以调用的方法和操作(operation)。
- 向服务器发送的信息。
- 从服务器获得什么数据。

为了说明问题，我们将在下面的示例中使用一个简单的 WSDL 文档，该文档位于 http://www. webservicex.net/ WeatherForecast.asmx?WSDL。选择这个 Web 服务的原因，主要是它比较简单，只包含两种操作和最低限度的输入。接下来先看 WSDL 文档中最重要的部分——操作(或方法)。

14.2.1 理解 WSDL 文档：操作

打开上述 WSDL 文档，滚动到文档中部，可以看到有关操作的定义。这部分定义了可以在当前 Web 服务中使用的所有方法。代码清单 14-1 展示操作定义。

代码清单14-1 WSDL操作示例

```
<wsdl:portType name="WeatherForecastSoap">
   <wsdl:operation name="GetWeatherByZipCode">
      <documentation xmlns="http://schemas.xmlSOAP.org/wsdl/">
            Get one week weather forecast for a valid Zip Code(USA)
      </documentation>
      <wsdl:input message="tns:GetWeatherByZipCodeSoapIn" />
      <wsdl:output message="tns:GetWeatherByZipCodeSoapOut" />
   </wsdl:operation>
   <wsdl:operation name="GetWeatherByPlaceName">
      <documentation xmlns="http://schemas.xmlSOAP.org/wsdl/">
       Get one week weather forecast for a place name(USA)
      </documentation>
     <wsdl:input message="tns:GetWeatherByPlaceNameSoapIn" />
     <wsdl:output message="tns:GetWeatherByPlaceNameSoapOut" />
   </wsdl:operation>
</wsdl:portType>
```

<wsdl:portType>标签中包含方法的定义，调用这些方法可以对服务器执行操作。代码清单 14-1 中包含两个方法定义：getWeatherByZipcode 和 getWeatherByPlaceName，由<wsdl:operation>标签定义。定义内容之后是说明和信息。

在找到操作之后，可以进一步找到输入和输出类型。请记住这个位置(最好是作笔记)，因为后面需要多次查看这部分的代码。在了解可以对当前 Web 服务采取什么操作后，接下

来要知道向它们发送什么信息。

14.2.2　理解 WSDL 文档：输入类型

在代码清单 14-1 所示的对象定义中，有一个名为<wsdl:input>的标签。这个标签中包含一个指针，指向文档中的另一个部分，该部分详细描述了系统期望接收什么样的输入参数。这些参数是变量或小块信息，服务器根据它们来决定如何处理请求及返回什么结果。如代码清单 14-2 所示，<wsdl:message>标签的 name 属性包含的信息是"tns:GetWeatherByZipCodeSoapIn"。这条信息与前面的代码清单 14-1 中的 WSDL 文档中的代码匹配。

代码清单14-2　WSDL文档的输入消息

```
<wsdl:message name="GetWeatherByZipCodeSoapIn">
  <wsdl:part name="parameters" element="tns:GetWeatherByZipCode"/>
</wsdl:message>
```

代码清单 14-3 所示为包含在 WSDL 文档类型(types)部分的 GetWeatherByZipCode 元素(与定义的方法同名)。据此可知，GetWeatherByZipCode 方法接收一个字符串参数——ZipCode。

代码清单14-3　WSDL文档中定义的输入元素

```
<s:element name="GetWeatherByZipCode">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="ZipCode"
          type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
```

读者可以通过翻看整个 WSDL 文档查找所有信息；但首先找到操作名称，再根据相关信息向下查找会更方便。

在弄清楚输入参数后，接下来理解输出对象。

14.2.3　理解 WSDL 文档：输出

跟大多数 Web 服务一样，返回值总比输入数据复杂得多。要查找输出定义，仍然可以从代码清单 14-1 中的同一位置找起。在操作定义中，可以看到<wsdl:output>标签指向代码清单 14-4 所示的 message 属性。

代码清单14-4　WSDL文档中定义输出的代码

```
<wsdl:message name="GetWeatherByZipCodeSoapOut">
  <wsdl:part name="parameters" element="tns:GetWeatherByZipCodeResponse" />
</wsdl:message>
```

继续在 WSDL 文档的类型部分查找，将会看到定义当前 Web 服务应该返回什么数据类型的元素(代码清单 14-5)。

代码清单14-5 WSDL文档中定义的元素

```
<s:element name="GetWeatherByZipCodeResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="GetWeatherByZipCodeResult"
         type="tns:WeatherForecasts" />
    </s:sequence>
  </s:complexType>
</s:element>
```

根据 WSDL 文档可以了解到，GetWeatherByZipCode()方法的响应应该包含 WeatherForecasts 项(即 type="tns:WeatherForecasts")。如代码清单 14-6 所示，名为 WeatherForecasts 的 complexType 的定义如下。

代码清单14-6 WSDL文档中定义的WeatherForecasts对象的代码

```
<s:complexType name="WeatherForecasts">
   <s:sequence>
      <s:element name="Latitude" type="s:float" />
      <s:element name="Longitude" type="s:float" />
      <s:element name="AllocationFactor" type="s:float" />
      <s:element name="FipsCode" type="s:string" />
      <s:element name="PlaceName" type="s:string" />
      <s:element name="StateCode" type="s:string" />
      <s:element name="Status" type="s:string" />
      <s:element name="Details" type="tns:ArrayOfWeatherData" />
   </s:sequence>
</s:complexType>
```

WeatherForecasts 对象中包含确定的 ZipCode 的所有详细信息。包括纬度(Latitude)、经度(Longitude)及使用邮政编码表示的地名(PlaceName)等。此外，还包含一个 type="tns:ArrayOfWeatherData"元素，这是一个 WeatherData 对象数组(代码清单 14-17)。

代码清单14-17 WSDL文档中定义输出类型的代码

```
<s:complexType name="ArrayOfWeatherData">
   <s:sequence>
      <s:element name="WeatherData" type="tns:WeatherData" />
   </s:sequence>
</s:complexType>
<s:complexType name="WeatherData">
   <s:sequence>
      <s:element name="Day" type="s:string" />
      <s:element name="WeatherImage" type="s:string" />
      <s:element name="MaxTemperatureF" type="s:string" />
      <s:element name="MinTemperatureF" type="s:string" />
      <s:element name="MaxTemperatureC" type="s:string" />
      <s:element name="MinTemperatureC" type="s:string" />
   </s:sequence>
</s:complexType>
```

如代码清单 14-7 所示，Web 服务会返回一个包含天气对象的数组。数组中将包含与每天的天气有关的一幅图像和最高、最低温度(有华氏和摄氏两种单位)。

提示：

在开发基于 Web 服务的应用程序时，有许多不错的产品可以用来辅助调试。例如，

Charles 和 ServiceCapture 等产品可以帮助我们查找和分析 Web 服务，还能显示出实际的请求和响应格式。在调试和解决持续连接(persistent connection)问题时，这些工具的作用不容忽视。

在日常开发中，不可能通过手工分析 WSDL 文档来收集信息。绝大多数发布 Web 服务的组织都会提供一份文档，描述需要发送的参数和将要收到的内容，还会明确告诉使用者可以调用哪些方法。以本书作者的经验来看，需要自己费力地从 WSDL 查找适当方法的情况并不多见。

14.3　使用 WebService 组件

在知道了 Web 服务提供的方法之后，可以在应用程序中调用相应方法并解析返回的结果。有几种与外部 Web 服务通信的方法可供选择，包括 SOAP Web 服务、HTTP REST Web 服务和远程对象(RemoteObject)，这些方法各有利弊，下面将逐一介绍。

首先我们看 SOAP Web 服务对象。在连接到 SOAP Web 服务的情况下，系统(客户端应用程序及服务器端应用程序)之间通常都要交换一些 XML 信息，其中包含数据并对数据进行解释。14.2 节介绍的 WSDL 文档就是这种情况下的常见 XML 数据。

在 Flex 开发中集成 Web 服务很简单，往往只需几行代码。例如，使用<mx:WebService>标签连接前面提到的天气预报服务时，相关代码如代码清单 14-8 所示。

代码清单14-8　mx:WebService标签

```
<mx:WebService id="weatherService" wsdl="http://www.webservicex.net/
WeatherForecast.asmx?WSDL" fault="wsdlFault(event)">
<mx:operation name="GetWeatherByZipCode" result="weatherResponse(event)"
   fault="weatherFault(event)"/>
<mx:operation name="GetWeatherByPlaceName"
   result="weatherResponse(event)" fault="weatherFault(event)"/>
</mx:WebService>
```

这个<mx:WebService>标签中包含将 Flex 应用程序指向 WSDL 文档的足够信息。当初始化<mx:WebService>标签时，相应的 WSDL 文档会被解析并从中提取出必要信息，进而生成 Flex 对象；然后通过生成的 Flex 对象即可与 Web 服务实现互操作。

其中的 operation 标签定义了可以对当前 Web 服务使用的各种操作。在此，定义了 GetWeatherByZipCode 和 GetWeatherByPlaceName 两项操作。每一项操作的定义都会创建一个句柄(handle)，通过这些句柄可以指定要调用的方法。操作定义中也包含将服务器返回结果传递到何处的指令。

如前所述，Flex 客户端与服务器通过交互传递 XML 包实现通信。但是，如果每次应用程序与服务器通信都要解析 XML 数据，那么处理 XML 数据就会成为一种开发负担；使用 XML 的效果也会大打折扣。为此，Flex 将 XML 包转换为容易使用的对象(如数组或集合)，既实现了 Web 服务与 Flex 环境的无缝集成，也卸下了开发人员肩上的这副重担。

Flex 在完成上述处理后，会将结果对象转交给为 ResultEvent 的 result 属性指定的函数。在上面的示例中，我们要求 Flex 把结果对象转交给 weatherResponse()函数。如代码清单 14-8 所示，事件被传递给 result 函数(result="weatherResponse (event)")。

在调用 Web 服务和操作后，<mx:WebService>标签会生成一个结果事件。这个事件中包含从服务器中请求的数据，以及与该事件有关的具体信息(如经历的传输时间)。知道了如何在 MXML 中创建 WebService 对象之后，接下来我们介绍如何使用 ActionScript 实现同样的任务。

14.3.1 用 ActionScript 创建 WebService 组件

虽然在 MXML 中创建 WebService 组件通常比较容易，但它却不是创建 WebService 组件的唯一方式。通过 ActionScript 也可以创建 WebService 组件，而且使用 ActionScript 操作 Web 服务具有更大的灵活性。代码清单 14-9 所示为配置 WebService 对象所需的 ActionScript 代码。

代码清单14-9 用ActionScript创建WebService对象

```
private var ws:WebService;

public function createWS(wsdl:String):void   {
  ws = new WebService();
  ws.requestTimeout = 300;
  ws.wsdl = wsdl;
  ws.addEventListener(FaultEvent.FAULT,wsdlFault);
  ws.GetWeatherByZipCode.addEventListener(ResultEvent.RESULT,
     weatherResponse);
  ws.GetWeatherByZipCode.addEventListener(FaultEvent.FAULT,weatherFault);
  ws.GetWeatherByPlaceName.addEventListener(ResultEvent.RESULT,
     weatherResponse);
  ws.GetWeatherByPlaceName.addEventListener(FaultEvent.FAULT,
     weatherFault);
  ws.addEventListener(LoadEvent.LOAD,wsdlLoaded);
  if (ws.canLoadWSDL())   {
    ws.loadWSDL();
  }
}
```

如果使用 ActionScript 而非 MXML 来添加 Web 服务，必须为每项操作分别定义结果处理程序。

提示：

有时候，我们想要连接的 Web 服务可能会提供以 Flex 保留字命名的方法。此时，可以通过 WebService.getOperation("nameOfOperation")函数取得该项操作的句柄。

介绍完使用 MXML 和 ActionScript 创建 WebService 的方法后，下面我们在实践中检验一下。

14.3.2 调用 WebService 组件

要调用<mx:WebService>，应该使用 webserviceID.operationname(parameters)的格式。代码清单 14-10 所示为调用 WebService 的过程。

代码清单14-10　通过ActionScript调用WebService

```
public function getWeather():void   {
 if ( zipcode.text.length > 0 )   {
    weatherService.GetWeatherByZipCode(zipcode.text);
  } else if ( placeName.text.length > 0   )   {
    weatherService.GetWeatherByPlaceName(placeName.text);
  }
}
```

代码清单 14-10 调用了相应的函数并传递了适当的参数，即 zipcode 和 placeName。请求会被打包成 SOAP 格式，然后发送给服务器。在等待服务器响应的同时，Flex 客户端可以自由地执行其他任务。

在此，有必要指出的是，Flex 发送给 Web 服务的请求是异步的。也就是说，请求和响应可以在独立的时间点分别发生。由于应用程序不必等待服务器的响应，因而可以避免出现宝贵的处理器资源闲置的情况。异步传输也有缺点，即无法保证请求返回的顺序。

14.3.3　处理结果

服务器返回响应后，Flex 将把响应结果转交给在 WebService 声明的 result 属性中指定的函数。

```
<mx:operation name="GetWeatherByZipCode"  result="weatherResponse(event)"
  fault="weatherFault(event)"/>
```

对以上代码而言，会调用 weatherResponse 函数。而传递给该函数的请求结果是 ResultEvent 的实例。在 weatherResponse 函数中，可以进一步处理返回的结果并按照显示需要对其格式化(有关格式化数据的内容参见第 4 章)。

14.3.4　ResultEvent

当 Web 服务成功地返回结果时，会触发 ResultEvent 事件。ResultEvent(mx.rpc.events.ResultEvent)是一个非常独特的事件对象，它包含标准事件中所没有的 result、token 和 headers 属性。添加这几个属性的目的在于辅助处理 Web 服务及调用方法返回的结果。在 <mx:WebService>、<mx:HTTPService>和<mx:RemoteObject>等几种 Web 服务调用中，ResultEvent 的结构都是相同的。

ResultEvent 的行为与 Flex 框架中的其他事件一样。可以忽略、可以冒泡到父对象以便处理，也可以通过任何 Flex 模块或组件注册并侦听 ResultEvent。

此外，以取得最后结果的方式也可以访问 Web 服务操作返回的结果。

```
weatherService.GetWeatherByZipcode.lastResult;
```

WebService 的操作组件保存着上一次请求返回的最后结果，除非新结果返回或者该结果被清除。要清除结果，可以调用 clearResult()函数，如下所示。

```
weatherService.GetWeatherByZipcode.clearResult();
```

想取得由 ResultEvent 返回的结果，需要访问 ResultEvent.results 对象。这个对象包含的数据实事求是地反映了 Web 服务返回值的格式及结构。Flex 会替我们将接收到的任何数据

格式(XML、远程对象或名值对)转换为对应的变量。这些变量与 XML 中的标签具有相同的名称(区分大小写)。例如，调用 GetWeatherByZipcode()方法后，将得到与代码清单 14-6 所示相同的结构。

14.3.5 处理结果

默认情况下，SOAP 包返回的结果会被传递给 ActionScript 对象，或者传递给简单值，这取决于发送的实际数据。每个对象都根据 SOAP 对象类型及 SOAP 包所包含的信息进行转换。

也就是说，可以直接根据 Web 服务的响应来操作返回对象。对于前面执行 GetWeatherByZipcode()函数的 Web 服务而言，当结果返回时，可以像访问任何 ActionScript 对象一样访问返回的值。看一看代码清单 14-11 中的 WSDL 信息，就会明白这个对象包含了哪些信息。

代码清单14-11 处理Web服务返回的响应

```
[Bindable] public var longitude:String;
[Bindable] public var placeName:String;
[Bindable] public var arrDetails:ArrayCollection;

public function weatherResponse(e:ResultEvent):void      {
  longitude = e.result.WeatherForecasts.Longitude;
  placeName = e.result.WeatherForecasts.PlaceName;
  arrDetails = e.result.WeatherForecasts.Details;
}
```

注意：

从 Web 服务接收到的数据中包含数组的情况并不常见。事实上，本章示例中连接的 Web 服务就返回了包含数组的数据。数组虽然容易操作，但在以下情况下却会导致问题。如果返回的数组包含一个元素，由于该元素表现为普通对象，Flex 不知道要将其视为数组。此时，我们必须事先了解数组的内容并以手工方式加以处理。在 Web 服务返回的结果中包含数组时，最好将它转型为数组，例如：

```
arrDetails = e.result.WeatherForecast.details.toArray();
```

任何服务器端语言中的类型都可以被转换为 ActionScript 中的变量类型。ActionScript 中的变量类型不变，但服务器端数据类型会因使用的服务器端技术不同而异。表 14-1 所示为 ColdFusion 服务器端语言的变量类型。

表 14-1 ColdFusion 数据类型

ColdFusion 类型	ActionScript 类型
String	String
Number	Int、Uint、Number
Boolean	Boolean
Date	Date
Array	Array 或 ArrayCollection

(续表)

ColdFusion 类型	ActionScript 类型
Struct	Object(未指定类型)
Query	ArrayCollection(对象数组)
Binary	ByteArray
CFC	Object(未指定类型，除非使用 Import Web Service 向导)

查阅 Flex 在线文档可以了解与其他服务器端语言对应的 ActionScript 类型。通常，主数据类型保持不变；数值类型会被转换为数值类型，布尔类型会被转换为布尔类型，依此类推。

ResultEvent 标志着连接成功，且 Web 服务返回可以接受的结果。不过，即使 ResultEvents 被触发，最好也要检查返回的结果，以确保反馈无误。因为有些 Web 服务会在响应中包装错误信息。但是，有些 Web 服务会在发生错误时返回出错信息。下一节讨论这个问题。

14.3.6 FaultEvent

Web 服务不可能永远都返回正确的值。有时候会出现差错——连接到 Web 服务时可能出错、返回的值也可能有误。此时就要用到 FaultEvent。

FaultEvent 中包含一些与 ResultEvent 相同的信息。而且与 ResultEvent 一样，FaultEvent 也同时适用于<mx:WebService>、<mx:HTTPService>和<mx:RemoteObject>标签。 换句话说，FaultEvent 是 Flex 系统中通用的错误通信机制。当 Web 服务出错时，FaultEvent 将成为错误消息的主要包装器。这个事件对象中包含目标(即错误来源——WebService、RemoteObject 或 HTTPService)和返回的头部信息。另外，取决于哪里出现问题，这个事件对象中还会包含来自 Web 服务或 Flex 客户端的消息。

多数情况下，FaultEvent 都是由 WebService 操作或 RemoteObject 方法生成的。RemoteObject 会在 Web 服务要提醒我们请求中包含错误时发送出错事件。而当主服务对象不能连接到 Web 服务或者在读取 WSDL 遇到问题时，主服务对象也会发送出错事件。

一般来说，处理出错事件就是告知用户存在问题，或者尝试重新获取数据。无论是在哪种情况下，如代码清单 14-12 所示，都可以在不干扰用户的情况下捕获出错事件。

代码清单14-12　处理出错事件

```
public function weatherFault(e:FaultEvent):void      {
  Alert.show("an error was encountered  /n" + e.fault.faultString +
   "/n/n here are the details: /n" + e.fault.faultDetail,
   "There was an error in retrieving the weather");
}
```

初次处理出错事件时，读者可能会倾向于选择使用弹出式窗口(参考代码清单 14-8)。实际上，处理出错事件的机制是应用程序中的一次调用，这种机制为管理错误提供了较大的灵活性。例如，可以利用错误使当前会话超时。

以上介绍的用于处理错误(FaultEvent)和结果(ResultEvent)的方法，同样适用于其他数据服务。换句话说，RemoteObject、WebService 和 HTTPService 都采用相同的方法处理 Web 服务返回的错误和结果。后面将继续介绍如何创建每种数据服务组件，但处理结果的方法没有区别。

14.3.7 使用异步记号

有时候，为了确保正确解析响应，需要跟踪结果并为请求附加一些额外信息。例如，面对一组需要执行的操作，必须保证从适当的数据源获得正确结果。或者需要通过相同的操作执行几次请求，但必须知道如何处理传回来的信息。

在上述两种情况下，可以先使用异步记号(token)保存数据，然后在请求返回时取得相应数据。每一次 Web 服务请求都会生成一个唯一的异步记号(AsyncToken)，用于跟踪请求和处理返回的响应。

AsyncToken 是一个动态对象，也就是说，可以为这个对象动态添加其他参数或变量(第 16 章将详细介绍动态对象)。代码清单 14-13 显示如何为 AsyncToken 添加更多处理指令。

代码清单14-13 使用AsyncToken

```
public function getWeather():void   {
  var wASync:AsyncToken;

  if ( zipcode.text.length > 0 )   {
    wASync = weatherService.GetWeatherByZipCode(zipcode.text);
   wASync.zipCode = zipcode.text;

  }

}
```

可以在 AsyncToken 中保存任何信息；然后当结果返回时，再从中取得信息，如代码清单 14-14 所示。

代码清单14-14 从AsyncToken中提取值

```
public function weatherResponse(e:ResultEvent):void      {
  var myZip:String;
  Alert. show("looks like we got a good result","s'all good");
  myZip = e.token.zipCode;
}
```

ResultEvent.token 就是前面初始化请求时使用的异步记号。通过使用异步记号，可以确定哪个响应属于哪个请求，然后适当地处理响应。这种技术之所以有用，是因为在创建 Web 服务请求时，往往要涉及与返回结果不一样的信息。

在实际应用中，我们可能无法控制向 Web 服务发送请求和取得响应之间的延迟时间。从请求到响应的过程中，应用程序可以继续运行并响应用户操作。用户很可能会在前一次请求的结果返回之前，在文本框中修改 zipcode。或者，如果 Web 服务器响应时间过长，用户可能会提交第二次请求，而此时应用程序还在等待第一次请求的结果。在这些情况下，很容易把错误的结果返回给用户。通过使用 AsyncToken，就可以跟踪每次请求中使用的 zipcode，并据以确定返回的结果是否有效。

14.4 使用 HTTPService 检索结果

有时候，我们要连接的 Web 服务并非是 SOAP 风格的服务。但也有一些常见的框架支

持使用 HTTP 的 get 和 post 方法请求和响应数据。在 Flex 开发中，使用<mx:HTTPService>标签可以实现对这种 Web 服务的访问。

基于 HTTP 的 get 和 post 方法操作的 Web 服务有时被称为 REST(Representational State Transfer，表述性状态转移)服务。多数情况下使用 HTTP 的 post 方法就可以满足应用程序的需要。而 REST 服务也常见于大公司的大型应用程序，如雅虎和谷歌的 Web 服务。

使用 HTTP Web 服务的好处在于，它传输的数据比 SOAP 风格的 Web 服务更少。SOAP 服务通常都要向客户端发回大量文本内容，而且 SOAP 结果中也包含数量可观的元数据。不过，虽然数据传输量少了，但解析数据所需的开销相对会多一些。HTTP Web 服务的另一个好处，就是它支持的客户端比 SOAP 范围更广泛。

14.4.1　连接 HTTP Web 服务

使用<mx:HTTPService>标签很简单。为了演示需要，我们将在下面的示例中连接到 Yahoo! 并执行一次 Web 搜索。<mx:HTTPService>标签的选项与<mx:WebService>和<mx:RemoteObject>标签的选项类似。<mx:HTTPService>也会发送可以侦听的 FaultEvent 和 ResultEvent 事件，如下所示。

```
<mx:HTTPService id="yahooHTTPService"
   url="http://search.yahooapis.com/WebSearchService/V1/webSearch"
   method="GET"
   makeObjectsBindable="true" result="httpServiceResult(event)"
   fault="httpServiceFault(event)" showBusyCursor="true">
</mx:HTTPService>
```

与<mx:WebService>标签不同的是，你不必定义操作。发送到服务器进行处理的所有变量，都会以名值对形式追加到 url 字段中指定的 URL 后面。如果读者有 HTML 中的 Web 处理经验，一定熟悉这种方式。<mx:HTTPService>标签的行为与 HTML 表单执行 get 和 post 请求时的行为非常类似。HTML 表单在向服务器发送参数时使用参数名和参数值的形式，HTTPService 组件也是一样。要向服务器发送数据，只需发送一个包含变量的对象即可；Flex 会以正确的格式封装这些变量。代码清单 14-15 所示为处理 HTTPService 组件所需的代码。

代码清单14-15　操作HTTPService对象

```
public function httpServiceResult(e:ResultEvent):void   {
  Alert.show("received a result");
}

public function httpServiceFault(e:FaultEvent):void      {
  Alert.show("received a Fault");
}

public function sendHttpRequest():void   {
  var requestObj:Object = new Object();
  requestObj.appid = new String("YahooDemo");
  requestObj.query = new String("persimmon");
  requestObj.results = new int(2);
  yahooHTTPService. request = requestObj;
  httpAsyncToken = yahooHTTPService.send();
}
```

sendHTTPRequest()函数创建了一个 HTTPService 对象，接着以名值对形式对其进行填

充，然后使用 send()方法发送请求。响应的格式由 Web 服务决定。与 WebService 标签一样，响应的结果会被自动解析为变量，并通过为 HTTPService 标签定义的 result 函数返回。而返回的错误则被传递给已经定义的 fault 函数。

注意：

Flex 操作 REST 服务时并非没有问题。例如，有些 Web 服务使用 HTTP 响应代码来表示错误类型(例如，500 表示服务器错误)。可是这些错误信息往往由浏览器控制，而浏览器不会向 Flex 通报任何 500 错误消息。这个问题有时候会导致不能与 Web 服务正常通信。

与大多数 Flex 标签一样，也可以使用 ActionScript 动态生成 HTTPService 对象。代码清单 14-16 展示了这一过程。

代码清单14-16 使用ActionScript创建HTTPService对象

```
public function createHS():void   {
  yahooHTTPService = new HTTPService();
  yahooHTTPService.url =
     "http://search.yahooapis.com/WebSearchService/V1/webSearch";
  yahooHTTPService.requestTimeout;
  yahooHTTPService.request = "GET";
  yahooHTTPService.showBusyCursor = true;
   yahooHTTPService.addEventListener(ResultEvent.
      RESULT,httpServiceResult);
  yahooHTTPService.addEventListener(FaultEvent.FAULT,httpServiceFault);
}
```

上面的示例向服务传递了参数。但是，利用 Flex 的绑定特性可以自动做到这一点。

14.4.2 显式参数绑定

另一种向服务器发送信息的方式，是在 HTTPService 组件中使用参数绑定技术。代码清单 14-17 展示了一个例子。

代码清单14-17 在MXML中使用显式参数绑定

```
<mx:HTTPService id="yahooHTTPService2"
   url="http://search.yahooapis.com/WebSearchService/V1/webSearch"
   method="GET"
   makeObjectsBindable="true" result="httpServiceResult(event)"
   fault="httpServiceFault(event)" showBusyCursor="true">

  <mx:request xmlns="">
    <appid>{yahooAppID}</appid>
    <query>{queryText}</query>
    <results>{numYahooResults}</results>
  </mx:request>
</mx:HTTPService>
```

绑定参数也可以简化验证工作，因为可以将用户界面控件中的数据直接作为请求参数。而在将这些数据发送到服务器之前，可以对它们进行验证。例如，在代码清单 14-17 中，可以将相应的值与验证器关联起来，以便验证发送到服务器的数据是正确的。要了解有关验证器的更多信息，参见第 6 章。

显式参数绑定结合变量绑定既方便了对变量的访问，又可以保证变量绑定始终正确。使用参数绑定的问题在于，所有绑定都是通过硬编码实现的。因此，如果需要发送更多的参数，那么就必须按照代码清单 14-17 所示的结构类型输入变量名称。而且，参数绑定也只能在 XML 中使用，不能移植到纯 ActionScript 方式下。

14.5　使用 RemoteObject 标签

RemoteObject 与前面介绍的 Web 服务标签不同，它在发送和接收数据时使用的是二进制连接套接字。HTTPService、WebService 和 URLLoader 标签直接把对象转换成普通文本，然后再把该文本发送给服务器。

RemoteObject 方法以二进制格式维护数据对象，以 Adobe 专有的 AMF 格式发送数据，因而在基于服务器返回的数据转换和创建对象时更加方便。

例如，在 Flex 客户端与服务器之间通信的过程中，两端都需要按照原有格式接收常规的变量和对象，然后再将它们转换为接收设备可以识别的格式。这样就会带来一些开销。把对象转换为 XML 或 SOAP 更是耗费资源的过程，会导致两端都付出相当大的处理成本。

在大多数情况下，把对象转换为文本格式与转换为 AMF 二进制格式相比，需要的处理环节和资源占用更多。而基于文本创建对象也会产生同样的开销。从大小上来讲，AMF 格式的对象也要小于 SOAP 或 XML 编码的变量。

不过，AMF 也存在问题。由于 AMF 是一种专有格式，因而很多情况下都必须为服务器端提供一个脚本，以便它能够理解并将 AMF 转换为所用语言能够使用的数据类型。虽然有一些支持本地使用 RemoteObject 的工具，但 Web 服务必须针对这些工具做出显式配置。

以下是支持服务器端使用 RemoteObject 与 Flex 通信的一些工具。

- Adobe LiveCycle Data Services(Java)
- Adobe ColdFusion
- AMFPHP(http://amfphp.sourceforge.net/)
- SabreAMF (http://www.osflash.org/sabreamf)
- Midnight Coders WebORB(http://www.themidnightcoders.com/)

如果服务器端使用的是 ColdFusion，那么会方便很多。ColdFusion 7 不仅支持本地使用 RemoteObject，而且支持精简的 Flex Data Services 组件。ColdFusion 8 还在管理员配置项中提供对启用或禁用 RemoteObject 的支持。

如果想在使用 Java 整合而成的 Web 服务中支持 RemoteObject，应该获取 LiveCycle Data Services(LCDS)。LCDS 提供了与 Flex 通信的完备支持，而且还具有很多我们在此没有介绍的功能。基于 LCDS 还可以实现消息机制，即向客户端推送信息。这样就可以在维护客户端与服务器连接的同时，避免执行客户端轮询。在编写本书时，可以从 Adobe 站点免费下载到 LCDS Express。总而言之，无论服务器端采用何种技术，只要想使用 RemoteObject

组件，就必须支持基于 AMF 协议的通信。设置 RemoteObject 连接与使用其他 Web 服务标签非常相似。如代码清单 14-18 所示，还需要额外创建一个通道(Channel)对象，并在通道组(ChannelSet)中指定 URL。

代码清单14-18 在使用RemoteObject时创建一个ChannelSet

```
import mx.messaging.ChannelSet;
import mx.messaging.channels.AMFChannel;
import mx.messaging.Channel;

private function initFunc():void
{
  var cs:ChannelSet = new ChannelSet();          ← 创建通道组

  var customChannel:Channel = new AMFChannel("my-cfamf",   ← 创建通道
     "http://meatwad/Flex2gateway/");

  cs.addChannel(customChannel);                  ← 将通道添加到通道组中

  roTest.channelSet = cs;                        ← 把通道组指定给RemoteObject
}

public function roTestHelloResult(e:ResultEvent):void
{
  Alert.show("received a result from remote services");
}

public function roTestFault(e:FaultEvent):void
{
  Alert.show("received a Fault from remote services");
}

public function sendRORequest():void
{
  roAsyncToken = roTest.hello();
}

<mx:RemoteObject id="roTest" destination="ColdFusion"
  source="helloWorld" fault="roTestFault(event)" requestTimeout="500">
   <mx:method name="hello" fault="roTestFault(event)"
      result="roTestHelloResult(event)" />
</mx:RemoteObject>
```

RemoteObject 声明中的 ChannelSet 描述了如何与服务交互，指定应该针对特定的 URL(http://www.cFlex.net/Flex2gateway/)使用 AMF 协议。

可以从 Web 服务提供者那里获得这些指定的信息。在代码清单 14-18 中，我们尝试连接到 CFlex.net 的服务器。Web 服务源对象引用 Web 服务器上文件的位置。在这个示例中，Web 服务器上的文件是 http://www.CFlex.net/helloworld.cfc。如果 Web 服务的 URL 是 http://www.CFlex.net/webservice/helloworld.cfc，那么 source 参数应该扩展为 webservice.helloworld(本质上是被调用对象的路径)。

接下来的代码片段(即<mx:method>标签)引用的是服务器端 Web 服务对象(CFC，即 ColdFusion Component)的方法，名为 hello.cfc。同样，当出现错误时会发生 FaultEvent，而当成功接收到响应时，则会触发 ResultEvent。

```
<mx:method name="hello" fault="roTestFault(event)"
        result="roTestHelloResult(event)" />
```

如果服务器端使用的是 ColdFusion 或 LCDS，而且你有权控制 Web 服务，那么可以通过添加命名服务(named service)来简化访问操作。在使用命名服务(可选)时，可以指定服务器上服务对象的位置，因而在 Flex 应用程序中就不必指定该位置了。

14.6　使用 URLLoader

有时候，我们只想通过 Flex 下载一个文件。例如，从服务器上下载一幅图像或一个包含 XML 属性的文件。此时，可以使用 URLLoader。

通过 URLLoader 既可以加载并向用户显示 HTML 页面，也可以动态下载图像。另外，使用 URLLoader 加载因服务器(开发、暂存或生产服务器)而异的配置页面，也是一种常见用途。代码清单 14-19 所示为创建 URLLoader 的例子，目的是从服务器取得一个 XML 属性列表。在部署服务器上，可以使用一些配置文件形式，使用这组属性来设置环境变量。这种方法也可以用于从不同服务器上加载文件，例如从不同网站上加载 HTML 文件或从来源服务器上加载 HTML 文件。

代码清单14-19　创建URLLoader

```
public function createLoader():void   {
  var XML_URL:String = "http://www.cflex.net/urlLoaderTest.xml";
  var myXMLURL:URLRequest = new URLRequest(XML_URL);
  var myLoader:URLLoader = new URLLoader(myXMLURL);
  myLoader.dataFormat = URLLoaderDataFormat.TEXT;
  myLoader.addEventListener(Event.COMPLETE,xmlLoaded);
  myLoader.addEventListener(IOErrorEvent.IO_ERROR,loaderError);
  myLoader.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                    loaderError);
}

public function xmlLoaded(event:Event):void     {
  var myXML:XML = new XML();
  myXML = XML(myLoader.data);
  trace("Data loaded. ");
}

public function loaderError(e:FaultEvent)   {
  Alert.show("received a Fault from remote services  "+
          e.fault.message);
}
```

URLLoader 是本章讨论的最后一个 Web 服务标签。每个标签都有各自适用或不适用的情形。但对于主要的数据加载任务，HTTPService 和 WebService 标签很可能是最常用的。Flex Builder 3 还内置了一个向导，可以帮助我们完成数据加载任务，同时减少对 SOAP Web 服务的使用。

14.7 使用 Flex Import Web Service 向导

Adobe Flex Builder 3 内置了 Import Web Service 向导，能够自动读取 Web 服务并创建在 Flex 应用程序中可用的类型对象。

在前面的示例中，接收到的大多数数据都是未指定类型的对象(untyped object)。未指定类型的对象不能与 Flex 系统中的任何现有对象产生关联。通常，这都是因为 Flex 不知道返回了什么数据，而呈现这些数据并不能使内部对象与 Web 服务返回的外部对象一致。在这种情况下，可以通过手工进行繁琐地处理，以便维持服务器与客户端的联系；或者可以利用 Import Web Service 向导来创建这些对象。

虽然基于 Web 服务返回的结果创建具体对象并不是必需的，但创建正在使用的传输对象却是一种最佳实践。这样做有利于将应用程序结合在一起。

- **代码提示**：在代码中创建并使用对象，可以利用代码提示功能(Flex Builder 会显示某个类中可用的属性)。
- **代码验证**：在使用确定类型的类保存值时，Flex 会验证该值确实属于相应类型。如果访问了某个类中的无效属性，它会给出警告。关键是在编写代码期间就可以隔离问题。

有了类和对象中明确定义的各种关系，应用程序中使用这些对象的任何代码在任何时候，都能够明确知道什么可用什么不可用。显然，这对于减少开发过程中无谓的时间损失和困扰非常有好处。

14.7.1 Import Web Service 向导

要在 Flex Builder 3 中使用 Import Web Service 向导，打开 Data | Import Web Service (WSDL)。

打开 Import Web Service 向导后，它会要求我们输入要导入的 WSDL 的 URL。在下面的示例中，我们将展示使用 Import Web Service 向导，把 zipcode 天气预报服务导入应用程序的过程。

如图 14-1 所示，向导要求我们指定从何处导入 Web 服务。指定的 Web 服务必须包含一个有效的 WSDL 文件，以便正确地创建对象；否则，向导要求连接到一个 LCDS。向导会从中下载 WSDL 文件，将按照类似 14.1 节中介绍的顺序解析该文件——只不过速度更快，而且几乎无需人工参与。

单击 Next 按钮，向导会从这个 Web 服务中自动提取出所有方法，这里分别是 GetWeatherByZipcode 和 GetWeatherByPlaceName。有时候，我们可能只会使用一个特定 Web 服务其中的一个方法，这也可以在向导中进行配置。

由于这个 Web 服务允许使用不同的方法(HTTP GET、HTTP POST 和 SOAP)，因此可以根据需要从中选择要导入的版本。在本示例中，我们选择 SOAP 版本。图 14-2 所示的每

种方法看起来都很相似；但对于其他 Web 服务，这些方法恐怕会有所不同。

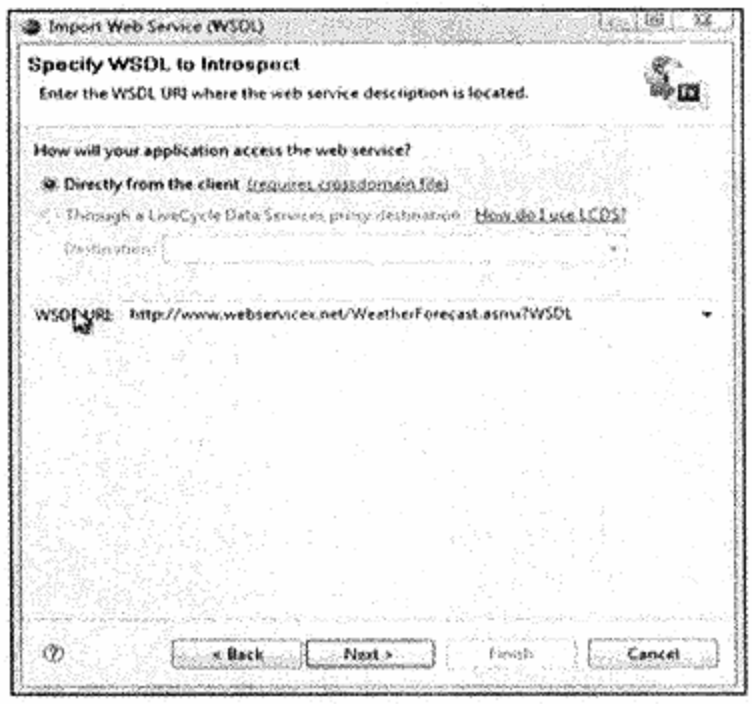


图14-1　Import Web Service向导的第一个窗口

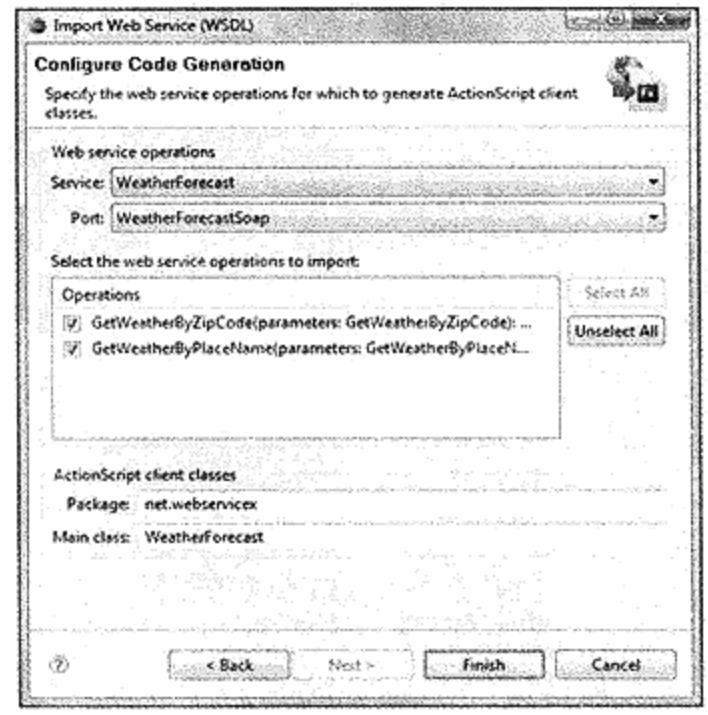


图14-2　Import Web Service向导的第二个窗口

如图 14-3 所示，通过修改包路径可以配置放置代码的位置及相应目录结构。根据配置，所有生成的 Web 服务类都将属于 Flex 中的同一个包。要了解有关包的更多内容，参见第 18 章，该章讨论如何操作自定义类。

向导会自动在 net\webservicex\文件夹中创建相应的 ActionScript 文件。然后，通过这些对象就可以与 Web 服务进行交互操作了。以这种方式使用 Web 服务的好处在于，可以根据命名的类创建内部对象。

图 14-3　在文件系统中创建的文件

14.7.2　使用生成的 Web 服务代码

使用生成的 Web 服务代码与使用常规 Web 服务很相似，但也有一些区别。首先，不使用<mx:WebServices>标签，而是使用生成的 Web 服务代码，即由生成的代码处理连接及转换对象的任务。

为了使用生成的代码，必须创建一个链接，以便 Flex 知道它们的位置。为此，可以在应用程序标签中添加如下代码。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="vertical"
   xmlns:srv="net.webservicex.*" creationComplete="initFunc()">
```

其中，xmlns:srv="net.webservicex.*"告诉 Flex Builder 在哪里能找到所需的文件。对于这个例子而言，相关文件位于系统的<FlexProjectRoot>\net\webservicex\目录中。如果修改了这个路径，只需重新指定 xmlns 属性即可。

在添加到应用程序栏之后，就可以使用 srv:扩展名来引用生成的对象了。如代码清单14-20 所示，其 MXML 代码中直接引用了向导创建的文件。而主文件则通过 WSDL 的 porttype 标签给出的名称(porttype="WeatherForecastSoap")来确定。

代码清单14-20 通过向导生成的代码来创建WebService对象

```
<srv:WeatherForecast id="myService"
   GetWeatherByPlaceName_result="placeResponse(event)"
   GetWeatherByZipCode_result="zipResponse(event)"
   fault="weatherFault(event)">
  <srv:getWeatherByZipCode_request_var>
    <srv:GetWeatherByZipCode_request ZipCode="{zipcode.text}" />
  </srv:getWeatherByZipCode_request_var>
    <srv:getWeatherByPlaceName_request_var>
   <srv:GetWeatherByPlaceName_request PlaceName="{wPlaceName.text}"/>
 </srv:getWeatherByPlaceName_request_var>
</srv:WeatherForecast>
```

如果查看向导创建的文件，会发现它创建了针对请求的对象、ResultEvent 对象、数据对象，还有一些基本对象。通过每个文件名的前缀和后缀可以确定文件类型。例如，请求对象的文件后缀为_request，基本对象的文件前缀为 Base，结果事件的后缀为 ResultEvent，数据对象没有前缀和后缀。

其中，主对象中包含与<mx:WebService>对象类似的属性。例如，它包含相应的属性，以指示用于处理任何响应的函数，以及处理任何请求错误的函数。

仔细看一看代码清单 14-20，有读者可能会问<srv:getWeatherByZipCode_request_var>的文件在哪里。文件系统不包含这个文件。但需要注意的是，它本身并非一个独立的对象或文件；它是主 Web 服务对象的一个属性，用于定义使用哪个请求对象。不过，不是必须使用生成的请求对象，也可以拿其他请求对象来代替 Flex Builder 生成的请求对象。到底用哪个请求对象，取决于实际需要。

通过 ActionScript 创建相同的 Web 服务请求元素与使用 MXML 一样简单。只不过在创建 Web 服务时，需要以稍微不同的方式设置结果处理函数，即要调用主对象自定义的添加事件侦听器函数。事件侦听器会添加所有对象并确保创建了结果时，能够返回适当类型的对象，而不是返回匿名对象。

调用这个新的自定义 Web 服务跟使用匿名的 Web 服务没有明显区别，如代码清单14-21 所示。最大的区别就是它调用了什么。

代码清单14-21 为基于向导生成的代码设置事件侦听器

```
private var myASService:WeatherForecast = new WeatherForecast();

private function initFunc():void   {
  myASService.addgetWeatherByZipCodeEventListener(zipResponse);
  myASService.addgetWeatherByPlaceNameEventListener(placeResponse);
  myASService.addEventListener(FaultEvent.FAULT,weatherFault);
}
```

真正令人兴奋的是操作响应对象(见代码清单 14-22)。虽然与处理匿名 Web 服务返回的数据没有天壤之别，但操作对象的方式变了。这一次，从服务器接收到的不是普通对象，而是强类型对象。也就是说，Flex Builder 会告诉我们对象都包含哪些属性，而且能通过编译提醒我们使用正确的值。在保存文件时，Flex 会就拼写不正确的属性给出提示，也会告知在何处尝试访问不存在的属性。这些特性一定会节省我们的时间和精力。

代码清单14-22　处理Web服务的响应

```
public function zipResponse(e:GetWeatherByZipCodeResultEvent):void
{
  Alert.show("looks like we got a good result by zipcode",
     "s'all good");
  longitude = e.result.Longitude;
  sPlaceName = e.result.PlaceName;
  arrDetails = e.result.Details;
}

public function placeResponse(e:GetWeatherByPlaceNameResultEvent):void
{
  Alert.show("looks like we got a good result by place name",
     "s'all good");
  longitude = e.result.Longitude;
  sPlaceName = e.result.PlaceName;
  arrDetails = e.result.Details;
}
```

读者大概已经注意到，代码清单 14-22 中的两个响应处理函数极为相似。会针对 Web 服务系统中的每个方法生成两个不同的事件。Import Web Service 向导也会自动为每一个响应分别生成文件。但是，由于响应结构相同，本质上会生成两个相同文件。乍一看向导生成的结果，恐怕会认为应该分别处理两个响应，但事实却不是这样。事实上，可以像代码清单 14-23 所示的那样将代码合并到一起。

代码清单14-23　以合并方式处理响应

```
public function zipResponse(e:Event):void
{
  Alert.show("looks like we got a good result by zipcode","s'all good");
  Var myEvent = e as GetWeatherByZipCodeResultEvent;
  longitude = myEvent.result.Longitude;
  sPlaceName = myEvent.result.PlaceName;
  arrDetails = myEvent.result.Details;
}
```

从维护代码的角度看，合并后的代码具有更少的函数，显然维护起来更轻松。

14.8　小结

Adobe Flex 内置了多种与 Web 服务交互的方法。从 HTTPService 组件到 RemoteObject 和 WebService 组件，Flex 为我们提供了与更广泛的 Web 服务交互的必要工具。而每种不同的组件都有各自独特的功能和用途。

理解 WSDL 文档有助于迅速从中找到给定 Web 服务支持的方法和调用以及请求返回的数据类型。尤其是要注意<wsdl:portType>和<wsdl:operation>标签。这两个标签是用于确定针对 Web 服务可以调用哪些方法。

大体而言，每个 Web 服务组件的工作原理都很相似。首先，通过传入 Web 服务的位置(或 URL)初始化组件；然后设置变量并调用操作名或 send()函数，以执行命令。最后等待 ResultEvent(或者，如果事情没有按计划顺利完成，就是 FaultEvent)以确定是否成功返回响应。如果成功获得响应数据，就可以在应用程序的组件中使用这些数据。

除了上述内置组件外，Flex Builder 还提供了 Import Web Service 向导。使用这个向导可以自动把返回结果转换成创建的组件，从而确保能够操作强类型对象。然后使用调用 Web 服务的结果，来填充这些自定义类。

在掌握了为应用程序获得数据的方法后，我们将继续沿着数据这条主线前进。第 15 章将探讨 XML 及如何在 Flex 应用程序中分析 XML 树。

第 15 章 使用 XML

15

本章内容：

- XML 基础知识
- XML 相关组件和 ActionScript 类
- 绑定到 XML 数据
- 使用 E4X 操作 XML

XML 取得的成功已经达到能够与 HTML 相媲美的程度。不过，HTML 主要是在基于互联网的应用程序的客户端使用，而 XML 已经几乎渗透到计算的每个方面。

既然 MXML 基于 XML，Flex 理所当然也支持 XML。本章，我们介绍 Flex 对 XML 的强大支持，从将 XML 作为源数据使用，到如何通过 Flex 直接操作 XML 代码。

15.1 XML 基础知识

如果读者熟悉 XML，可以跳到下一节；否则，还是有必要花点时间继续看下去，以便掌握足够的基础知识。不过，读者也不用担心得学习多少 XML 知识；其实，你大可不必成为精通 XML 的专家，只要理解 XML 的规则和结构，就有助于筛选和操作 XML 数据。

15.1.1 什么是 XML

XML 是 Extensible Markup Language(可扩展标记语言)的简写。随着它的日益普及，关于 XML 的严格定义也出现了分歧。XML，一个最广泛接受的定义，作为一门语言，它是一种描述信息的灵活机制。但很多人认为 XML 是一种平台中立的数据结构。从技术上讲，上述理解都没有问题。

从功能上讲，XML 其实是上述两者的混合物——XML 描述信息的能力源自它中立的数据结构语法，但还不止于此。因为 XML 用于描述信息，它也被称为元语言，且更像是一种标准(而非一门语言)。

抛开中立的数据结构与数据描述机制的争论不谈，任何符合 XML 语法和规则的标记语言都可以被看成是 XML，它们与 XML 自动兼容。例如，RSS(Really Simple Syndication，

真正简单的聚合)博客和新闻源，或 BPEL(Business Process Execution Language，商业流程执行语言)，都是使用 XML 定义的标记语言。

与 HTML 不同，XML 没有预定义的标签。相反，XML 允许创建者定义自己的标签并描述标签间的关系，这就是 Extensible Markup Language 中的 *Extensible*(可扩展性)。而定义好的标签可以由其他应用程序使用或处理。例如，Web 服务使用 XML 来描述支持函数，以及这些函数接收和返回的数据，其中也包括复杂数据类型的定义。

15.1.2 XML 的优点

前面提到过 XML 的一个优点是它的中立性。所谓中立，即基于 XML 的数据可以在系统和系统之间共享，无论是什么硬件、编程语言或操作系统。

“系统到系统”(system to system)这个概念的含义非常广泛；它可以应用于在笔记本电脑上运行的应用程序(在 API 之间传输 XML)；也可以广泛地应用于互联网上，例如 Web 服务；但最终结果都是鼓励重用和便于协同工作。

由于 XML 本身是结构化的，因此它很适合表现具有树形结构的数据。大多数语言都没有表现树形结构的本地数据类型。在这些语言中，为了表现层次化的数据，我们通常要不厌其烦地通过结构创建链表。而 XML 是基于文本的，因此可以使用它原封不动地保存树形结构的数据。

XML 还有下列优点。

- 简化数据共享
- 简化数据传输
- 增加数据可用性
- 支持创建自定义语言

不知何故，很少有人提及 XML 的缺点。但正如本书一直强调的，任何事物都有两面性，XML 当然也不例外。因此，我们先承认 XML 有缺点，接着再看一看它有哪些缺点。

15.1.3 XML 的缺点

有时候，太通用或者太中立了也会带来问题。XML 就是这样一种通用型语言，但它不会针对任何应用做出优化。

具体来说，XML 本身往往十分冗长。因此，在传输大规模数据的情况下，XML 的效率极低。尽管使用压缩技术可以加速数据传输，但许多编程语言即使在处理全都是文本的 XML 数据时，也会产生较大开销。毕竟，XML 不是大多数语言中的原生数据类型。

所幸的是，Flex 可以通过 E4X 将大规模 XML 数据作为本地数据类型来处理。本章后面将对此展开讨论。

15.1.4 XML 语法及规则

为了保证互操作性，XML 定义了一组严格的规则。好在这组规则并不多，因而遵守起来也不困难。

1. 根标签

所有 XML 结构都必须包含在一个主根标签中，这个标签称为*文档元素*。

```
<root>
  <node/>
  <node/>
</root>
```

在这个例子中，根(root)标签就是顶级标签；其他标签都不能与它处在同一个层次上。

2. 格式正确

以下是必须严格遵守的一些语法上的基本规则：所有标签必须使用左尖括号(<)和右尖括号(>)括起来。每个标签都必须有一个对应的结束标签，结束标签的标签名前面包含一个正斜杠(/)。HTML 在这方面比较宽松，但 XML 不是。必须正确地嵌套标签——被嵌套的标签组必须在更高级标签结束前结束。

请看以下示例，这是嵌套正确的代码。

```
<root>
  <people>
    <person>…</person>
  </people>
</root>
```

这是嵌套不正确的代码。

```
<root>
  <people>
    <person>
  </people>
  </person>
</root>
```

在第二个示例中，即使删除结束标签</person>，代码还是无效。因为开始标签<person>依然存在。

3. XML 声明

XML 文件的第一行可以是一条 XML 声明语句(可选)。除了指定数据内容是 XML 格式外，这条语句还标识当前使用的 XML 版本以及采用的字符编码类型(字符集)。

在 Flex 中，每个 MXML 文档的第一行都是这样的一条声明。

```
<?xml version="1.0" encoding="utf-8"?>
```

下面我们看一看什么是 XML 实体。

4. XML 实体

实体(entity)是一个技术术语，指的是某个字符的简写或等效形式。所有 XML 文档都默认包含的实体是那些在 XML 文档中具有特定含义的替换字符。表 15-1 所示为这些替换字符。

表 15-1 XML 实体及等效字符

字 符	说 明	相应的 XML 实体
&	和号	&
<	小于号(左尖括号)	<
>	大于号(右尖括号)	>
'	单引号	'
"	双引号	"

读者可能熟悉这些实体，它们与 HTML 中的实体是一样的。

5. 存储数据

在 XML 中，可以通过两种方式存储数据。把数据放在开始和结束标签之间，或者把数据作为标签的属性。在使用属性时，必须使用匹配的引号(单引号或双引号)将值引起来。

下面是把数据放在标签中。

```
<person>Tariq Ahmed</person>
```

下面则是把数据放在属性中。

```
<person name="Tariq Ahmed"/>
```

这两种方式各有各的优点，但属性方式涉及的文本较少一些。在传递数千条记录时，这种优势会成为决定性因素(毕竟，减少必须传输的数据量是我们始终如一的目标)。

根据前面使用 MXML 的经验，我们知道某些字段对于 XML 解析器具有特殊含义。如表 15-1 所示，和号(&)和尖括号(<和>)如果出现在数据中会导致问题。在 XML 中，和号用来引用实体字符，而尖括号是 XML 标签的组成部分。如果数据中包含尖括号，XML 解析器将把尖括号看成程序代码，而不是数据。

在这些情况下，需要使用 CDATA 块封装数据。CDATA 告诉解析器忽略包含在 CDATA 块中的语法性字符，从而允许它们随同其他数据一起传输。代码清单 15-1 说明在节点中包含 XML 语法性字符(即其中的-->)为什么会导致问题。

代码清单15-1 会导致错误的脚本

```
<root>
  <people>
    <person>
      John & Mary Smith Email -->jmsmith@jmsith.com
    </person>
  </people>
</root>
```

这个问题在使用 CDATA 块的情况下很容易解决，如代码清单 15-2 所示。

代码清单15-2 使用CDATA块解决问题

```
<root>
 <people>
  <person>
   <![CDATA[
    John & Mary Smith --> jmsmith@jmsith.com
   ]]>
  </person>
 </people>
</root>
```

另外，也可以使用等效实体字符来转义其中的右尖括号，如代码清单 15-3 所示。

代码清单15-3 使用字符实体

```
<root>
 <people>
  <person>
   <![CDATA[
    John &amp; Mary Smith --&gt; jmsmith@jmsith.com
   ]]>
  </person>
 </people>
</root>
```

不过，我们还是建议使用 CDATA 块方式。原因是：有些数据可能来自数据库，其中会有很多特殊字符。既然 CDATA 块就可以解决捕获和转换所有特殊字符的问题，又何乐而不为呢？

6. 区分大小写

XML 区分大小写，换句话说，XML 中的大写字母和小写字母分别表示不同含义。因此，对于混合大小写的内容必须多加注意。例如，下面的代码是有效的。

```
<root>
<someData>...</someData>
<SomeData>...</SomeData>
</root>
```

但是，如果要通过解析查找某个命名约定，那么很可能会漏掉另一个。

15.1.5 DTD 与 XML Schema

XML 文档受到一组叫做 DTD(Document Type Definition，文档类型定义)的规则约束。DTD 基于元素和属性描述文档的结构。

可以编写自定义 DTD 并将它提供给其他系统，以便该系统了解相应 XML 文档的结构，同时也可以基于 DTD 验证 XML 数据。代码清单 15-4 所示为一个简单的 DTD。

代码清单15-4 DTD描述XML文档中的元素和属性

```
<?xml version="1.0"?>
<!DOCTYPE root [
  <!ELEMENT root (people)>
  <!ELEMENT people  (person+)>
  <!ELEMENT person  (#PCDATA)>
  <!ATTLIST person age CDATA #REQUIRED>
]>
```

在这个 DTD 中，root 包含一个元素 people。而 people 元素可以包含一个或多个 person 元素。在 person 元素中，应该包含由 XML 解析器解析的数据。此外，这个 DTD 还定义了一个必需的属性 age。

代码清单 15-5 所示为一个符合代码清单 15-4 中 DTD 标准的文档。该文档可以被称为格式良好的 XML 文档。

代码清单15-5 符合代码清单15-4中DTD标准的XML文档

```
<root>
  <people>
    <person age="35">Tariq Ahmed</person>
    <person age="34">Jon Hirschi</person>
 </people>
</root>
```

DTD 从 XML 诞生之日起就成为它的一个组成部分。但是，DTD 正在慢慢被另一种叫做 XSD(XML Schema，XML 模式)的新机制取代。XSD 是一种更稳健的方式，它使用面向对象的手段描述文档结构，能够描述数据类型和文档结构之间的关系。代码清单 15-6 所示为一个例子。

代码清单15-6 person.xsd：使用XML Schema定义文档结构

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="person" type="Person"/>
      <xs:complexType name="Person">
        <xs:sequence>
           <xs:element name="firstname " type="xs:string"/>
           <xs:element name="lastname" type="xs:string "/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

要使用这个 XML Schema，先将它保存为.xsd 文件，然后在 XML 文档中引用该文件，如代码清单 15-7 所示。

代码清单15-7 将XML Schema投入使用

```
<person
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="person.xsd">
    <firstname>Tariq</firstname>
    <lastname>Ahmed</lastname >
</person>
```

15.1.6 名称空间

如果有多个系统分别维护了各自对 XML 文档的定义集合，而每个系统都以不同方式描述相同的元素和属性，该怎么办呢？这时候就要用到名称空间了。名称空间是使用名为 xmlns 的属性针对给定上下文指定的一个前缀。而通过这个前缀可以区分不同上下文中的相同元素。代码清单 15-8 所示为使用名称空间避免元素冲突的例子。

代码清单15-8 使用名称空间添加上下文

```
<root
xmlns:cflex="http://www.cflex.net " xmlns:dopejam="http://www.dopejam.com">
<cflex:people>
  <cflex:person username="tahmed">
  <cflex:person username="jhirschi">
</cflex:people>
<dopejam:people>
  <dopejam:person>
    <dopejam:firstname>Faisal</dopejam:firstname>
    <dopejam:lastname>Abid</dopejam:lastname>
  </dopejam:person>
</dopejam:people>
</root>
```

从这个示例中可以看到每个结构都使用<people>和<person>标签。虽然这两个标签都有不同的表现形式，但解析起来不会有任何问题，因为它们通过各自的名称空间标识了具体数据的定义。

15.2 XML 组件和类

Flex 提供了很多可以用来存储 XML 数据的内置组件和类。本节就来介绍这些组件和类，同时介绍它们在工作流中的应用。

首先，看一看 Flex 都提供了哪些 XML 相关组件和 ActionScript 类，如表 15-2 所示。

表 15-2 XML 相关组件和类

MXML 组件	ActionScript 类	说 明
<mx:XML>	XML	以单节点形式存储标准 XML 数据
<mx:XMLList>	XMLList	以多节点形式存储 XML 数据
<mx:XMLListCollection>	XMLListCollection	具有强大集合特性的 XMLList。例如，XMLListCollection 在其状态变化时可以分派通知事件。而其他组件可以侦听这个事件并做出响应

第 11 章曾大致提到过这些组件和类，当时是使用它们为导航组件提供数据。现在，我们要深入介绍它们的函数和属性。

15.2.1 XML 组件

XML 组件是操作 XML 数据的基本机制。通过 XML 组件可以将 XML 数据嵌入到应

用程序代码中，也可以在代码中引用外部 XML 源文件。但由于 XML 不是集合组件，因此它不具备高级的数据同步和自动感知数据变化能力。

调用 XML 组件

要使用 XML 组件，可以显式地嵌入 XML 数据或从外部文件中引入 XML 数据。代码清单 15-9 所示为一个嵌入 XML 数据的例子。

代码清单15-9 使用XML组件嵌入XML数据

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:XML id="usersXML">
  <users>
    <user id="1">Tariq Ahmed</user>
    <user id="2">Jon Hirschi</user>
  </users>
</mx:XML>
</mx:Application>
```

这种方式经常用于以嵌入方式在代码中保存应用程序的配置项，或者以变量形式保存结构化的数据。不过，引用外部数据或配置则是更常用的方法。要在应用程序中引入一个外部源文件，需要使用 XML 的 source 属性(见代码清单 15-10)。

代码清单15-10 使用source属性引用一个外部文件

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:XML id="usersXML" source="users.xml"/>
</mx:Application>
```

source属性的值可以是相对或绝对路径

必须注意的是，Flex 会在编译时纳入此 XML 文件。也就是说，除非再次执行编译，否则对 XML 文件的任何修改都不会在应用程序中反映出来。

15.2.2 XML 类

XML 类与 XML 组件相似，只不过可以通过 ActionScript 来创建它的实例。它们的主要区别是不能通过 XML 类在编译时嵌入外部 XML 文件。

1. 调用 XML 类

如代码清单 15-11 所示，在 ActionScript 中使用 XML 类的方式与使用 XML 组件很相似。

代码清单15-11 通过XML类嵌入XML数据

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
  <![CDATA[
    public var usersXML:XML=
      <users>
        <user id="1">Tariq Ahmed</user>
        <user id="2">Jon Hirschi</user>
      </users>
  ]]>
</mx:Script>
</mx:Application>
```

虽然这和使用 XML 组件没有太大区别，但我们可以把 ActionScript 代码保存为一个单独的文件，然后再在 MXML 文件中包含该文件(而这是使用 MXML 方式做不到的)。

例如，可以把 ActionScript 代码提取出来并保存为 loadUsersVariable.as 文件。然后再通过下面的代码将它引入到 MXML 文件中。

```
<mx:Script source="loadUsersVariable.as"/>
```

这种 ActionScript 与 MXML 代码分离的好处是，可以在多个文件中共享 ActionScript 代码，因而能够提高代码的重用率。

2. 把对象转换为 XML

还有一种技术，可以利用 XML 类的初始化函数(即构造函数)把包含 XML 的对象转换为真正的 XML 对象。例如：

```
public var simpleString:String = "<users><user>Tariq Ahmed</user></users>";
public var usersXML:XML = new XML(simpleString);
```

public var usersXML:XML = new XML(simpleString);

由于 XML 类没有 source 属性(不像与它对应的组件)，因此不能通过它加载外部文件。不过，如果在编译时就知道 XML 文件的内容，则大可使用 XML 组件。否则，如果想动态加载外部文件，可以将这个类与专门用于获取数据的功能结合使用，例如 HTTPService(参见第 14 章了解更多有关获取远程信息的内容)或者另一种机制 URLLoader，后者将在 15.3 节再做介绍。

3. XML 类的属性

XML 类有很多静态属性(见表 15-3)。这些属性保存有确定的、全局性的设置，不会随实例不同而改变。如果修改了其中某个属性，那么修改内容会反映到 XML 对象的所有实例。

表 15-3　XML 类的全局(静态)属性

属　性	类　型	说　明
ignoreComments	Boolean	指定是否过滤掉 XML 源文件中包含的注释。有读者可能认为这个选项可有可无；实际上，有时候就需要纯 XML 以便显示用户(或生成日志)，有时候还可能会处理 XML 中的注释。默认值为 true
ignoreProcessing-Instructions	Boolean	指定是否忽略 XML 中包含的处理指令(处理指令的格式为 <?PROC_INSTR fooBar?>)。设置为 false 表示允许访问相应信息，设置为 true 表示过滤掉处理指令。默认值为 true
ignoreWhiteSpace	Boolean	指定是否保留或删除文本首尾的空白字符。保留空白可能会便于显示。默认值为 true(删除空白字符)
prettyIndent	int (integer)	指定不同缩进级别所需的空格数(在 prettyPrinting 属性值为 true 时有效)。默认值为 2
prettyPrinting	Boolean	指定在通过 toString()和 toXMLString()函数输出时，是否以结构化的形式显示 XML 内容。默认值为 true

代码清单 15-12 显示这些属性的工作原理，它切换静态 prettyPrinting 属性。

代码清单15-12 使用XML类的静态属性

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
  <![CDATA[
   public var usersXML:XML=
     <users>
       <user id="1">Tariq Ahmed</user>
       <user id="2">Jon Hirschi</user>
     </users>

   public function togglePretty():void
    {
       XML.prettyPrinting = !XML.prettyPrinting;
       myTextArea.text = usersXML.toString();
    }
  ]]>
  </mx:Script>

  <mx:TextArea id="myTextArea" width="200" height="100"/>
  <mx:Button label="Toggle Pretty Printing" click="togglePretty()"/>
</mx:Application>
```

静态属性不需要特定实例

代码清单 15-12 的结果如图 15-1 和图 15-2 所示。其中，XML 输出结果分别是非结构化和结构化形式。

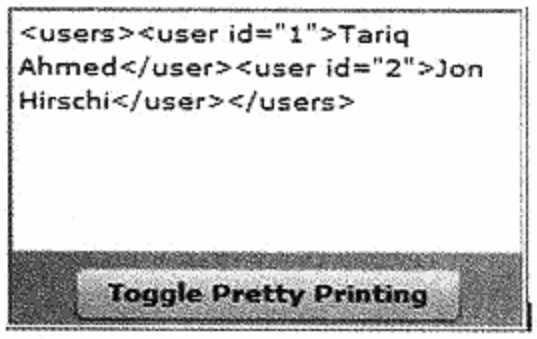


图 15-1 在 XML. prettyPrinting 值为 false 的情况下，显示非结构化内容

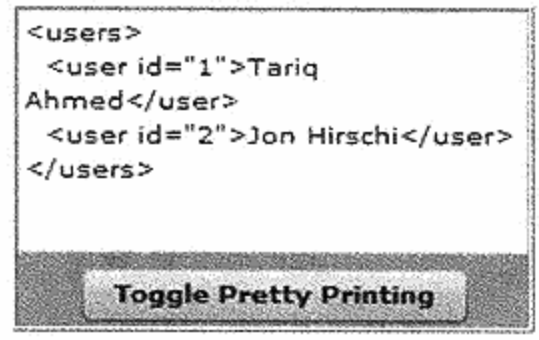


图 15-2 在 XML. prettyPrinting 值为 true 的情况下，显示结构化内容

同样，试验其他属性也将看到显示结果的相应变化。但最重要的是，要知道设置这些属性可以指定 XML 在整个应用程序范围内的呈现方式。

4. XML 类的函数/方法

你可以自定义很多函数。根据用途可以将它们分类如下。

- 获得及操作结构的函数
- 获得及操作属性的函数
- 操作内容及显示的函数

表 15-4 至表 15-6 按照上面的分类列出了所有与 XML 相关的函数。

表 15-4　与 XML 结构相关的函数

函　　数	说　　明	返回值
appendChild	在给定节点的最后一个子节点之前添加节点	XML
child	获得给定节点所有子节点的列表	XMLList
childIndex	获得某个子节点的位置	int
children	从 XML 对象中获得所有子节点	XMLList
comments	从 XML 对象中获得注释	XMLList
descendants	获得与给定名称匹配的节点的所有子节点	XMLList
elements	获得 XML 对象中的元素	XMLList
insertChildAfter	在指定节点后面插入一个子节点	Dynamic
insertChildBefore	在指定节点前面插入一个子节点	Dynamic
parent	获得给定节点的父节点	XMLList
prependChild	在给定节点的其他任何子节点之前插入子节点	XML
processingInstructions	从 XML 对象中获得处理指令	XMLList
contains	类似于搜索函数，用于确定 XML 中是否包含一个匹配项	Boolean
copy	创建现有 XML 对象的副本	XML
length	确定 XML 对象的大小。在循环遍历 XML 内容时，用于计算需要执行多少次迭代	int

表 15-5　与 XML 属性相关的函数

函　　数	说　　明	返　回　值
attribute	获得指定的属性值	XMLList
attributes	获得所有属性值	XMLList
hasOwnProperty	确定某个属性是否存在	Boolean
propertyIsEnumerable	确定指定属性可否可以枚举	Boolean
replace	替换属性的值	XML
setChildren	替换子节点属性的值	XML

表 15-6　与内容相关的 XML 函数

函　　数	说　　明	返　回　值
hasComplexContent	确定 XML 结构是否包含一层以上的内容	Boolean
hasSimpleContent	确定 XML 结构是否只包含一层内容	Boolean
nodeKind	返回节点类型。通常在迭代遍历 XML 结构时使用。可能的值有：text、comment、processing-instruction、attribute 和 element	String

(续表)

函　数	说　明	返 回 值
Text	只获得 XML 文档中的文本；注释、处理指令及属性全部过滤掉。即只返回元素中的文本	XMLList
defaultSettings	在重写静态 XML 属性(参见表 15-3)的情况下，展示默认设置下的 XML	Object
setSettings	间接设置其他属性，包括：ignoreComments、ignoreProcessingInstructions、ignoreWhitespace、prettyIndent 和 prettyPrinting	void
settings	设置 XML 属性(参见表 15-3)的值	Object
normalize	通过合并邻近节点和空节点清理 XML 文档	XML
toString	尝试以简单字符串形式返回 XML 内容。如果 XML 内容太复杂，则原样返回(包括标签及所有内容)。	String
toXMLString	原样返回 XML 内容	String

这些函数确实够多的！不过，由此我们也可以感受到 Flex 为操作 XML 提供了多么丰富的工具。稍后我们会看一些使用这些函数的例子，在此之前，我们需要先了解另外一些内容。

15.2.3 XMLList 组件

XMLList 组件/类就像是一个辅助对象，因为它只能处理 XML 的子集。实际上，XMLList 是用来保存 XML 片段的，类似于在数据库中基于给定条件筛选出的子记录集视图。

所以，XMLList 可以不受顶级根节点限制。但是，XMLList 不支持引用外部文件。

使用 XMLList 组件

使用 XMLList 组件与使用 XML 组件非常相似；不过，如代码清单 15-13 所示，可以在 XMLList 中忽略根节点。

代码清单15-13　创建XMLList组件，以便为List组件提供数据

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:XMLList id="myXMLList">
      <user id="1">Tariq Ahmed</user>
      <user id="2">Jon Hirschi</user>
   </mx:XMLList>
   <mx:List dataProvider="{myXMLList}"/>
</mx:Application>
```

编译并运行代码清单 15-13，将看到与图 15-3 所示类似的结果。

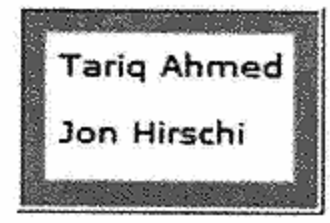


图 15-3　List 组件的数据来自于 XMLList 组件

15.2.4 XMLList 类

XMLList 类是非常实用的一个类。使用 XMLList 类时，甚至不需要创建它的新实例；而是可以将其中一个 XML 类的函数的返回结果直接作为 XMLList 使用。

下面来看代码清单 15-14，其中使用了这两个概念。

代码清单15-14 使用XMLList

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
     <![CDATA[
    public function displayIt():void
    {
     var admins:XMLList = myXML.children();
     myTextArea.text = admins.toXMLString();
    }
     ]]>
  </mx:Script>
  <mx:XML id="myXML">
    <users>
      <administrators>
        <user id="1">Tariq Ahmed</user>
        <user id="2">Jon Hirschi</user>
      </administrators>
    </users>
  </mx:XML>
  <mx:TextArea id="myTextArea" width="200" height="100"/>
  <mx:Button click="displayIt()" label="Try It"/>
</mx:Application>
```

单击 Try It 按钮，将看到与图 15-4 所示类似的结果。

图 15-4 直接将执行 XML children 函数的结果作为 XMLList 使用

如图 15-4 所示，显示结果并没有包含根节点，因为它们全都是根节点的子节点。假如想让显示结果再深入一层，可以将代码清单 15-14 中的相关代码修改为：

```
var admins:XMLList =
  myXML.administrators.children();
```

这样修改后，结果将只显示两个用户(user)节点。

15.2.5 XMLListCollection 组件和类

XMLListCollection 是一个包装对象，用于包装已有的 XMLList 对象。通过包装，可以赋予 XML 集合操作的特性。而集合特性在显示数据驱动的组件时非常方便。特别是，当底层数据发生变动时，显示组件会收到变动通知，并做相应更新。

1. 包装 XMLList——直截了当

如代码清单 15-15 所示，把 XMLList 转换为 XMLListCollection 的最简单方式，就是直截了当地用 XMLListCollection 来包装 XMLList。

代码清单15-15 使用XMLListCollection来包装XMLList

```
<mx:XMLListCollection id="myXLC"  >
<mx:XMLList id="myXML">
  <users>
    <administrators>

      <user id="1">Tariq Ahmed</user>
      <user id="2">Jon Hirschi</user>
    </administrators>
  </users>
</mx:XMLList>
</mx:XMLListCollection>
```

读者如果觉得这种技术太过直白，也可以采取另一种方式，即使用 XMLListCollection 的 source 属性。

2. 通过 source 属性指向 XML 和 XMLList 来源

请读者不要与 XML 类的 source 属性混淆，这里的 source 属性用于指向当前的 XMLList 或 XML 对象。这种方式的好处是，通过降低依赖性提高了灵活性，因为我们可以随时改变作为目标的 XML 源。

如代码清单 15-16 所示，利用这种方式也可以过滤掉特定的节点。

代码清单15-16 通过source属性来过滤和绑定XML数据

```
<mx:XMLList id="myXML">
  <users>
    <administrators>
      <user id="1">Tariq Ahmed</user>
      <user id="2">Jon Hirschi</user>
    </administrators>
  </users>
</mx:XMLList>
<mx:XMLListCollection id="myXLC" source="{myXML.administrators.user}"/>
```

当然，如果 myXML 是一个<mx:XML>对象(而非<mx:XMList>)，这种技术也同样有效。

15.3 加载外部 XML 源数据

前面曾经提到过，XML 类没有可以引用和加载外部 XML 源数据的 source 属性。如果我们想根据需要加载 XML 数据，可以通过以不同方式使用数据服务(参见第 14 章)来实现。但是，如代码清单 15-17 所示，另一种比较简单的技术是使用 URLLoader 类。

代码清单15-17　使用URLLoader来引用外部XML源文件

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >
<mx:Script>
 <![CDATA[
 import flash.events.Event;
 import flash.net.URLLoader;
 import flash.net.URLRequest;

 public function loadXML():void
 {
  var xmlLoader:URLLoader  = new URLLoader();

  xmlLoader.load(new URLRequest("users.xml"));

  xmlLoader.addEventListener(Event.COMPLETE,
     handleLoad);
 }

 public function handleLoad(event:Event):void
 {
  var usersXML:XML = new XML(event.target.data);

  myTextArea.text = usersXML.toXMLString();
 }

 ]]>
</mx:Script>
<mx:TextArea id="myTextArea" width="250" height="100"/>
<mx:Button click="loadXML()" label="Load in XML"/>
</mx:Application>
```

初始化请求

创建URLLoader实例

指定要使用的URL

指定URLLoader结束时调用的函数

当URLLoader结束时调用

把数据转换为XML对象

显示结果

编译并运行代码清单 15-17，将看到与图 15-5 所示类似的结果。

Flex 会将 URL 作为事件发送。而使用 addEventListener 函数可以告诉应用程序在结果返回时调用适当的函数。

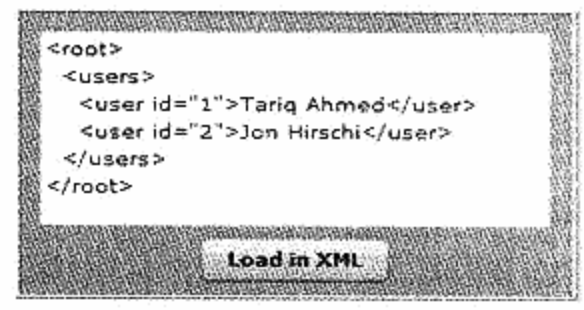


图15-5　通过URLLoader加载了外部XML数据

15.4　XML 中的绑定

尽管对最终用户来说，绑定是透明的，但采取这种技术往往能节省可观的时间。而且由于绑定与 XML 的关系尤为特殊，因此这种技术也能够极大地简化编程工作。

15.4.1　绑定自 XML

如代码清单 15-18 所示，可以将 XML 绑定到其他某些目标。

代码清单15-18　绑定XML数据以驱动显示

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
 <mx:XML source="users.xml" id="usersXML"/>
 <mx:List id="myList" dataProvider="{usersXML.users.user}"/>
</mx:Application>
```

以上绑定可以取得位于 XML 标签<user>...</user>中的文本。如果想取得某个标签(对于

本例而言，是<user>...</user>)的属性，可以使用@字符。例如，要取得 id 属性，可以使用如下代码：

```
<user id="1">Tariq Ahmed</user>
```

可以在代码清单 15-18 中将原有代码修改为如下所示：

```
<mx:List id="myList" dataProvider="{usersXML.users.user}"
   labelField="@id"/>
```

如果不使用绑定，恐怕就得创建循环遍历数据的函数，然后再根据显示组件的需要构建适当的数据结构以便输出值。

15.4.2 绑定到 XML

到目前为止，绑定数据源已经不是什么神秘的技术了。但是，从外部数据源绑定到 XML 呢？(还没有介绍过，但确实很棒)。下面我们通过代码清单 15-19 展示这一技术。

代码清单15-19 通过绑定外部数据源填充XML

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white">
  <mx:Script>
    [Bindable]
    public var userid:int=0;
  </mx:Script>
  <mx:XML id="usersXML">
    <root>
      <users>
        <user userid="{userid}">{myTextInput.text}</user>
      </users>
    </root>
  </mx:XML>
  <mx:Label text="Name:"/>
  <mx:TextInput id="myTextInput"
            change="myTextArea.text=usersXML.toXMLString()"/>
  <mx:TextArea id="myTextArea" width="250" height="100"/>
</mx:Application>
```

绑定到XML中的变量

绑定文本输入框中动态输入的值

在输入发生变化时更新TextArea

图 15-6 所示为当用户在 Name 文本输入框中输入值时，通过绑定将该值动态复制到 XML 文档<user>标签中的过程。

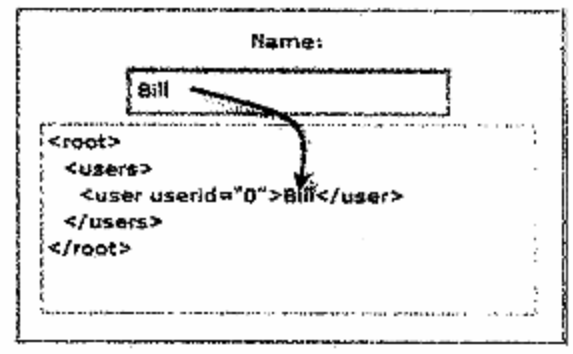


图15-6 通过绑定将文本输入框中的值复制到XML中

代码清单 15-19 配置了一个简单的可绑定(Bindable)变量，用户会在文本输入框中输入值。而这两个数据来源的值都被绑定到 XML 中。

在实际开发中，使用这种技术可以节省构建 XML 文档的时间。例如，可以动态构建一个小型 XML 对象，用于映射表单的值；随着用户不断输入表单各项的值，相应的 XML 对象被同步添加到主 XML 对象中。

15.5 ECMAScript for XML(E4X)

Flex 中的 ActionScript 语言遵循了 ECMAScript 标准的约定、结构和规则。ECMA-357 是对这个标准的扩展，即我们所说的 ECMAScript for XML，又简称为 E4X。

注意：

ECMA 指的是 European Computer Manufacturers Association(欧洲计算机制造商协会)。ECMAScript，也就是 ECMA-262 规范，还得到 JavaScript、Jscript 及 DMDScript 的共同遵守。

遵从标准自然有其优势，因为有关 E4X 的知识可以在不同语言中得以共享。但在 ActionScript 中，Flex 将 E4X 和 XML 都视为本地数据类型，这就为我们带来两个重要的好处。

- **速度**：如果不是本地数据类型，那么 Flex 就必须动用很多资源，将数据转换为可用的本地数据类型。
- **简化访问及操作信息的语法**：可以使用与操作其他 Flex 对象时相同的语法。

在不知不觉中，我们已经在 Flex 中使用 E4X 了。本书所有涉及 XML 的操作都是在 Flex 对 E4X 支持的基础上实现的——从语法到 XML 和 XMLList 对象。

15.5.1 E4X 语法

理解 XML 的一个简便方法，就是把它包含的每一个层次都看成一个对象数组。对于 ColdFusion 开发人员来说，就像是嵌套的结构数组。

学习语法的最简单方式就是看实例。代码清单 15-20 所示为示例 XML 文档，后续示例将基于这个文档展开。

代码清单15-20　用于说明E4X语法的XML文档

```
<mx:XML id="usersXML">
  <root>
    <users>
      <user id="1" lovesDonuts="Yes">
        <firstname>Tariq</firstname>
        <lastname>Ahmed</lastname>
      </user>
      <user id="2" lovesDonuts="Yes">
        <firstname>Jon</firstname>
        <lastname>Hirschi</lastname>
      </user>
    </users>
  </root>
</mx:XML>
```

1. 示例 1：根据位置(索引)访问特定的值

第一个示例展示如何提取第二个用户的名字(Jon)。

```
usersXML.users.user[1].firstname
```

在知道元素位置的情况下，这种方法很方便。可以使用这种技术循环遍历每个用户(或其他什么节点)，通过索引计数器取得所需的值。为了方便，也可以使用绑定语法(可以绑定任何 E4X 表达式)。例如：

```
<mx:TextArea text="{usersXML.users.user[1].firstname}"/>
```

下面再看另外一个示例。

2. 示例 2：根据属性值访问特定的节点

与示例 1 类似，如果不知道节点的位置，但是知道它的属性和值，则可以像下面这样访问用户 Jon：

```
usersXML.users.user.(@id==2).firstname
```

因为使用的是属性，所以必须使用@字符来引用属性。

3. 示例 3：根据邻近节点的值访问节点

这个示例假设要提取 Jon 的姓，而姓与名是相邻的。为此，可以循环遍历 XML(这在 E4X 中还算容易)，但现在，我们要做的是根据邻近节点的值找到目标节点。

```
usersXML.users.user.(firstname=='Jon').lastname
```

实际上，也可以访问节点的属性，如下一个示例所示。

4. 示例 4：访问节点的属性

一定要记住，在访问属性值时，始终使用@字符：

```
usersXML.users.user.(firstname=='Jon').@id
```

最后一个示例展示如何赋值。

5. 示例 5：赋值

访问 XML 信息的语法同样支持赋值。下列代码示范了如何赋值：

```
usersXML.users.user.(@id==2).firstname = 'Faisal'
usersXML.users.user.(firstname=='Faisal').lastname = 'Abid'
usersXML.users.user[1].firstname = 'Mike'
usersXML.users.user[1].@id = 3
```

接下来，我们讨论如何操作 XML 结构。

15.5.2 使用 XML 结构

本节将继续对 E4X 的讨论，但关注的焦点将转移到如何迭代或修改 XML 结构。为此，可以利用 XML 函数。

我们还要继续用事实说话的做法，准备好，现在开始了。

1. 示例 1：基于索引循环遍历 XML 结构

由于可以把 XML 看成一个数组，因此迭代遍历数组的方法也适用于 XML。在本示例

中，我们想要遍历所有用户并显示它们的名字，相应代码如代码清单 15-21 所示。

代码清单15-21　基于索引循环遍历XML结构

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white"
            creationComplete="tryme()">
<mx:XML id="usersXML" source="users.xml"/>

<mx:Script>
 <![CDATA[
   [Bindable]
   public var infoString:String = "";
   public function tryme():void
   {
     for(var i:int=0; i<usersXML.users.user.length(); i++)
     {
       infoString = infoString + usersXML.users.user[i].firstname;
       infoString = infoString + usersXML.users.user[i].lastname;
     }
   }
   ]]>
</mx:Script>
 <mx:TextArea text="{infoString}" width="250" height="100"/>
</mx:Application>
```

在Flex创建应用程序时调用相应函数

加载示例XML并创建XML对象

检查项数

注意，通过 length()函数可以确定需要迭代多少项。图 15-7 所示为代码清单 15-21 的运行结果。

图15-7　基于索引遍历XML结构生成的字符串

2. 示例 2：基于键循环遍历节点

这里，我们在示例 1 的基础上稍加修改，使用 for...each 循环来迭代每个节点。修改后的代码如代码清单 15-22 所示。

代码清单15-22　基于键迭代节点

```
public function tryme():void
{
 for each(var node:XML in usersXML.users.user)
 {
   infoString = infoString + node.firstname;
   infoString = infoString + node.lastname;
 }
}
```

在 ColdFusion 中，这种技术类似于使用 StructKeyList()函数循环遍历每个键。

3. 示例 3：循环遍历 XML 片段中的节点

如果我们想把示例 2 中的函数设计得更容易被重用，就需要将函数配置为只理解包含 user 结构的 XML 片段。这样一来，函数就不必了解 XML 的整体情况了。

代码清单 15-23 所示为用于访问 XML 片段的 E4X 语法。

代码清单15-23 循环遍历XML片段

```
var xmlFragment:XMLList = usersXML.users.user;
for each(var node:XML in xmlFragment)
{
  infoString = infoString + node.firstname;
  infoString = infoString + node.lastname;
 }
```

以上代码将 XML 片段赋值给一个 XMLList 类型的变量，而 XMLList 的用途恰好就是保存 XML 片段。

4. 示例 4：取得后代节点

E4X 定义了后代(descendant)操作符，语法为连续的两个点(..)。这个语法是用来取得给定节点所有后代节点的便捷方式(无论后代节点位于何处)。

例如，下面这行代码可以取得所有<firstname>节点。

```
usersXML..user.firstname
```

这行代码返回的结果如下。

```
<firstname>Tariq</firstname>
<firstname>Jon</firstname>
```

下面我们来看如何使用通配符。

5. 示例 5：使用通配符取得节点和属性

如果只想取得某一层次的所有子节点，可以使用 E4X 支持的星号(*)。星号在这里叫做通配符，它的用法如下。

以下代码会提取出完整的 XML 片段，而 user 在这里就像是一个包含用户信息的数组。

```
<mx:TextArea text="{usersXML.users.user}" width="250" height="200"/>
```

这行代码的结果如图 15-8 所示。

然而，在使用通配符的情况下，只会提取出<user>包含的节点。

```
<mx:TextArea
   text="{usersXML.users.user.*}"
   width="250" height="200"/>
```

图 15-9 所示为使用通配符只提取了<firstname>和<lastname>节点。

```
<user id="1" lovesDonuts="Yes">
  <firstname>Tariq</firstname>
  <lastname>Ahmed</lastname>
</user>
<user id="2" lovesDonuts="Yes">
  <firstname>Jon</firstname>
  <lastname>Hirschi</lastname>
</user>
```

图 15-8 不使用通配符取得节点

```
<firstname>Tariq</firstname>
<lastname>Ahmed</lastname>
<firstname>Jon</firstname>
<lastname>Hirschi</lastname>
```

图 15-9 使用通配符取得节点

还可以组合使用@和通配符来取得属性，例如：

```
usersXML.users.user.@*
```

这行代码将提取出 user 节点中所有属性的值。

6. 示例 6：前置和后置子节点

有两个根据作用命名的函数(参见表 15-4)，可以帮助我们在已有的节点前后插入节点。

- prependChild(newChild)：在现有结构的开始处插入一个子节点。
- appendChild(newChild)：在某个结构的结束处插入一个子节点。

代码清单 15-24 通过向示例 usersXML 对象中添加一个新 user 节点，示范了这两个函数的用法。

代码清单15-24　在最后添加一个XML子节点

```
var newUser:XML =
  <user id="3" lovesDonuts="no">
    <firstname>Faisal</firstname>
    <lastname>Abid</lastname>
  </user>
usersXML.users.appendChild(newUser);
```

以上代码会在 user 父节点下方添加一个新的 user 节点。而最终的 XML 结构如代码清单 15-25 所示。

代码清单15-25　以编程方式添加第三个user节点

```
<root>
  <users>
    <user id="1" lovesDonuts="Yes">
      <firstname>Tariq</firstname>
      <lastname>Ahmed</lastname>
    </user>
    <user id="2" lovesDonuts="Yes">
      <firstname>Jon</firstname>
      <lastname>Hirschi</lastname>
    </user>
    <user id="3" lovesDonuts="no">
      <firstname>Faisal</firstname>
      <lastname>Abid</lastname>
    </user>
  </users>
</root>
```

相反，如果代码清单 15-24 中使用的是 prependChild()函数，那么第三个 user 节点将出现在这个列表的前面。

7. 示例 7：在现有节点之间插入节点

插入节点与后置节点类似，只不过要指定在现有节点的什么位置插入新节点。换句话说，通过指定与新节点位置相邻的节点来执行此操作。插入的子节点位于指定的子节点旁边。

表 15-4 中的以下两个函数可以帮助我们插入子节点。

- insertChildAfter(existingChild,newChild)：在现有节点后面插入新子节点。
- insertChildBefore(existingChild,newChild)：在现有节点前面插入新子节点。

代码清单 15-26 在第一个和第二个 user 节点之间插入一个新 user 节点。

代码清单15-26 在现有节点后面插入一个节点

```
var newUser:XML =
  <user id="3" lovesDonuts="no">
    <firstname>Faisal</firstname>
    <lastname>Abid</lastname>
  </user>
usersXML.users.insertChildAfter(usersXML.users.user[0],newUser);
```

类似地，也可以使用 insertChildBefore()函数在第二个 user 节点前面插入该 user 节点。

```
usersXML.users.insertChildBefore(usersXML.users.user[1],newUser);
```

由于原来的 XML 中只有两个 user 节点，因此上面这行代码也将得到与使用 insertChildAfter()的代码清单 15-26 相同的结果。

8. 示例 8：复制节点

同样，根据表 15-4 可知，使用 copy()函数可以复制已有的 XML 结构。与表 15-4 中的任何函数一样，也可以在任何层次上调用 copy()，结果就是从整个 XML 对象的任意位置提取出任何节点结构的副本。

由于 copy()函数只能提取节点结构，因此还必须配合使用其他函数(如 appendChild()、insertChildBefore()等)来添加或插入节点以便实际完成复制。

```
var copiedUser:XML = usersXML.users.user[1].copy();
usersXML.appendChild(copiedUser);
```

第一行代码提取出一段 XML 并将它赋值给一个新 XML 变量。然后，新 XML 变量又被添加到原来的 XML 对象中。

9. 示例 9：删除节点

删除节点很容易，只需使用 Flex 的 delete 关键字来销毁对象。

假设要删除第二个 user(抱歉了，Jon)，使用像下面这样简单的代码即可。

```
delete usersXML.users.user[1];
```

至此，我们就结束了使用 E4X 操作 XML 结构的全部讨论。

15.6 小结

XML 是一种描述和组织数据的方法。前几章曾在示例中大量使用 XML 为很多组件提供数据，而这些组件又将获得的数据显示给用户。本章详细讨论了充分利用 XML 数据的各种方法。

与 XML 文档可能达到的复杂程度及其结构深度相比，操作 XML 的方法反倒简单得有点令人难以置信。操作 XML 的简单主要得益于 E4X 标准，该标准将 XML 视为一个对象数组。另一个原因是，与 Flex 对 E4X 的支持共存的两个辅助类——XML 和 XMLList。

XML 类定义了很多函数，这些函数能帮助我们实现几乎任何目标。函数太多也会有缺点，例如许多函数之间的差别非常小(有些函数甚至连名字都非常相似)；这样一来，一时间就很难确切地搞清楚它们到底有什么区别，或者什么时候该选择使用哪一个。为此，希望读者能够以本章给出的大量示例为原型，基于相同的数据亲自动手试验一下没有演示到的函数。

第 16 章 对象和类

16

本章内容：

- OO 的概念
- 使用 ActionScript 实现 OO 概念
- 创建自定义类
- 包

到目前为止，想必读者已经掌握了有关 ActionScript 的足够知识，包括通过它响应事件、响应用户与应用程序的交互以及操作属性。现在到了提升 ActionScript 技能的时候。

本章我们探讨 Flex 中的一个基础特性——对象和类。同时，还会介绍一些有关 OO(Object Oriented，面向对象)的理论。掌握本章的内容将有助于节省可观的开发时间，并且让你的软件更容易维护。

本章读者将学会如何创建对象和类，调用和销毁它们，以及扩展现有的类。这些内容看起来好像很多，实际上掌握它们不会超过 30 分钟。而且学习完本章，读者也将创建一个真正意义上的 OO 程序。

16.1 用五分钟的时间理解 OO 理论

Flex 是一个基于 OO 的应用程序。因此，理解一些 OO 编程的概念对更好地利用 Flex，以及提升编程技能和更快速地解决问题的能力都非常有帮助。最重要的是，掌握 OO 编程的知识可以为创建可重用代码夯实基础。

在以下几节中，我们介绍使用对象和类编程的基本概念。实践证明，这是开发人员应该掌握的计算机编程方法中一个最重要的分支。理解对象和类也有助于迅速掌握其他编程语言，如 C#、C++和 Java。

16.1.1 对象和类的关系

为了更清楚地理解对象，我们来做一个心理测验。为保证测验成功，读者不能把对象

看作某种抽象的概念，而是要把它当成一个你非常熟悉的东西。

平复一下心情，然后环顾四周。你会注意到周围到处都是对象：汽车、电视、电脑以及书。每种对象都有特定的功能和角色。而且对象还有特征，如它们的颜色和尺寸。

有些对象依赖于其他对象(电脑依赖于 CPU)，而有些对象之间则没有任何关系。例如，你的电脑跟本书就没有什么关系。如果读者理解了以上关于对象的种种说法，那么就已经成功了一半。另一半就是实现。

对象是一种自包含的部件。它有自知之明，知道自己有多大能力。对象会表现出双重性。

- 它们通常只关心自己。(太自私了！)
- 它们可以被开发人员方便地使用，在后台默默地完成艰苦的工作。(太无私了！)

现在我们大致了解了对象的概念，那么什么是类呢？

类的含义

从纯粹的 OO 理论来看，只存在对象。但本书到目前为止经常提到对象和类这两个概念。OO 理论只是一种理论，它不关心实现。Flex 必须把理论付诸实践；因此，与大多数 OO 语言一样，Flex 也采用以类作为模板(或蓝图)创建对象的思路实现这种概念。简而言之，对象就是类的实例。

例如，如果有一个名为 Human(人类)的类，而且该类有一个名叫 Derek Douville 的实例，那么 Derek Douville 就是一个对象。

16.1.2 对象拥有属性和方法

对象了解自身和自己能力的方式有两个。

- 属性
- 方法(有时也被称为函数)

仍然以 Human 类为例，图 16-1 所示为人类应该具备的一些属性。其中一些属性在具体的实例中只会被设置一次，例如 Place of Birth(出生地)；而另外一些属性则经常改变，例如 Mood(心情)。当然，人还能做一些事，比如吃饭、睡觉。

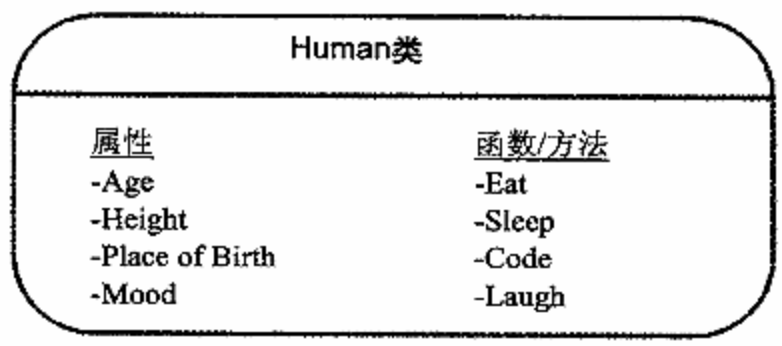


图 16-1 Human 类定义了区别于其他类的属性和能够执行的一些函数

属性和方法有作用域

作用域的目的是为了控制谁可以访问什么。例如，某人要对自己的 SSN(Social Security number，美国社会保障号码)严格保密，以防被公之于众。但在某些情况下，还需要通过某

个方法来使用该号码，例如用于申请信用卡。通过定义作用域，既可以确保 SSN 对外界保密，又可以做到通过类中的方法使用它。

类似地，作用域也适用于方法。人可以拥有一些公共函数(功能)，如工作，以便外部的老板来调用。但也可能有一些私有方法，只能在内部使用，不允许其他人使用(如抠鼻子)。Flex 支持针对属性和方法的如下作用域。

- **私有(private)**：只能在当前类中使用的属性和方法。
- **公共(public)**：任何人都可以使用的属性和方法。
- **受保护(protected)**：只能在当前类及扩展当前类的类中使用的属性和方法。
- **内部(internal)**：可以被同一个包中的类访问的属性和方法。
- **静态(static)**：直接通过类访问的属性和方法。

如果要创建一个对象，让它能够完成人类所能完成的一切任务，应该怎么做呢？可以复制人类的所有方法和属性，然后添加特有的方法和属性。但是，这样一来如果想对两者都做出相同修改，就会出现问题——需要在两个位置做相同的修改。好在，一种叫做继承的机制可以解决这个问题。

16.1.3 继承

一个对象可以从另一个对象继承属性和特征。通过继承，一个类(也称为基类或父类)的属性和方法可以传递给子对象。而且可以继续为继承父类的子类添加属性和方法。

在 Flex 中，有一个非常有名的自定义组件，名叫 PromptingTextInput，可通过 FlexLib 包获得它(http://code.google.com/p/flexlib/)。这个组件是 TextInput 组件的子组件，它通过扩展 TextInput，不仅支持 TextInput 的本地特性，而且还能够显示一条消息(例如“请输入某种类型的值”)，以提示用户输入内容。

某些 OO 语言(包括 Flex)把这一过程叫做扩展类。打个比方，可以将 Human 类作为基础，通过扩展创建出 Male(男人)和 Female(女人)子类(见图 16-2)。每个子类都有自己独特的属性和函数。更重要的是，它们可以按不同的方式实现的相同函数。

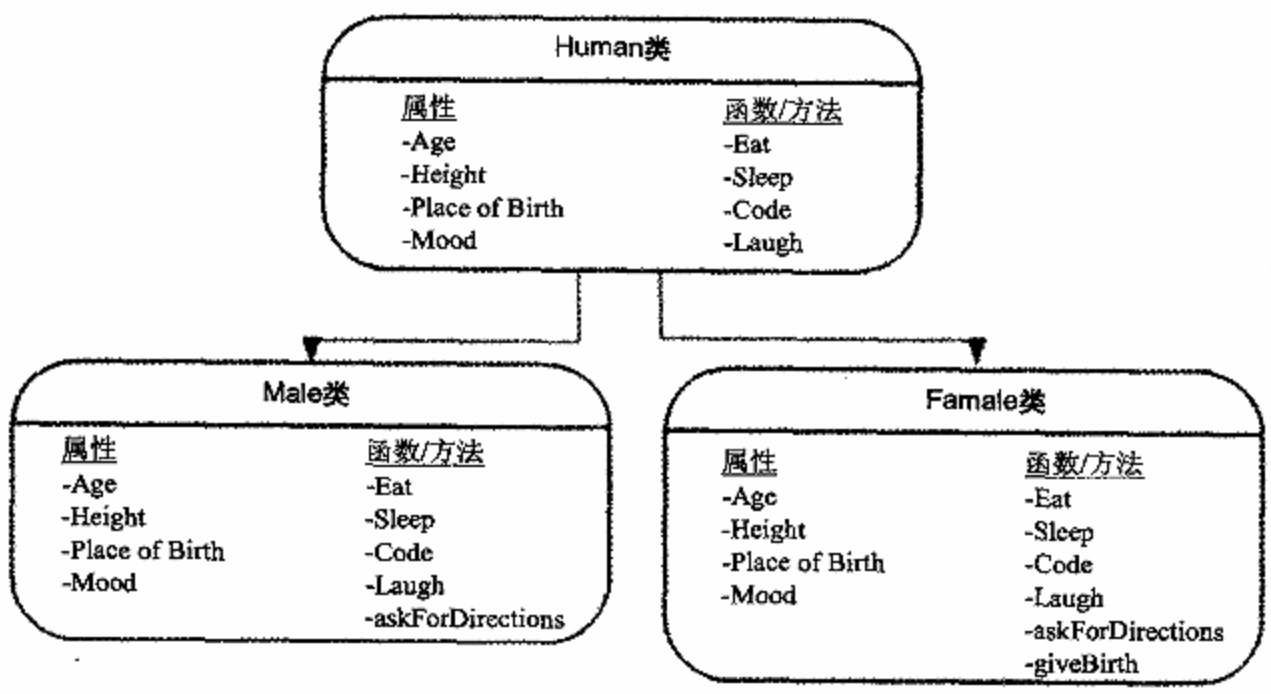


图 16-2 子类可以继承父类的所有属性，也可以添加新属性

例如，虽然 Female 和 Male 都有 askForDirections 函数，但 Female 中的这个函数会返回一系列说明，借以能够成功找到目标位置。而 Male 类的同一个函数则什么也不返回(男人不懂得怎么问路)。

通过继承，可以节省在每个子对象中重复定义属性和方法的时间。在明白了什么是继承之后，接下来我们要介绍怎么保持对象独立——是不是有几分反其道而行之的味道？这个过程叫做封装，它能够促进代码重用。

16.1.4 封装和耦合

封装的关键在于重用代码。封装是指在一个自包含的环境或模块中定义对象。这个对象不需要知道所处的应用程序如何运作。同样，应用程序也不必了解对象的工作过程。

封装做得越到位，代码就越容易重用，因为构成应用程序的各个元素之间都不用关心对方如何执行任务。而对象的首要目标就是在自己的范围内把一切问题搞定。

如果想让其他代码元素使用对象，可以向调用代码公开抽象了内部工作流程的方法和属性。这样，即使修改了对象的内部工作流程，调用代码仍然能够若无其事地正常运转；反之亦然。

举个例子，假设有一个对象，为了执行自己的任务，必须从应用程序中某个地方获得一条信息(如 UserID)。如果这个对象脱离自身而亲自到主应用程序中寻找 UserID，它必须知道 UserID 是什么、有什么特征以及到哪里去找。虽然可以为对象明确提供一条信息(以硬编码方式)，但如果 UserID 发生了变化，导致原先指定给对象的信息失去意义，就必须以手工方式进行更新。

此时，对象必须知道怎样以及到哪里获得所需的值。这样不仅破坏了对象的封装(它不可能做到无视外部世界的存在)，而且导致对象与它所在的应用程序形成了紧密耦合。

解决这个问题的更好的办法，是通过相互传值的方式让对象(及其交互对象)不必知道或关心到哪里去找 UserID。让它只管接收传给它的信息，然后根据指令做出相应处理，从而形成松散耦合的关系。

16.1.5 对象：无处不在

在 ActionScript 3/Flex 3 中，一切皆是对象。从面板到格式化程序，再到用户输入组件，一切都遵循了 OO 编程的概念。

当然，MXML 组件(如<mx:Panel/>)和 ActionScript 类(如 ArrayCollection)都不例外。更重要的是，通过 MXML 创建的一切都将被转换成 ActionScript 类。

严格来讲，说一切都是对象只说对了 99%。从真正意义上来说，原始数据类型，如 int 和 Boolean，就不是对象。但是，除了原始数据类型之外，一切都是对象。

理论讲得够多了，我们还是赶紧转入实战吧！

16.2 操作对象

我们始终都坚持一个原则，即不到用时先不学。本书前几章也经常涉及有关对象的内容，现在则是了解底层机制的时候了。挽起袖子，准备开始吧。

16.2.1 进一步了解对象

首先，通过执行下列步骤建立一个公共起点。

(1) 创建一个名为 ObjectsandClasses 的 Flex 项目。

(2) 打开 ObjectsandClasses.mxml 文件。

(3) 创建<mx:Script>块，以便放置 ActionScript 代码。

完成这些步骤后，将得到如代码清单 16-1 所示的代码。

代码清单16-1 外壳应用程序

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute">
   <mx:Script>
      <![CDATA[

      ]]>
   </mx:Script>
</mx:Application>
```

这说明你已经迈出了宝贵的第一步。接下来，我们向其中添加一个简单的对象。

1. 添加原始对象

在此我们要创建一个对象的实例，借此来了解如何添加对象。准确地说，是原始对象。原始对象是指那些仅支持基本对象特性的对象。

在<mx:Script>块中，输入如图 16-3 所示的代码。

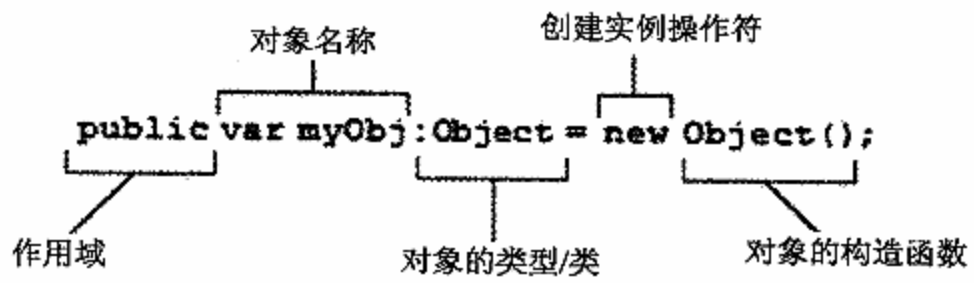


图16-3 创建一个对象的实例

这样就创建了一个名为 myObj 的新对象。变量定义 myObj:Object 之后的 new 操作符，会通知 ActionScript 编译器你要在此创建一个 Object 类的新实例。

既然创建了对象，那么就来揭开它神秘的面纱，看看内部都包含些什么。

2. 剖析对象

要想知道这个对象是由什么构成的，最简单的办法就是另起一行，输入 myObj 并紧接着敲入一个句点。Flex Builder 会立即显示一组关联的属性和方法，如图 16-4 所示。

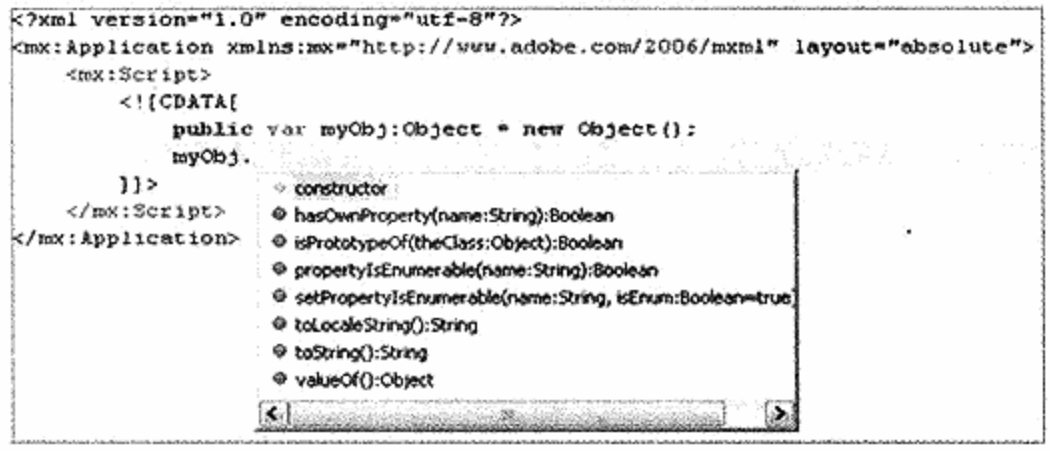


图 16-4 在实例后面输入一个句点会提示 Flex 列出与该实例相关的公共属性和方法

16.2.2 对象的方法

每个 ActionScript 类都包含方法。方法是对象的操作者，它们执行由程序员或开发人员为它们定义的操作。下面我们就以 Array 类为例，看看它包含什么方法。

跟前面创建简单的对象一样，再创建一个名为 myArr 的新数组变量。

```
var myArr:Array = new Array();
```

同样，使用 new 操作符创建一个 Array 类的实例。另起一行，输入 myArr.(别忘了最后的句点)。此时将看到如图 16-5 所示的结果。

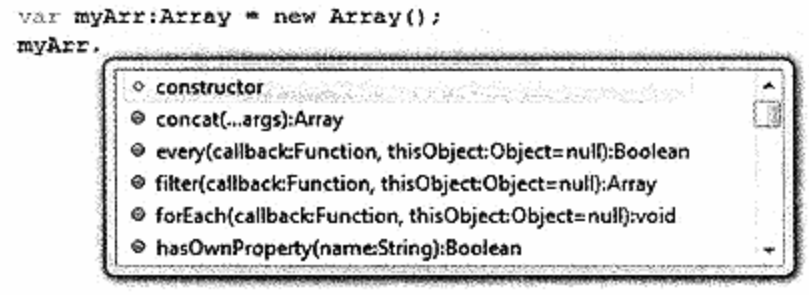


图 16-5 Flex Builder 的自动建议功能给出一些方法

Flex Builder 显示了与 myArr 相关的方法和属性。其中，后跟一对圆括号的是方法，包括 concat()、every()和 forEach()等。

对象的主要使命就是让编程变得更轻松。现在浏览一下这个列表，相信读者能够发现一些有助于达成这一使命的实用方法。

这些都是 Flex 开发团队专门为与 Array 对象交互而编写的方法。例如，数组方法 split()和 push()的用途如下。

- split()：把数组分割成多个较小的部分。
- push()：向数组中添加项。

请记住，对象本身并不关心外界事物。因此，它们的方法也与对象所在环境无关。如果要使用对象的方法，想为它们提供执行任务所需的信息，那么应该传递参数。

16.2.3 方法的参数

可以通过参数(arugment)向方法中传递信息。再看一看 Array 的方法列表，注意圆括号

中的文本，它们的格式是 text:ObjectName。

当我们在圆括号中输入数据时，就是在为方法传递参数。如果想为数组添加一个新值，应该像下面这样提供这个项的值。

```
myArr.push("some Value");
```

在此我们把数据("some Value")传递给了方法，然后方法会按照既定指令处理数据。别忘了，在 OO 概念中，对象做什么、怎么做，都与其他代码无关。对于这个例子而言，"some Value"只是 Array 的参数。

16.2.4 方法返回信息

方法也可以向对象返回信息或数据，这种发送信息的机制叫做返回值。在 Flex Builder 自动建议功能中，我们注意到，除了每个方法使用的参数外，还标明了方法返回值的类型。例如，图 16-5 所示的 concat()方法就会返回另一个数组。

16.2.5 对象的属性

属性是与对象沟通的另一种方式。通过属性既可以配置对象，也可以确定它的特征(当前状态)。以 Array 为例，通过名为 length 的属性可以确定当前数组中包含多少个项。

对象是构造类的基础。下面我们就把前面介绍的与对象有关的知识应用到创建类上。

16.3 创建类

为了学习类，我们要编写一个简单的数学程序，用于计算给定形状的面积。形状也将由创建的类来绘制，同时我们还要定义计算面积的方法。

16.3.1 创建类文件

在这个示例中，请读者按照下列步骤创建一个新项目。

(1) 选择 File | New | Flex Project。

(2) 将新项目命名为 Shapes。

(3) 单击 Finish 按钮。

然后按照下列步骤创建一个 ActionScript 类文件。

(4) 选择 File | New | Create New ActionScript Class。

(5) 根据图 16-6 所示的对话框添加相应字段，但先不要单击 Finish 按钮。下面我们还要详细介绍这个 New ActionScript Class 向导。

至此，我们创建了一个项目和一个简单的类。但是，我们还要解释一下在图 16-6 所示对话框中输入的内容。

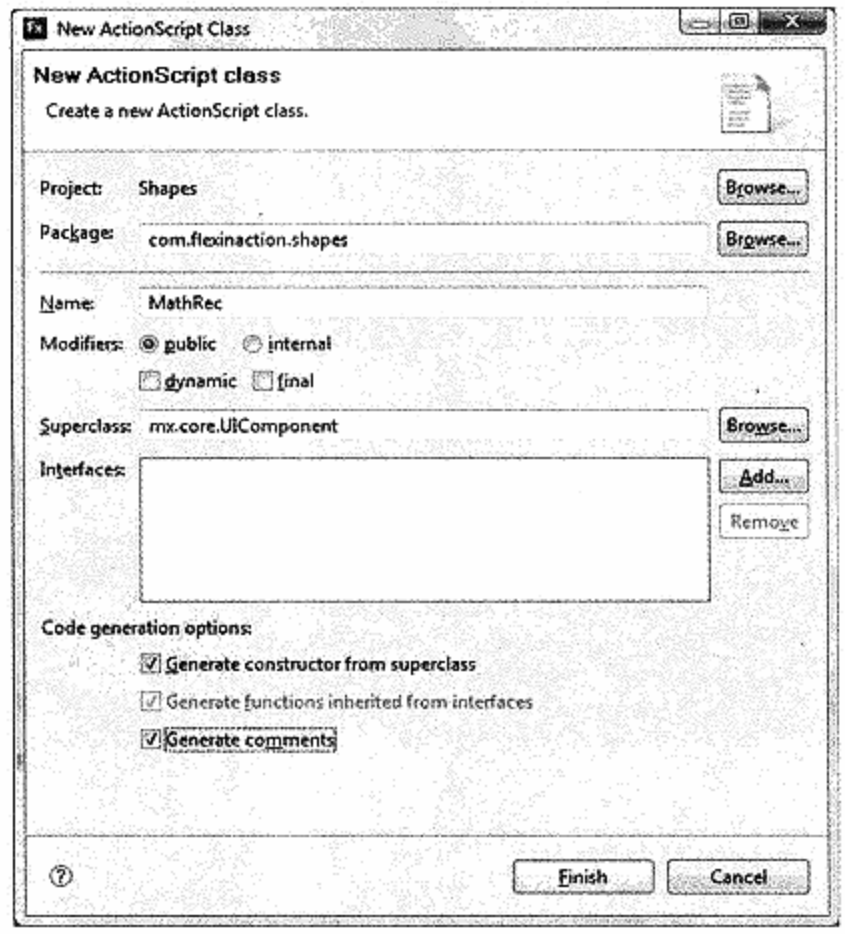


图 16-6 Flex Builder 的类向导

16.3.2 指定包

第一个字段用于指定类的包名。包是用于保存类文件的文件夹。将类分别放在不同文件夹中比直接放在顶级目录中更好。通过包来组织元素，能够有效地加快程序的开发进程。

虽然每个人都可以根据自己的喜好来为包起名，但大多数开发人员更倾向于选择反向域名(reverse-domain)表示法，即把域名反过来写(如 com.mycompany)，然后再定义所需的任何标识符。

例如，包名 com.mycompany.sales.management 表示的文件夹结构是 com/mycompany/sales/management。

这个练习中使用的是 com.flexinaction.shapes。之所以把 shapes 作为组名，是因为这个文件夹中将包含创建的所有形状类。

16.3.3 命名类

接下来的输入字段不言自明——指定类的名称。我们不建议随意给类命名，最好为它取一个与其用途有关的名字。在此，我们打算创建一个矩形，因此使用的名称是 `MathRec`。在为类命名时，注意不要跟项目中的其他类重名。

16.3.4 类修饰符

类修饰符用于控制其他哪些应用程序元素能够访问这个类。通过选择 Name 字段下方的一对单选按钮(标签分别为 public 和 internal)来设置类修饰符。

形象一点说，修饰符可以决定要创建的类与其他类的交往程度。如果选择了 public 修饰符，表明允许其他类及应用程序访问这个类。如果选择了 internal，则意味着只有位于同一个包中的类才能访问这个类。实际上，可以从下列 4 个作用域修饰符中选择。

- public：任何代码都可以使用这个类。
- internal：默认值。同一包内的类可以使用这个类。
- dynamic：可选，能与 public 或 internal 共同使用，表示允许在运行时添加属性。
- final：可选，能与 public 或 internal 共同使用，表示不允许其他类扩展这个类。

至于选择哪个更合适，这是在设计应用程序时需要考虑的一个架构问题。影响选择的因素可能有几个。

- 是否希望其他类访问这个类？
- 公开这个类有没有安全问题？
- 这个类是否只需要支持它所知道的其他类？

找到这几个问题的答案，基本上就大致了解了类并知道如何创建对象。但是，怎么利用应用程序已经定义的类呢？这就涉及扩展类。

16.3.5 超类：扩展类

不，超类跟来自氪星的超人、超女可不是一回事。超类用来扩展一个类的属性和方法(通过添加另一个类的属性和方法)。例如，假设有两个类 A 和 B。A 有 20 个方法，B 有 10 个。B 通过扩展 A 不仅可以保留原先的方法，还能拥有 A 的所有方法。因此，B 最终会有 30 个方法。

在此，我们要使用 mx.core.UIComponent 来扩展计算面积的类，因为 mx.core.UIComponent 属于 Flex 的核心类，而且它与在屏幕上显示的视觉对象有关——非常符合我们的思路(创建形状)。稍后，我们将介绍以编程方式(而不是依托向导)修改和扩展类的方法。

提示：

超类也被称为父类或基类。

然后介绍接口。接口与类相似，但没有类那么充实(或者完善)。

16.3.6 接口

如果不单纯从技术的角度来讲，接口就是预定义的模板，其中定义了一组方法和属性。如果某个类声称遵守某个接口，那么该类就必须实现相应接口定义的方法。例如，我们定义了一组形状类(矩形、圆形和三角形等)，而这些形状计算面积和周长的公式各不相同。此时，可以通过一个接口来强制它们都拥有支持每种计算公式的方法。

接口是作为类的契约(contract)存在的。契约用于声明某个类将会实现由相应接口定义的方法。可以创建一个名叫 Shape 的超类，并定义 area 和 perimeter 两个方法，然后再通过子类(如 Circle 类)来重写这些方法。但这不是最佳方式。因为计算圆形面积的公式与计算矩形面积的公式不一样，所以 Shape 的 area 方法只能白白地占用资源。

我们想要解决的问题是，让所有形状的类都支持既定的方法。显然，接口既能解决这个问题，又不会浪费资源(因为它们不包含实现代码，也只需很少的系统处理)。

下面我们就来理论联系实际——实际地创建并使用类。

16.3.7 分析类

严格按照前面指示做的读者，现在可以单击 New ActionScript Class 向导中的 Finish 按钮。Flex Builder 会自动创建如代码清单 16-2 所示的代码。

代码清单16-2 基本的类结构

```
package com.flexinaction.shapes    ❶
{
 import mx.core.UIComponent;
 public class MathRec extends UIComponent    ❷
 {    ❸
  public function MathRec()
  {
     super();
  }
 }
}
```

读者或许觉得代码中有很多地方都很熟悉。

- package 标识符❶后面跟着我们输入的包名。
- public 修饰符❷被指派给类，因此这个类可以被其他类和应用程序本身访问。
- MathRec 类扩展了 UIComponent❸。

注意：

本章我们一直在谈论方法，但代码清单 16-2 中展示的却是一个函数——是不是搞错啦？没有。函数是对方法的另一种叫法。编程和开发社区的人们经常叫方法，但方法和函数其实是一回事。

如代码清单 16-2 所示，我们创建一个名为 MathRec()的函数。太巧了，MathRec 也是类的名字。每当创建类的一个实例时，ActionScript 都会自动执行该类的一个默认方法(这个方法与类同名)。而这个被执行的默认方法，叫做构造函数。

在这个方法中，还包含另一个名为 super()的方法。这是调用父(超)类构造函数的一种方式。由于我们扩展的是 UIComponent，因此 super()调用的就是 UIComponent 的构造函数(用于初始化对象)。可以不定义构造函数，而且省略构造函数还会带来一点性能上的提升；但是最佳实践还是包含构造函数。

下一节，我们介绍如何为类赋予唯一的特征——添加属性。

16.4 添加属性

在创建了类的框架之后，接下来该考虑绘制图形的问题了。但在此之前，必须了解一

点用户需求。例如，用户是喜欢红色的矩形，还是喜欢蓝色的？另外，要是用户喜欢大的不喜欢小的呢？如果能让用户来输入相应的值，直接控制形状就好了。可是，类怎么知道用户输入了什么值，或者说通过什么方式来接收这些值呢？

在真实的世界里，要想改变周围的环境，可以把说明或资料交给相关的组织或个人。例如，可以告诉施工人员按照指定长度和宽度画一个矩形，还可以告诉他们把这个矩形刷成红色。

通过类也能实现这一切。方法是创建用于保存说明信息的变量。请读者在类标识符的下一行，创建以下 3 个私有变量。

- _color，类型为 uint。
- _height，类型为 uint。
- _length，类型为 uint。

每个变量开头的下划线字符是一个常见的业界约定，表示该变量是私有变量。虽然不加下划线也可以，但加上它有助于在代码中把这些私有作用域的变量区分出来。

添加了新变量后，原来的代码应该变成代码清单 16-3 所示的样子。

代码清单16-3 定义了私有属性的类

```
package com.flexinaction.shapes
{
  import mx.core.UIComponent;
  public class MathRec extends UIComponent
  {
    private var _color:uint = 0x000000;
    private var _length:int = 10;
    private var _height:int = 20;
    public function MathRec()
    {
      super();
    }
  }
}
```

提示：

为什么创建 int 和 uint 类型的变量不使用 new Object()语法呢？因为它们是原始数据类型，谈不上对象实例化。关于原始数据类型的详细内容参见第 3 章。

这些变量都是类的属性。

16.5 添加 getter/setter 方法

可以简单地把新属性定义为公共属性，允许外部代码访问和操作它们。这样做虽然不能算错，但为了通过 Flex 适当而有效地编程，最好还是采用访问这些变量的另一种机制——getter 和 setter 方法。

所谓 getter 和 setter 方法，它们都只有一个任务，即读取和修改变量。这种机制可以保

证读取和修改变量都是内部行为，不会受到调用代码干扰。

首先，我们创建用于设置 3 个变量的 setter 方法。创建 setter 方法要遵循下面的简单模式：

```
public function set yourPropertyname(value:DataTypeOfProperty):void{
   this._propertyname = value; }
```

在 setter 方法前面冠以 public 标识符，表明该方法可以被其他类和主应用程序访问。使用 function 标识符表示创建方法，而使用特殊关键字 set 标明这是一个 setter 方法。

之后的属性名应该尽力避免混淆，不能让人分不清是设置还是读取属性。虽然可以任选，但最好还是将它确定为与私有变量同名。接着，在圆括号中指定想要通过参数传递的值及其数据类型。:void 表示这个函数不会返回值。最后，在花括号中编写要执行的代码。代码清单 16-4 展示了几个例子。

代码清单16-4 setter函数的例子

```
public function set color(value:uint):void
{
 this._color = value;
}

public function set Reclength(value:int):void
{
 this._length = value;
}

public function set Recheight(value:int):void
{
 this._height = value;
}
```

代码中位于变量名前的 this 是引用类中变量的一种语法。与下划线字符一样，使用 this 不仅使代码易懂，而且能清晰地标明变量来源(在此来源就是当前类)。

其次，我们要创建 3 个用于访问变量的 getter 函数。语法与创建 setter 函数类似，只不过要使用 get 关键字而不是 set，如代码清单 16-5 所示。

代码清单16-5 getter函数的例子

```
public function get color():uint
{
 return this._color;
}

public function get Reclength():int
{
 return this._length;
}

public function get Recheight():int
{
 return this._height;
}
```

显然，getter 函数与 setter 函数最大的区别在于前者不是传入值，而是返回值。

16.6 创建类的方法

有了 getter 和 setter 方法之后，就该想办法绘制矩形并添加计算矩形面积和周长的方法。ActionScript 内置了很多绘制图形的类，其中就有可以用来创建矩形的类。

请读者按照代码清单 16-6 所示，创建一个名为 DrawRectangle()的新方法。

代码清单16-6 绘制矩形的方法

```
public function DrawRectangle():void
{
 this.graphics.clear();    ❶
 this.graphics.lineStyle(1,_color);    ❷
 this.graphics.drawRect(120,120 ,_length,_height);    ❸
}
```

在以后调用这个方法时，它会执行下列操作：

- 清除舞台中的所有矩形❶。
- 将线条的颜色设置为私有变量_color 中保存的颜色(在类定义中，该颜色默认为黑色)❷。
- 从屏幕上坐标为(120,120)的点开始绘制矩形，长度和高度分别使用_length 和_height 变量的值。

在作为构造函数生成的默认方法中，调用 DrawRectangle 函数。这样，就会在创建类的实例时绘制一个默认的矩形。相应的代码如下所示。

```
public function MathRec()
{
 super();
 DrawRectangle();
}
```

有了绘制形状的方法后，接下来创建用于计算面积和周长的辅助方法。代码清单 16-7 所示为用于执行相应计算的函数。

代码清单16-7 支持方法

```
public function getArea():int
{
 return (this._length * this._height);    面积=长度*高度
}

public function getPerim():int
{
 return ((2*(this._height)) + (2*(this._length)));    计算周长
}
```

现在使用该应用程序。尝试在高度和长度字段输入值，并观察结果，如图 16-7 所示。

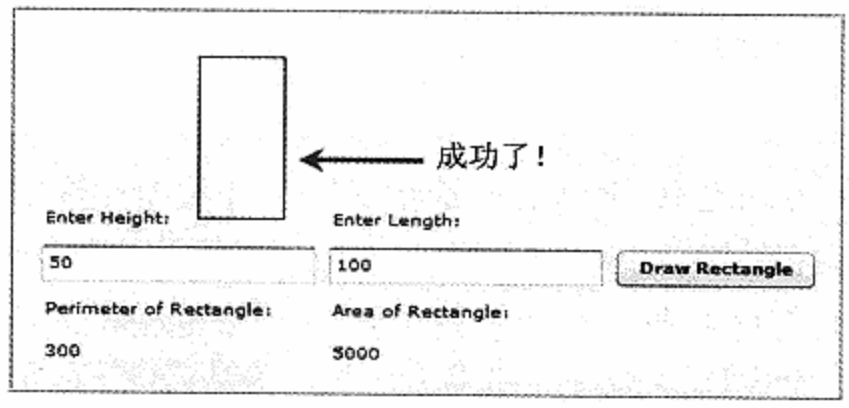


图 16-7 使用 MathRec 类

代码清单 16-8 所示为这个类的完整代码。

代码清单16-8 CH16.mxml：使用创建的类

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            layout="absolute" backgroundColor="white"
            initialize="addChild(myRectangle);" >
 <mx:Script>
  <![CDATA[
   import com.flexinaction.shapes.MathRec;
   public var myRectangle:MathRec = new MathRec();

   private function drawMyRectangle():void
   {
    myRectangle.Recheight = Number(inputHeight.text);
    myRectangle.Reclength = Number(inputLength.text) ;
    myRectangle.DrawRectangle();
    outputArea.text = myRectangle.getArea().toString();
    outputPerim.text = myRectangle.getPerim().toString();
   }
  ]]>
 </mx:Script>

 <mx:Button x="365" y="207" label="Draw Rectangle"
  click="drawMyRectangle()"/>
 <mx:Label text="Enter Length:" x="197" y="181"/>
 <mx:TextInput x="29" y="207" id="inputLength"/>
 <mx:Label text="Enter Height:" x="29" y="181"/>
 <mx:TextInput x="197" y="207" id="inputHeight"/>

 <mx:Label text="Area of Rectangle:" x="197" y="237"/>
 <mx:Label id="outputArea" x="197" y="263"/>
 <mx:Label text="Perimeter of Rectangle:" x="29" y="237"/>
 <mx:Label id="outputPerim" x="29" y="263"/>
</mx:Application>
```

把myRectangle添加到应用程序中

导入类

创建MathRec的实例

setter函数

getter函数

调用公共方法

如前所述，要使用对象，必须先导入相应的类。这就是代码清单 16-8 中使用 import 语句导入 MathRec 类的原因。

```
import com.flexinaction.MathRec
```

而 drawMyRectangle()方法负责将用户输入的高度和长度值转换为数值，然后使用 setter 函数设置 MathRec 类的属性。

16.7 小结

在 Flex 开发中，一切都可以归结为 ActionScript 编程。熟悉标签语言(如 HTML)的读者，一开始可能会不太适应；不过，随着时间的推移，相信这些读者也能体验到使用 ActionScript 编程的乐趣。在体验到编程乐趣的同时，ActionScript 类的威力也就发挥出来。

在学习 OO 编程时，有人光在研究各种理论和高级技术上就可以花上几个月时间。但经过对本章的短暂学习之后，读者已经成功掌握了 OO 编程的各种基本知识。而且自己的 Flex 开发技能也得到前所未有的提升。

熟练掌握 OO 编程，有利于创建出设计良好的应用程序，而且这些应用程序将构建于自包含、可重用的对象基础之上。第 17 章，我们将通过探讨自定义组件的应用，进一步加深读者对可重用性的理解。

第 17 章 自定义组件

17

本章内容：

- 创建自定义组件
- 扩展现有组件
- 为自定义组件传递参数
- 从自定义组件检索值
- 利用 getter 和 setter 函数
- 通过自定义组件传播事件
- 创建 ActionScript 组件

读者在学习 Flex 的短暂过程中已经掌握了大量新知识。事实上，以你现在掌握的知识为起点，已经完全可以创建出标准的 Flex 应用程序。

无论是捕获和验证用户输入，还是与后端数据服务交互以便获取和传输数据，乃至格式化并显示信息，都已经不成问题。能够做到这些当然是一件好事；不过，除非你想把所有代码都堆在一个大文件中，否则还需要学会把应用程序分割成更小的、可重用的块。

这正是自定义组件的目的所在。自定义组件虽然不是实现代码重用的唯一手段，但却是最受欢迎的，因为它很容易实现。

注意：

本章我们将向读者传授在构建 Flex 应用程序时思考问题的方法。我们不会直接把最后的答案端出来，而是会引导你寻找解决方案。

本章介绍创建和利用自定义组件的基本知识，包括向组件中传递和从组件中输出信息。就像在许多编程语言中一样，实现同一任务的方式有很多种，而每种方式各有利弊。我们将深入探讨在面临这些问题时该如何做出取舍。不过，首先还是要解释一下什么是自定义组件。

17.1 什么是自定义组件

从本书开始我们就在使用组件，包括诸如 Button(<mx:Button>)的接收用户输入的视觉组件乃至诸如 VBox 的用于布局的组件。

所谓自定义组件，指的是开发人员自己创建的组件，不是 Flex 内置的组件。在实践中，也可能会出现一种情况，即自定义组件的可重用程度很高，几乎达到 Flex 内置组件的水平。

提示：

这一点对于 ColdFusion 开发人员也一样。虽然无法与 ColdFusion 直接对应，但我们仍然可以说，Flex 中的自定义组件类似于 ColdFusion 中包含用户自定义函数(UDF，User-Defined Function)的自定义标签。但从后台来看，Flex 实现自定义组件的方式非常类似于 CFC。

为什么 Adobe 不提供那些明显有用的组件呢？根据 Flex 工程团队的既定目标，构建这个框架是为了让开发人员社区自行创建组件。如果 Flex 团队把时间都花在创建更多的组件上，用在关注 Flex 整体功能上的时间将会减少。

自定义组件示例

下面我们看一个简单示例。假设有一个全球中心型应用程序，该程序中肯定会用到位置数据。因此，会有一些表单要求用户填写地址等信息。为便于用户填写美国州名，需要创建一个如图 17-1 所示，包含美国州名的下拉菜单。

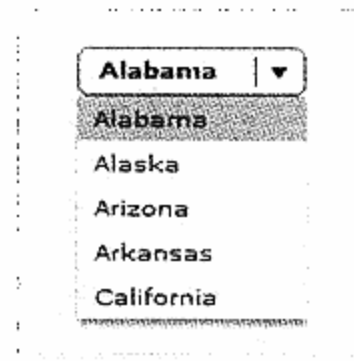


图17-1 使用ComboBox展示美国州名的自定义组件

代码清单17-1 cbStates.mxml：基于ComboBox的自定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:dataProvider>
      <mx:Object stateCode="AK" label="Alaska"/>
      <mx:Object stateCode="AL" label="Alabama"/>
      ...
    </mx:dataProvider>
</mx:ComboBox>
```

省略的美国州名

代码清单17-2 HelloWorld.mxml：主应用程序文件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white" xmlns:local="*">
   <local:cbStates/>
</mx:Application>
```

如果在应用程序中显示这个下拉菜单只需一行代码，不会是耸人听闻吧？当然，这是一条加载语句，而且对于这个示例来说，就这么简单：

```
<local:cbStates/>
```

完整的代码如代码清单 17-1 和代码清单 17-2 所示。在创建自定义组件时，我们的目

标应该是让其他开发人员更方便地使用相应的组件(即使组件内部非常复杂)。

稍后，我们将对如何达成这一目标给出提示。如果读者认同我们陈述的这个观点，那么本章其余内容也就算不上什么难题了。

17.2　自定义组件的工作原理

构建自定义组件的总体策略，就是以 Flex 现有组件为基础，添加所需的功能和自定义逻辑。

前面展示的美国州名的下拉菜单是基于 ComboBox 组件构建的，相应代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml"/>
```

如果把这两行代码保存为一个文件，那么就可以把它们用作一个功能完备的 ComboBox 组件。如果读者觉得我们是在谈面向对象的继承，你是正确的！创建自定义组件靠得就是继承，即创建一个组件，让它扩展现有组件。这样一来，就可以继承：

- 属性
- 函数
- 事件
- 样式

通过组件继承组件，继承的东西可以逐步累积起来。而这正是我们的总体策略——继承链中的每一个新级别会逐渐增加更多功能。后代组件会变得越来越强大，但每一代之间的差别则不会太悬殊，我们可以让自定义组件继承其中任意一代，然后再添加额外功能。

公共域(Public domain)组件

如果读者在查找通用组件，那么很可能它已经存在了。查找通用域组件的流行站点包括：

- CFLEX.Net(http://www.cflex.net)
- FlexLib(http://code.google.com/p/flexlib/)
- FlexBox(http://flexbox.mrinalwadhwa.com/)
- RIAForge(http://www.riaforge.org)

别忘了回馈这些社区；如果你创建了好用的组件，一定要共享出来！

有趣的内容还不止这些。为了实现更高的灵活性，可以让自定义组件囊括各种不同的组件，这种组件就是复合组件。

17.2.1　简单组件与复合组件

自定义组件主要有两种：简单组件和复合组件。简单自定义组件基于一个组件构建，就像前面显示州名的 ComboBox 一样。复合自定义组件则基于一个容器构建，其中又包含一个或多个组件。

除了可以选择要创建的组件类型，我们还可以选择在应用程序中保存它们的方式。

17.2.2 实现方式

实现自定义组件的手段有好几种，每种都有优势和不足。别忘了，Flex 编译器在构建应用程序时，会把所有自定义组件都转换成纯粹的 ActionScript 类。

1. 纯 MXML 方式

这种方式最容易上手。相应的自定义组件将以 MXML 文件形式存在，同时在文件顶部声明用于扩展的 Flex 组件。

这样说吧，打开主 MXML 应用程序文件，然后把<mx:Application>标签中的 Application 修改为另一个容器名，如 Canvas。知道这样做的结果吗？你的第一个自定义组件诞生了。

这种方式直观方便，而且利用了迄今所学的知识。此外，由于使用的是视觉组件，因此布局问题也较容易解决。但是，不能重写自定义组件继承的组件的函数。

2. 纯 ActionScript 方式

以纯 ActionScript 方式定义的自定义组件要保存为.as 文件，创建方法是扩展 Flex 组件的 ActionScript 版(也称组件类)。这种技术相对更高级一些，因为它要求开发人员掌握更多的 ActionScript 技能。当然，可以重写从中扩展的组件类的函数，还可以对自定义组件进行精确地控制。不过，使用这种方式不能借助 Flex Builder 以可见形式设置组件的布局。

3. 代码隐藏方式

这种方式综合了前面两种技术，它使用 ActionScript 创建扩展的组件类，使用 MXML 文件(基于 ActionScript 类)以可见形式布局组件。

虽然这种技术兼有 MXML 和 ActionScript 组件的优势，但工作量较大。而且，由于是双重扩展，消耗的资源也稍微多一些。

17.3 简单自定义组件

理论就到此为止，好戏开场了。代码清单 17-1 和代码清单 17-2 所示为如何重新生成本章前面示例的代码。

在 Flex Builder 中把这两个文件保存到同一个文件夹(见图 17-2)，然后运行这个应用程序，将看到如图 17-1 所示的下拉组合列表，其中包含所有美国州名。

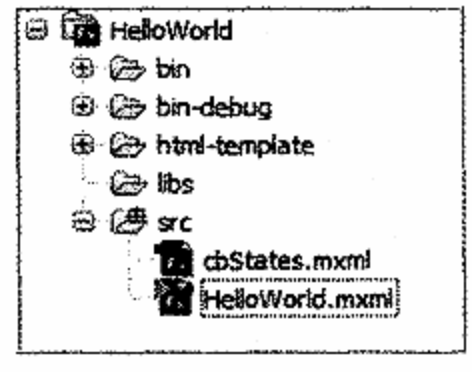


图17-2 主应用程序文件和自定义组件文件的位置

这是一个基于非容器组件创建的自定义组件。对于这种组件而言，使用者根据继承关系很容易知道它的属性和行为。具体来说，如果知道这个简单自定义组件基于 ComboBox 构建，也就知道这个自定义组件具有与 ComboBox 相同的属性、事件、样式及函数。

知道如何创建自定义组件后，下面我们介绍怎样通过参

数为组件传递信息。

17.4　传递参数

仍以前面创建的自定义组件为例。现在，我们想为它传递一个值——用于指定默认显示的美国州名的参数(在真正的项目开发中，如果是从数据库中取出记录并填充到编辑表单中，那么，默认情况下，所有的值都会基于相应的数据)。

Flex 支持一些传递参数的机制，同样各有利弊。好在，我们既可以选择一种机制，也可以组合使用多种机制。

17.4.1　通过函数传递值

指定默认值的一种最简单方式，莫过于调用一个设置该值的函数。根据这个思路，代码清单 17-3 改进了前面的自定义组件，为它定义了一个公共函数，以便调用代码通过它设置默认的美国州名。

代码清单 17-4 所示为调用代码如何使用该函数。

代码清单17-3　cbStates.mxml：组件中包含设置默认值的函数

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      public function setDefaultState(stateCode:String):void
      {
          for(var x:int=0;x<this.dataProvider.length;x++)
            if(this.dataProvider[x].stateCode == stateCode)
              this.selectedIndex = x;
      }
    ]]>
  </mx:Script>
      <mx:dataProvider>
        <mx:Object stateCode="AK" label="Alaska"/>
        <mx:Object stateCode="AL" label="Alabama"/>
        ...
      </mx:dataProvider>
</mx:ComboBox>
```

代码清单17-4　HelloWorld.mxml：调用setDefaultState函数的主应用程序文件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()" backgroundColor="white" xmlns:local="*">
  <mx:Script>
    <![CDATA[
      public function initApp():void
      {
        myStates.setDefaultState('CA');
      }
        ]]>
        </mx:Script>
      <local:cbStates id="myStates"/>
</mx:Application>
```

现在的自定义组件更好用，更像那么回事了。像这样定义公共函数的做法有一个优点，如果调用程序需要执行一次复杂的初始化例程并调用设置各种参数的函数，那么它恰好可以派上用场。

虽然通过公共函数可以传递信息，但这样的函数也可以肩负操作自定义组件的功能。例如，可以通过这样的函数重置自定义组件以及实现其他任何操作。

17.4.2 通过属性传递值

读者肯定是一位善解人意的程序员，因此想让使用这个自定义组件的人更方便一些。你会想，如果能像下面这样以设置属性的方式传递值就好了。

```
<local:cbStates id="myStates" defaultValue="AZ"/>
```

你打算使用的这种传统方法确实可行。为了方便起见，我们这次让自定义组件扩展 TextInput(见图 17-3)。

图 17-3 自定义 TextInput 组件显示通过属性设置的值

相应的代码很简单。在代码清单 17-5 中可以看到，调用应用程序文件(代码清单 17-6)中视为属性的项恰好是 myTextInput.mxml 内部的一个公共变量。

代码清单17-5 myTextInput.mxml：将自身设置为公共变量的值

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TextInput xmlns:mx="http://www.adobe.com/2006/mxml"
          text="{defaultValue}">
  <mx:Script>
    <![CDATA[
        [Bindable]
         public var defaultValue:String = "";
      ]]>
    </mx:Script>
</mx:TextInput>
```

代码清单17-6 HelloWorld.mxml：设置自定义组件的属性

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white" xmlns:local="*">
    <local:myTextInput defaultValue="hi mom"/>
</mx:Application>
```

真是够简单的。这种技术在某些情况下有效，但它对自定义 ComboBox 也有效吗？

1. 需要考虑的问题

看一看代码清单 17-7，其中也使用了相同的手段。你觉得在此应该怎么使用 defaultValue？

代码清单17-7　使用公共变量的cbStates.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      [Bindable]
      public var defaultValue:String = "";
    ]]>
    </mx:Script>
    <mx:dataProvider>
      <mx:Object stateCode="AK" label="Alaska"/>
      <mx:Object stateCode="AL" label="Alabama"/>
      ...
    </mx:dataProvider>
</mx:ComboBox>
```

这里没有简单的文本字段可以用来原封不动地显示值。能不能通过组件的 creationComplete 事件设置它呢？这样，当 Flex 创建完该组件后，相应的函数会运行并恰当地设置默认的美国州名。

```
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml"
        creationComplete="setDefaultState()">
```

但这种方式只在始终事先设置 defaultValue 的情况下才有效。

```
<local:cbStates defaultValue="AZ"/>
```

如果调用代码修改了 defaultValue，又该如何呢？如果真发生了这种事，我们也不会知道。应该了解发生已经更改的值，然后通过应用适当的逻辑来确定为 ComboBox 的 selectedIndex 属性使用哪个值。

2. 利用绑定事件跟踪变化

一种解决方案是利用 Flex 的绑定机制，捕获绑定事件——该事件在值改变时发生。我们的新对策如下。

- 使用前面提到的 creationComplete 事件设置初始值。
- 使用 Flex 的 ChangeWatcher 对象监控在 defaultValue 属性上发生的数据绑定事件。每当发生绑定事件时，就更新选择的美国州名。

代码清单 17-8 和代码清单 17-9 把这两个对策付诸实践，添加了自动截获变化事件和予以处理的机制。

代码清单17-8　cbStates.mxml：经过改进后，可以自动检测defaultValue的变化

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml"
        creationComplete="initComponent()">
  <mx:Script>
    <![CDATA[
      import mx.binding.utils.*;
      [Bindable]
      public var defaultValue:String = "";
```

```
      private function initComponent():void
      {
        setDefaultState(null);
        ChangeWatcher.watch(this, "defaultValue", setDefaultState);
      }

      public function setDefaultState(evt:Event):void
      {
        for(var x:int=0;x<this.dataProvider.length;x++)
          if(this.dataProvider[x].stateCode == defaultValue)
            this.selectedIndex = x;
      }
    ]]>
  </mx:Script>
  <mx:dataProvider>
    <mx:Object stateCode="AK" label="Alaska"/>
    <mx:Object stateCode="AL" label="Alabama"/>
    ...
  </mx:dataProvider>
</mx:ComboBox>
```

如果指定了，则会设置默认值

监控默认值的绑定事件

代码清单17-9 HelloWorld.mxml：支持对默认值的修改

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white" xmlns:local="*">
  <local:cbStates id="stateSelect" defaultValue="AZ"/>
  <mx:Button click="stateSelect.defaultValue='AL'" label="Change State"/>
</mx:Application>
```

经过一番改造之后，现在的自定义组件既可以接收初始的设置值，又能跟踪设置值的变化(见图 17-4)。太棒了！

虽然把最终代码和盘托出也是一种方式，但我们的想法是向读者传达思考的过程，以便读者能在开发 Flex 项目时举一反三。而且，我们还会展示完成相同任务的不同方式。

实际上，设置值的技巧还有很多。另一个技巧就是使用 setter 函数，这种方式巧妙得让函数在外界看来就像是一个属性。

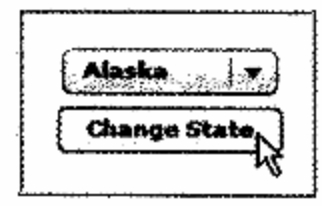


图17-4 显示美国州名的自定义组件。单击按钮，州名改变

17.4.3 使用 setter 函数作为属性代理

Flex 支持 setter 函数。从调用代码的角度看，代码好像在设置一个普通的属性；但实际上，它是在调用函数。

这种方式的优点很多，几乎是集各家之所长。

- 具有设置属性般的方便。
- 在设置属性的同时可以应用相关逻辑。
- 对外部世界抽象了内部的处理细节。
- 简化了实现过程。

注意：

既然这种方式好处多多，为什么还要介绍其他方式呢？是为了让读者了解各种手段。这样，当读者在开发项目时遇到特殊的困难，就有选择最佳解决方案的余地。

在使用 setter 函数时，该函数需要以某种方式设置内部的一个私有变量。把前面的示例(代码清单 17-8)修改成使用 setter 函数，将得到代码清单 17-10。

代码清单17-10　cbStates.mxml：经过修改后，使用setter函数

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      private var _defaultValue:String="";
      public function set defaultValue(statecode:String):void
      {
        _defaultValue = statecode;
        for(var x:int=0;x<this.dataProvider.length;x++)
          if(this.dataProvider[x].stateCode == _defaultValue)
            this.selectedIndex = x;
      }
    ]]>
  </mx:Script>
  <mx:dataProvider>
    <mx:Object stateCode="AK" label="Alaska"/>
      <mx:Object stateCode="AL" label="Alabama"/>
    ...
  </mx:dataProvider>
</mx:ComboBox>
```

真正的属性，设置为私有

set关键字表示这是setter函数

此后，任何认为是给 defaultValue 属性赋值的代码，实际上都是在调用一个名为 defaultValue 的函数，而该函数又会相应地设置一个内部变量。

```
<local:cbStates id="stateSelect" defaultValue="AZ"/>
```

提示：

为与 setter 函数相关的变量添加下划线前缀(如_defaultValue)是一种行业约定。这种变量跟普通变量没有功能上的区别。

显然，setter 函数为调用者和实现者都提供了方便；但是，我们还想为未来的 Flex 开发人员介绍另一个工具。想象一下，如果把 setter 函数和绑定结合起来呢——那就是强强联合啦！

17.4.4　使用绑定参数

不要忽略一个事实：调用代码能够传递绑定的值，而不仅仅是设置一个静态属性的值。因此，我们可以像为许多列表组件的 dataProvider 属性使用绑定一样，在自定义组件中也使用绑定。

```
<mx:List dataProvider="{myArrayCollection}"/>
```

还以显示美国州名的下拉菜单为例，假设要支持传入一组州名的操作，因为我们决定将这个组件的功能一般化为只展示数据。至于从哪里获得数据并不是这个组件所要关心的事(对于一个国际化的应用程序而言，必须根据国家显示相应的行政区名，因此这种情况完全可能)。

代码清单 17-11 把值传递给代码清单 17-12，后者可接收这些值。

代码清单17-11 HelloWorld.mxml：为自定义组件传递一个绑定的属性

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white" xmlns:local="*">
  <mx:Script>
   <![CDATA[
    import mx.collections.ArrayCollection;
    [Bindable]
    public var myAC:ArrayCollection = new ArrayCollection([
    {label:"Alaska", stateCode:"AK"},
    {label:"Alabama", stateCode:"AL"},
    {label:"Arizona", stateCode:"AZ"}]);
    ]]>
  </mx:Script>
  <local:cbStates id="stateSelect" useThisData="{myAC}"/>
</mx:Application>
```

将更改信息传递给myAC

代码清单17-12 cbStates.mxml：使用绑定的属性

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml"
   dataProvider="{useThisData}">
  <mx:Script>
   <![CDATA[
    import mx.collections.ArrayCollection;
    [Bindable]
    public var useThisData:ArrayCollection;
    ]]>
  </mx:Script>
</mx:ComboBox>
```

列表会随着这个值改变

绑定是 Flex 中一个无所不在的特性，绝不应该忽视它；它能帮助我们节省大量时间。

当然到目前为止，无论用不用绑定，读者都可以选择很多方式为自定义组件传入信息。下一个必然的步骤是从自定义组件中取得信息。取得信息的“作战计划”应该是相同的，只不过方式相反而已。

17.5 检索值

设置和传递值只是一方面，另一方面是取得值。在本章示例中，如果用户选择了一个美国州名，调用代码需要取得用户当前的选项(在真实的应用程序中，调用代码将在某一时刻把取得的值提交到数据库)。

与传递值相似，取得值的方式也有很多种，有的简单，有的复杂。我们会把最佳方式(针对这个例子而言)放到最后讨论，不过读者应该知道如何做出选择。以下介绍的方式包括使用函数和属性取得值。

17.5.1 通过函数检索值

通过函数不仅可以设置值，而且可以取得值。通过函数取得值的好处是，可以在返回

值前执行必要的逻辑处理。这种方式也隔离和抽象了决定返回什么数据的逻辑；可以根据需要添加自定义业务逻辑，调用代码不用关心这些逻辑。这样虽然增加了自定义组件的工作量，但还是值得的。另外要注意的是，不能使用 Bindable 属性。

考虑一个购物车的例子，用户可以向其中添加商品，而每个商品都有自己的价格及用户选择数量。在购物车内部，可能没有一个跟踪总金额的变量。不过，对使用你的自定义组件的人来说，你是个好心人，会提供一个用于动态计算总金额的函数。

对于本示例而言，需要在代码清单 17-10 的自定义组件(cbStates.mxml)中添加如下函数，以便返回值。

```
public function getValue():String
{
 return this.selectedItem.stateCode;
}
```

这是一个公共函数，因此访问起来相对容易，如代码清单 17-13 所示。

代码清单17-13　HelloWorld.mxml：使用函数检查当前值

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white" xmlns:local="*">
 <mx:Script>
  <![CDATA[
   import mx.controls.Alert;
   public function checkValue():void
   {
    Alert.show(stateSelect.getValue(),"Current U.S. state");
   }
   ]]>
 </mx:Script>
 <local:cbStates id="stateSelect" defaultValue="AZ"/>
 <mx:Button click="checkValue()" label="Check which state is selected"/>
</mx:Application>
```

编译并运行这个应用程序，将看到与图 17-5 所示类似的结果。这是从自定义组件中取得值的一种简单方式。

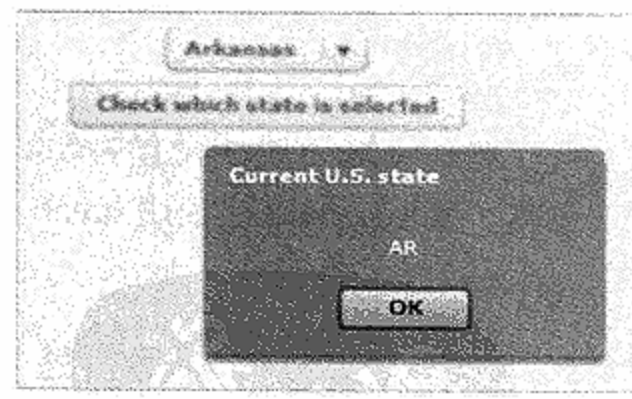


图 17-5　通过访问自定义组件的公共函数查看当前选择的美国州名

现在，我们面临一个使用函数发送信息的反问题：能不能利用属性的便利呢？例如，像下面这样绑定当前值。

```
<mx:Label text="The current state is:
  {stateSelect.getValue()}"/>
```

非常遗憾，代码清单 17-13 中的代码无效，因为不能这样来绑定。不过别担心，这个问题还是有解的。但是我们要先来看一看必须提供什么属性。

17.5.2　通过属性检索值

没人喜欢受限制。为什么不定义一个公共量，然后让调用代码访问它呢？这是可行的。而且，由于这个自定义组件基于 ComboBox，已经可以访问 selectedIndex 和 selectedItem 等常

见属性了(见代码清单 17-14)。

代码清单17-14 HelloWorld.mxml：访问存在的公共属性

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white" xmlns:local="*">
  <mx:Script>
   <![CDATA[
     import mx.controls.Alert;
     public function checkValue():void
     {
      Alert.show(stateSelect.selectedItem.stateCode,"Current State");
     }
     ]]>
  </mx:Script>
  <local:cbStates id="stateSelect"/>
  <mx:Button click="checkValue()" label="Check which state is selected"/>
</mx:Application>
```

运行代码清单 17-14 后，将看到如图 17-6 所示的结果。

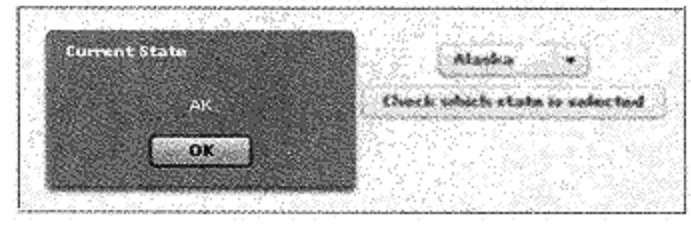


图 17-6 通过属性也可以访问相同的信息

此时你应该扪心自问，愿不愿意这样来设计应用程序。虽然这种想法听起来不错，但却存在一个关键问题：调用程序必须知道怎样取得美国州名。

无论你喜不喜欢，这种做法都破坏了封装规则。问题在于，如果将来修改了自定义组件，而调用代码因此要改变使用这个组件的方式，就必须在所有使用这个自定义组件的地方做此修改(有可能要涉及多个应用程序)。

所以，必须要改进这种做法。方法是使用一个属性，让自定义组件基于内部业务规则设置它的值，然后再从外部访问这个属性。第一步是创建一个 Bindable 且 public 的属性，以便外部代码使用。

```
[Bindable]
public var selectedValue:String="";
```

然后将这个属性设置为初始值(可能是列表中的第一项或者是由调用代码传入的值)。为此，要像下面这样使用 creationComplete 事件。

```
creationComplete="selectedValue=selectedItem.stateCode"
```

最后一件事，就是确保在用户选择不同的美国州名时更新 selectedValue，而这要通过 change 事件来实现：

```
change="selectedValue=selectedItem.stateCode"
```

提示：

在实际项目中，可能会通过 change 事件调用一个函数(如 setValue)，以便在设置适当的值时应用必要的逻辑。而这里的逻辑也不一定只是为了确定正确的值；可以将组件从视觉上变灰或者将组件重置为一个全新的状态，以便开始新的处理。

综合起来，一个好用的自定义组件就诞生了，如代码清单 17-15 所示。

代码清单17-15　cbStates.mxml：使用公共可绑定属性更新的组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml"
         creationComplete="selectedValue=selectedItem.stateCode"
         change="selectedValue=selectedItem.stateCode">
<mx:Script>
 <![CDATA[
  private var _defaultValue:String="";

  [Bindable]
  public var selectedValue:String="";

  public function getValue():String
  {
   return selectedValue;
  }

  public function set defaultValue(statecode:String):void
  {
     _defaultValue = statecode;
     for(var x:int=0;x<this.dataProvider.length;x++)
      if(this.dataProvider[x].stateCode == _defaultValue)
       this.selectedIndex = x;
   }
  ]]>
 </mx:Script>
 <mx:dataProvider>
  <mx:Object stateCode="AK" label="Alaska"/>
  <mx:Object stateCode="AL" label="Alabama"/>
  . . .
 </mx:dataProvider>
</mx:ComboBox>
```

组件使用者可以绑定这个属性

更新函数以便返回值

这里还更新了上一节的 getValue 函数，让它返回 selectedValue 的值。

调用代码中也使用了一个新特性——数据绑定。此时，调用代码既可以像以前一样，在需要时访问值，也可以通过绑定实时获得该值，如代码清单 17-16 所示。

代码清单17-16　利用可绑定的公共属性

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white" xmlns:local="*">
 <mx:Script>
  <![CDATA[
   import mx.controls.Alert;
   public function checkValue():void
   {
    Alert.show(stateSelect.selectedValue,"Current State");
   }
   ]]>
 </mx:Script>
 <local:cbStates id="stateSelect" defaultValue="AZ"/>
 <mx:Button click="checkValue()" label="Check which state is selected"/>
 <mx:Label text="The current state is: {stateSelect.selectedValue}"/>
</mx:Application>
```

绑定到组件的属性

如图 17-7 所示，通过绑定可以实时显示当前选择的美国州名。

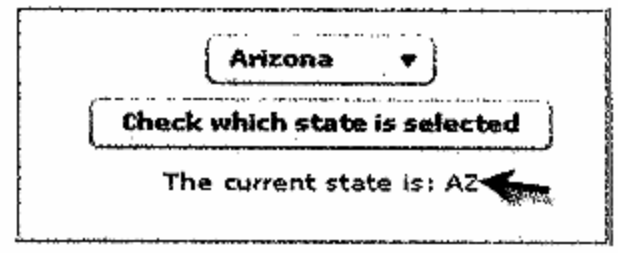


图 17-7 绑定到一个自定义组件的公共属性

现在的自定义组件封装了一些业务逻辑，同时也提供了 Bindable 公共属性的便利。虽然实现起来麻烦一些，但还是有必要的。

假如没有实现这个 Bindable 属性，组件就会存在一个明显的缺点，即它会强迫使用者必须知道怎样根据已有的公共属性确定当前值。仅就目前这个示例而言，这个缺点可能并不明显，因为我们都知道 ComboBox 有一个 selectedItem 属性。可是，还应该考虑如下情况。

- 如果需要添加自定义逻辑怎么办？
- 是否希望调用代码必须知道如何取得这个值？

函数有其优势，属性有其便利，而把它们组合起来的 getter 函数则兼具二者之长。

17.5.3 使用 getter 函数作为属性代理

读者大概早就料到，有 setter 函数，就会有 getter 函数，对吗？

一点不错！使用 getter 函数可以综合函数和属性的长处。按照使用 setter 函数的思路，接下来我们按照下列步骤操作。

(1) 把 selectedValue 修改为私有属性，把 cbStates.mxml 中的所有 selectedValue 重命名为 _selectedValue。

(2) 创建公共的 getter 函数。

```
[Bindable]
private var _selectedValue:String="";

public function get selectedValue():String
{
 return _selectedValue;
}
```

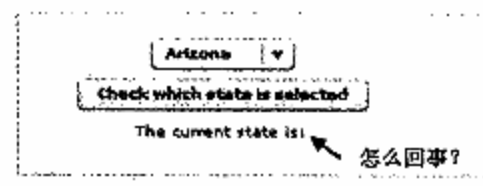


图 17-8 绑定在这里似乎没有效果

然后不修改调用代码，可以运行它并看到如图 17-8 所示的结果。

出问题了！基于_selectedValue 属性的数据绑定似乎并没有把值传送给调用代码。原来，调用代码中绑定的是属性而不是函数。因此，接下来应该让函数也可以被绑定。

```
[Bindable]
public function get selectedValue():String
{
 return _selectedValue;
}
```

注意：

一般来说，不能绑定到函数——可绑定的函数只适用于 getter 和 setter 函数。

如果此时保存代码，Flex Builder 会给出一个错误提示：[Bindable] on read-only getter is unnecessary and will be ignored(为只读的 getter 函数指定[Bindable]关键字没有必要，因此将

被忽略)。出现这个提示的原因是，getter 没有对应的 setter。Flex 很清楚，既然那个属性永远不会改变(没有 setter)，绑定它又有什么用呢？

显然，那个属性是可以改变的，只不过没有通过 setter 改变而已。此时，我们又面临另一个有关实现的抉择。

- 定义一个真实的 setter 函数，这个函数允许设置_selectedValue。可能是想让调用代码可以任意修改选项，或者是不想让这个变量被直接设置。
- 创建一个伪 setter 函数，除了欺骗 Flex Builder 之外，什么也不干。

大多数情况下，第一个选择是正确的。既可以通过真正的 setter 函数添加必要逻辑，对相应属性施加保护；又可以借此机会将设置 defaultValue 的 setter 转换为设置 selectedValue 的 setter。因此，最终的解决方案只需要一个而不是(潜在的)两个 setter 函数。

代码清单 17-17 所示为 cbStates.mxml 的最终版本。

代码清单17-17　cbStates.mxml：getter和setter都针对一个属性

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="selectedValue=selectedItem.stateCode"
    change="selectedValue=selectedItem.stateCode">
  <mx:Script>
    <![CDATA[
      [Bindable]
      private var _selectedValue:String="";
      [Bindable]
      public function get selectedValue():String
      {
        return _selectedValue;
      }

      public function set selectedValue(statecode:String):void
      {
        _selectedValue = statecode;
        for(var x:int=0;x<this.dataProvider.length;x++)
          if(this.dataProvider[x].stateCode == _selectedValue)
            this.selectedIndex = x;
      }
    ]]>
  </mx:Script>
  <mx:dataProvider>
    <mx:Object stateCode="AK" label="Alaska"/>
    <mx:Object stateCode="AL" label="Alabama"/>
    . . .
  </mx:dataProvider>
</mx:ComboBox>
```

使用Setter函数实现隐式更新

最后还要修改调用代码，让它使用 getter 和 setter 函数设置并绑定自定义组件认定的属性。

```
<local:cbStates id="stateSelect" selectedValue="AZ"/>
<mx:Label text="The current state is: {stateSelect.selectedValue}"/>
```

到现在为止，想必读者已经弄懂了封装的含义。创建一个自定义组件，既要考虑到调用代码的方便，又要在公开与控制之间做出正确的选择。

在掌握了创建简单自定义组件及发送和取得数据的技术之后，接下来我们讨论复合组件。

17.6 创建复合组件

上一节我们学习了如何创建简单自定义组件，也介绍了与自定义组件交换信息的各种技术。这些内容涉及创建自定义组件的大多数知识点，本章剩余部分将讨论其他知识点。

谈到复合组件，很多人的第一感觉是复杂，但事实并非如此。所谓复合，指的是让自定义组件基于容器而不是另一个组件。在基于容器的条件下，可以向其中放入多个组件，差不多相当于一个小型应用程序。

下面我们就以本章前面的示例(选择美国州名的下拉菜单)为基础，扩展并创建一个复合组件，实现收集送货地址的功能(见图 17-9)。

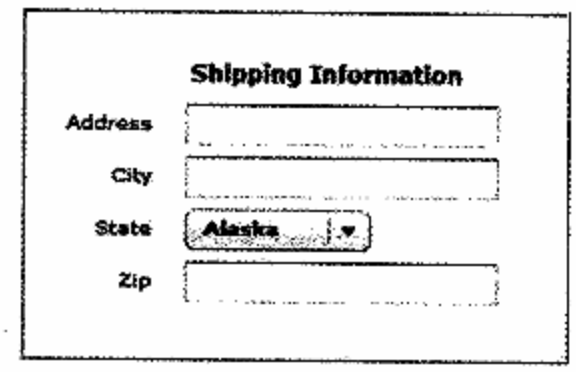


图 17-9 复合组件中的送货信息

代码清单 17-18 所示为生成这个复合组件的代码。这个自定义组件通过容器包含了一些 Flex 内置的组件。代码清单 17-19 中的应用程序调用了这个复合组件。

代码清单17-18 shippingAddress.mxml：复合自定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:local="*">
  <mx:Form>
    <mx:FormHeading label="Shipping Information"/>
    <mx:FormItem label="Address">
      <mx:TextInput id="frmAddress"/>
      </mx:FormItem>
    <mx:FormItem label="City">
      <mx:TextInput id="frmCity"/>
    </mx:FormItem>
      <mx:FormItem label="State">
      <local:cbStates/>
    </mx:FormItem>
    <mx:FormItem label="Zip">
      <mx:TextInput id="frmZip"/>
    </mx:FormItem>
  </mx:Form>
</mx:VBox>
```

自定义组件中的自定义组件

代码清单17-19 HelloWorld.mxml：使用复合自定义组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white" xmlns:local="*">
  <local:shippingAddress id="shipping"/>
</mx:Application>
```

注意到什么有趣的事了吗？自定义组件中包含了另一个自定义组件！把大任务分解成小代码块是创建自定义组件的关键。

前面讨论的一切知识点都适用于复合自定义组件，包括为调用代码提供函数以及传递和取得值。

复合组件的优点是，可以对布局和定位拥有更大的控制权(这还不算可以在其中放置很多东西)，但你的责任仍然是提供足够的函数及属性，以方便调用代码使用相应的复合组件。由于复合组件基于容器构建，因此除了根容器提供的内容外，用户不能指望也无法预言其他任何内容。

迄今为止，我们讨论的都是以 MXML 方式实现自定义组件。但是，另一种重要的实现方式也要介绍，这就是 ActionScript 方式。

17.7　创建 ActionScript 组件

我们曾经提到，前面所做的一切都可以通过 ActionScript 实现。对于熟悉标签语言(像 ColdFusion 和 HTML)的读者来说，自然想使用 MXML 方式，而且也完全没有问题。

不过，随着 Flex 开发技能的增长和经验的丰富，读者会发现 ActionScript 其实也很方便。下面我们就来看是否可以通过 ActionScript 实现供选择美国州名的自定义组件(见代码清单 17-20)。

代码清单17-20　cbStatesScripted.as：cbStates.mxml的ActionScript版本

```
package
{
 import flash.events.Event;

 import mx.controls.ComboBox;
 public class cbStatesScripted extends ComboBox
 {
 [Bindable]
 private var _selectedValue:String="";

 public function cbStatesScripted()
 {
  super();
  dataProvider = [{label:"Alaska", stateCode:"AK"},
                  {label:"Alabama", stateCode:"AL"},
                  {label:"Arizona", stateCode:"AZ"}];
   addEventListener("creationComplete", setItem);
   addEventListener("change", setItem);
 }

 public function setItem(evt:Event):void
 {
  selectedValue=selectedItem.stateCode;
    }

 [Bindable]
 public function get selectedValue():String
 {
  return _selectedValue;
 }

 public function set selectedValue(statecode:String):void
 {
    _selectedValue = statecode;
    for(var x:int=0;x<this.dataProvider.length;x++)
```

❶ 声明包

❷ 导入并扩展类

❸ 创建构造函数

❹ 调用超类中的函数

❺ 导入和扩展类

```
            if(this.dataProvider[x].stateCode == _selectedValue)
             this.selectedIndex = x;
        }
      }
    }
```

对应的操作如下。

(1) 创建名为 cbStatesScripted.as 的文件。

(2) 代码块主体由 package 声明封装❶。package 表示一个目录，用于把多个 ActionScript 类组织在一起。现在使用的是主 MXML 文件所在的目录；后面将讨论如何构造自定义组件。

(3) 导入要从其扩展的 ActionScript 类(即 ComboBox)，然后通过扩展该类创建一个新类❷。

(4) 构造函数与类同名❸。当 Flex 创建这个类的实例时，将自动调用它。

(5) 调用 super 是为了执行父类(ComboBox)的构造函数❹。

(6) 为 creationComplete 和 change 事件注册侦听器❺。

(7) 其余代码不变，只不过使用<local:cbStatesScripted/>而不是<local:cbStates>进行调用。

通过 ActionScript 实现自定义组件的工作量大一些，优点是可以重写父类的构造函数。当然，同时还有一个缺点，即不能在 Flex Builder 中以可见方式创建组件。

此时此刻，读者可能会有种站在丁字路口的感觉，因为必须决定使用 MXML 还是 ActionScript。事实上，还有兼容这两种方式的第三条路可走，这就是隐藏代码。

使用隐藏代码技术

MXML自定义组件不能重写父类的函数，而在Flex Builder中看不到ActionScript组件的直观效果。但使用代码隐藏技术可以发挥这两种方式的长处。

首先创建一个典型的ActionScript自定义组件(像17.8节一样)。然后再创建一个MXML自定义组件，让它扩展前面的ActionScript组件(像扩展其他组件一样)。

代码清单17-21所示为如何使用代码隐藏技术，基于代码清单17-20中的cbStatesScripted.as创建一个MXML组件。

代码清单17-21 cbStatesFromScripted.mxml：使用代码隐藏技术

```
<?xml version="1.0" encoding="utf-8"?>
<local:cbStatesScripted xmlns:mx="http://www.adobe.com/2006/mxml"
   xmlns:local="*"/>
```

仅此而已！如有必要，还可以向这个MXML组件中添加其他代码(如样式)，当然，这完全取决于实现特定逻辑的位置。

在掌握了如何创建自定义组件，谋划自定义组件的结构，以及与自定义组件交换数据之后，还有一个不容忽视的问题——自定义事件。Flex 是一个事件驱动的开发环境，因此必须学会利用事件与自定义组件沟通。

17.8　自定义事件

自定义组件继承自另一个组件，所以父组件的所有事件(creationComplete、change 等)都是可用的。不过，自定义组件也可以分派自己的事件，以便其他组件通过侦听与它实现互动。

代码清单 17-22 和代码清单 17-23 做了一个试验，定义了一个名叫 invalidState 的事件，该事件会在调用代码为自定义组件设置不存在的美国州名时分派。

代码清单17-22　cbStates.mxml：分派invalidState事件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml"
         creationComplete="selectedValue=selectedItem.stateCode"
         change="selectedValue=selectedItem.stateCode">
 <mx:Metadata>
  [Event(name="invalidState")]
 </mx:Metadata>

<mx:Script>
   <![CDATA[
   [Bindable]
   private var _selectedValue:String="";

  [Bindable]
   public function get selectedValue():String
   {
    return _selectedValue;
   }

   public function set selectedValue(statecode:String):void
   {
      var isFound:Boolean=false;
      _selectedValue = statecode;
      for(var x:int=0;x<this.dataProvider.length;x++)
       if(this.dataProvider[x].stateCode == _selectedValue)
       {
        this.selectedIndex = x;
        isFound=true;
       }
      if(!isFound)
        dispatchEvent(new Event("invalidState"));
    }
   ]]>
  </mx:Script>
  <mx:dataProvider>
   <mx:Object stateCode="AK" label="Alaska"/>
   <mx:Object stateCode="AL" label="Alabama"/>
   . . .
  </mx:dataProvider>
</mx:ComboBox>
```

❶ 定义事件

❷ 分派事件

首先使用 Metadata 标签定义这个事件❶。告诉 Flex 编译器和 Flex Builder，当前组件可能会分派这个事件。然后使用 dispatchEvent 函数分派这个事件❷。

代码清单17-23 HelloWorld.mxml：捕获invalidState事件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white" xmlns:local="*">
  <mx:Script>
     <![CDATA[
     import mx.controls.Alert;
     public function invalidStateHandler():void
     {
       mx.controls.Alert.show("Invalid State Specified");
     }
     ]]>
  </mx:Script>
  <local:cbStates id="myStates" invalidState="invalidStateHandler()"/>
  <mx:Button click="myStates.selectedValue='BB'"
          label="Set to fake State"/>
</mx:Application>
```

处理事件的函数 ❶

这里的调用代码中设置了一个处理事件的函数❶。

运行以上代码将看到如图 17-10 所示的弹出式对话框。

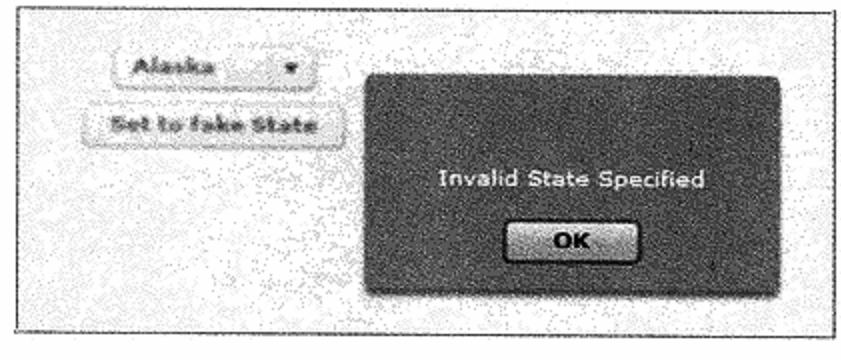


图 17-10 捕获到自定义事件后，会出现弹出式对话框

非常好，不必明确地检查某个属性或函数来确定事件是否发生。只要告诉组件“在它发生时告诉我”，Flex 的事件机制就会严阵以待。

传递事件对象

必要的情况下，可以传递分派事件时生成的事件对象。为此，需要把 Metadata 标签修改为如下所示。

```
<mx:Metadata>
[Event(name="invalidState", type="flash.events.Event")]
</mx:Metadata>
```

为利用这个事件对象，可以把它传递给调用代码的事件处理函数。

```
<mx:Script>
  <![CDATA[
    import mx.controls.Alert;
    public function invalidStateHandler(evt:Event):void
    {
      mx.controls.Alert.show("Invalid State Specified");
    }
  ]]>
</mx:Script>
<local:cbStates id="myStates" invalidState="invalidStateHandler(event)"/>
```

为了简单起见，我们把所有资源都保存在一个目录中。但在真实的项目开发中，可能必须要以某种目录结构来组织自定义组件。为了实现这一点，就要用到名称空间，以方便对组件的引用。

17.9　名称空间和结构

本章介绍的最后一个知识点是在哪里保存自定义组件。为了简单起见，本章示例把所有文件都保存在同一个目录中，但 Web 上的许多例子都使用了以子目录来命名的组件。在构建真正的应用程序时，必须设计一个有意义的文件夹结构。

本节我们讨论设计这个文件夹结构时应该重点考虑的问题，还有 Flex 为此提供的相关机制。

17.9.1　mx 名称空间

名称空间是一种简单的机制，它告诉 Flex 要使用哪个组件。例如，所有内置组件都有以下代码中指定的名称空间。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"/>
```

其中，xmlns 告诉 Flex 编译器，当遇到前缀为 mx 的组件(如<mx:VBox/>)时，要使用 Flex 内置的组件。为什么还要加前缀呢？因为没有谁可以阻止开发人员创建自己的 VBox 组件，(如果这种情况发生)Flex 必须知道要引用哪一个。

17.9.2　local 名称空间

在前面的示例中，只要引用自定义组件，我们都会加上 local 前缀。例如：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:local="*">
  <local:cbStates/>
</mx:Application>
```

这里的 local 就是一个名称空间，而 xmlns:local="*"告诉 Flex，如果组件前缀为 local，则该组件就在当前目录中。如果 Flex Builder 的默认行为是不指定名称空间，就在当前目录中查找，那么在引用当前目录中的组件时，可以不指定 local 前缀。

17.9.3　目录结构与名称空间

Web 上的很多例子都把自定义组件放在名为 components 的子目录中，如图 17-11 所示。

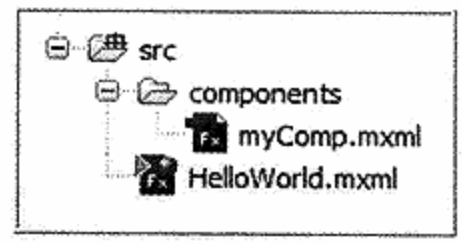


图 17-11　应该把组件放在适当的文件夹结构中

如代码清单 17-24 所示，可以将名称空间 comp 映射到相应的文件夹，并调用其中的组件。

代码清单17-24　通过子目录名称空间调用组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   xmlns:comp="components.*">
  <comp:myComp/>

</mx:Application>
```

文件夹结构可以达到任意深度，用点号隔开子目录。例如， components/userinterface/widgets

要映射为 components.userinterface.widgets。

文件夹结构用于组织组件，而名称空间则用于映射和引用它们。不过，有一个显而易见的问题：如果有人也和你使用了一样的文件夹结构怎么办？有一个业界人士普遍接受的标准做法，就是使用反向域名(reverse-domain)表示法。

反向域名语法

在实际的项目开发中，可能下载使用第三方组件，也可能自己开发组件。如果有很多人创建相同的目录结构或组件名，就会导致冲突。

为避免这种冲突，我们推荐采用反向域名表示法。所谓反向域名表示法，就是使用公司的域名作为文件夹结构，但域名书写方向相反。

假设我们为 dopejam.com 开发了一些 UI 组件，那么就可以把这些组件打包为 com.dopejam. components.ui。还有其他一些流行的命名方式，例如在路径中包含应用程序名称(如果公司同时开发了多个应用程序，而这些应用程序具有公共的结构，就可以采用这种命名方式加以区分)。

名称空间和目录结构不仅适用于 MXML，也可以作为 ActionScript 包特性的一部分。

17.9.4 ActionScript 中的名称空间

在 ActionScript 中，可以把目录结构整合到包声明中。如果不指定名称空间，相当于指定了本地名称空间；但如果把 ActionScript 自定义组件保存到 components 子目录中，应该像下面这样更新包名。

```
package components
{
  import mx.controls.ComboBox;
  public class widget extends ComboBox
  {
    . . .
  }
}
```

如图 17-12 所示，以上代码应该被保存在 widget.as 文件中。

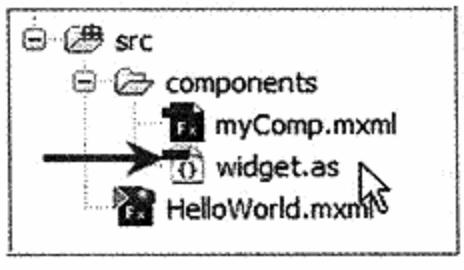


图 17-12 widget.as 组件使用了包组件名称空间

注意，包的命名必须与目录结构完全一致，否则 Flex Builder 无法找到组件。

17.10　小结

随着 Flex 应用程序规模的增大，设计自定义组件的时间必然相应增加。因此，Flex 开发人员应该尽可能打下创建自定义组件的坚实基础。

所有视觉组件都扩展自顶级界面对象 UIComponent。因此，我们创建的自定义组件，同样也扩展自 UIComponent。通过这种继承关系，自定义组件可以从父组件那里获得所有事件、样式、属性和函数。

创建自定义组件的关键在于封装，应该通过减少自定义组件对应用程序工作方式的了解程度确保其可以重用；而自定义组件本身也应该尽可能保持独立，以便在多个应用程序中使用。如果你创建的自定义组件不关心会被谁调用以及调用者的目的，而且调用者也不用关心自定义组件的内部实现，我们就要祝贺你了！

第 18 章将以本章内容为线索，继续探讨如何在多个应用程序间共享资源(如自定义组件)，以减少冗余代码。

第 18 章 Flex 中的高级可重用性

18

本章内容：

- 运行时共享库
- 共享的可重用性
- 模块 API
- 添加补丁
- 重构

前面我们学习了 OOP 的基本知识以及如何创建自定义类，也讨论了怎样开发封装功能的实用组件。本章，我们探索 Flex 支持的高级可重用特性。

什么是可重用性？大致来说，可重用性是指在相同或不同应用程序中重用某些代码块的能力。可重用性并不是 Flex 中独有的概念，任何语言都有实现该特性的机制。可重用程度越高，基于已有代码开发新应用程序的速度就越快。从维护的角度看，也可以更快地实施更改。在 Flex 开发中，提高可重用性能够改善应用程序的性能。

前几章已经涉及一些常见的可重用技术，包括使用自定义类和自定义组件。Flex 还支持一个高级的可重用特性——RSL(Runtime Shared Library，运行时共享库)。学习完本章之后，读者可以为进一步理解 RSL 及其他 Flex 支持的可重用能力打下良好基础。

Flex 应用程序(将被编译为 SWF 文件)可能会包含一些高级元素，如 SWC 文件、RSL 和模块。

有效利用这些高级元素不仅可以提高加载应用程序的速度，而且能够带来长期维护中的便利。本章将向读者介绍这些元素的基本概念，以及如何在 Flex 开发中有效利用它们。

首先介绍主要的元素——SWC 文件。

18.1 SWC 文件

不要把 SWC(音“swick”)文件和 SWF(音“swiff”)搞混了。如果用 Java 来打比方，SWF 文件就是.class 文件(可执行文件)，而 SWC 文件则类似于.jar 文件(包含支持项的归档文件)。

SWC 文件是包含类、资产(asset)和组件的容器，你将这些资源放入 SWC 文件供重用。开发中使用的大多数第三方组件和库都是 SWC 文件，因为 SWC 文件比.as 文件更便于分发。

用 WinZip 或其他压缩程序打开 SWC 文件，会在其中发现一个 library.swf 和一个 catalog.xml 文件。SWF 文件中包含着所有代码/资产，而 catalog.xml 文件负责告诉编译器到哪里查找类或资产。

SWC 最常用于打包在运行时共享的主题和代码。稍后我们会简单介绍在开发主题时使用 SWC 文件的好处。

下面我们先看一看如何在运行时共享库中使用 SWC 文件。

18.2 运行时共享库

RSL 是指在运行期间由多个应用程序或域加载的代码(对域来说，存在某些限制)。假设我们创建了一个可供 20 个应用程序共享的新类，但没有必要让 20 个应用程序都编译这个新类，从而多占用 300KB 空间。让应用程序在运行时加载这个类会节省下这 300KB。

后面也将讨论到，在发布应用程序的下一个主版本之前，以这种方式提供补丁也是一种简捷的方式。而只有在部署大型 Flex 应用程序时(需要数量与此相对的组件和类)，才会看到性能的明显提升。

我们要讨论 3 种 RSL。

- **标准型**：由浏览器缓存且只对一个域有效。
- **跨域型**：由浏览器缓存，但可以被其他域访问。
- **框架型**：由 Flash Player 缓存。

听到标准型、跨域型、框架型 RSL 这些术语，恐怕有读者会感到太复杂了。但实际上，这些概念都非常简单。首先来看一看标准 RSL。

18.2.1 标准 RSL

为理解如何使用标准 RSL，最好是创建一个简单的类，然后把它放到一个 RSL 中，以便同一个域中的其他应用程序使用。第一步是创建一个新 Flex 库项目。

(1) 在 Flex Builder 中，选择 File | New | Flex Library Project，创建一个新 Flex 库项目。

(2) 将该项目命名为 MyRSL。

(3) 单击 Finish 按钮。

这样我们就有了一个空的库项目。如果已经定义好要转换为 SWC 文件的类，直接把它们拖到库文件夹中就可以了；Flex Builder 会把它们编译成 SWC 文件。

不过，这里我们还是要创建一个类。

(1) 右击项目文件夹，创建一个新类。

(2) 为该类指定 com.gunix.myrsl 包并将其命名为 MyClass。

(3) 扩展 Button 类(将 Button 指定为超类)。

注意：

前面我们曾讨论过对象和类，因此读者对创建新类应该不会陌生。如果有什么不清楚的话，参见第 16 章。

在这个类中，我们为按钮添加一个小标签。代码清单 18-1 所示为这个类的完整代码。

代码清单18-1　在RSL中使用一个类

```
package com.gunix.myrsl
{
  import mx.controls.Button;

  public class MyClass extends Button
  {
    public function MyClass()
    {
      super();
      this.label = 'Hey Look, I am Dynamically Loaded';
    }
  }
}
```

为按钮创建一个新标签

创建了新类后，通常要清理一下项目。在 Flex Builder 中清理项目的步骤如下。

(1) 单击 Flex Builder 工具栏中的 Project 菜单项。

(2) 选择 Clean 项。

(3) 选择 Clean 时，会重新编译项目，并在/bin 文件夹中创建一个新的 SWC 文件以便部署。

这样就创建了第一个 SWC 文件。接下来，我们看一看怎样在多个项目中使用该文件。

18.2.2　在 Flex 应用程序中使用 RSL

我们刚才创建了一个类并将其编译为 SWC 文件。这样，就可以从正在构建的任何应用程序访问这个 SWC 文件及其中包含的类。

在多个项目都会使用这种共享类的情况下，把 SWC 文件集中放到一个集中文件夹(如 C:\Flex\SWC\)中会比较方便。当然，这个文件夹里的 SWC 文件不仅仅有我们创建的，还可能包括下载的。

一个 SWC 就是一个共享库，可以通过静态或动态方式引用。在静态方式下，要把 SWC 中的一切资源都嵌入到最终的 SWF 文件中。从应用程序开发角度看，如果类和资产(如图像)要在多个应用程序中使用，这种方式比较方便。但是，这种方式会导致 SWF 文件比较大。

动态链接就是把 SWC 作为 RSL 的方式(也是更值得推荐的方式)，即在运行期间根据需要加载类。下面抛开 SWC 文件不管，先来创建一个使用它的项目。

1. 创建使用 RSL 的项目

现在不用管 SWC 文件放在哪里。我们先创建一个 Flex 项目，然后通过配置，在该项目中使用相应的 RSL。请读者按照下列步骤操作。

(1) 在 Flex Builder 中，创建一个名为 RSLTest 的新项目。

(2) 在项目 Navigator 窗格中，右击 RSLTest 项目并选择 Properties。

(3) 选择 Flex Build Path(见图 18-1)。

(4) 单击 Library Path 标签页，查看加载到当前 Flex 应用程序中的外部库。

(5) 在使用 RSL 时，首先要做的一件事，就是让整个 Flex 框架都作为 RSL 使用。为此，在 Properties for RSLTest 对话框中，把 Framework Linkage 项修改为 Runtime Shared Library。

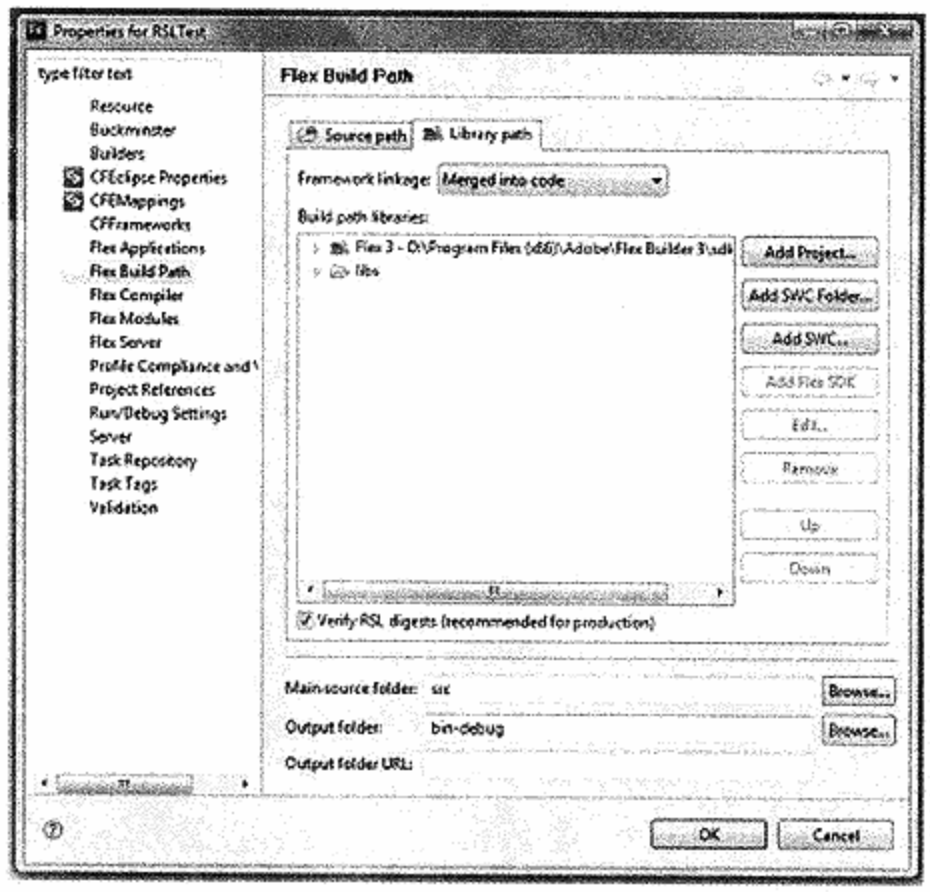


图 18-1 在 SWC 构建对话框中选择 Flex Build Path

然后会创建包含 Flex 框架的一个 SWZ 文件和一个 SWF 文件。SWZ 文件是经由 Adobe SWF 数字签名的 SWF 文件，可以被缓存到 Flash Player 中。当最终用户第二次打开同一个 Flex 应用程序时，能够显著缩短加载时间(因为 Flash Player 不必重新加载 Flex 框架了)。

不过，Flex Builder 为什么要在创建了 SWZ 文件后，还创建一个 SWF 文件呢？因为 Flash Player r115 版之后才支持框架缓存功能。虽然读者在看到本书时，r115 的安装率已经高达 95%，但 Flex 仍然会创建相应的 SWF 文件，以便在最终用户没有安装 r115 的情况下，由浏览器缓存这个 SWF 文件，将其作为一个标准 RSL 或一个跨域 RSL 使用。请注意，用户随时可能清除浏览器缓存；如果这样的话，则下一次还要重新缓存框架。

接下来，我们把 MyRSL 项目转换为 RSL。

2. 创建 RSL

在把 Flex 框架设置为 RSL 后，必须再把 MyRSL 项目也设置为 RSL。首先必须把它导入到库中。

(1) 在 Properties for RslTest 对话框中单击 Add SWC。

(2) 找到并选中相应路径下的 SWC 文件。

(3) 单击 OK 按钮。

之后会看到 MyRSL SWC 文件被添加到应用程序的库中，但此时它还不是 RSL。在 Build path libraries 窗格中，如果扩展 MyRSL 树节点，会发现 Link Type(链接类型)为 Merged into code(合并到代码中)。我们不希望把库合并到代码中，因为这相当于在应用程序中(通过静态链接方式)编译 MyRSL。

在此我们把 SWC 作为一个可以动态链接的 RSL。因而，接着上面最后一步，继承执行如下操作：

(1) 单击 Link Type，再单击 Edit 按钮(见图 18-2)。

(2) 在打开的 Library Path Item Options 对话框中，勾选 Use same linkage as framework 复选框。这样就会为 MyRSL 赋予与 Flex 框架相同的链接方式，即让 MyRSL 也可以动态加载，因而成为了一个 RSL。

(3) Deployment Path/URL 文本框中显示的是 MyRSL.swf。这就是将被浏览器缓存，并由所有使用该类的应用程序共享的文件。

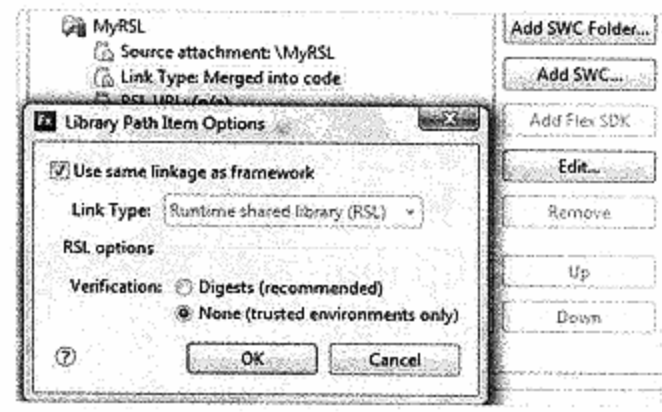


图18-2　选择Link Type，然后单击Edit按钮来修改SWC库的链接方式

(4) 单击 OK 按钮。

(5) 再单击 Library 对话框中的 OK 按钮。Flex Builder 会以新属性编译应用程序。

这样就链接了我们的自定义类，并设置为由浏览器缓存它，因此同一个域中的所有应用程序都可以使用它。

针对每个要使用这个类的应用程序，都要执行相同的设置操作。但在把应用程序上传到域中时，则只需：

- 一个 MyRSL.swf 文件
- 一个 Framework.swz 文件
- 一个 Framwork.swf 文件(每个域一个)

要保证 RSL 的路径在每个应用程序中都是相同的，以便 Flash Player 知道检查哪里来确定文件是否有缓存；否则就需要加载。在有缓存的情况下，由于 Flash Player 无需每次都加载 200KB～300KB 的 Flex 框架及自定义类，因此会明显减少应用程序的加载时间。

下一节介绍有效组织 SWF 文件的机制——模块。

18.3　开发模块化的 Flex 应用程序

Flex 3 新增的另一个特性，就是可以使用模块来开发 Flex 应用程序。通过使用模块，可以封装应用程序的不同部分。而封装的好处在于，模块之间可以互不依赖。这样在应用程序中包含故障时，就更容易查找和解决问题。

模块化编程与组件驱动编程有一个主要区别，模块在运行时加载和卸载，而不是被编译在 SWF 文件中。这样 SWF 文件的大小会大幅缩小，Flex 应用程序的结构也会更加合理，

而且还可以在必要时在其他应用程序中重用模块。

18.3.1 组件与模块

如前所述，组件与模块的主要区别是组件会被编译到 Flex 应用程序中，而模块不会。但是通过模块仍然可以实现与使用组件相同的任务。

虽然程序由相同的代码构成，但模块以<mx:Module/>标签开始，而组件则以其扩展类的标签开始。在<mx:Module/>标签中，可以随意复制粘贴组件代码，最终的模块行为实际上跟组件一样。

当 Flex 显示列表认为需要展示模块时，才会加载模块。换句话说，如果模块表示另一种视图状态(有关视图状态的内容，参见第 13 章)，那么在需要展示该视图状态之前，不会从外部 SWF 文件加载模块。

但是如果使用组件，则无论是否需要展示，组件都将被嵌入在 Flex 应用程序中。因此，组件会增大 SWF 文件。

18.3.2 创建简单的模块

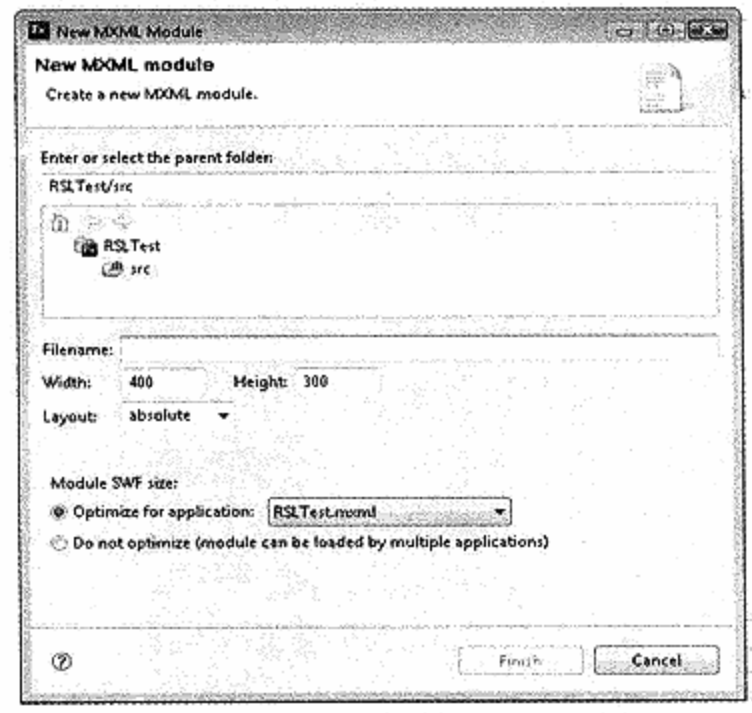


图 18-3 创建一个新 MXML 模块

下面我们就来创建一个包含按钮的简单模块。读者可以创建一个新 Flex 项目或者使用前面创建的项目，然后按照下列步骤操作。

(1) 在 Flex Builder 菜单栏中，选择 File | New | MXML Module，打开如图 18-3 所示的对话框。

(2) 输入想要创建的模块的名称，同时指定是否想在其他应用程序中使用该模块。

如果选择 Optimize for application，则说明只想在当前 Flex 应用程序中使用这个模块，Flex 会删除模块中不必要的类并加以优化，以减小文件尺寸，加快加载速度。如果选择 Do not optimize，即要在其他应用程序中使用这个模块，Flex 会为这个模块添加独立于应用程序的特性，让它可以在别的项目中使用。

在这个例子中，我们选择 Optimize for application。

(3) 单击 Finish 按钮。

创建模块后，可以随意向其中添加按钮、面板和画布，以及任何实现应用特性或用户界面所需的组件。

读者可能想知道怎样使用应用程序模块中的数据。Flex 为此给出了一个名为 Module API 的解决方案。

18.3.3 通过模块 API 以 MXML 方式加载模块

既然已经创建了模块，那么怎样在 Flex 应用程序中使用它呢？有读者可能会本能地到

Flex Builder 组件库的 Custom 文件夹中找到自定义模块，然后把它添加到应用程序中。

这种方式可行；但是以这种方式使用模块与使用组件没有什么不同，对应用程序的加载时间或大小没有任何影响。那么应该怎样在 Flex 中加载模块呢？

有两种方式。比较简单的方式是使用<mx:ModuleLoader/>标签并为其 url 属性指定模块的路径。

```
<mx:ModuleLoader url="MyModule.swf" x="100" y="100" width="356"
   height="274" />
```

这样当调用<ModuleLoader/>标签时，就会创建一个 MyModule.swf 模块的新实例，并将其加载到 Flex 中。对于刚开始利用模块的人而言，这称得上是一个较好的解决方案。但是，如果希望对模块施加更多控制或在不需要时主动卸载它们，可以选择 ActionScript 方式。

18.3.4　通过 ActionScript 加载和卸载模块

在介绍使用 ActionScript 加载和卸载模块之前，需要先了解模块 API 的工作原理。模块 API 位于 mx.modules 包中，总共是 1 个接口 4 个类。读者不必知道这些类和接口的细节，但可以通过表 18-1 了解它们的大概情况。

表 18-1　模块 API 包

名　称	类　型	说　　明
IModuleInfo	接口	作为当前使用模块的句柄
Module	类	MXML 模块的基类
ModuleBase	类	ActionScript 3 模块的基类
ModuleLoader	类	类似于 SWFLoader，但可以与当前模块交互
ModuleManager	类	管理 Flex 应用程序中模块的加载与卸载

为加载和卸载模块，需要使用 ModuleManager 类管理 ModuleLoader 标签。首先在 Flex 应用程序中创建一个新 ModuleLoader 标签，将其 id 属性指定为 MyModule，如代码清单 18-2 所示。

代码清单18-2　使用MXML的新模块

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            layout="absolute">
   <mx:ModuleLoader id="MyModule" url="MyModule.swf" x="100" y="100"
      width="356" height="274" />
</mx:Application>
```

运行这个应用程序，确认已经加载了新模块；读者可能会注意到，加载模块时有 1 秒钟左右的延迟，具体时间长短取决于模块大小及其内容。

然后创建名为 unloadBtn 的新按钮，并使用标签属性为它指定一个适当的标题，如代码清单 18-3 所示。

代码清单18-3 使用MXML卸载模块

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute" >
   <mx:ModuleLoader id="MyModule" url="MyModule.swf" x="100"
      y="100" width="356" height="274" />
   <mx:Button x="348" y="382" label="Unload Module" id="unloadBtn"/>
</mx:Application>
```

现在创建一个<mx:Script/>标签，以编写 ActionScript。再定义一个名为 unloadMyModule()的函数。

在这个函数中调用 MyModule 的 unloadModule()函数(ModuleLoader 的一个函数)，如代码清单 18-4 所示。

代码清单18-4 使用ActionScript卸载模块

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
           layout="absolute" >
<mx:Script>
 <![CDATA[
  private function unloadMyModule():void
  {
     MyModule.unloadModule();
  }
 ]]>
</mx:Script>
 <mx:ModuleLoader url="MyModule.swf" x="100" y="100" width="356"
            height="274"  id="MyModule"/>
   <mx:Button x="348" y="382" label="Unload Module" id="unloadBtn"
          click="unloadMyModule()"/>
</mx:Application>
```

在调用 unloadModule()函数时，Flex 显示列表会从 Flash Player 渲染器列表中删除该模块，然后再将其从内存中完全删除。再加载这个模块也很简单，调用 MyModule.loadModule()，Flex 就会加载它。

模块是开发成规模(无论大规模还是小规模)应用程序的最有效方式。从长期来讲，无论是增加还是减少功能，模块都具有非常高的效率。毕竟，模块不依赖于应用程序代码，也不会导致故障。

18.3.5 模块的利弊

模块似乎好得难以置信了。然而，跟现实中的事物一样，模块也有利有弊。图 18-4 所示为模块的两个方面及相应的影响。

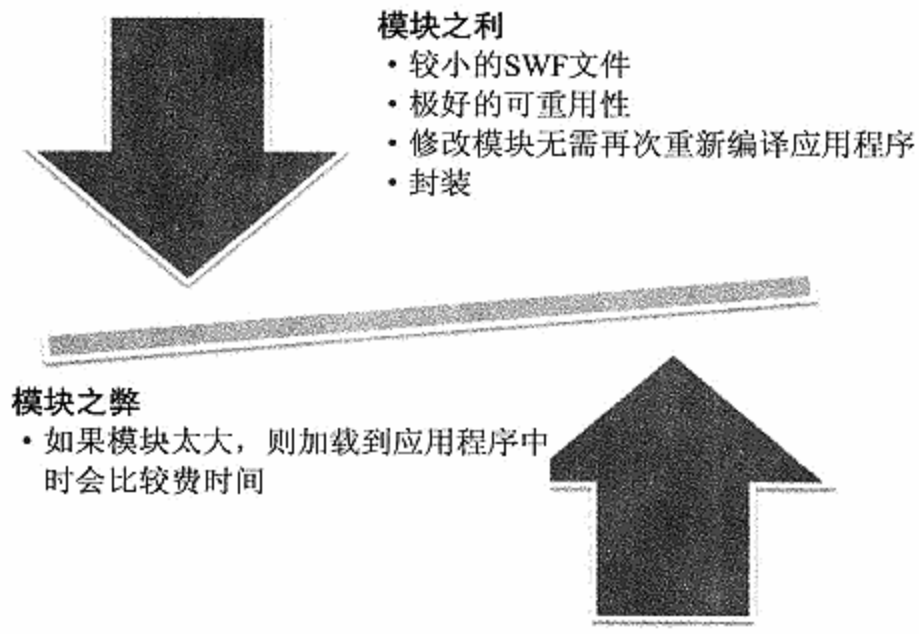


图18-4　模块的利弊

如图 18-4 所示，使用模块的好处大大超过了使用它们的负面影响。下一节，我们将探讨如何利用前面所学的知识为应用程序打补丁。

18.4　在 Flex 应用程序的生命周期中添加补丁

对于大型应用程序，有时候必须要在下一个主版本发布前为它添加补丁。不用编译应用程序，不必在服务器上重新部署整个应用程序，为应用程序打补丁的最简便方式就是利用强大的 RSL。

通过前面的学习，我们知道 RSL 会在运行时以 SWF 形式被加载并缓存到浏览器中。而且只有在清除浏览器缓存或文件有更新时，才需要重新缓存。利用这一点，我们可以创建相应的 SWC 文件，让它作为应用程序运行期间可以检查更新的一个补丁。但是在为某个功能打补丁时，通常涉及替换或更新类，而这并不是 SWC 的特长。不用担心，使用我们随后介绍的一个技巧，可以得到相同的结果。

使用 SWC 更新和替换类

Flex 编译器会按照从上到下的顺序，依次加载 RSL，这是一个非常有用的特性。假设应用程序中的某个类包含两个函数，而我们希望添加第三个函数，同时更新原有函数以便它们使用新函数。为此，可以将类代码复制粘贴到 Flex 库项目中，按需要加以修改，再基于修改结果创建一个 patches.swc 文件。

接着打开应用程序的 Properties 窗口并编辑 Flex Build Path 对话框(见图 18-5)以便添加 patches.swc 文件。现在，将此文件移动到列表顶部，然后构建应用程序。

通过把文件移到列表顶部，可以让 Flex 编译器先加载 patches.swc 中的类，然后抛弃后来遇到的同名类。这是在发布新版本之前临时解决问题的一个不错方式。

在开发过程中，有时候也需要重新命名代码中的某些标识符。下一节，我们学习如何在不复制粘贴或重写代码的前提下实现重命名。

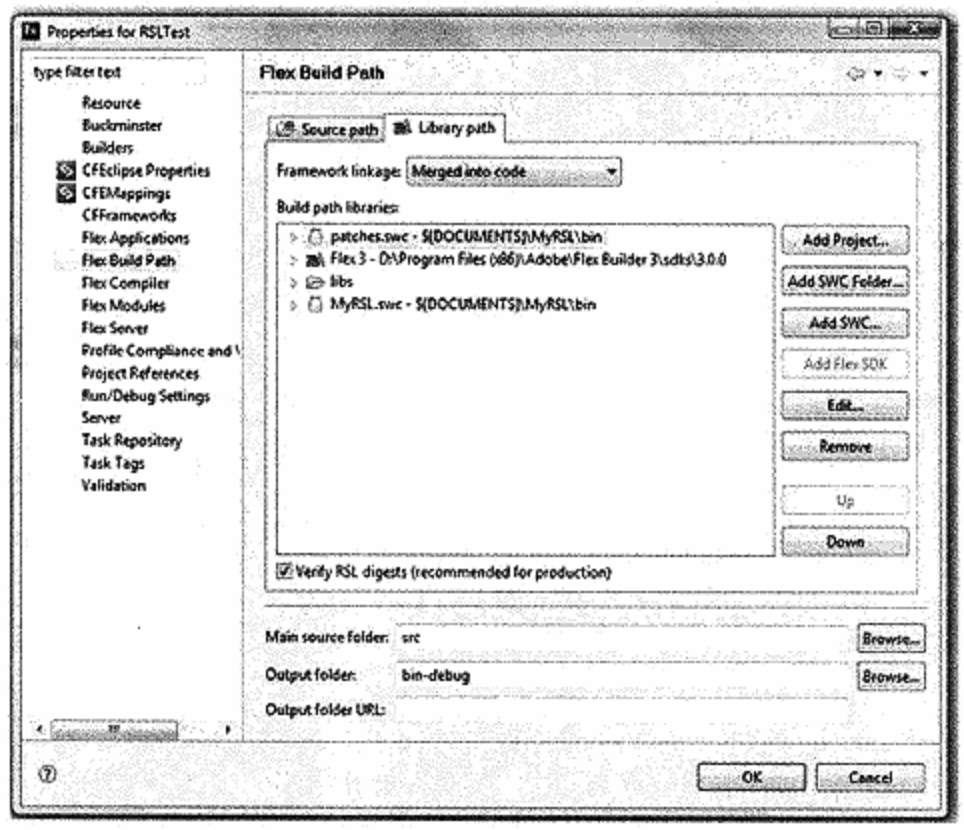


图 18-5 在 Flex Build Path 对话框中添加作为 RSL 的 patches.swc

18.5 重构

重构是 Flex Builder 3 中增加的一项重要特性，也是在应用程序中重用代码的一种好办法。

任何人都不可能回避一个现实：项目开发往往从小型概念验证做起，或者说一开始只能搭建起应用程序框架，然后再随时间推移逐步丰富其内容。在此期间，有可能出现命名不当或用途发生变化的函数，而现在，我们想把这些问题都处理掉。

为此，如果通过手工找到每个问题并逐个加以修改，一定会浪费大量宝贵时间，而且往往效率低下。全局性的查找替换可能会奏效，但这种方式并不绝对可靠，经常还会制造出新问题。

那么该怎么办？重构！让 Flex Builder 来帮助我们实现重构——重命名一切标识符，并且一切完好。

代码清单 18-5 所示为一个命名有问题的函数。

代码清单18-5 不适当的方法名

```
package com.a.b.c
{
  public class Alphabet
  {
    public function Alphabet()
    {}
    public function badMethodName():void
    {}

      }
}
```

整个应用程序调用了这个函数 100 多次。如果是在 Flex Builder 2 中，恐怕必须要手工搜索代码并把每一个 badMethodName()都重命名为 goodMethodName()。

但是通过使用 Flex Builder 3 强大的重构特性，则可以这样来实现重命名。

(1) 单击函数名，右击鼠标，选择 Refactor。

(2) Flex Builder 会显示重构提示框。输入新函数名。

(3) 单击 Preview 按钮，Flex Builder 会扫描代码并显示将要更新的所有引用。此时，我们可以辅助 Flex Builder 正确地完成工作。

(4) 单击 OK 按钮。Flex Builder 会搜索代码并重命名函数及对该函数的调用。

读者如果是第一次听说重构，可能会觉得它难以名状或者比较复杂。事实上，重构不过如此，它只是 Flex 为我们提供的一个最简单的工具。

18.6　小结

在开发大型应用程序时，SWF 文件的大小会很快从几百 KB 增加到一两 MB。通过使用 RSL 及模块等技术，可以有效减少 SWF 文件的大小，同时还能使代码在其他应用程序中被更好地重用。

在第 II 部分中，我们学习了提升代码可重用性和应用程序性能的技术。但是无论应用程序的运行速度有多快，我们仍然希望它的外观能够更漂亮、更协调一些。第 19 章的主题是如何增加应用程序外观的吸引力。

第Ⅲ部分

高级应用

从技术上讲，学习完第Ⅱ部分后，读者就拥有了创建完整 Flex 应用程序的能力。第Ⅲ部分将教读者如何创建非同凡响的应用程序，特别是自定义用户体验。

首先将介绍类似 HTML 中 CSS 的样式表，以及包装 Flex 应用程序整体观感的主题(包括转换任何 Flex 组件外观的皮肤)。

在这一部分中，我们还要学习如何添加令人拍案叫绝的效果，让应用程序博得用户的由衷赞叹。这些效果添加起来极其简单，却能给人带来强烈的视觉冲击。

一切围绕用户体验展开。除了本身具有的炫酷天性外，利用 Flex 的拖放特性还可以增加应用程序的易用性。

最后，但并非最不重要的是 Flex 的图表功能，图表支持以多种方式形象地展示数据。

在第Ⅲ部分末尾将讨论 Flex 项目开发周期的最后阶段。其中包括测试、调试及针对应用程序发布所要考虑的各种问题。

第 19 章 自定义体验

19

本章内容：

- 通过样式控制应用程序的外观
- 使用字体
- 嵌入图像和图标
- 为 Flex 组件创建皮肤

开发 Flex 应用程序，想让界面不好看都难——这一点，说起来还真有点酷！许多开发人员其实没有设计背景，但 Flex 应用程序的默认界面却让人觉得它出自设计行家之手。

不用关心界面，就可以把更多时间花在特性和功能上。然而，还是有一个小问题：默认情况下，每个人开发的 Flex 应用程序看起来都一样！本章我们就来探讨如何添加符合你或用户个性的自定义体验。

首先从回顾 Flex 的样式开始。

19.1 样式

有一个好消息：读者如果熟悉 CSS 语法，那么告诉你，这些语法在 Flex 中同样适用。Flex 遵循 CSS 2.0 语法且支持许多常用属性，但除了字体大小和背景颜色外，Flex 还允许我们控制圆角、透明度甚至阴影的角度。

在 Flex 中定义样式及样式属性的方式有 3 种。

- **行内样式**：设置组件的样式属性。
- **本地样式**：把样式定义集中在一个公共区域。
- **外部样式表**：导入包含所有样式的外部文件。

考虑到这些相似性，因此读者学习本章的进度可以快一些。下面我们就来对比一下这几种方式在 Flex 与 HTML/CSS 中的异同。

19.1.1 行内样式

行内样式就是在具体的组件上直接应用样式属性。如果读者用过 HTML，那么这种方式应该是会偶尔为之的。

```
<span style="font-size:14px;font-family:Tahoma;color:#AE0000">
   Inline Style</span>
<textarea style="background-color:#CC0033">Text Area</textarea>
```

在 Flex 中，通过设置组件的样式属性也可以达到相同的目的。

```
<mx:Text text="Inline Style" fontFamily="Tahoma" fontSize="14"
   color="#AE0000"/>
<mx:TextArea backgroundColor="#CC0033" text="Text Area"/>
```

图 19-1 所示为 HTML 和 Flex 中的结果。如图 19-1 所示，结果大致一样，但并不相同。

图19-1 为HTML和Flex应用相同的样式属性，得到的结果会有所差异

差别不算太大，对吗？接下来，为了确定可以使用哪些样式属性，有 3 种简便的技术可供选择。

1. 使用 Flex Builder 的属性建议功能

第一种方式是在 Flex Builder 中键入组件名称。此时，Flex Builder 会自动给出全部可用属性的建议，如图 19-2 所示。

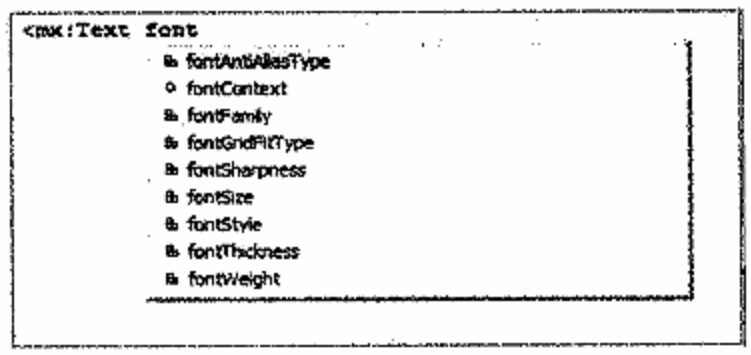


图 19-2 使用 Flex Builder 的自动建议功能查找属性可以节省时间

不过在给出的建议属性列表中，并非只包含样式属性。但通过属性名称旁边的样式图标可以识别哪些是样式属性。

2. 使用设计视图

在 Flex Builder 编辑窗口的上方，有两个标签分别为 Source 和 Design 的按钮。单击 Design 按钮，可以切换到设计视图，然后单击想要添加样式的组件。

右侧 Flex Properties 区域会显示与所选组件有关的全部属性，而通过 Style 窗格可以调整一些样式属性(图 19-3)。

如果要调整所有属性，单击右上角的 Category View 按钮(图 19-4)，会显示所有属性列表(最关键是要注意其中的样式属性)。

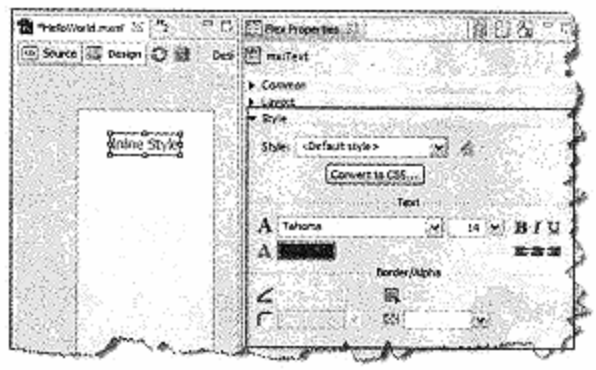


图 19-3　通过 Flex Builder 设计视图可视地调整样式属性

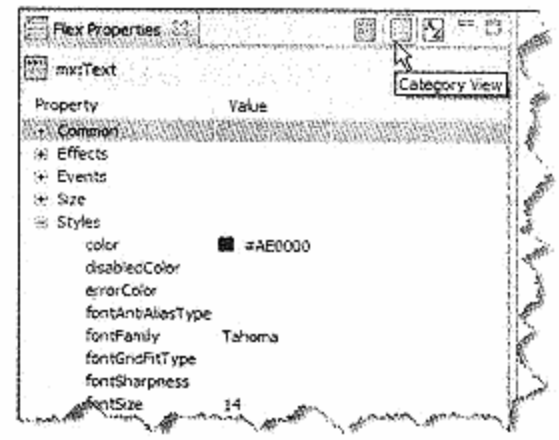


图 19-4　分类视图(Category View)展示了可以调整的所有样式属性

在刚开始使用 Flex 时，这种方式最为实用。而且一旦深入进去，还可以养成利用 API 语言参考的好习惯。

3. 使用 API 语言参考

有些属性建议了期望值的类型。例如，fontSize 期望一个数值。如果想进一步明确这一点，可以如下操作。

(1) 选中 Flex Builder 中的组件名，然后按 Shift+F2(图 19-5)，结果会显示语言参考。

(2) 向下滚动到 Styles 部分。

(3) 单击 Show Inherited Styles 下的链接。这样就会显示可以为当前组件设置的所有样式属性，包括每个属性的期望值类型(图 19-6)。

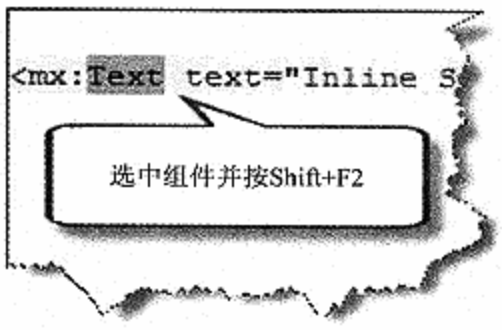


图19-5　选中组件并按Shift+F2可以打开API语言参考

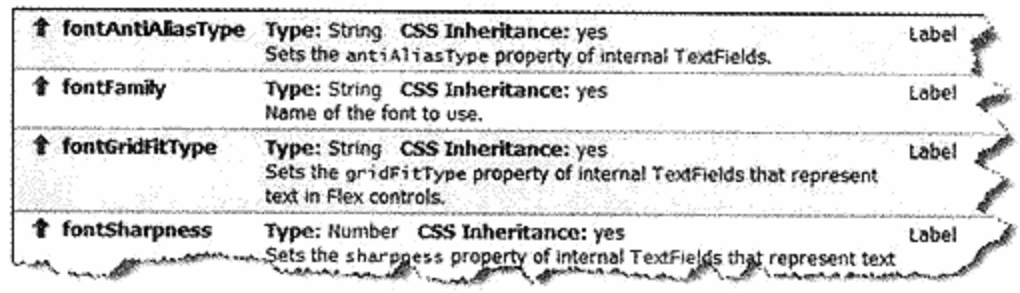


图19-6　API语言参考的Styles部分列出了所有可用的样式属性

无论通过何种方式找到样式属性，使用行内样式都免不了会有优点和缺点。

4. 什么情况下使用行内样式

一般来说，在开发 Flex 应用程序的概念验证阶段，可以使用行内样式。此时，我们只想简单地定义一下外观，或者只想知道某个组件是否具有某些特别的样式属性。如果过度使用这种方式，那么当将来需要修改设计方案时，就要面临在大型应用程序中做大量工作以便更新所有设置的问题。

19.1.2　本地样式定义

本地样式定义比行内样式更实用，因为通过把样式定义与具体的组件分开，能够方便

样式重用。本地样式定义有两种。

- **类型定义**：样式应用于相应类型的所有组件。
- **类定义**：各个组件指定要应用的样式类。

下面将分析每种类型。

1. 类型定义

如果我们想让某个组件的所有实例共享一组样式，那么类型定义最合适。在 HTML 中，需要通过页面中的<style>标签来指定 CSS 类型定义(代码清单 19-1)。

代码清单19-1 在HTML CSS中使用类型定义为文本输入框应用样式

```
<style>
   input
   {
      font-family:Tahoma;
      font-size:14px;
      color:#AA4400;
      background:#CCCCFF;
   }
</style>
<input value="Input 1">
<input value="Input 2">
```

如图 19-7 所示，无需分别为每个输入框指定样式，即可让所有输入框的实例都获得相同的外观。

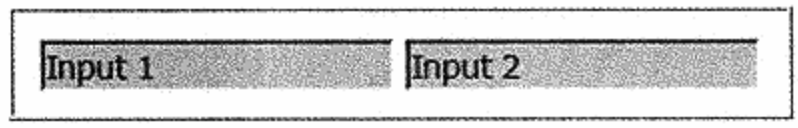


图 19-7 使用 CSS 的类型定义让两个 HTML 文本框具有了相同外观

在 Flex 中，情况几乎一样，如代码清单 19-2 所示。

代码清单19-2 Flex中的类型定义与HTML中的CSS定义几乎相同

```
<mx:Style>
   TextInput
   {
      fontFamily:Tahoma;
      fontSize:14px;
      color:#AA4400;
      backgroundColor:#CCCCFF;
   }
</mx:Style>
<mx:TextInput text="Input 1" />
<mx:TextInput text="Input 2"/>
```

图 19-8 所示为在 Flex 中使用 CSS 类型定义，能够得到与在 HTML 中使用 CSS 类型定义相同的效果——所有 TextInput 实例的外观都一样。

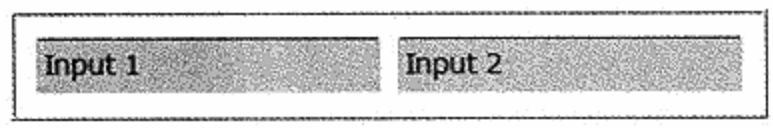


图 19-8 两个 Flex 的 TextInput 组件外观都遵循 CSS 类型定义

如果想根据任意标准为组件应用样式，应该使用类定义。

2. 类定义

类定义的层次更高一些，它并不针对具体组件类型。类定义最适合为特定组件应用上下文相关的样式。例如，想为地址搜索表单中的输入控件应用一种样式，而为用户信息表单中的输入控件应用另一种样式。而且同一个类定义可以应用给不同类型的组件(假设相应的 CSS 属性都能得到支持)。

在 HTML 中，像代码清单 19-3 所示的那样使用类定义。

代码清单19-3　HTML CSS：可以随意使用的类定义

```
<style>
   .coolStuff
   {
      font-family:Tahoma;
      font-size:14px;
      color:#AA4400;
      background:#CCCCFF;
   }
</style>
<!-- Style reused on multiple HTML tags -->
<input   class="coolStuff" value="Input 1">
<textarea class="coolStuff" rows="2" cols="40">Input 2</textarea>
<!-- Same tag, but no style -->
<input value="Input 3">
```

如图 19-9 所示，可以看到类定义只应用给了某些 HTML 元素；更重要的是，可以为不同类型的元素指定相同的样式。

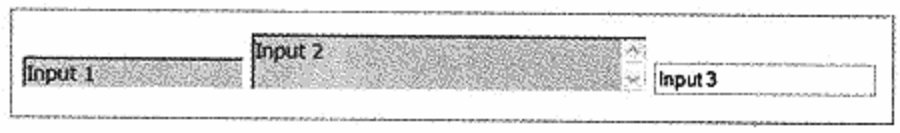


图 19-9　HTML 中 CSS 的类定义可以任意使用

Flex 中的类定义也使用点加类名的格式，如代码清单 19-4 所示。

代码清单19-4　Flex CSS：与HTML中CSS的类定义相同

```
<mx:Style>
   .coolStuff
   {
      fontFamily:Tahoma;
      font-Size:14px;
      color:#AA4400;
      backgroundColor:#CCCCFF;
   }
</mx:Style>
<mx:TextInput text="Input 1" styleName="coolStuff"/>
<mx:TextArea  text="Input 2" styleName="coolStuff"/>
<mx:TextInput text="Input 3"/>
```

如图 19-10 所示，结果也是一样的。使用类定义可以根据需要应用样式，也可以为不同类型的组件应用样式。

图 19-10 使用 CSS 类定义可以为 Flex 中的不同组件应用样式

为了确保向 Flex 中 CSS 的过渡尽可能直截了当，Flex 支持了一些相同的样式属性名称。

3. 确定有效的 CSS 属性

要确定可用的属性名称，可以使用前面介绍的几种技术——Flex Builder 的自动建议特性、设计视图或语言参考。不过，也有一个小陷阱。Flex 支持很多 HTML 中常见的 CSS 样式属性。例如，可以使用 font-Family 而非 fontFamily，使用 font-Size 而非 fontSize。

```
TextInput
{
   font-Family:Tahoma;
   font-Size:18px;
}
```

这在一定程度上便于把基于 HTML 的应用程序转换到 Flex 框架中，因为差不多只要原样复制即可。但在实践中，与内置的 Flex CSS 属性相比，要想搞清楚某个 HTML 中的 CSS 属性是否得到 Flex 的支持并没有那么容易。Flex 支持的 HTML CSS 属性，只是它所支持的所有 CSS 属性的一个较小的子集。而且在使用帮助时，帮助里只会显示 Flex 内置的样式属性。

为了直观地说明问题，代码清单 19-5 所示为可以在 Panel 组件中使用的所有样式定义。

代码清单19-5 在Flex中使用CSS控制特性

```
Panel {
  borderColor: #33cc33;
  borderAlpha: 0.27;
  borderThicknessLeft: 15;
  borderThicknessTop: 6;
  borderThicknessBottom: 21;
  borderThicknessRight: 19;
  roundedBottomCorners: true;
  cornerRadius: 27;
  headerHeight: 26;
  backgroundAlpha: 0.5;
  highlightAlphas: 0.19, 0.52;
  headerColors: #990033, #ccff99;
  footerColors: #000000, #ccff33;
  backgroundColor: #ffcccc;
  shadowDistance: 8;
  shadowDirection: center;
  dropShadowColor: #9933ff;
  titleStyleName: "mypanelTitle";
}

.mypanelTitle {
  color: #000099;
  textAlign: center;
  fontFamily: Palatino;
  fontSize: 14;
  fontWeight: bold;
}
```

这种方式是可行的，但与其他方式一样，它也有自己的适用范围。下面我们就讨论一下何时应该使用本地样式。

4. 什么情况下使用本地样式

对于真正面向用户的应用程序而言，必须采用某些可重用的机制来管理样式。本地样式就是一种不错的手段。但是从另一个角度说，本地样式也可用来定义不太可能重用的一次性样式。

19.1.3　外部样式表

这种方式是指把样式定义保存在一个专门的文件中，然后在应用程序中引用这个文件。这种方式与使用本地样式的效果几乎相同，区别仅在于要把样式保存在 CSS 文件中。

以前面的 HTML 代码为例，可以把样式定义分离到独立的文件中，如代码清单 19-6 所示。

代码清单19-6　my.css——把HTMLCSS分离到独立的文件中

```
.coolStuff
{
   font-family:Tahoma;
   font-size:14px;
   color:#AA4400;
   background:#CCCCFF;
}
```

在 HTML 中使用<link>标签，可以再次引入 CSS，如代码清单 19-7 所示。

代码清单19-7　HelloWorld.html——引用CSS

```
<link rel=stylesheet type="text/css" href="my.css">

<input   class="coolStuff" value="Input 1">
<textarea class="coolStuff" rows="2" cols="40">Input 2</textarea>
```

在 Flex 中，仍然是使用 style 标签，但要指定相应的文件。文件内容如代码清单 19-8 所示。

代码清单19-8　my.css——把CSS分离到独立的文件中

```
.coolStuff
{
   fontFamily:Tahoma;
   font-Size:14px;
   color:#AA4400;
   backgroundColor:#CCCCFF;
}
```

在 Flex 中引入 CSS 也和 HTML 中差不多，只不过要使用<mx:Style>标签而非<style>标签，如代码清单 19-9 所示。

代码清单19-9　HelloWorld.mxml——使用Style标签引入CSS文件

```
<mx:Style source="my.css"/>

<mx:TextInput text="Input 1" styleName="coolStuff"/>
```

这种技术简单方便。下面我们看一看什么时候适合使用外部样式表。

什么情况下使用外部样式表

理想情况下，应该尽最大可能使用外部样式表。这样做可以在更改设计方案时免去编辑 MXML 文件的工作。而且其他开发人员或设计人员就可以只关注 CSS 定义，而无需访问 MXML 文件。另外，还可以在其他应用程序中重用同一 CSS 文件。

19.1.4 样式浏览器

图 19-11 所示的 Style Explorer(样式浏览器)是 Adobe 创建的一个非常棒的资源。这是一个交互性的工具，通过它可以调整常见视觉组件的样式属性，然后动态生成相应的 CSS。样式浏览器的地址为 http://www.adobe.com/devnet/flex/samples/style_explorer/。

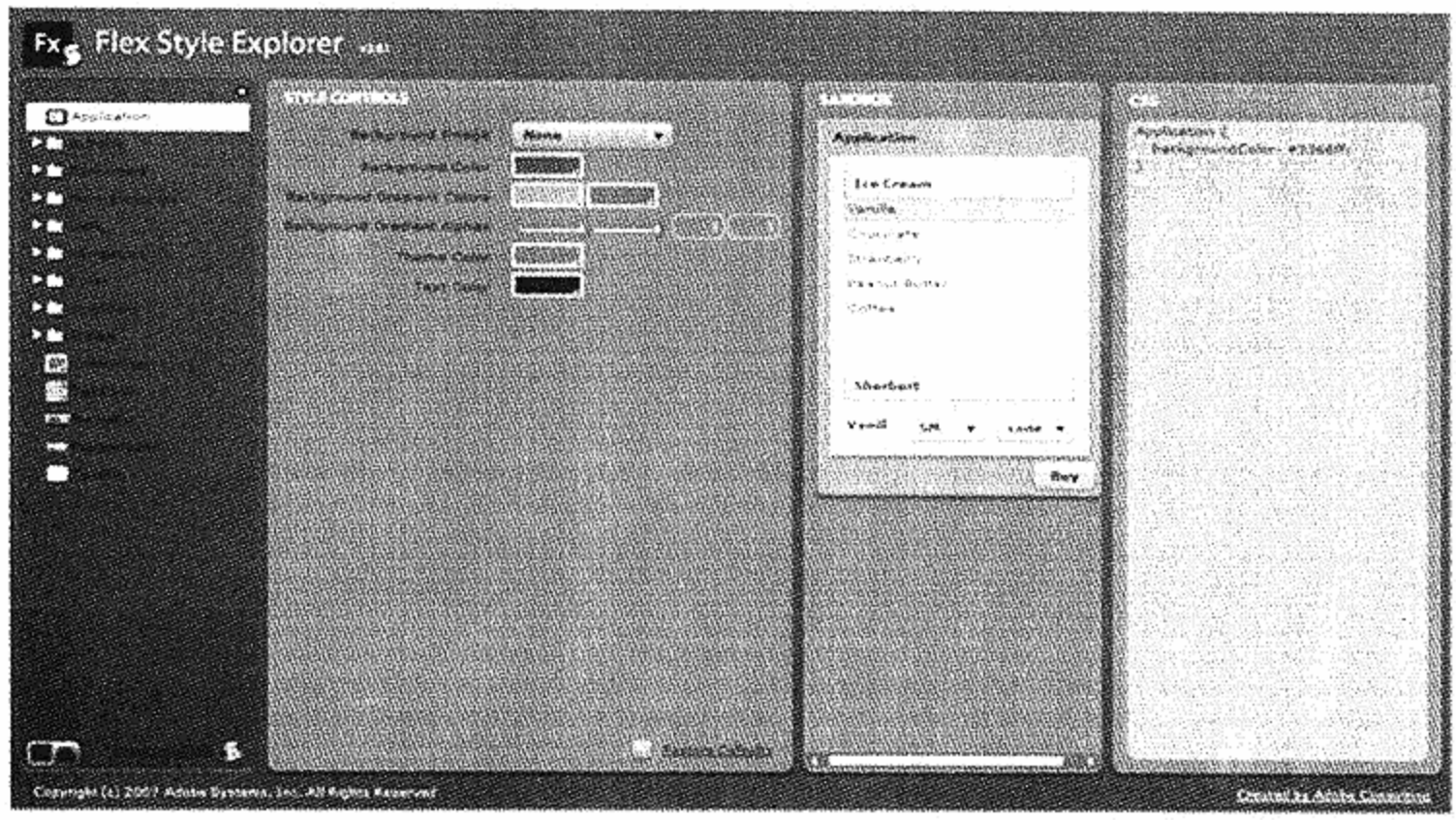


图 19-11 用好 Flex Style Explorer 可以提高应用样式的效率

使用这个工具来验证应用程序的配色及设计方案非常方便，而且可以把样式浏览器生成的 CSS 直接复制粘贴到应用程序中。

19.1.5 处理颜色

可以通过几种不同的格式指定颜色，最常用的是颜色名(Flex 称为 VGA 颜色名)和以十六进制格式表示的 RGB 颜色值，而这也都是 HTML 中的 CSS 支持的格式。

1. 颜色名

颜色名是以纯文本形式表示的一些重要颜色。相对于 HTML 的 CSS 而言，Flex 中可用的颜色名要少一些。

Flex 中的颜色名包括 aqua、black、blue、fuchsia、gray、green、lime、maroon、navy、olive、purple、red、silver、teal、white 和 yellow。可以像代码清单 19-10 所示的那样使用任何样式。

代码清单19-10　在CSS样式定义中使用颜色名

```
<mx:Style>
.myTeal
   {
      backgroundColor:teal;
   }
</mx:Style>
<mx:TextInput text="Input 1" styleName="myTeal"/>
<mx:TextArea text="Input 2" backgroundColor="Teal"/>
```

为了更精确地控制颜色，还可以使用数字表示颜色值。

2. 数字格式的颜色值

在任何领域中，指定颜色值的一种通用方法，是使用它们的数字表示法——RGB 值(Red、Green 和 Blue)。其中，每种颜色的取值范围是 0～255，而 3 种颜色混合起来可以产生数百万种颜色。

Flex 支持两种以数字格式指定颜色值的方法，第一种是流行的十六进制格式。也就是说，不使用逢 10 进一的十进制数(数字范围为 0～9)，而使用逢 16 进一的十六进制数(数字范围为 0～F，其中 F 表示 15)。

无论在 HTML 还是 Flex 支持的 CSS 中，都可以通过如下方式指定颜色。

```
.coolStuff
{
   color:#FF0000;
}
```

这个例子指定的是纯红色(十六进制的 FF 是十进制的 255)。这种方法既可以在样式定义中使用，也可以在行内格式中使用。

特别建议：在使用样式表时，推荐使用#FF0000 表示法，但在使用行内样式时，Adobe 建议使用 0xFF0000 表示法(即以 0x 取代#)。

以下是使用十六进制数指定颜色的两种方法。

```
<mx:TextArea backgroundColor="0xAEAEAE" text="Text Area 1"/>
<mx:TextArea backgroundColor="#AEAEAE" text="Text Area 2"/>
```

读者对第二行中使用的方法可能会更熟悉。

第二种以数字格式指定颜色的方法是使用 RGB 饱和度形式，即通过 0%～100%的百分比值指定颜色强度，如代码清单 19-11 所示。

代码清单19-11　使用RGB饱和度值定义颜色

```
<mx:Style>
   TextArea
   {
      backgroundColor: rgb(100%,0%,0%);
   }
</mx:Style>
```

这是指定颜色的另一种方式。不过这种方式并不怎么常见，因为大多数设计和颜色工具都不以这种格式给出颜色值。

19.1.6 透明度

在一层一层地放置组件(例如，在容器中放置容器)时，可以控制每个组件的透明度。为此需要设置组件的 Alpha 级别，取值范围为 0.00(完全透明)～1.00(完全不透明)。

任何 Alpha 属性的名称中都包含 Alpha，但有的颜色属性也有对应的 Alpha 属性。表 19-1 所示为几个这样的属性。

表 19-1 许多颜色属性支持对应的 Alpha 属性

颜 色 属 性	Alpha 属 性
borderColor	borderAlpha
backgroundColor	backgroundAlpha
fillColors	fillAlphas

通过一个基准 Alpha 属性可以控制整个组件的透明度。代码清单 19-12 和代码清单 19-13 所示为一个几乎透明的组件和一个几乎不透明的组件，结果如图 19-12 和 19-13 所示。

图 19-12 几乎透明的面板

图 19-13 几乎不透明的面板

代码清单19-12 几乎完全透明的面板组件

```
<mx:Panel alpha="0.16" >
   <mx:Label text="Label"/>
   <mx:Text text="Some Text"/>
</mx:Panel>
```

代码清单19-13 几乎完全不透明的面板组件

```
<mx:Panel alpha="0.90" >
   <mx:Label text="Label"/>
   <mx:Text text="Some Text"/>
</mx:Panel>
```

代码清单 19-14 所示为如何充分运用此功能。

代码清单19-14 使用各种Alpha属性

```
<mx:Panel alpha="0.80" borderColor="#33ff99" borderAlpha="0.37"
  backgroundAlpha="0.29" backgroundColor="#ff0000">
   <mx:Label text="Label"/>
   <mx:Text text="Some Text"/>
</mx:Panel>
```

这个示例不仅控制了整个组件的 Alpha，也控制了组件边框和背景的 Alpha。在 HTML

的 CSS 中，无法如此明确地控制的 Alpha，而控制渐变也是如此。

19.1.7　使用渐变

渐变(gradient)是一组颜色之间均匀过渡的效果。这种效果通过 HTML 的 CSS 不容易实现，除非利用某些技巧(例如，拉伸渐变图像或使用多个指定过渡颜色的每个更改的 span/div 标签)。

不过，在 Flex 中实现渐变效果可是小菜一碟。除了要指定两个而非一个颜色值外，指定渐变与指定纯色样式属性一样。许多纯色属性(如 backgroundColor)都有对应的渐变属性。这些属性名称使用复数形式，如 fillColors、headerColors 等。

注意：

Scale Nine 站点中包含使用渐变及完整主题的优秀资源，地址为 http://www.scalenine.com。

要试验各种渐变效果，Flex Style Explorer 是最简便的方式。此外，也可以看一看语言参考中的样式属性。

注意，渐变属性的值是数组，通常包含两个值。而使用数组的简写表示法，可以在行内定义渐变。

```
<mx:Panel headerColors="[0x4D0808, 0x7A93B8]"/>
```

但使用样式定义更加简单，因为只需指定两个值即可。

```
<mx:Style>
   Panel {headerColors: #4D0808,#7A93B8}
</mx:Style>
<mx:Panel label="My Panel"/>
```

在某些情况下，渐变属性可能要用到 4 个值，以便实现常规的填充渐变和鼠标悬停在组件上时的渐变。请看下面这个例子。

```
Button {fillColors: #000000, #006666, #00ff99, #cccc00}
```

最后在指定渐变的同时指定 Alpha 属性，可以控制渐变的透明度。

```
Button
{
  fillAlphas: 0.33, 0.67, 0.75, 0.65;
  fillColors: #000000, #006666, #00ff99, #cccc00;
}
```

目前我们为样式指定的值都使用了硬编码方式。如果应用程序要求动态操作样式，也可以使用编程方式。

19.1.8　以编程方式操作样式

也可以不使用静态样式表，而是通过 ActionScript 在运行期间以编程方式设置样式。

为此要用到任何组件都有的 setStyle()函数。该函数接收两个参数——要设置的样式属性和值。

例如，可以使用 CSS 修改 TextArea 的背景颜色，而使用 ActionScript 也可以做到(见代码清单 19-15)。

代码清单19-15 通过setStyle函数以编程方式设置样式属性

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="0xFFFFFF">

<mx:Script>
   <![CDATA[
      public function changeColor():void
      {
         myTextArea.setStyle("backgroundColor",0xFF0000);
      }
   ]]>
</mx:Script>
<mx:TextArea text="Text Area" id="myTextArea"/>
<mx:Button click="changeColor()" label="Try Me"/>

</mx:Application>
```

图 19-14 所示为修改样式前后的文本区。

图19-14 以编程方式修改TextArea的背景颜色

整个过程很直观。首先找到想要操作的样式，然后将其设置为所需的值。下面我们介绍使用 setStyle()函数的注意事项，其中涉及性能问题。

1. 使用 setStyle()函数的注意事项

如果在大型应用程序中频繁使用 setStyle()函数，一定要考虑此项操作对性能的影响。设置样式之所以会影响性能，主要是因为样式是继承的。因此，如果在上层某个组件上设置了样式，就会沿着组件嵌套层次向下依次应用该样式设置。

消除影响的关键是尽量少调用 setStyle()函数。

- 首先要使用常规的 CSS 样式表定义应用程序的基准样式，而不是通过 setStyle()函数定义所有样式。
- 只通过 setStyle()函数修改 CSS 未设置的样式。
- 在应用程序或组件初始化并需要使用 setStyle()函数时，使用 preinitialize 事件而不是其他事件(如 creationComplete)。

```
<mx:TextArea text="Text Area" id="myTextArea"
   preinitialize="changeColor()"/>
```

以编程方式不仅可以设置样式，还可以使用 ActionScript 取得样式的当前值。

2. 取得样式值

setStyle()函数有一个“同胞兄弟”，名叫 getStyle()，后者的作用显然是取得组件的样式

信息。getStyle()函数接收一个参数——要取得其值的样式属性名称。属性类型不同，取得的变量类型也不相同(通常是字符串，但有些属性是数字值)。

代码清单 19-16 演示了如何使用 getStyle()属性取得 TextArea 的背景颜色值。

代码清单19-16　以编程方式取得TextArea当前的颜色

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="0xFFFFFF">
   <mx:Script>
      <![CDATA[
         [Bindable]
         public var currentColor:Number;

         public function changeColor():void
         {
            myTextArea.setStyle("backgroundColor",myCP.selectedColor);
            currentColor= myTextArea.getStyle("backgroundColor");
         }

      ]]>
   </mx:Script>

   <mx:TextArea id="myTextArea" text="My Color is {currentColor}"/>

   <mx:Label text="Change the Text Area's color:"/>
   <mx:ColorPicker id="myCP" change="changeColor()"/>
</mx:Application>
```

图 19-15 所示为结果。尽管颜色一般以十六进制表示，但由于此处将取得的值赋给了 Number，所以显示结果是十进制。

只要知道样式属性，就没有问题，因为属性名是 getStyle()函数需要的唯一参数。

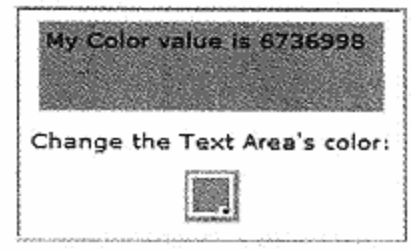


图 19-15　以编程方式取得当前样式属性

19.2　嵌入字体

传统的 HTML 网页存在一个限制，即如果用户没有安装使用的字体，则用户就无法体验到原汁原味的应用程序。虽然在 Flex 中也存在这种情况，可是如果我们知道自己使用了不在标准范围内的字体，那么可以在应用程序中嵌入该种字体。

嵌入字体可以通过使用 CSS 定义@font-face 实现。@font-face 声明的属性如表 19-2 所示。

表 19-2　@font-face 声明的属性

属　性	说　明
src:url	字体文件的路径(True Type 或 Open Type)。如果未指定 src:local 属性，则必须指定此属性。可以使用相对路径(如 myfonts/dopejam.ttf)或绝对路径(如 c:\windows\fonts\cflex.otf)
src:local	字体在系统中的名称，如 Tahoma
fontFamily	用来引用当前字体的名称

(续表)

属　性	说　明
fontStyle	默认值为 normal，其他值包括 italic 和 oblique
fontWeight	默认值为 normal，其他值包括 bold 和 heavy。有些组件默认会使用加粗字体，因此有时候需要把 fontWeight 设置为 bold，有时候则可能需要将其设置为 normal
advancedAntiAliasing	默认值为 true，表示平滑边缘。false 表示不平滑边缘

在定义了 CSS 样式后，把组件的 fontFamily 属性(直接或通过 CSS)设置为引用该字体的别名。

19.2.1 通过字体的系统名嵌入

可以通过字体的系统名嵌入字体，如代码清单 19-17 所示。

代码清单19-17 通过系统名嵌入字体

```
<mx:Style>
   @font-face
   {
     src:local("Tahoma");
      fontFamily: localFont;
      fontWeight: bold;
   }
</mx:Style>
<mx:Button fontFamily="localFont" label="Try Me"/>
```

这种方法的好处是不用关心字体的位置，却依赖于用户计算机中安装的字体；使用字体文件则不存在这种依赖性。

19.2.2 通过字体文件嵌入

还可以通过引用字体文件的方式嵌入字体。请记住，只能嵌入 True Type 和 Open Type 字体，拜托！代码清单 19-18 使用了位于项目 assets 子文件夹中的 Tahoma 字体文件。

代码清单19-18 通过指定文件名嵌入字体

```
<mx:Style>
   @font-face
   {
     src:url("assets/tahoma.ttf");
      fontFamily: localFont;
      fontWeight: bold;
   }
</mx:Style>
<mx:Button fontFamily="localFont" label="Try Me"/>
```

这种方法的优点是，可以把字体文件放在应用程序构建文件夹中，而用户系统中是否安装了该字体则无所谓。

19.2.3　使用 CSS

可以像往常一样使用 CSS 来创建使用字体的完整样式定义(见代码清单 19-19)。这样组件就不必知道字体的别名。

代码清单19-19　在其他样式定义中重用嵌入的字体

```
<mx:Style>
   @font-face
   {
      src:url("assets/Abduction.ttf");
      fontWeight: normal;
      fontFamily: localFont;
   }
   .coolStyle
   {
      fontFamily: localFont;
      fontWeight: normal;
   }
</mx:Style>
<mx:Button styleName="coolStyle" label="Try Me"/>
```

代码清单 19-19 以本地样式定义的方式嵌入字体，然后又像引用常规字体一样引用该字体。

19.3　图像和图标

Flex 中的图像通常泛指某种媒体资产，是设计领域中的一个术语。添加图像很简单，不过也有几种不同的方式，而每种方式又各有利弊。

19.3.1　图像类型

Flex 支持下列类型。

- JPG
- GIF
- PNG
- SVG(Scalable Vector Graphic，可伸缩矢量图)
- SWF(Flash animation，Flash 动画)

可以在 Flex 应用程序中像嵌入字体一样，嵌入以上图像类型，也可以在应用程序运行后加载它们。

19.3.2　嵌入还是不嵌入

由于 Flex 应用程序需要编译，因此必须选择是嵌入图像还是在运行时加载图像。

1. 嵌入图像

嵌入图像的优点是，即时显示。缺点是，图像会导致 Flex 应用程序增大。因此，嵌不嵌入图像决定了加载应用程序的时间长短(不过加载完成后，就没有速度问题了)。嵌入图像还有一个缺点，就是在图像改变时必须重新编译应用程序。

嵌入图像要使用@Embed 指令，该指令告诉 Flex 把相应图像作为文件的一部分。

```
<mx:Image id="myImage" source="@Embed(source='image.jpg')"/>
```

另外，还有以下几点要注意的问题。

- 可以使用绝对或相对路径。
  - 相对路径的例子：source='../../image.jpg'(在 Flex Builder 应用程序中，相对于项目的 src 文件夹)。
  - 绝对路径的例子：source='c:/Documents/Images/myImage.png'。
- 一定要在路径中使用正斜杠(/)，不能使用反斜杠(\)。
- 不能在源中使用 URL。

也可以加载图像，加载图像也有一些独特的好处。

2. 加载图像

在运行时加载图像可以减小文件大小，从而提升加载速度；但在应用程序需要图像时，则会因动态加载图像而有所延迟。加载图像的另一个好处是，修改图像非常容易，因为图像与应用程序之间没有耦合关系。

加载图像时不需要使用@Embed 指令，只要指定相对或绝对 URL 即可。

```
<mx:Image id="myImage" source="assets/myImage.png')"/>
```

请记住如下事项。

- Flex 使用沙盒安全模型。默认情况下，Flex 应用程序只能访问其来源服务器。要访问其他服务器，必须配置一个 crossdomain.xml 文件。
- 相对路径是相对于应用程序被加载的位置。
- 绝对路径既可以从根目录开始(如/images/myImage.png)，也可以是一个完整的 URL(如 http://myserver/images/myImage.png)。
- 不能在运行时加载 SVG 图像。

在决定嵌入还是加载图像时，要根据实际情况比较这两种方式的优缺点，最终确定对应用程序最有利的方式。在 Flex 中使用图像的另一种方式是，把图像作为变量。

19.3.3 作为变量的图像

想象一下这种情况：我们已经在应用程序中很多地方，基于特定的文件嵌入一幅图像；而现在我们需要更换一个新文件。此时，想想要搜索代码，查找所有所有需要更新的位置……肯定是一件令人头疼的事。

好在，Flex 支持先将图像赋给一个变量，然后再引用变量，来解除这种紧密的耦合关系。这样，如果需要更换一个新文件，只要在一个地方将图像赋给变量即可。如代码清单

19-20 所示为作。

代码清单19-20　把图像指定给变量以分离其位置信息

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">

   <mx:Script>
      <![CDATA[
         [Bindable]
         [Embed(source="assets/star.gif")]
         public var imgStar:Class;
      ]]>
   </mx:Script>
   <mx:Image source="{imgStar}"/>
</mx:Application>
```

Image 组件可以直接绑定相应变量，如果将来需要指向不同的图像，在一个位置进行修改即可。

与图像类似的是图标，图标是在特定环境下使用的图像。

19.3.4　图标

Flex 中某些视觉组件有一个专用属性，名叫 icon。而且许多组件也支持不同环境下的 icon 属性(例如，默认状态下的图标和被用户单击过的图标)。

图标是个非常不错的特性，通过它可以快速赋予应用程序某种个性化特征。代码清单 19-21 使用了@Embed 指令，因为图标必须是嵌入的图像。

代码清单19-21　为按钮添加图标

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Button label="my button" icon="@Embed(source='assets/star.gif')"/>
</mx:Application>
```

图 19-16 所示为显示的图标，而图 19-17 所示为这个示例中文件的具体存储位置。

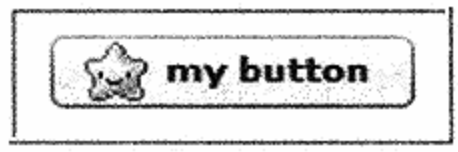


图19-16　Button组件支持icon属性

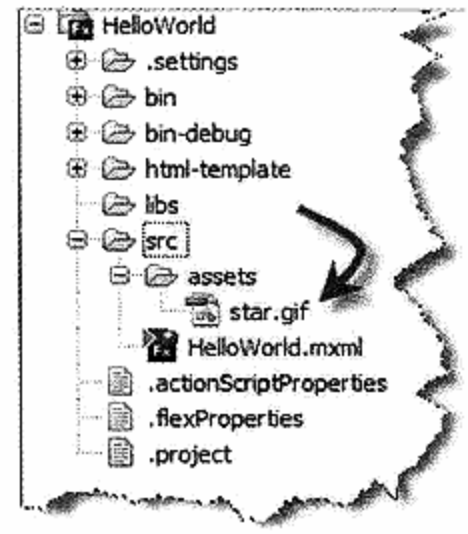


图19-17　这里的图标位于项目的一个子文件夹内

按钮是最适合使用图标的组件，因为可以为按钮的不同状态指定不同图标。

- downIcon

- overIcon
- selectedDisabledIcon
- selectedDownIcon
- selectedOverIcon
- selectedUpIcon
- upIcon

前面的示例使用的是非数据驱动的组件，即 Button。在使用数据驱动的组件时，往往也需要基于数据动态显示图标。

1. 动态列表中的图标

一些包含动态列表项的组件允许在项的旁边显示图标。这些组件有 List、DataGrid、Tree，甚至 ComboBox 等输入控件。

为此，创建图像变量，然后在数据中通过一个字段声明使用哪个图像变量。代码清单 19-22 使用相应的数据。

代码清单19-22 显示数据中指定的图标的列表组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
   <mx:Script>
      <![CDATA[
         import mx.collections.ArrayCollection;

         [Bindable]
         public var myData:ArrayCollection = new ArrayCollection([
         {label:"Jeff", icon:"imgYellow"},
            {label:"Mike",icon:"imgRed"}]);

         [Bindable]
         [Embed(source="assets/yellowstar.gif")]
         public var imgYellow:Class;

         [Bindable]
         [Embed(source="assets/redstar.gif")]
         public var imgRed:Class;

      ]]>
   </mx:Script>

   <mx:List labelField="label" iconField="icon"
      dataProvider="{myData}" height="50"/>
</mx:Application>
```

❶ icon字段包含变量名的文本

❷ 为图标设置嵌入图像的变量

❸ 列表会查找指定的iconField

可见，数据中通过简单字符串指定了图像变量的名称❶，而且程序中也为每个图标设置了可绑定的图像❷。最后，List 组件会遍历每个数据项并通过 iconField 查找指定字段❸，从而确定使用哪个图像作为图标(见图 19-18)。

虽然 ComboBox 不是基于列表的组件，但作为数据驱动的组件，它同样可以通过类似方式使用图标。ComboBox 使用内部列表管理其包含的项。它没有 iconField 属性，但有一个子组件 dropdown，dropdown 是基于列表的组件(见图 19-19)，因此支持图标。

图 19-18　List 组件基于数据动态显示图标

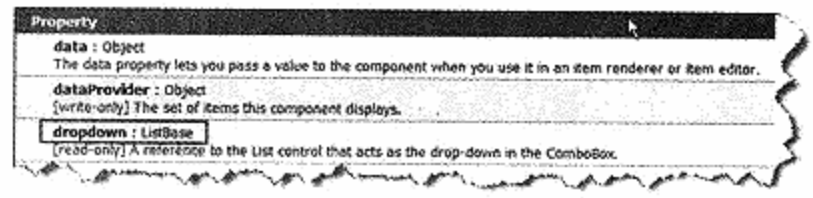


图 19-19　Flex API 语言参考显示 ComboBox 包含一个 List

可以把前面示例中的<mx:List>替换成下面的 ComboBox。

```
<mx:ComboBox id="myCB"  labelField="label"
   creationComplete="myCB.dropdown.iconField='icon'"
   dataProvider="{myData}"/>
```

关键在于，即使某个组件表面上不支持图标，但该组件的某个属性有可能支持。因此问题是，必须知道到哪里查找相关的图标属性。

2. 到哪里查找图标属性

在 Flex 语言参考中查找组件时，展开 Styles 部分，以便查看继承的所有样式。因为图标本身是样式，所以从中就能找到可用的图标属性(图 19-20)。

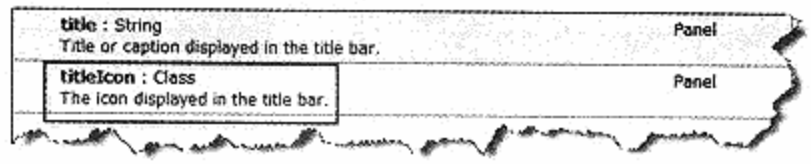


图19-20　使用Flex API语言参考，可以在Styles部分中找到与图标相关的属性

与图像和图标相关的一个概念是皮肤(skin)，皮肤是一种可以配置的界面资产包。

19.4　皮肤

如果读者是一位 WinAmp 或 Firefox 的忠实用户，一定明白怎样通过皮肤变换应用程序的外观。样式表虽然非常有用，但皮肤的作用更大。

19.4.1　皮肤的类型

皮肤有两种类型。

- **图形式皮肤**：使用图像替换组件的某些部分。
- **编程式皮肤**：通过编写脚本定义组件的外观和行为。

皮肤本质上就是样式，因此创建皮肤需要用到前面介绍的有关 Flex 样式的所有内容。不过，皮肤毕竟以图像为基础，所以也要用到刚刚讨论的图像知识。

19.4.2　使用图像的图形皮肤

图形式皮肤可以使用 JPG、PNG 和 GIF 图像。

提示：

PNG 和 GIF 图像支持透明背景，因此可以用来创建图形式皮肤。

皮肤的样式表可以在行内定义，也可以引入外部样式表，这取决于读者自己。不过，我们还是建议使用本地样式，或者最好使用外部 CSS 样式表。

在谈到 UI 自定义时，Button 组件通常最适合用来举例。因此，我们将继续使用 Button 作为创建皮肤的起点。API 语言参考中列出了包含 skin 这几个字母的样式属性(表 19-3)。

表 19-3 Button 组件的皮肤属性

属 性	类 型	说 明
disabledSkin	Class	当 Button 组件的 enabled 属性被设置为 false 时使用的图像
downSkin	Class	当在 Button 上单击鼠标左键时使用的图像
overSkin	Class	当鼠标悬停在 Button 上时使用的图像
upSkin	Class	当鼠标未悬停在 Button 上时使用的图像
selectedDisabledSkin	Class	当选中被禁用的 Button(即 enabled 值为 false，但 selected 值为 true)时使用的图像
selectedDownSkin	Class	当选中 Button 且单击鼠标左键时使用的图像
selectedUpSkin	Class	当选中 Button 但未将鼠标悬停在 Button 上时使用的图像
selectedOverSkin	Class	当选中 Button 且鼠标悬停在 Button 上时使用的图像

知道了这些皮肤属性后，就可以为按钮创建皮肤了。第一种方式是使用行内样式。

1. 通过行内样式创建皮肤

使用行内样式的语法，可以在组件上直接设置相关属性。如代码清单 19-23 所示，我们为按钮皮肤指定了常见的状态：默认状态、鼠标悬停状态和鼠标按下状态。

代码清单19-23 使用行内样式为Button组件创建皮肤

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Button label="My Skinned Button"
   upSkin="@Embed(source='assets/ButtonUp.png')"
   overSkin="@Embed(source='assets/ButtonOver.png')"
   downSkin="@Embed(source='assets/ButtonDown.png')"/>
</mx:Application>
```

图 19-21 至图 19-23 依次展示了不同状态下的按钮皮肤。

图 19-21 鼠标未悬停时的皮肤

图 19-22 鼠标悬停时的皮肤

图 19-23 鼠标按下时的皮肤

目前来看，使用行内样式比较方便。但从长远看，如果有 50 个按钮，这种方式就不方便了。因此在经过原型阶段后，应该考虑使用本地或外部样式表。

2. 通过本地或外部样式创建皮肤

代码清单 19-24 把前面示例中的样式定义转换成本地样式定义，由于使用了类定义，因此组件可以在必要时选择相应的皮肤(通过样式)。

代码清单19-24　使用本地样式表分离样式的实现

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Style>
   .styledButton
   {
     upSkin: Embed("assets/ButtonUp.png");
     downSkin: Embed("assets/ButtonDown.png");
     overSkin: Embed("assets/ButtonOver.png");
   }
</mx:Style>
<mx:Button label="My Skinned Button" styleName="styledButton"/>
</mx:Application>
```

如果想把样式应用到所有按钮，可以把.styledButton 修改为 Button。另外，如果想使用外部样式表，那么把样式定义复制到一个单独的 CSS 文件中即可。

图形式皮肤并不是只能使用传统意义上的图像。由于 Flex 的皮肤在 Flash Player 中运行，因此也可以使用 Flash 动画。

19.4.3　使用 SWF 的图形皮肤

SWF(Flash 动画)是编译后的 Flash 应用程序(也是 Flex 应用程序)。通过 SWF 可以打包字体、声音和图像等媒体资产。因此，可以创建包含图像资产(要用作皮肤)的静态 SWF 文件。

在图形式皮肤中使用 SWF 文件必须遵守一些规则，这些规则都同 Flash 有关。

- 所有图像都必须以元件(symbol)形式存在。
- 所有在皮肤中使用的图像必须位于舞台(stage)上。
- 导出时，要使用 Export for ActionScript 功能。不要忘了指定链接名称(以便在 Flex 中引用)。
- Flash 源文件(FLA 文件)中不能包含 ActionScript。
- 导出的所有元件(symbol)都应该以左上角为注册点。

除非熟悉 Flash，否则读者不太可能使用 Flash 来创建皮肤。作为一名 Flex 开发人员，只有当团队中的设计人员以 SWF 形式提供图像资产时，才有必要使用 SWF。

从使用者的角度讲，可以把 SWF 文件想象成一个压缩文件。而且压缩文件中包含许多图像，我们必须知道在什么情况下使用哪幅图像。要指定图像，指定元件即可。代码清单 19-25 仍以前面的示例为基础，为 Embed 指令添加了另一个参数，用以指定使用哪个图像元件(代码清单 19-25)。

代码清单19-25 使用Flash的SWF文件创建皮肤

```
<mx:Style>
   .styledButton
   {
      upSkin: Embed("assets/ButtonSkin.swf",symbol="up");
      downSkin: Embed("assets/ButtonSkin.swf",symbol="down");
      overSkin: Embed("assets/ButtonSkin.swf",symbol="over");
   }
</mx:Style>
```

这样，我们就为相应的按钮应用了皮肤，而该按钮恐怕还毫无察觉呢！

事实上，使用上述方法创建皮肤是相当费时的。为此，Adobe 工具套件提供了更简便的转换方式。

19.4.4 Flex 皮肤设计扩展(SDE)

Adobe CS3 中的 Photoshop、Illustrator、Flash 和 Fireworks 已经集成了相应的皮肤设计功能。使用方法很简单，大致如下。

(1) 在 CS3 工具(如 Photoshop)中，选择 File | Scripts | New Flex skin。此时会看到一个提示，询问想为哪个组件创建皮肤；可以在此选择为组件的所有实例创建皮肤，也可以只为使用样式名(styleName)的组件创建皮肤。

(2) 然后会加载一幅包含默认皮肤的图像。可以根据需要修改每个状态(鼠标悬停和鼠标按下等)。

(3) 隐藏第一层，也就是包含辅助线的非皮肤内容层。

(4) 选择 File | Scripts | Export Flex skin，然后选择保存皮肤的位置。这样皮肤图像就会被保存在一个名为 assets 的文件夹中。

(5) 在 Flex 中，选择 File | Import | Skin Artwork，然后找到第(4)步创建的 assets 文件夹。

(6) 此时将看到可供选择的皮肤。从中选择想要的皮肤，单击 Continue 按钮。

(7) Flex Builder 会自动创建样式定义。

注意：

在本书写作时，Adobe CS4 尚未发布。不过，CS4 中可能会有些变化。19.4.4 小节中介绍的步骤基本上会保持不变。但在 Illustrator CS4 中，可以在 File | Scripts | Flex Skins 中打开 Create Skins 对话框。Fireworks CS4 则会把该选项放在 Commands | Flex Skinning 中。

总之，这些 SDE(Skin Design Extensins，皮肤设计扩展)的目标，就是让设计人员创建的皮肤能够几乎毫不费力地用在应用程序中。

19.4.5 图像切割

在使用图像创建皮肤时有一个问题：组件大小的改变会导致图像变形，因为 Flex 只能线性地缩放图像。例如，下面这行代码使用图 19-21 中的图像，但通过设置按钮高度让它变大了。

```
<mx:Button label="My Skinned Button" styleName="styledButton" height="75"
   width="140"/>
```

结果如图 19-24 所示，图像在垂直方向上被拉长了。尽管 Flex 是按比例缩放的，但恐怕设计人员也不愿意看到这种缩放法。

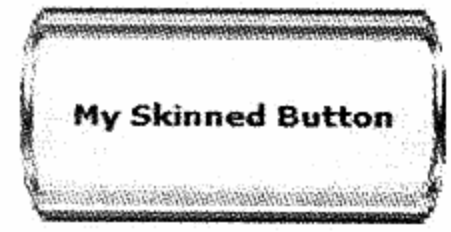


图 19-24　注意图像的上沿由于缩放被拉伸了

好在有一种技术可以解决这个问题，叫做 9 切片缩放(9-slice scaling)。9 切片缩放类似于使用表格为 HTML 切割图像，然后再把切割结果组合起来，因此可以适应各种大小。

在 HTML 中，通常利用重复单元格背景来适应大小变化，而固定宽度的列始终保持不变。在 Flex 中则有所不同：需要定义可变大小图像的周长，导致把 4 个角排除在外，当图像内部大小变化时，4 个角的大小不变。

为此需要在 Embed 指令中指定 4 个 9 切片属性，如表 19-4 所示。

表 19-4　在 Embed 指令中使用的 4 个 9 切片缩放属性

属　性	类　型	说　明
scaleGridLeft	Number	从左侧(像素)
scaleGridRight	Number	从右侧(像素)
scaleGridTop	Number	从上部(像素)
scaleGridBottom	Number	从底部(像素)

如图 19-25 所示，这些数值都是相对于左上坐标而言的。

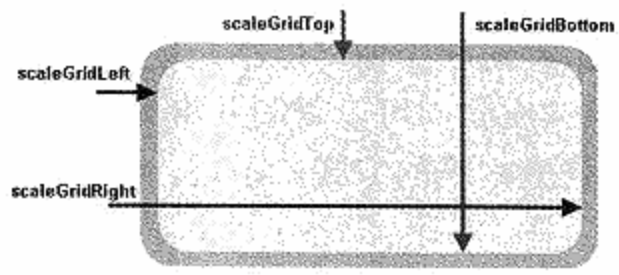


图 19-25　通过 9 切片方法控制图像边缘的缩放方式

代码清单 19-26 所示为这几个属性的用法，注意是在 Embed 指令中指定它们。

代码清单19-26　使用9切片属性

```
<mx:Style>
   .styledButton
   {
      upSkin: Embed("assets/ButtonUp.png",
         scaleGridLeft="5",scaleGridRight="194",scaleGridTop="5",
            scaleGridBottom="31");
      downSkin: Embed("assets/ButtonDown.png",

         scaleGridLeft="5",scaleGridRight="194",
            scaleGridTop="5",scaleGridBottom="31");
      overSkin: Embed("assets/ButtonOver.png",
         scaleGridLeft="5",scaleGridRight="194",scaleGridTop="5",
            scaleGridBottom="31");
   }
</mx:Style>
```

现在图像 4 边的拉伸范围完全处于受控状态(见图 19-26)。图像的上、下边保持了原来的 5 像素高，图像整体效果得到了保持。

9 切片缩放技术并不只对皮肤有效；可以在任何 Embed 指令中使用 9 切片缩放属性，包括常规图像。

```
<mx:Image width="100" height="50" id="myImag
   source="@Embed(source='assets/
   ButtonUp.png',scaleGridLeft='5',
   scaleGridRight='194',scaleGridTop='5',scaleGridBottom='31')"/>
```

图 19-26 使用 9 切片缩放技术保持图像整体效果

不过，对于经常需要重用的组件而言，9 切片缩放技术与皮肤可是一对形影不离的好伙伴。

9 切片缩放技术确实很有效，但也存在一个限制，即图像必须是“矩形”，这样才能保证 4 个角不被拉伸。

19.4.6 编程式皮肤

编程式皮肤属于高级内容，已经超出本书讨论的范围。不过，我们在此仍想为读者介绍一下这种创建皮肤的方法。本质上讲，编程式皮肤就是完全用 ActionScript 创建皮肤类，而这些类必须实现最低限度的功能。为此需要导入其他辅助类(特别是 Graphics)；以便绘制图形，修改颜色，完成其他与图形有关的操作。

在创建编程式皮肤时，有 3 种方式可供选择。

- **扩展已有的皮肤**：这是优先选择的方式，因为通过扩展可以得到所有功能和行为，相当于有了良好的基础。然后可以再增加新的特性。
- **修改已有的皮肤**：与扩展皮肤类似，但不是新增皮肤，而是修改已有皮肤。Flex 有一个名叫 Halo 的默认皮肤，可以复制并将其修改成自己的皮肤。
- **从头创建新皮肤**：这是最困难的方式。不过，要想创建令人耳目一新的皮肤，可以选择这种方式。

愿意冒险创建编程式皮肤的开发人员并不多，因为投入的时间相当可观。虽然最终结果可能会令人惊叹，但从投资的角度看，却很可能得不偿失。除非应用程序的业务模型对界面外观有特殊要求。

19.5 小结

本章讨论的内容不少，但新知识并不多(特别是对具有 Web 开发背景的读者而言)。现在读者应该对如何通过使用样式表(CSS)定义样式、嵌入自定义字体、利用图像和图标以及使用皮肤美化应用程序，来创建个性化界面有了深刻的认识。

在自定义界面这个大主题中，第 20 章将介绍的 Flex 效果也非常重要。多数人认为效果是界面中的装饰品，但只要使用得当，效果也会提升可用性。

第 20 章 使用效果

20

本章内容：

- 定义和使用效果
- 效果的类型
- 缓和函数
- 在效果中使用字体

效果(effect)会让 Flex 应用程序与众不同。Flex 基于 Flash 平台，而后者在创建迷人的动画界面方面可谓源远流长。因此通过 Flex 创建效果具有坚实的基础。

本章我们先从介绍和使用效果开始，然后讨论 Flex 内置的各种效果。在掌握了本书前面章节中介绍的厚重内容后，现在也该来点轻松的了。

20.1 效果的含义

效果就是随着时间的推移修改组件的视觉属性。这些属性包括组件的位置、透明度和大小等。例如，可以在 5 秒钟内逐渐改变其中一种属性。

20.1.1 创建效果的方式

可以通过两种方式生成效果。

- **使用触发器**：在某个事件发生时自动触发效果。例如，可以在鼠标移动到组件上，或组件状态从可见变为不可见时触发效果。
- **编程方式**：可以动态创建并触发效果并根据需要使用它。例如，可以通过编程方式确定某笔账目是否过期，如果是，则为该过期账目的 Date Due(逾期)字段应用红色发光效果。

这两种方式都可以用于 Flex 内置的效果。

20.1.2 Flex 内置的效果

Flex 提供了下列内置效果。

- AnimateProperty
- Blur
- Dissolve
- Fade
- Glow
- Iris
- Move
- Pause
- Resize
- Rotate
- SoundEffect
- WipeLeft、WipeRight、WipeUp、WipeDown
- Zoom

把这些效果组合起来，可以创建合成效果(Composite effect)。

20.1.3 合成效果

合成效果的方式有以下两种。

- **串行合成**：逐一播放每个效果。
- **并行合成**：同时以并行方式一次性播放所有效果。

在对效果有了总体认识后，下面我们来详细介绍每个方面。首先从触发效果开始。

20.2 触发效果

如前所述，触发效果是由事件导致的。所有视觉组件都包含一组属性，通过这些属性可以指定在相应事件发生后播放什么效果。

例如，可以像下面这样指定当组件上发生悬停事件时播放的效果。

```
<mx:SomeComponent rollOverEffect="{idOfEffect}"… />
```

Flex 提供的视觉组件至少支持表 20-1 所示的效果触发器。

表 20-1 所有视觉组件都支持的效果触发器属性

属 性	类 型	触发时机
addedEffect	Effect 对象	组件正在被添加到容器中
creationCompleteEffect	Effect 对象	Flex 创建完一个组件后
focusInEffect	Effect 对象	键盘焦点转到当前组件上时

(续表)

属　性	类　型	触发时机
focusOutEffect	Effect 对象	键盘焦点移出组件时
hideEffect	Effect 对象	组件的 visible 属性被修改为 false 时
mouseDownEffect	Effect 对象	在组件上单击鼠标左键时
mouseUpEffect	Effect 对象	鼠标在组件上被按下又释放时
moveEffect	Effect 对象	组件改变位置时
removedEffect	Effect 对象	组件被所在容器删除时
resizeEffect	Effect 对象	调整组件大小时
rollOutEffect	Effect 对象	鼠标从组件上移开时
rollOverEffect	Effect 对象	鼠标移动到组件上时

接下来我们通过一个示例看一看这些属性的用法。相册应用程序中有一个流行的效果，即在鼠标移动到缩略图上时，小照片会逐渐变大。代码清单 20-1 所示为创建这个 Resize 效果的过程。

代码清单20-1　在鼠标悬停时通过Resize效果放大图片

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="white">
  <mx:Resize id="fxEnlarge" widthTo="300" heightTo="176"/>
  <mx:Resize id="fxShrink"  widthTo="150" heightTo="88"/>
  <mx:Image  id="thumbnail"
          rollOverEffect="{fxEnlarge}"
          rollOutEffect="{fxShrink}"
          source="assets/bmw.jpg"/>
</mx:Application>
```

恐怕没有比这更容易的了吧！如图 20-1 所示，当鼠标移至和移出图像时，会触发已经应用的两个效果。

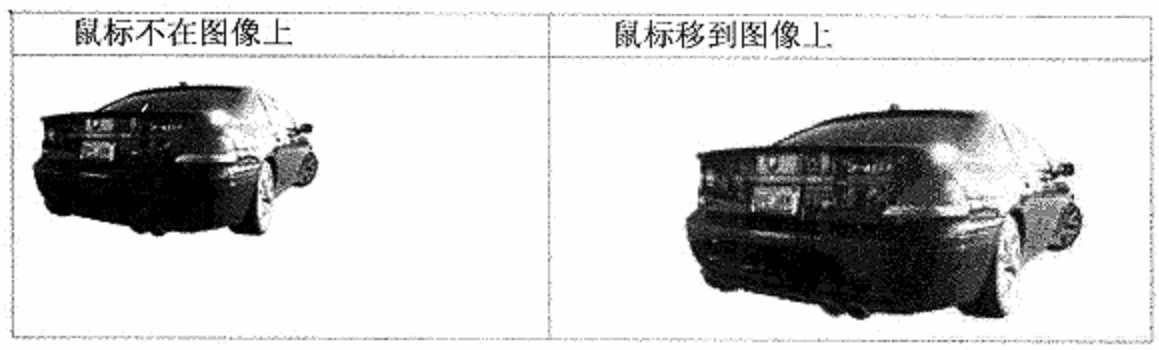


图 20-1　鼠标移到图像上会触发 Resize 效果，导致图像变大

注意：

触发器与效果合在一起就是行为。

创建触发效果很容易，但仅限于组件支持的已知触发器。如果想基于业务逻辑应用效

果，则必须使用编程方式。

20.3 以编程方式应用效果

通过事件，可以触发 Flex 根据“常规”情况提供的效果。但是，在实际开发中总会有一些特殊情况也需要应用效果。

20.3.1 使用 ActionScript 创建效果

假设有一个提示功能，它会在某个值符合条件时以发光效果显示消息。代码清单 20-2 所示为以倒计时方式实现这个功能的过程；如果达到了临界条件，则每次检查到条件匹配都会以发光效果显示消息。

代码清单20-2 当满足某些业务规则时播放效果

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white" initialize="initMe()">
  <mx:Script>
  <![CDATA[
   import flash.utils.Timer;
   [Bindable]
   public var secondsTillDue:int=120;
   public var myTimer:Timer;

   public function initMe():void
   {
    myTimer = new Timer(1000);
    myTimer.addEventListener('timer',warnIfClose);
    myTimer.start();
   }

   public function warnIfClose(event:TimerEvent):void
   {
    secondsTillDue=secondsTillDue-1;
    if(secondsTillDue<90)
    {
     myEffect.target = idDueLabel;
     myEffect.play();
    }
   }
     ]]>
  </mx:Script>
  <mx:Glow id="myEffect" duration="750"
      alphaFrom="1.0" alphaTo="0.3"
      blurXFrom="0.0" blurXTo="50.0"
      blurYFrom="0.0" blurYTo="50.0"
      color="0xFF0000"/>

  <mx:Label text="This is due in {secondsTillDue} seconds"
        id="idDueLabel"/>

</mx:Application>
```

设置计时器，每秒执行一次检查

如果值小于90秒则播放动画

将相应标签组件指定为应用效果的目标

运行这个应用程序后，当倒计时少于 90 秒时，将看到如图 20-2 所示的消息发光效果。

在这个示例中，我们使用了一个 MXML Effect 组件的实例(myEffect)。这种做法可以避免在修改效果时修改函数中执行的操作。如果不想使用 Glow 效果了，只要保证 id 属性不变，myEffect.play()部分就照常运行。

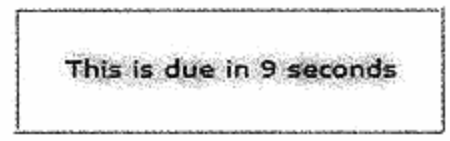


图 20-2　以编程方式应用的发光效果

20.3.2　只使用 ActionScript

如果你愿意，当然也可以只使用 ActionScript。此时，要使用相应效果的 ActionScript 类，而不是组件。代码清单 20-3 通过 ActionScript 创建与前一个示例相同的效果。

代码清单20-3　完全使用ActionScript创建效果的实例

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="white" initialize="initMe()">
  <mx:Script>
  <![CDATA[
    import flash.utils.Timer;
    import mx.effects.Glow;
    [Bindable]
    public var secondsTillDue:int=120;
    public var myTimer:Timer;
    public var myEffect:Glow;

    public function initMe():void
    {
      myTimer = new Timer(1000);
      myTimer.addEventListener('timer',warnIfClose);
      myTimer.start();
      myEffect = new Glow();
      myEffect.alphaFrom=1.0;
      myEffect.alphaTo=0.3;
      myEffect.blurXFrom=0.0;
      myEffect.blurXTo=50.0;
      myEffect.blurYFrom=0.0;
      myEffect.blurYTo=50.0;
      myEffect.color=0xFF0000;
      myEffect.duration=750;
    }

    public function warnIfClose(event:TimerEvent):void
    {
      secondsTillDue=secondsTillDue-1;
      if(secondsTillDue<90)
      {
        myEffect.target = idDueLabel;
        myEffect.play();
      }
    }
  ]]>
  </mx:Script>

  <mx:Label text="This is due in {secondsTillDue} seconds"
          id="idDueLabel"/>

</mx:Application>
```

设置计时器，每秒执行一次检查

如果值小于90秒则播放动画

将相应标签组件指定为应用效果的目标

注意在一开始导入 Glow 类，以便 Flex 知道所谓的“发光”是什么意思。

还记得前一个示例的好处是什么吗？切换到其他效果相对比较容易。虽然使用ActionScript也可以切换效果，但这种方式的最大优点是可以动态创建Effect对象。例如，为了顺利进行，可以把各种情况下要播放的效果保存为配置信息。

20.4 创建合成效果

可以把几个独立的效果组合到一起，创建出合成效果。合成效果可以是几种效果的并行播放，也可以是串行播放。

20.4.1 串行效果

按照顺序一个接一个地播放效果，应该使用<mx:Sequence>标签，如代码清单20-4所示。

代码清单20-4 组合并按顺序播放效果

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Sequence id="sequenceEffect">
      <mx:Fade alphaFrom="0.0" alphaTo="1.0"/>
      <mx:Zoom zoomWidthFrom="0.5" zoomWidthTo="1.0"
         zoomHeightFrom="0.5" zoomHeightTo="1.0"/>
   </mx:Sequence>
   <mx:Button id="myButton" label="Mouse Over To Play Effect"
      rollOverEffect="{sequenceEffect}"/>
</mx:Application>
```

当鼠标移到按钮上时，Fade效果开始播放。Fade效果播放完后，Zoom效果接着播放。如果想同时播放所有效果，可以使用并行效果。

20.4.2 并行效果

要同时播放一组效果，应该使用<mx:Parallel>标签。代码清单20-5修改了前面的示例以便实现并行播放效果。

代码清单20-5 同时播放多个效果

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Parallel id="parallelEffect">
  <mx:Fade alphaFrom="0.0" alphaTo="1.0"/>
  <mx:Zoom zoomWidthFrom="0.5" zoomWidthTo="1.0"
         zoomHeightFrom="0.5" zoomHeightTo="1.0"/>
  </mx:Parallel>
  <mx:Button id="myButton" label="Mouse Over To Play Effect"
          rollOverEffect="{parallelEffect}"/>
</mx:Application>
```

如果想混合串行和并行呢？Flex支持组合合成效果。

20.4.3 组合合成效果

只有串行和并行还不够。实际上，可以任意混合搭配串行和并行效果。例如，可顺序播放两组并行效果，如代码清单 20-6 所示。

代码清单20-6 在串行效果中嵌套两个并行效果

```
<mx:Sequence id="compositeEffect">
  <mx:Parallel>
    <mx:Fade alphaFrom="0.0" alphaTo="1.0"/>
    <mx:Zoom zoomWidthFrom="0.5" zoomWidthTo="1.0" zoomHeightFrom="0.5"
          zoomHeightTo="1.0"/>
  </mx:Parallel>
  <mx:Parallel>
    <mx:Zoom zoomWidthFrom="1.0" zoomWidthTo="0.5" zoomHeightFrom="1.0"
          zoomHeightTo="0.5"/>
    <mx:Fade alphaFrom="1.0" alphaTo="0.0"/>
  </mx:Parallel>
</mx:Sequence>
```

理论上，合成效果的嵌套没有深度限制。但是从可用性角度来讲，仅仅为传达变化无需过分使用效果。

在知道了如何创建和实例化效果之后，下面我们看一看 Flex 内置了哪些类型的效果。

20.5 使用 Flex 内置的效果

前面曾经提到过，Flex 提供了一组非常不错的现成效果。这些效果都有各自独特的属性，同时也共享了一组属性(继承的结果)。表 20-2 所示为这些共享的属性。

表 20-2 所有效果都支持的属性

属 性	类 型	说 明
duration	Number	效果持续的时间，单位是毫秒。例如，5 000 意味着效果要持续 5 秒钟
easingFunction	Function	缓和(easing)函数的名称。默认为 none。20.6 节将具体讨论缓和函数
repeatCount	Number	效果重复播放的次数。默认为 1；要无限次播放，可设置为 0
repeatDelay	Number	如果设置了重复播放效果，则通过这个属性设置下一次播放效果之前等待的时间。默认值为 0(没有延迟)
startDelay	Number	效果开始播放前等待的时间，单位是毫秒
id	String	效果名，用于让组件引用当前效果

通过这些属性，基本可以实现对效果各方面的控制，它们在运行时起作用。知道了这些公共属性后，下面我们开始介绍 Flex 内置的典型效果，先从 AnimateProperty 开始。

20.5.1 AnimateProperty

AnimateProperty 是一个通用效果，可以将组件的某个属性从一个值变化为另一个值。例如，可以使用 AnimateProperty 效果将某组件的宽度变化成一个特定的值，或者通过它修改组件的 Alpha 属性令其消失。

有读者可能会说，这好像是其他一些效果啊。没错，实际上，一些效果通过 AnimateProperty 来操作预先选定的属性实现。但是，如果需要更细致的控制，可以直接使用 AnimateProperty。代码清单 20-7 修改了 Button 的透明度属性(alpha)，在 3 秒钟内将按钮从完全不透明变成完全透明。

代码清单20-7 使用AnimateProperty效果实现按钮淡出

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:AnimateProperty id="myEffect" property="alpha"
                      fromValue="1.0" toValue="0.0" duration="3000"/>
  <mx:Button label="Click me" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

提示：

duration 是所有效果共享的一个继承属性。

而使用 Fade 效果(20.5.4 小节讨论)也能实现相同的行为。

20.5.2 Blur

顾名思义，Blur(模糊)就是把内容搞得看不清楚。可以从两个方向加以控制：水平(X 轴)和垂直(Y 轴)。表 20-3 所示为操作 Blur 效果的属性。

表 20-3 Blur 效果特有的属性

属　性	类　型	说　明
blurXFrom	Number	水平模糊的起始值。0～255
blurXTo	Number	水平模糊的终止值。0～255
blurYFrom	Number	垂直模糊的起始值。0～255
blurYTo	Number	垂直模糊的终止值。0～255

代码清单 20-8 使用 Blur 效果实现了用户按下按钮时的水平模糊效果。

代码清单20-8 为按钮应用Blur效果

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="white">
  <mx:Blur id="myEffect" blurXFrom="0" blurXTo="255"/>
  <mx:Button label="Click me" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

如图 20-3 所示，当鼠标单击按钮时，按钮会慢慢变得模糊。

图 20-3　按钮应用 Blur 效果后的变化

下面我们看一看 Dissolve(溶解)效果。

20.5.3　Dissolve

Dissolve 效果与 Fade 效果类似，但实现方法不同。Dissolve 是把一个透明矩形放在组件上面，然后这个矩形会慢慢变得不透明。实际上，看不到这个矩形，因为它的颜色与它所覆盖的背景相同。表 20-4 所示为 Dissolve 效果特有的属性。

表 20-4　Dissolve 效果特有的属性

属　性	类　型	说　明
alphaFrom	Number	起始的透明度值。0.0～1.0
alphaTo	Number	终止的透明度值。1.0～0.0
color	Color	矩形溶入的颜色。默认为组件的 backgroundColor(可选)

代码清单 20-9 所示为在 Dissolve 效果中使用这些属性实现了按钮由完全不透明到几乎透明的变换。

代码清单20-9　为按钮应用Dissolve效果

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:Dissolve id="myEffect" alphaFrom="1.0" alphaTo="0.1"/>
  <mx:Button label="Click me" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

如前所述，从行为上看，Fade 效果与 Dissolve 效果很相似。

20.5.4　Fade

Fade 效果就是在一定时间内改变组件的 alpha 属性，因此相当于使用 AnimateProperty 效果操作 alpha。表 20-5 所示为 Fade 效果特有的属性。

表 20-5　Fade 效果特有的属性

属　性	类　型	说　明
alphaFrom	Number	起始的透明度值。0.0～1.0
alphaTo	Number	终止的透明度值。1.0～0.0

代码清单 20-10 使用这些属性将按钮从完全不透明淡出为几乎透明。

代码清单20-10 为按钮应用Fade效果

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:Fade id="myEffect" alphaFrom="1.0" alphaTo="0.1"/>
  <mx:Button label="Click me" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

单击按钮，按钮就会淡出至几乎透明，这与 20.3 小节所示的效果相同。

20.5.5 Glow

Glow 效果类似于为组件加上边框，同时又为这个组件应用模糊和淡出效果。因此 Glow 效果具有 Fade 和 Blur 二者的属性。表 20-6 所示为 Glow 效果特有的属性。

表 20-6 Glow 效果特有的属性

属 性	类 型	说 明
blurXFrom	Number	水平模糊的起始值。0～255
blurXTo	Number	水平模糊的终止值。0～255
blurYFrom	Number	垂直模糊的起始值。0～255
blurYTo	Number	垂直模糊的终止值。0～255
alphaFrom	Number	起始的透明度值。0.0～1.0
alphaTo	Number	终止的透明度值。1.0～0.0
color	Color	发光的颜色
inner	Boolean	控制内发光，还是外发光。默认值 false 表示外发光，true 表示内发光
knockout	Boolean	如果值为 true，则组件内部将变成透明的。默认值为 false
strength	Number	发光区域的强度。0～255

代码清单 20-11 创建了在水平和垂直方向同时发光的效果，颜色为红色。

代码清单20-11 为按钮应用Glow效果

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:Glow id="myEffect" duration="750"
      alphaFrom="1.0" alphaTo="0.3"
      blurXFrom="0.0" blurXTo="50.0"
      blurYFrom="0.0" blurYTo="50.0" knockout="true"
      color="0xFF0000"/>
  <mx:Button label="With Knock Out" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

图 20-4 所示为通过设置 knockout 属性，避免 Glow 效果盖住组件，而是在按钮上抠出

了一块实心区域。

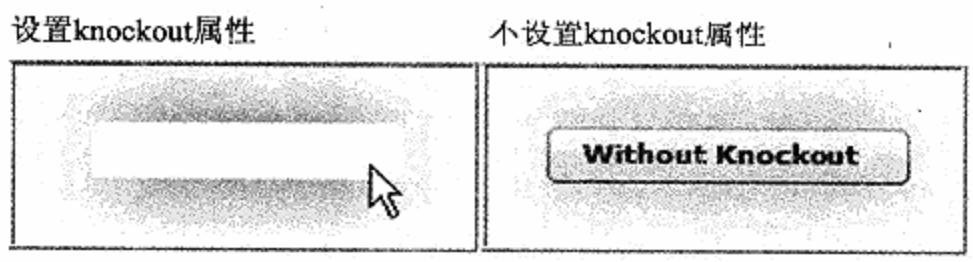


图 20-4 通过 knockout 属性控制 Glow 效果

接下来是 Iris(虹膜)效果。

20.5.6 Iris

Iris 效果使用了蒙板技术。读者可能在其他 Adobe 工具(如 Flash 和 Photoshop)中接触过蒙板的概念；通过蒙板可以针对可见区域创建黑白混合的形状(白色表示可见，黑色表示不可见)。

在默认情况下，Iris 效果会在组件中心放置一个矩形的小蒙板，然后将蒙板向四周扩大。随着蒙板扩大，组件的更多部分会逐渐显露出来。

Iris 效果没有任何专用属性——只有默认效果，如代码清单 20-12 所示。

代码清单20-12 为按钮应用Iris效果

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  backgroundColor="white">
 <mx:Iris id="myEffect"/>
 <mx:Button label="Click on me" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

这个效果用得不多，主要原因是效果的名称，大多数人不能根据 *Iris* 这个词想象出实际效果。不过这个效果还是值得一试，毕竟很多用户都认为它很少见。

20.5.7 Move

Move 效果可以把组件从一个位置移到另一个位置。表 20-7 中所有属性的单位都是像素，而且都相对于组件所在容器的左上角。

表 20-7 Move 效果特有的属性

属　性	类　型	说　明
xFrom	Number	水平起始位置。如果不指定，则以当前位置为起点
xTo	Number	水平终止位置。如果不指定，则以当前位置为终点
yFrom	Number	垂直起始位置。如果不指定，则以当前位置为起点
yTo	Number	垂直终止位置。如果不指定，则以当前位置为终点
xBy	Number	水平移动的像素数。负值表示向左移动，正值表示向右移动
yBy	Number	垂直移动的像素数。负值表示向上移动，正值表示向下移动

代码清单 20-13 给出了一个示例，每次单击按钮都会导致按钮沿对角线向右下角移动。

代码清单20-13 使用Move效果沿对角线移动按钮

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
   <mx:Move id="myEffect" xBy="50" yBy="50"/>
   <mx:Button label="Click on me" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

注意，在使用这个效果时，组件每次都会停留在终点上。如果想把组件移回起始位置，则需要应用相反的效果。

有读者可能觉得这跟手工操作组件的 X 和 Y 属性没有什么不同，从某种角度讲，确实如此。不过，别忘了所有效果都共享公共属性，如 repeatCount 和 duration。另外，还可以在合成效果中使用 Move。

20.5.8 Pause

Pause 效果可以单独使用，但通常不会单独使用，因此它的作用就是使时间延迟。有时候，在合成效果中的不同效果间加入一段延迟是很有用的。

Pause 也没有特殊属性，只要使用共享的 duration 属性控制其延迟时间即可。下面我们在上一个示例中加入 Pause 效果，以便在单击后向下移动按钮，等 5 秒钟后再把它移回原位(代码清单 20-14)。

代码清单20-14 在两个效果间使用Pause效果加入暂停

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:Sequence id="myEffect">
   <mx:Move xBy="50" yBy="50"/>
   <mx:Pause duration="5000"/>
   <mx:Move xBy="-50" yBy="-50"/>
  </mx:Sequence>
  <mx:Button label="Click on me" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

如果是在并行合成效果中使用 Pause，暂停效果不会很明显，因为所有效果都是同时发生的。

20.5.9 Resize

在前面介绍触发效果时，我们已经看到 Resize 效果——沿水平和垂直方向缩放组件。可以将指定大小作为缩放的起/终点，也可以按照指定增量放大/缩小组件。表 20-8 所示为 Resize 效果特有的属性。

表 20-8 Resize 效果特有的属性

属 性	类 型	说 明
heightFrom	Number	起始高度。如果未指定，则使用当前高度(例如，只指定了 heightTo)
heightTo	Number	终止高度。如果未指定，则使用当前高度(例如，只指定了 heightFrom)
widthFrom	Number	起始宽度。如果未指定，则使用当前宽度(例如，只指定了 widthTo)
widthTo	Number	终止宽度。如果未指定，则使用当前宽度(例如，只指定了 widthFrom)
heightBy	Number	高度需要变化的像素数。负值表示减少高度，正值表示增加高度
widthBy	Number	宽度需要变化的像素数。负值表示减少宽度，正值表示增加宽度

代码清单 20-15 给出了一个示例，当单击按钮时，按钮高度会从 0 像素增加到比原始高度高 50 像素。

代码清单20-15 从小到大增加按钮高度

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:Resize id="myEffect" heightFrom="0" heightBy="50"/>
  <mx:Button label="Click on me" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

由于并未改变 width 属性，因此按钮会从扁扁的一条线开始沿垂直方向拉伸。

20.5.10 Rotate

让组件转起来！使用 Rotate 效果，可以围绕所选的点旋转组件。表 20-9 所示为 Rotate 效果特有的属性。

表 20-9 Rotate 效果特有的属性

属 性	类 型	说 明
angleFrom	Number	起始角度。如果未指定，从当前角度开始。0～360
angleTo	Number	终止角度。如果未指定，到当前角度结束。0～360
originX	Number	相对于组件中心点的水平位置。负值向左移动旋转点，正值向右移动旋转点
originY	Number	相对于组件中心的垂直位置。负值向上移动旋转点，正值向下移动旋转点

代码清单 20-16 所示为将组件顺时针旋转 45°。

代码清单20-16 将按钮旋转45°

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:Rotate id="myEffect" angleTo="45"/>
  <mx:Button label="Click on me" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

使用 ActionScript 以自定义逻辑操作按钮的属性，也可以实现旋转。不过，使用 Rotate 效果更方便一些。别忘了，必要时还得把旋转的组件转回去。

20.5.11 SoundEffect

还记得最近一次在 AJAX 应用程序中看到以音效作为沟通方式是什么时候吗？使用 SoundEffect 效果，能够为应用程序添加一种可用性元素，让用户什么也不用看就能领会某种意图。

SoundEffect 支持一种名为“平移(panning)”的技术，可以用来从左到右地控制音量的大小转换(如果你愿意)。表 20-10 所示为 SoundEffect 效果特有的属性。

表 20-10 SoundEffect 效果特有的属性

属 性	类 型	说 明
autoLoad	Boolean	是否自动加载 MP3。默认值为 true，false 表示不自动加载
bufferTime	Number	缓冲的毫秒数。默认值为 1 000
loops	Number	重复播放的次数。默认值为 0，只播放一次
panEasingFunction	Function	平移效果的缓和函数
panFrom	Number	音量起始的平移值。默认为 0(居中)。-1.0 表示完全靠左，1.0 表示完全靠右
panTo	Number	音量终止的平移值。默认为 0(居中)。-1.0 表示完全靠左，1.0 表示完全靠右
source	Object	引用 MP3 音频文件
startTime	Number	音频文件中的启动毫秒数。默认值为 0
useDuration	Boolean	是否考虑 duration 属性。true(默认值)或 false
volumeEasingFunction	Boolean	调节音量的缓和函数
volumeFrom	Number	起始音量。0.0～1.0(默认)
volumeTo	Number	终止音量。0.0～1.0(默认)

代码清单 20-17 所示为单击按钮播放音频的示例。

代码清单20-17 单击按钮播放音频

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
 <mx:SoundEffect id="myEffect" source="@Embed(source='assets/bing.mp3')"/>
 <mx:Button label="Click on me" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

然而，音频在 RIA 领域中使用得还不够广泛，这很大程度上是因为具有 Web 应用程序开发背景的开发人员不习惯于利用声音。我们希望读者能够因地制宜地在应用程序中使用音效。

20.5.12 Wipe

Wipe 会通过在组件上滑入/滑出一个矩形，实现逐渐隐藏/显示组件的效果。这个矩形与背景颜色相同，因此看起来就像组件本身滑入或滑出一样。

虽然不是必须的，但通过切换组件的 visible 属性可以最大限度地利用 Wipe 效果。有 4 种 Wipe 效果：

- WipeLeft
- WipeRight
- WipeUp
- WipeDown

这些效果没有特殊属性。

代码清单 20-18 使用 Wipe 效果来隐藏和显示面板。

代码清单20-18 单击按钮时滑入或滑出面板

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:WipeLeft id="wipeleft" />
  <mx:WipeRight id="wiperight" />
  <mx:Panel id="myPanel" label="Wipe Me"
         showEffect="{wipeleft}" hideEffect="{wiperight}"/>
  <mx:Button label="Toggle the Panel"
          click="myPanel.visible=!myPanel.visible"/>
</mx:Application>
```

单击按钮，会重复打开或关闭面板的 visible 属性。这样就会触发 showEffect 和 hideEffect 事件发生，进而播放相应的 Wipe 效果。

20.5.13 Zoom

Zoom 效果有点像 Resize 效果，但它在后者基础上经过重新配置，专门针对把小组件放大为较大的组件(当然也可以缩小)。Zoom 效果有用于指定缩放原点的额外参数(默认情况下，自中心点开始缩放)。表 20-11 所示为 Zoom 效果特有的属性。

表 20-11 Zoom 效果特有的属性

属性	类型	说明
originX	Number	缩放原点的水平位置。如果不指定，则是组件中心点水平位置
originY	Number	缩放原点的垂直位置。如果不指定，则是组件中心点垂直位置
zoomWidthFrom	Number	起始宽度。如果不指定，默认为当前宽度
zoomWidthTo	Number	终止宽度。如果不指定，默认为当前宽度
zoomHeightFrom	Number	起始高度。如果不指定，默认为当前高度
zoomHeightTo	Number	终止高度。如果不指定，默认为当前高度

运行代码清单 20-19 后，单击按钮，Zoom 效果会把按钮从无形(宽和高都等于 0)放大到原始大小。

代码清单20-19 使用Zoom效果放大按钮

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="white">
  <mx:Zoom zoomWidthFrom="0" zoomHeightFrom="0" id="myEffect"/>
  <mx:Button label="Click on me" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

这是 Flex 内置的最后一个效果。我们注意到，以上示例中的所有运动都是线性的，有一种机械的感觉。要改变这种状况，可以使用缓和函数(*easing function*)。

20.6 缓和函数

默认情况下，Flex 在播放效果时，会随时间均匀地分布每次变化。例如，Move 效果要在 100 秒时间内将一个组件移动 100 像素，那么这个组件移动的速度就是每秒 1 像素。

假设要为上述这个 Move 效果添加一点趣味性，让组件开始时移动较快，然后逐渐减慢直至停止(就像汽车刹车一样)。这时候，就要用到缓和函数了。

20.6.1 内置的缓和函数

Flex 提供了一个内置的缓和包(mx.effects.easing)，其中包含许多带有缓和函数的类。表 20-12 所示为这些类和函数。

表 20-12 Flex 内置提供的缓和函数

类	函数	类	函数	类	函数
Back	easeIn，easeOut，easeInOut	Elastic	easeIn，easeOut，easeInOut	Quartic	easeIn，easeOut，easeInOut

(续表)

类	函 数	类	函 数	类	函 数
Bounce	easeIn, easeOut, easeInOut	Exponential	easeIn, easeOut, easeInOut	Quintic	easeIn, easeOut, easeInOut
Circular	easeIn, easeOut, easeInOut	Linear	easeIn, easeOut, easeInOut, easeNone	Sine	easeIn, easeOut, easeInOut

这些函数还真不少。好在，使用缓和函数很简单，如代码清单 20-20 所示。

代码清单20-20 在Move效果中使用缓和函数生成非线性运动

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="white">
  <mx:Script>
   import mx.effects.easing.*
  </mx:Script>
  <mx:Move id="myEffect" xBy="300" easingFunction="{Elastic.easeOut}"/>
  <mx:Button label="Make me move" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

这个示例中的按钮就像皮球被突然向右射出，然后在对面的边缘上反弹了几下，最后停止不动。

20.6.2 创建自定义缓和函数

也可以创建自定义缓和函数。在定义缓和函数时，Flex 会为函数传递表 20-13 所示的参数。

表 20-13 自定义缓和函数接收的参数

参 数	类 型	说 明
t	Number	当前时间点
b	Number	组件的初始位置
c	Number	组件位置的更改总计
d	Number	效果的总持续时间(以毫秒为单位)
s	Number	过冲量(默认值为 0)；可选

Flex 会按照既定时间间隔调用函数，以便确定组件的当前位置(返回的数值)。代码清单 20-21 所示为一个简单的缓和函数。

代码清单20-21 使用自定义缓和函数

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  backgroundColor="white">
  <mx:Script>
  public function myEasing(t:Number, b:Number, c:Number, d:Number):Number
  {
    t/=d;
    return b+c*(t);
  }
  </mx:Script>
  <mx:Move id="myEffect" xBy="300" easingFunction="{myEasing}"/>
  <mx:Button label="Make me move" mouseDownEffect="{myEffect}"/>
</mx:Application>
```

提示：

如果读者想深入了解缓和函数，可以试一试 Timothee Groleau 创建的在线缓和函数生成器，地址为 http://timotheegroleau.com/Flash/experiments/easing_function_generator.htm。这个程序可以辅助生成缓和函数代码。

在结束本章之前，我们还要讨论一个与字体和效果有关的问题。

20.7 字体与效果

在试验前面的效果时，读者可能发现某些组件中的文本并未受到效果影响。为什么会这样呢？

默认情况下，Flex 会使用系统字体。因而，Flex 操作字体的能力会受到限制。尤其是在 Dissolve、Fade 和 Rotate 效果中。不过，有一个简单的解决方案——嵌入字体。

如代码清单 20-22 所示，可以通过样式来嵌入字体。

代码清单20-22 嵌入字体以保证对文本也应用效果

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="white">
  <mx:Style>
    @font-face
    {
      src:local("Tahoma");
      fontFamily: localFont;
      fontWeight: bold;
    }
  </mx:Style>
  <mx:Fade id="myEffect" alphaFrom="1.0" alphaTo="0.0"/>
  <mx:Button fontFamily="localFont" label="Try Me"
          mouseDownEffect="{myEffect}"/>
</mx:Application>
```

嵌入字体的问题是应用程序大小会因字体占用的空间而增大。所以要注意不能嵌入过多字体。

20.8 小结

效果可以把应用程序提升到一个新的层次。本章介绍的丰富效果一定会让人情不自禁地发出赞叹——特别是那些我们期望能接受 Flex 的管理人员。

但是，不要只把效果当成漂亮的点缀。效果还可以增强应用程序的可用性，它能把用户的注意力吸引到应用程序中正在发生的事件上，也能充当“运动驱动”的沟通机制。

Flex 刚刚出现时，它的另一个特性——拖放，同样也令人侧目和惊叹不已。第 21 章，我们讨论拖放。

第21章 拖　放

21

本章内容：

- 拖放控制
- 实现多项拖放操作
- 创建自定义拖放功能
- 自定义拖放体验

Flex 刚刚问世的时候，就已经内置了拖放(D&D，Drag-and-Drop)特性，而拖放正是所有 RIA 的必备能力。从开发人员的角度看，Flex 的 D&D 特性很容易实现，因为很多组件都具有相应的内在支持机制。与此同时，还可以引入或重写拖放过程各个阶段的默认行为。

我们不仅想通过本书让读者知道怎样实现某项功能，而且还想告诉读者为什么要实现该功能。希望读者不要为了 D&D 而 D&D，或者就因为 D&D 比较酷才使用它。应该通过 D&D 为应用程序增强可用性。

由于使用 HTML/JavaScript 开发 Web 应用程序的限制，早在桌面 GUI 问世时期就引入的功能在 Web 上却一度被遗忘。在传统 Web 应用程序中，用户必须通过点击各种链接或按钮才能移动数据——Web 开发人员似乎总也想不起来使用 D&D(在 Web 环境下)，尽管他们在桌面应用程序中每天都会用到拖放。因为 Web 上的 D&D 难得一见，最终甚至连用户都忘记了这个特性。

我们强烈建议读者以可用性为出发点考虑 D&D，不要仅仅因为你会用或者 D&D 非常酷就使用它。换句话说，使用 D&D 的难点不在于编程(读者在本章后面会看到，编程很容易)，而在于是否用得其所。可以沿袭 Web 用户的惯性思维，在 Web 应用程序中继续提供典型的数据操作方式；但更值得尝试的则是培养他们使用 D&D 的习惯。

毕竟用户的操作越成功，你的应用程序就越成功。在通往成功之路的起点上，最好先总览一下 D&D 的过程。

21.1 拖放过程

你特别喜欢事件，对吧？好，D&D 过程本质上就是在拖动和放置期间的不同阶段触发事件。

首先来明确一些概念。

- **拖动启动器**：启动拖动的组件。
- **拖动代理**：在 D&D 过程中显示给用户的鼠标图标。
- **拖动源**：在拖动启动器与拖动目标之间传递的数据。这个概念比较容易让人误解，因为拖动源听起来好像是拖动操作的来源，实际上拖动操作的来源是拖动启动器。
- **放置目标**：接收数据的组件。

现在我们对某个对象会出现在哪个点上还不清楚，因此应该再了解一下 D&D 的过程。根据用户的操作不同，D&D 会包括许多步骤。为了让读者了解每一步用户在做什么、会触发什么事件以及用户有什么体验，下面比较详细地分析每个步骤。

(1) 启动拖动：

- 用户单击组件中允许拖动的项(即 dragEnabled="true")。触发 mouseDown 事件。
- 如果用户在项上按住鼠标左键不松手，则意味着拖动开始。开始时鼠标所在的组件就是拖动启动器。触发 mouseMove 和 dragStart 事件。

(2) 在拖动过程中，会创建一个拖动源对象。这个对象包含与被拖动项有关的所有数据。例如，从 DataGrid 组件中拖动一行时，如果记录中包含 10 个字段，可能只会显示两个字段。但在拖动时，拖动的却是整个记录行。

(3) 拖动代理出现，为用户给出视觉提示。红色的⊘表示鼠标下方的组件不接受放置。绿色的⊕表示允许放置。

(4) 用户鼠标移动到目标组件。拖动代理继续检查鼠标经过的每个组件，以确定该组件是否接受放置。

如果鼠标经过的组件是潜在的放置目标，则这些组件也会检查拖动源，看是否能够接纳用户拖动的数据(数据类型是否一致？是否存在业务规则？等等)。触发 dragEnter 和 dragOver 事件。

放置目标是接受(见图 21-1)还是拒绝，可以通过拖动代理明确地传达给用户。

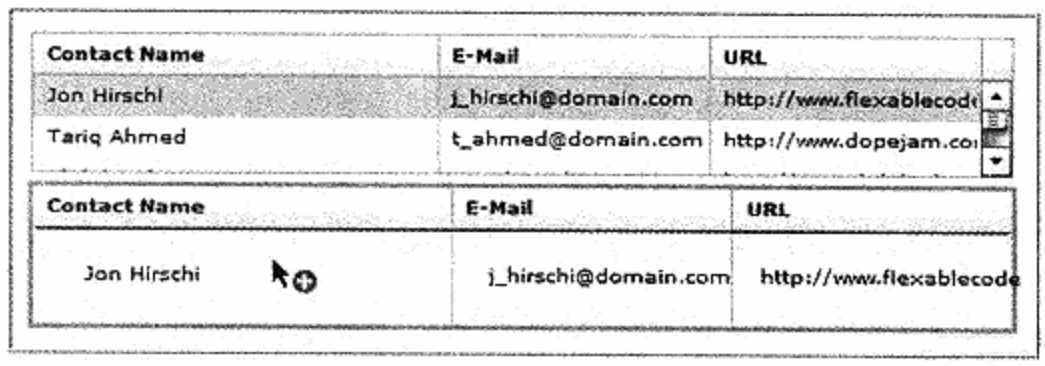


图 21-1 拖动代理为用户提供了视觉指示器，据此可知是否允许相应操作

(5) 发生下列两种情况。

- 用户在放置目标上释放鼠标左键。放置目标处理拖动并适当地操作数据项(默认是将数据添加到 dataProvider 中)。如果放置成功，用户会在放置目标中看到新增的数据项。触发 dragDrop 和 dragComplete 事件。
- 用户鼠标从潜在的放置目标上移开，但同时仍然按住鼠标左键不放。拖动代理改变图标。触发 dragExit 事件。

可见，整个过程会伴随各种事件的发生，因为 Flex 是事件驱动的系统。如果发生了上述事件，则说明当前处于 D&D 过程。

21.2　拖放事件

前面介绍了 D&D 过程中发生的一些事件。下面我们就把这些事件拿出来加以详细介绍。表 21-1 所示的“接收者”列表示获得事件的组件。

表 21-1　拖放事件

事　　件	接　收　者	说　　明
mouseDown	拖动启动器	当用户在数据项上按下鼠标左键时发生。利用这个事件可以添加业务逻辑，以确定是否允许启动拖动操作
mouseMove	拖动启动器	用户开始移动数据项
dragStart	拖动启动器	拖动启动器自行触发这个事件(这个事件看起来好像有点奇怪)
dragEnter	放置目标	拖动代理刚刚进入潜在的放置目标。放置目标可以利用这个事件检查数据并确定接受还是拒绝放置
dragOver	放置目标	在 dragEnter 事件发生后立即发生。利用这个事件可以执行接受后的处理(如果拒绝放置，则可以在 dragEnter 中执行相应操作)
dragDrop	放置目标	由用户释放鼠标左键引发。利用这个事件可以提交数据并终结操作
dragExit	放置目标	如果用户没有释放鼠标左键，而且从当前放置目标上移开时触发，即从 dragOver 直接变成了 dragExit。利用这个事件可以撤销/重置在 dragOver 事件发生时所做的处理
dragComplete	拖动启动器	无论因为什么原因(用户放置了数据项或当鼠标未处在放置目标上时，释放鼠标左键以取消拖放操作)。当 D&D 过程结束时，会将此事件发送给拖动启动器

现在我们已经了解了拖放的过程及伴随拖放发生的事件。接下来，我们看一看支持 D&D 的组件。

21.3　支持拖放的组件

显然，D&D 操作比较适合在基于列表的组件中使用。在 Flex 中，支持拖放的(基于列表的)组件包括：

- List
- TileList
- HorizontalList
- Tree
- DataGrid
- AdvancedDataGrid
- PrintDataGrid

这些组件对 D&D 提供了内置支持。不过，也可以为组件之间的 D&D 操作添加自定义逻辑。

21.4 在基于列表的组件中启用拖放

启用 D&D 很容易——简直易如反掌。基于列表的组件支持两个与此相关的属性，如表 21-2 所示。

表 21-2 基于列表的组件支持的拖放属性

属 性	类 型	说 明
dragEnabled	Boolean	表示是否允许组件作为拖动启动器。true 或 false(默认值)
dropEnabled	Boolean	表示是否允许组件作为放置目标。true 或 false(默认值)

代码清单 21-1 使用这两个属性显示图 21-2 中使用的代码(D&D 过程)。

代码清单21-1 DataGrid接受来自另一个DataGrid的拖放

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Script>
 <![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var myAC:ArrayCollection  = new ArrayCollection([
  {name:"Jon Hirschi", email:"j_hirschi@domain.com",
     url:"http://www.flexablecoder.com"},
  {name:"Tariq Ahmed", email:"t_ahmed@domain.com",
     url:"http://www.dopejam.com"},
  {name:"Abdul Qabiz", email:"a_qabiz@domain.com",
     url:"http://www.abdulqabiz.com"},
  {name:"Faisal Abid", email:"f_abid@domain.com",
     url:"http://www.g-unix.com"},
  {name:"Frank Krul", email:"f_krul@domain.com", url:""}
  ]);
 ]]>
</mx:Script>
<mx:DataGrid id="dgSource" dataProvider="{myAC}" dragEnabled="true" >
   <mx:columns>
      <mx:DataGridColumn dataField="name" headerText="Contact Name"
         width="300" />
      <mx:DataGridColumn dataField="email" headerText="E-Mail"
         width="200"/>
      <mx:DataGridColumn dataField="url" headerText="URL" width="200"/>
   </mx:columns>
</mx:DataGrid>
<mx:DataGrid id="dgTarget" dropEnabled="true" >
   <mx:columns>
      <mx:DataGridColumn dataField="name" headerText="Contact Name"
         width="300" />
      <mx:DataGridColumn dataField="email" headerText="E-Mail"
         width="200"/>
      <mx:DataGridColumn dataField="url" headerText="URL" width="200"/>
   </mx:columns>
</mx:DataGrid>
</mx:Application>
```

本章将较为详细地讨论自定义和控制，但代码清单摘取自真实的应用程序。

21.5 移动与复制

读者大概已经注意到，运行代码清单 21-1 后，在 DataGrid 间拖放数据项执行的是复制操作，而不是移动操作。如果想用移动操作代替复制操作，修改起来也很简单。

基于列表的组件都支持 dragMoveEnabled 属性，默认值是 false，即拖动数据项时默认会执行复制操作(有一个例外，Tree 组件的默认值为 true。)

因此，如代码清单 21-2 所示，只要把 dragMoveEnabled 属性设置为 true，拖放操作就会变成移动而非复制。

代码清单21-2 设置dragMoveEnabled属性以便在拖放中移动而非复制数据项

```
<mx:DataGrid id="dgSource" dataProvider="{myAC}" dragEnabled="true"
          dragMoveEnabled="true" >
   <mx:columns>
      <mx:DataGridColumn dataField="name" headerText="Contact Name"
         width="300" />
      <mx:DataGridColumn dataField="email" headerText="E-Mail"
         width="200"/>
      <mx:DataGridColumn dataField="url" headerText="URL" width="200"/>
   </mx:columns>
</mx:DataGrid>
```

做此修改后，再运行这个示例，拖动数据项就可以将其从来源 DataGrid 中删除了。这种工作流应该是我们比较熟悉的，而且对于移动单个数据项也很有效。

提示：

在把 dragMoveEnabled 设置为 true 的情况下，也可以执行复制操作。方法是在拖动时按住 Ctrl 键。

可是，在想要移动多个数据项时，一个一个地拖动就太麻烦了。与 Windows 类似，Flex 也支持多项拖动。

21.6 多项拖动

如果想让用户能够同时移动多个数据项，试试 allowMultipleSelection 属性。把 allowMultipleSelection 属性设置为 true，即可实现多项拖动(见代码清单 21-3)。

代码清单21-3 让用户能够一次移动多个数据项

```
<mx:DataGrid id="dgSource" dataProvider="{myAC}"
   dragEnabled="true" dragMoveEnabled="true"
      allowMultipleSelection="true" >
   <mx:columns>
      <mx:DataGridColumn dataField="name" headerText="Contact Name"
         width="300" />
```

```
        <mx:DataGridColumn dataField="email" headerText="E-Mail"
           width="200"/>
        <mx:DataGridColumn dataField="url" headerText="URL" width="200"/>
      </mx:columns>
    </mx:DataGrid>
```

此时，用户拖动数据项时有两种选择：一是单个选择数据项，二是选择一组数据项。要逐个选择数据项，按照下列步骤操作。

(1) 按住 Ctrl 键，通过鼠标选择想要拖动的每一项。

(2) 选择完要拖动的所有项后，释放 Ctrl 键。

(3) 在选择的项上按下鼠标左键，并执行拖放。

图 21-2 所示为隔行移动数据项时的情景。

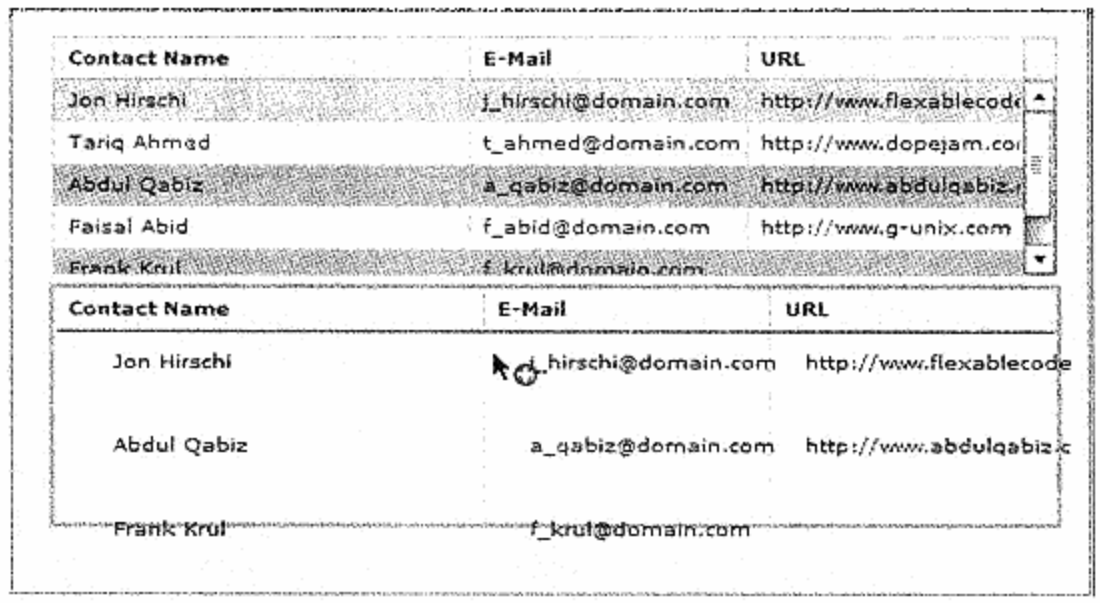


图 21-2 按住 Ctrl 键可以选择要拖动的多个数据项

Flex 也支持选择一组数据项。单击第一项和最后一项是选择一组连续数据项的便捷方式，Flex 会选择单击范围内的所有数据项。

为此按照下列步骤操作。

(1) 选择第一个数据项。

(2) 滚动到最后一个数据项，按住 Shift 键，然后选择该项。Flex 会突出显示其间所有选中的项。

(3) 释放鼠标左键，在选中的项上单击并执行拖放。

这种 UI 行为并不是 Flex 独有的，Windows 应用程序中一般都支持这种功能。

我们迄今展示的只是单向 D&D 操作，一边是数据源，另一边是目的地。Flex 也支持双向拖放。

21.7 双向拖放

拖放不一定只能将一个组件的项移动或复制到另一个组件，如果能够实现双向拖动一定会更有用。在 Flex 中，这不是个问题，具体做法下面就来介绍。

把有关组件的 dragEnabled 和 dropEnabled 属性都设置为 true。另外，如果想让默认行为是移动而非复制的话，别忘了把 dragMoveEnabled 属性也设置为 true。

代码清单 21-4 所示为一个虚构的离婚应用程序，让用户来分割家庭财产，决定谁该拿走什么。因此，3 个列表中的项是可以来回拖动的。

代码清单21-4 在3个List组件间拖动和放置

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Script>
 <![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var allAssets:ArrayCollection  = new ArrayCollection([
  {label:"House"},{label:"BMW 745i"},{label:"Lexus IS350"},{label:"Stocks"}
  ]);
 ]]>
</mx:Script>

<mx:Panel title="DivorceNET" width="300">

   <mx:Label text="Assets"/>
   <mx:List id="listAll" dataProvider="{allAssets}" width="100%"
      height="90"
           dragEnabled="true" dragMoveEnabled="true"
              dropEnabled="true"/>

   <mx:HBox width="100%">
      <mx:VBox width="50%">
         <mx:Label text="His"/>
         <mx:List id="listHis" width="100%" height="90"
              dragEnabled="true" dragMoveEnabled="true"
                dropEnabled="true"/>
      </mx:VBox>

      <mx:VBox width="50%">
         <mx:Label text="Hers"/>
         <mx:List id="listHers" width="100%" height="90"
              dragEnabled="true" dragMoveEnabled="true"
                dropEnabled="true"/>
      </mx:VBox>

   </mx:HBox>
</mx:Panel>

</mx:Application>
```

在把财产从一个人名下转移到另一个人名下后，财产就从来源处被删除。当然，还可以把财产再拖回去。

双向拖放的另一个特点是，可以在组件内部进行拖放。

21.8 使用拖放实现用户控制的排序

D&D 不仅可以用来从一个组件向另一个组件移动数据，而且可以用来在同一个组件中重新排列数据。不知道读者有没有开发或看到如图 21-3 所示的应用程序。

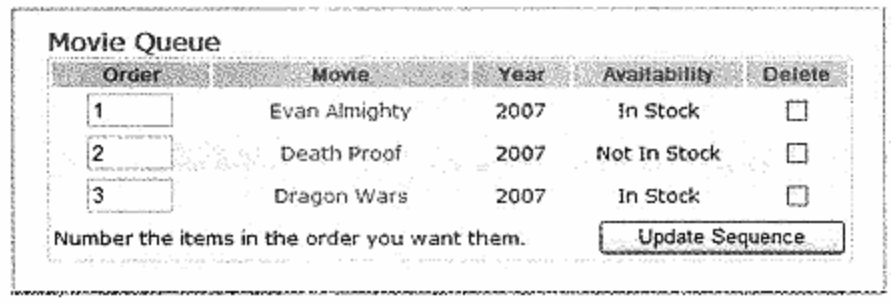


图 21-3 在标准 HTML 中，没有 D&D，因此用户要对一系列项进行排序很麻烦

从可用性的角度看，随着数据量的增加，处理列表内容排序的工作量也会同步增多。而使用 D&D，则可以为用户节省大量时间。

按照下面所示设置组件的属性，可以实现组件内部的拖放。

- dragEnabled="true"
- dropEnabled="true"
- dragMoveEnabled="true"

代码清单 21-5 所示为让 DataGrid 同时支持拖动和放置的例子。

代码清单21-5 允许在同一个组件中拖放，以便进行排序

```
<mx:DataGrid id="dgSource" dataProvider="{myAC}" dragEnabled="true"
                    dropEnabled="true" dragMoveEnabled="true" >
   <mx:columns>
      <mx:DataGridColumn dataField="name" headerText="Contact Name"
         width="300" />
      <mx:DataGridColumn dataField="email" headerText="E-Mail"
         width="200"/>
      <mx:DataGridColumn dataField="url" headerText="URL" width="200"/>
   </mx:columns>
</mx:DataGrid>
```

运行这个应用程序，用户将看到如图 21-4 所示的界面，并可以控制联系人的顺序。

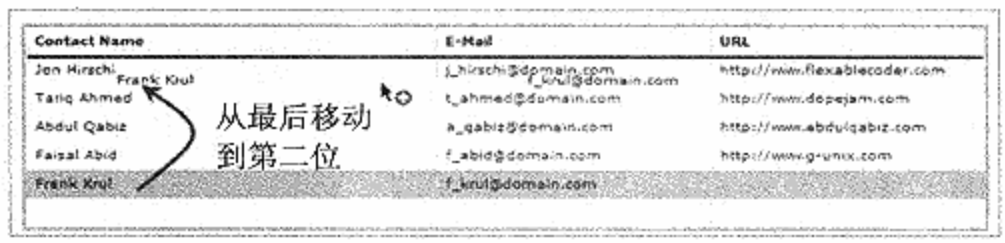


图 21-4 在 DataGrid 本身使用 D&D，用户可以方便地控制每一项的位置

这样就可以自行添加 D&D，即组件能够利用其内置的 D&D 行为。生活要总是那么简单就好了！在真正的项目中，有时候还要加入业务逻辑和业务规则，此时就要用到 DragManager。

21.9 使用 DragManager

DragManager 是一个全局对象，随时可以满足我们的需要——它在后台负责对 D&D 行为的控制和处理。无论什么时候需要处理某个对象，都可以获得 DragManager 的帮助。

要实现自定义处理，必须使用 DragManager，以下几节将讨论相关内容。但在此之前，有必要先概述一些关键的属性。

21.9.1 操作值

DragManager 有 4 个静态值(也称常量，即值永远不变的变量)，可以通过这个 4 个值确定用户当前的操作(例如移动还是复制)并为用户提供视觉反馈。表 21-3 所示为这几个值。

表 21-3 DragManager 中定义的常量

值	说 明
COPY	当前操作是复制被拖动的项。基于列表的组件假设用户按住 Ctrl 键，就是在复制(如果 dragMoveEnabled="true")
LINK	当前操作是链接——类似于复制，但不是创建副本，而是把多个项链接到一个源。这个操作在按住 Shift 键时发生
MOVE	当前操作是把拖动的项从一个位置移动到另一个位置
NONE	没有操作。操作被拒绝

这些值(也就是常量)用于支持函数。

21.9.2 DragManager 函数

在自定义 D&D 时，还需要利用 3 个重要的函数(也称方法)。

- doDrag()：启动 D&D 过程。通常在 mouseMove 事件处理程序中启动。
- acceptDragDrop()：让 DragManager 知道是否应该继续 D&D 过程。在 dragEnter 事件处理程序中调用。
- showFeedback()：就当前操作值(COPY、LINK、MOVE 或 NONE)为用户提供反馈。
- 有了 DragManager 的常量和支持函数，就有了接受或拒绝放置操作的必要工具。

21.10 接受或拒绝放置

前面讨论的内容如果应用到不受控制的 D&D 中，还是很好很有效的。不过在真正的项目开发中，下一个必然要考虑的问题，是添加业务逻辑以决定是否允许在给定组件上发生放置操作。常见的情况包括：在复制前先行检查，以免用户重复添加已经存在的项。

21.10.1 限制添加项

通过 dragEnter 事件，可以添加相应的逻辑，以确定是否接受放置。这个重要的事件在鼠标移动到组件上时触发。

下面将给出一个示例，让放置目标检查记录项，只有记录项处于活动状态时，才接受放置。以下是代码清单 21-6 中的主要逻辑过程。

(1) 为可能接受放置的 DataGrid 添加 dragEnter 事件处理程序。

(2) 这个事件处理程序会通过 DragEvent 事件来了解被拖动的数据。

数据格式可以是任何形式的对象数组。即使只拖动了一项，DragManager 也支持多项——因为可能接收多项(例如，启动组件设置了 allowMultipleSelection="true")。

(3) 检查数据，看是否有活动的项(检查 isActive 的值是否为 false)。

(4) 如果所有项都是活动的，则告诉 DragManager 可以继续；否则拒绝放置。

代码清单21-6 只允许放置活动项

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
<mx:Script>
 <![CDATA[
  import mx.collections.ArrayCollection;
  import mx.events.DragEvent;
  import mx.managers.DragManager;

  [Bindable]
  public var myAC:ArrayCollection  = new ArrayCollection([
  {name:"Jon Hirschi", email:"j_hirschi@domain.com", isActive:true},
  {name:"Tariq Ahmed", email:"t_ahmed@domain.com", isActive:false},
  {name:"Abdul Qabiz", email:"a_qabiz@domain.com", isActive:true}]);

    public function handleDragEnter(event:DragEvent):void
    {
       var arItems:Array= event.dragSource.dataForFormat("items")
          as Array;
       var isAllowed:Boolean = true;
       for (var i:int = 0; i < arItems.length; i++)
          if(arItems[i].isActive == false)
             isAllowed = false;
       if (isAllowed)
          DragManager.acceptDragDrop(dgTarget);
       else
       {
          DragManager.showFeedback(DragManager.NONE);
          event.preventDefault();
       }
    }

 ]]>
</mx:Script>
<mx:DataGrid id="dgSource" dataProvider="{myAC}"
         dragEnabled="true" dragMoveEnabled="true" >
   <mx:columns>
      <mx:DataGridColumn dataField="name" headerText="Contact Name"
         width="300" />
      <mx:DataGridColumn dataField="email" headerText="E-Mail"
         width="200"/>
      <mx:DataGridColumn dataField="isActive" headerText="Active"
         width="75"/>
   </mx:columns>
</mx:DataGrid>
```

获得移过的项的数组

检查每个项的状态

```
<mx:DataGrid id="dgTarget" dropEnabled="true"
   dragEnter="handleDragEnter(event)" >
   <mx:columns>
      <mx:DataGridColumn dataField="name" headerText="Contact Name"
         width="300" />
      <mx:DataGridColumn dataField="email" headerText="E-Mail"
         width="200"/>
      <mx:DataGridColumn dataField="isActive" headerText="Active"
         width="75"/>
   </mx:columns>
</mx:DataGrid>

</mx:Application>
```

运行这个示例，在尝试移动用户 Tariq 时，第二个 DataGrid 会拒绝放置。

如果 DragManager 未发现调用 acceptDragDrop 函数，则 D&D 的其余过程不会发生(因为不会触发 dragOver 和 dragDrop 事件)。

21.10.2 阻止事件传播

在前面的示例中，设置 dropEnabled="true"意味着 Flex 会执行其默认事件处理程序。即使为 dragEnter 指定了事件处理程序，在该函数运行后，Flex 也会执行自己的函数。

```
<mx:DataGrid id="dgTarget" dropEnabled="true"
   dragEnter="handleDragEnter(event)" >
```

但是，如果不想接受放置，就要阻止传播链中的其他事件。明确地讲，要避免执行 Flex 为这些事件定义的默认事件处理程序。这就是前面示例中调用 preventDefault()函数的原因。

```
else
{
   DragManager.showFeedback(DragManager.NONE);
   event.preventDefault();
}
```

如果不调用该函数，那么在函数执行后(无论是否接受放置)，默认的处理程序也将执行(假设数据符合要求)。

21.10.3 使用 DragEvent 对象找到放置目标

在调用 acceptDragDrop()时，告诉它 DataGrid(dgTarget)是要接受放置的组件。为了让这个函数更通用，可以访问 DragEvent 的 currentTarget 属性，因为该属性引用的是放置对象。

在 API 参考中查阅 DragManager 的 acceptDragDrop()函数，会发现该函数接收一个实现了 IUIComponent 接口类(所有 UI 组件的基类)的对象。

此时，如果只传入 currentTarget，Flex Builder 会给出一个类型不匹配的提示(见图 21-5)。因为在编译之前，currentTarget 可能引用任何对象。

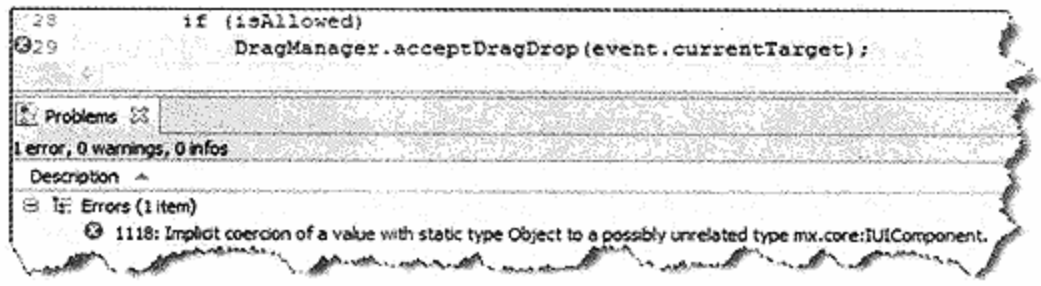


图 21-5 尽管 currentTarget 在运行时会指向一个 UI 组件，但 Flex Builder 编译器需要精确的类型匹配

解决方案很简单：创建一个与放置目标类型相同的对象。代码清单 21-7 所示为如何创建一个 DataGrid(是一个 UI 组件)类型的临时变量 dropTarget，然后将 dropTarget 传递给 acceptDragDrop()函数。

代码清单21-7 将currentTarget转换成一个UI组件

```
public function handleDragEnter(event:DragEvent):void
{
    .
    .
    var dropTarget:DataGrid=DataGrid(event.currentTarget);
    .
    .
    if (isAllowed)
        DragManager.acceptDragDrop(dropTarget);
    else
        DragManager.showFeedback(DragManager.NONE);
}
```

如果需要在 dropTarget 对象上执行很多操作(也许是通过修改其颜色来表明接受还是拒绝放置)，那么这应该是一个非常不错的方案。

更简单、便捷的方式则是告诉 Flex，currentTarget 指向的对象遵从 IUIComponent 接口。

```
DragManager.acceptDragDrop(event.currentTarget as IUIComponent);
```

可以依赖组件内置的方法处理放置操作；但也可以重写该方法，从而应用自定义方法。

21.11 应用自定义放置方式

接下来，我们要考虑如何自定义放置操作；这涉及将数据直接保存到组件的 dataProvider 或绑定组件的变量中。例如，在放入数据项时，把相应数据保存到数据库中。

为保持代码简短，我们扩展了前一示例，但只给出新增的代码。

21.11.1 显示添加到组件的 dataProvider

应该添加一个 dragDrop 事件处理程序，用于接收移动的数据并将数据复制到组件的 dataProvider 中。由于 dataProvider 有可能什么类型也不是(默认值为 null)，因此需要检查一下并在必要时对其初始化(见代码清单 21-8)。

代码清单21-8 通过创建函数直接将数据添加到组件中

```
<mx:Script>
 <![CDATA[
   public function handleDrop(event:DragEvent):void
   {
    var arItems:Array= event.dragSource.dataForFormat("items")
        as Array;
    if(dgTarget.dataProvider==null)
     dgTarget.dataProvider = new ArrayCollection();

    for(var i:int=0; i<arItems.length;i++)
     dgTarget.dataProvider.addItem(arItems[i]);
   }
 ]]>
</mx:Script>
<mx:DataGrid id="dgTarget" height="100"
            dragEnter="handleDragEnter(event)"
            dragDrop="handleDrop(event)">
  <mx:columns>
   <mx:DataGridColumn dataField="name" headerText="Contact Name"
        width="200" />
   <mx:DataGridColumn dataField="email" headerText="E-Mail"
        width="200"/>
   <mx:DataGridColumn dataField="isActive" headerText="Active"
        width="75"/>
  </mx:columns>
</mx:DataGrid>
```

将被拖动的数据作为数组进行访问

添加到放置目标的 ArrayCollection

如果读者感觉操作放置操作源的 dataProvider 不太好，也可以采用隐式方式。

21.11.2 隐式添加到组件的 dataProvider

另一种方式是创建一个变量，让基于列表的组件绑定到该变量，而在事件处理程序中操作这个变量。这样，就不必与组件紧密耦合，也体现了以数据为中心的思想。此外，因为不用管使用数据的组件，只需操作 ArrayCollection，所以 dragDrop 事件处理程序会更简单。

代码清单 21-9 定义了相同的 handleDrop()函数，但这次是操作 ArrayCollection——可以用于它了解的任意数量的组件。

代码清单21-9 操作驱动组件显示的变量

```
<mx:Script>
 <![CDATA[
   [Bindable]
   public var targetAC:ArrayCollection = new ArrayCollection();

   public function handleDrop(event:DragEvent):void
   {
    var arItems:Array= event.dragSource.dataForFormat("items") as Array;
    for(var i:int=0; i<arItems.length;i++)
         targetAC.addItem(arItems[i]);
   }
```

```
]]>
</mx:Script>
<mx:DataGrid id="dgTarget" height="100" dataProvider="{targetAC}"
            dragEnter="handleDragEnter(event)"
            dragDrop="handleDrop(event)">
   <mx:columns>
      <mx:DataGridColumn dataField="name" headerText="Contact Name"
         width="200" />
      <mx:DataGridColumn dataField="email" headerText="E-Mail"
         width="200"/>
      <mx:DataGridColumn dataField="isActive" headerText="Active"
         width="75"/>
   </mx:columns>
</mx:DataGrid>
```

使用有关自定义放置的知识，加上对 DragManager 的了解，还可以更上一层楼。下一节，我们介绍为本来不支持 D&D 的组件赋予 D&D 功能。

21.12 为非列表组件添加拖动行为

基于列表的组件可以自行处理所有 D&D 过程。但是，要为非列表组件赋予 D&D 行为，基本上就要靠我们自己来添加处理逻辑了。

为此，必须利用前几节提到的事件(dragEnter 和 dragDrop)，不过也需要用到另外一些事件，特别是 mouseMove 和 dragComplete。

21.12.1 设置示例

本节示例将展示购物车的基本概念，即通过把商品拖放到购物车中来购买商品(见图 21-6)。

这个示例的确很简单，但能够说明问题。在真正的项目中，可能需要查询数据库获得商品的详细信息(价格、名称、说明和缩略图等)，然后再构建一个包含详细信息的列表(可以使用 HorizontalList 和 DataGrid)。

为了简单起见，我们在此使用 Button 组件；不过使用 Image 组件也是可以的。在明确了要实现的拖放过程后，下面我们就从启动拖动操作开始。

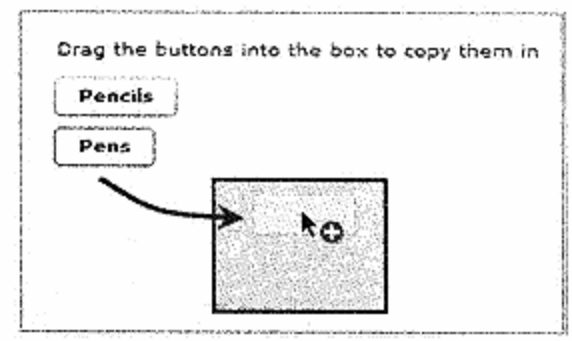


图 21-6 按钮本身不支持 D&D，但可以轻松地为其添加该功能

21.12.2 启动拖动

为启动 D&D 过程，要利用 mouseMove 事件并创建一个事件处理程序(通过 DragManager 实现)。这个拖动启动器函数主要执行 3 个任务。

- 创建一个对拖动启动器的引用。
- 创建一个拖动源对象，包含要通过 DragManager 传递的所有数据。
- 调用 DragManager 的 doDrag()函数，为其传递拖动启动器的引用、拖动源对象及由用户创建的 MouseEvent 事件对象。

代码清单 21-10 所示为这个函数。

代码清单21-10 用于启动拖动的函数

```
public function initiateDrag(event:MouseEvent):void
{
  var dragInitiator:Button=Button(event.currentTarget);
  var ds:DragSource = new DragSource();
  ds.addData(dragInitiator.label,"name");
   DragManager.doDrag(dragInitiator, ds, event);
}
```

对被拖动按钮的引

创建拖动源对象

添加要传递的数据

启动拖动

这是一个好的开始！无论如何，已经成功一半了，另一半是添加放置逻辑。在拖动和放置这两个步骤之间，可以选择是否对用户给出视觉反馈。

21.12.3 添加视觉反馈

这一步做不做皆可，但我们还是做一下。在 21.10.1 节中，dragEnter 事件处理程序曾对用户给出视觉反馈，当时是告诉用户要拒绝相应的操作。在本示例中，我们假设允许任意拖动和放置。紧随 dragEnter 事件之后的是 dragOver 事件，dragOver 事件是添加视觉反馈的恰当时机。尽管代码是执行简单的复制操作，但如果你想支持其他操作，或者想添加自定义的 D&D 业务逻辑，那么也可以在此一并完成。

在 Flex 中，基于列表的组件会在有拖动项悬停其上时自动改变边框颜色。颜色变化是对用户的一种视觉提示，让用户知道拖动项是否得到潜在放置目标的许可。不过，这种隐含的行为在按住 Shift 或 Ctrl 键时会有所不同。

代码清单 21-11 重现了上述视觉反馈行为，根据用户操作相应地改变了放置目标边框的颜色。

代码清单21-11 以不同方式对用户给出视觉提示

```
public function handleDragOver(event:DragEvent):void
{
if (event.ctrlKey)
  {
      DragManager.showFeedback(DragManager.COPY);
      event.currentTarget.setStyle('borderColor', 'blue');
  }
  else if (event.shiftKey)
  {
    DragManager.showFeedback(DragManager.LINK);
     event.currentTarget.setStyle('borderColor', 'green');
  }
  else
  {
      DragManager.showFeedback(DragManager.MOVE);
      event.currentTarget.setStyle('borderColor', 'yellow');
  }
}
```

之所以这样实现，还有一个不显眼但很关键的原因：在调用 showFeedback()函数时，该函数会把 DragEvent.action 属性设置为相应的操作。这样一来，后面需要知道具体操作

的事件处理程序就比较方便了。

21.12.4 处理放置

因为使用的不是基于列表的组件，所以没有处理放置操作的默认行为。在本示例中，需要访问放在拖动源对象中的数据并基于该数据创建一个按钮，然后把其放在表示购物车的盒子内。此后，还应该恢复盒子边框的初始颜色。

在此我们只看如何处理移动操作。首先要检查一下用户是否想执行移动操作，如果是，则销毁原始对象(见代码清单 21-12)。

代码清单21-12 添加支持移动D&D的逻辑

```
public function handleDrop(event:DragEvent):void
{
  var newButton:Button = new Button();
  newButton.label = event.dragSource.dataForFormat("name") as String;
  ShoppingCart.addChild(newButton);
  ShoppingCart.setStyle('borderColor','black');

  if(event.action == DragManager.MOVE)
    Catalog.removeChild(event.dragInitiator as Button);
}
```

创建按钮

使用传入的数据

添加到表示购物车的VBox中

如果移动则销毁源

此外，另一个适合处理删除逻辑的地方，是在 dragComplete 事件处理程序中(是由拖动启动器生成的事件)。

```
<mx:Button label="Pencils" mouseMove="initiateDrag(event)"
   dragComplete="handleDragComplete(event)"/>
```

这样就完成了本示例所需的大部分操作。不过，在自定义 D&D 时，还要考虑最后的清理工作，即要添加处理退出的代码。

21.12.5 处理退出

由于前面的 dragOver 事件改变了购物车的边框颜色，因此还要在用户鼠标离开放置目标并释放鼠标键时恢复原来的边框颜色。

```
public function handleDragExit(event:DragEvent):void
{
  ShoppingCart.setStyle('borderColor','black');
}
```

至此，为不支持 D&D 的组件添加 D&D 行为的所有代码都已经就绪。接下来，我们把这些代码放到一起。

21.12.6 综合起来

整个过程看起来似乎有点复杂，但这很大程度上是因为我们单独描述了每个阶段的相关细节。如果把所有函数集中起来，读者就会发现，原来自定义 D&D 的代码也不是很多。代码清单 21-13 完整展示了本节示例。

代码清单21-13 为非列表组件添加D&D行为

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
backgroundColor="white">
<mx:Script>
 <![CDATA[

 import mx.core.DragSource;
 import mx.managers.DragManager;
 import mx.events.*;
 import mx.controls.Button;
 import mx.collections.ArrayCollection;

  [Bindable]
  public var myAC:ArrayCollection  = new ArrayCollection([
   {name:"Pencil", price:1.25},{name:"Pen", price:1.78}]);

  public function initiateDrag(event:MouseEvent):void
  {
      var dragInitiator:Button=Button(event.currentTarget);
      var ds:DragSource = new DragSource();
      ds.addData(dragInitiator.label,"name");
      DragManager.doDrag(dragInitiator, ds, event);
  }

  public function handleDragEnter(event:DragEvent):void
  {
     DragManager.acceptDragDrop(event.currentTarget as VBox);
  }

  public function handleDragOver(event:DragEvent):void
  {
   if (event.ctrlKey)
   {
       DragManager.showFeedback(DragManager.COPY);
         event.currentTarget.setStyle('borderColor', 'blue');
   }
   else if (event.shiftKey)
   {
     DragManager.showFeedback(DragManager.LINK);
         event.currentTarget.setStyle('borderColor', 'green');
   }
   else
   {
     DragManager.showFeedback(DragManager.MOVE);
     event.currentTarget.setStyle('borderColor', 'yellow');
   }
  }

  public function handleDrop(event:DragEvent):void
  {
     var newButton:Button = new Button();
     newButton.label = event.dragSource.dataForFormat("name")
        as String;
      ShoppingCart.addChild(newButton);
      ShoppingCart.setStyle('borderColor','black');
      if(event.action == DragManager.MOVE)
         Catalog.removeChild(event.dragInitiator as Button);
  }
```

创建拖动源对象

创建对被拖动按钮的引用

启动拖动

添加要传递的数据

```
        public function handleDragExit(event:DragEvent):void
        {
            ShoppingCart.setStyle('borderColor','black');
        }
    ]]>
</mx:Script>
<mx:VBox id="Catalog">
        <mx:Label text="Drag the buttons into the box to copy them in"/>
        <mx:Button label="Pencils"  mouseMove="initiateDrag(event)"/>
        <mx:Button label="Pens" mouseMove="initiateDrag(event)"/>
</mx:VBox>
<mx:VBox id="ShoppingCart" borderColor="black" borderThickness="2"
        dragEnter="handleDragEnter(event)"
        dragOver="handleDragOver(event)"
      dragDrop="handleDrop(event)"
      dragExit="handleDragExit(event)"
       width="94" height="76" backgroundColor="#D7FE2E"
          borderStyle="solid"/>
</mx:Application>
```

如果读者从头到尾亲手完成了这个示例，给自己鼓鼓掌！这应该是本书里比较大的一个示例了。实际上，把这个示例分解为处理整个 D&D 过程中各种事件的小函数，理解起来会容易得多。

21.13 自定义拖放体验

本章最后我们看一看在使用 D&D 时如何发挥一下创造性。换言之，可以使用自定义的图标对用户的各种手势(gesture)给出反馈。首先要修改拖动图像。

21.13.1 修改拖动图像

正常情况下，Flex 会显示被拖动组件的轮廓线。可以将这个轮廓线修改成任何图像。这就要用到 DragManager 的 doDrag()函数，即启动拖动操作的函数。在前面示例中使用 doDrag()函数时，只为它传递了必需的参数。

```
DragManager.doDrag(dragInitiator, ds, event);
```

在 Flex API 参考中查阅 DragManager 类，会发现该函数的另外一些参数，特别是有一个参数名叫 dragImage。通过这个参数可以修改拖动图像。

因此下一步是创建一幅图像并将其传入 doDrag()函数。代码清单 21-14 借用了代码清单 21-13 中自定义 D&D 示例的一些代码。

代码清单21-14 创建自定义拖动图像

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
   <mx:Script>
   <![CDATA[
   import mx.managers.DragManager;
   import mx.core.DragSource;
```

```
    import flash.events.MouseEvent;
    import mx.controls.Image;

    public var imgCopy:Class;
    [Embed(source='assets/dragimage.png')]
    public var imgDrag:Class;

    public function initiateDrag(event:MouseEvent):void
    {
      var dragInitiator:Button=Button(event.currentTarget);
      var ds:DragSource = new DragSource();
      var dragImage:Image = new Image();
      var xOffset:int = -5;
      var yOffset:int = -5;
      var alphaLevel:Number = 1.00;
      dragImage.source =`imgDrag;
      dragImage.height=59;
      dragImage.width=57;
      DragManager.doDrag(dragInitiator, ds, event,
                    dragImage,xOffset,yOffset,alphaLevel);
    }
    ]]>
    </mx:Script>
    <mx:Button label="Pencils"  mouseMove="initiateDrag(event)"/>
  </mx:Application>
```

当用户拖动按钮时(见图 21-7)，会出现一个新颖的图标而不是通常的灰线框。这个示例的工作过程如下。

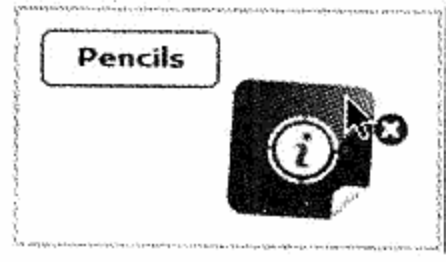


图 21-7 使用自定义的拖动图像自定义体验

(1) 将一幅图像以全局变量形式导入应用程序。

(2) 当拖动将要发生时，基于图像创建 Image 对象。

(3) 把这个 Image 对象连同其他(可选的)参数传入 doDrag()函数。其他参数控制图像相对于鼠标指针的 X 和 Y 偏移量以及透明度级别等。

另外也可以控制用于向用户传达有关拖动操作的信息(移动、复制和拒绝等)的图像。

21.13.2 修改拖动代理图标

前面曾提到过，拖动代理是在 D&D 过程中为用户给出的视觉反馈。最简单的修改方式是使用 CSS/样式(有关样式的更多信息参见第 19 章)。

体现在代码上，就是使用 CSS 声明来定义每种情况下拖动代理使用哪些图像，如代码清单 21-15 所示。

代码清单21-15 重写默认的拖动代理图像

```
<mx:Style>
 DragManager
 {
  copyCursor: Embed(source="assets/copy.png");
  linkCursor: Embed(source="assets/link.jpg");
  moveCursor: Embed(source="assets/move.png");
  rejectCursor: Embed(source="assets/reject.jpg");
 }
</mx:Style>
```

显然这比修改拖动图像容易多了，而且它利用的还是 CSS 技术。

21.14 小结

拖放确实很有意思，因为它可以复杂到我们能够想得出来的任何程度。不过，Flex 基本上已经承担了所有难处理的工作，我们要做的不过就是在适当时机插入自定义业务逻辑而已。

基于列表的组件内置支持大多数工作，开发人员可以不捕获任何事件，也可以捕获一部分或全部 D&D 事件。对于非列表组件而言，DragManager 是实现 D&D 的关键。Flex 会调用与拖动相关的事件处理程序，而且每次调用都会传递 DragEvent 对象，以便开发人员获得需要操作的信息。

别忘了通过自定义用户体验让应用程序与众不同。另外，一定要把可用性作为使用 D&D 的出发点，通过便捷的拖放来为用户节省时间。

第 22 章，将介绍不同的主题——图表。贯穿可用性这条主线，正如 D&D 可以增加界面可用性一样，图表能够提升信息的可用性。

第 22 章 图 表

22

本章内容：

- 理解图表构成
- 使用图表系列
- 图表类型
- 自定义图表

读者如果想找个卖点向公司推荐 Flex，那么图表一定是合适的选择。管理层喜欢图表，让老板体验一下 Flex 图表，相信他们很快就会爱不释手并请你多介绍一些特性。

Flex 图表之所以那么招人喜爱，与静态图表(包括带动画的图表)不同，主要是通过 Flex 能够创建交互式的、迷人的图表板(dashboard)风格的体验。尤其是在图表中加入效果的情况下更是如此。

本章将探讨图表的各个组成部分、Flex 内置的图表以及如何自定义图表。首先介绍一下图表。

22.1 图表简介

Flex Builder 的 Professional 版支持图表特性。不过，即使读者没有购买 Professional 版，仍然可以构建图表，只不过图像中会显示带有 Flex Charting Trial 字样的水印。

从开发人员角度来看，Flex 的图表引擎极为灵活。可以控制每个数据点的显示，构建功能强大的图表板，使用多轴，以及支持各种形式的用户交互。

例如，用户单击某个数据点会触发事件，通过事件处理程序可以打开另一个图表或者载入与该数据点有关的数据。可以通过编程实现图表的拖放操作，让用户能够围绕一组数据点做出选择。

在 Flex 中，所有图表都派生自 CartesianChart 类或 PolarChart 类。CartesianChart 类基于笛卡尔坐标系构建，通过水平、垂直坐标轴的上 X、Y 坐标来定义数据点。PolarChart 类用于定义圆形或放射形空间内的区域，派生于此类的唯一图表类型是饼图。

22.1.1 图表的构成

图表由多个部分构成，这些部分以 MXML 组件和 ActionScript 类的形式存在。表 22-1 所示为构成图表的各个部分。

表 22-1 图表的构成

图表的组成部分	说 明
图表	图表本身，用于联系各个组成部分的框架
系列	相关数据点的集合
轴	水平和垂直坐标轴。Flex 图表只支持二维图形，但可以有多个坐标轴
轴渲染器	控制线条和标签显示的一种强大机制
元素	显示在图表区域内的其他任何东西。例如，可以在某些点上显示特殊图标(以及网格线和注释等)
标签	说明图表坐标轴和系列的文本信息

利用这些为数不多的要素，可以为用户创建出丰富多样的图表。

22.1.2 图表类型简介

Flex 内置了 9 种图表，如表 22-2 所示。

表 22-2 Flex 3 中的图表类型

类 型	MXML 组件和类	系列组件和类
面积图(Area)	AreaChart	AreaSeries
条形图(Bar)	BarChart	BarSeries
气泡图(Bubble)	BubbleChart	BubbleSeries
烛台图(Candlestick)	Candlestick	CandlestickSeries
柱状图(Column)	ColumnChart	ColumnSeries
股价图(HLOC)	HLOCChart	HLOCSeries
折线图(Line)	LineChart	LineSeries
饼图(Pie)	PieChart	PieSeries
散点图(Plot)	PlotChart	PlotSeries

我们知道读者想早点使用图表。不过在介绍图表用法之前，最好先了解一下本章的示例。假设有一家名为 FlexMoto 的外资汽车公司，而我们要为该公司生成一些销售图表板。

22.2　准备系列和数据

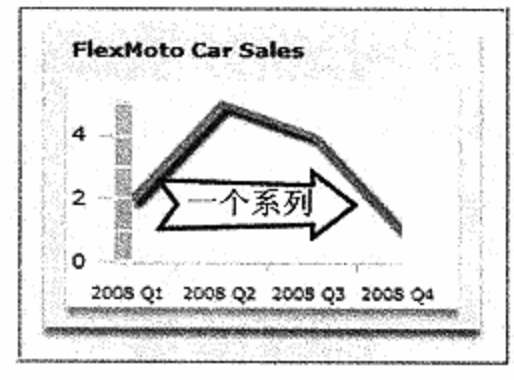


图 22-1　此图表通过一个系列展示了每季度的汽车销售量

要创建图表，首先必须知道图表如何显示数据以及如何简化显示。

图表显示的相关数据集合称为系列，每个图表至少有一个系列。系列常见的用途是按照既定的时间段显示财务数据。例如，月度销售额或季度销售收入。

系列是图表的核心组成部分，每种图表类型都有相应的系列组件/类。图 22-1 展示了一幅折线图，其中包含一个系列，该系列在一年中的每个季度有一个数据点。

系列只包含数据，而且通常是表格形式的数据。在 FlexMoto 的示例中，假设有表 22-3 所示的数据。

表 22-3　示例中使用的 FlexMoto 数据

Interval(时段)	FerrariUnits	FerrariSales	PorscheUnits	PorscheSales
2008 Q1	2	2.32	2	0.180
2008 Q2	5	7.82	3	0.240
2008 Q3	4	6.92	7	0.420
2008 Q4	1	1.80	12	1.180

在真实的项目中，应该通过诸如 Web 服务或远程对象的后端数据服务从数据库检索数据(参见第 14 章)，然后再将数据赋给 ArrayCollection。结果与代码清单 22-1 的硬编码示例类似的 ArrayCollection(读者明白这个意思就行了)。

代码清单22-1　ArrayCollection形式的FlexMoto销售数据

```
salesAC:ArrayCollection = new ArrayCollection( [
{interval: "2008 Q1", ferrariUnits:2, ferrariSales:2.32, porscheUnits:2,
   porscheSales: 0.180},
{interval: "2008 Q2", ferrariUnits:5, ferrariSales:7.82, porscheUnits:3,
   porscheSales: 0.240},
{interval: "2008 Q3", ferrariUnits:4, ferrariSales:6.92, porscheUnits:7,
   porscheSales: 0.420},
{interval: "2008 Q4", ferrariUnits:1, ferrariSales:1.80, porscheUnits:12,
   porscheSales: 1.180}
]);
```

此外 XML 的数据格式也是比较流行的一种选择。本例将使用 XML 格式。代码清单 22-2 所示为数据的 XML 版本。

代码清单22-2　data.xml：XML格式的FlexMoto销售数据

```
<salesdata>
 <all>
  <data interval="2008 Q1" ferrariUnits="2" ferrariSales="2.32"
                   porscheUnits="2" porscheSales="0.180"/>

  <data interval="2008 Q2" ferrariUnits="5" ferrariSales="7.82"
                   porscheUnits="3" porscheSales="0.240" />
```

```
    <data interval="2008 Q3" ferrariUnits="4" ferrariSales="6.92"
                     porscheUnits="7" porscheSales="0.420"/>

    <data interval="2008 Q4" ferrariUnits="1" ferrariSales="1.80"
                     porscheUnits="12" porscheSales="1.180"/>
  </all>
</salesdata>
```

有了数据后，就可以着手创建图表了。

22.3 创建图表

图表有一个方便的特性，即可以先创建一个图表，如果不喜欢的话，修改为其他图表类型也很容易。本节将介绍如何创建图表，如何添加图表要素，直至构成完成。

22.3.1 调用图表

创建图表涉及挑选适当的图表类型和使用其系列组件绘制数据点，而且至少要提供一条水平坐标轴。

提示：

图表是系列和其他要素赖以存在的基础。

下面我们就选择折线图，使用 FlexMoto 销售数据，展示法拉利(Ferrari)和保时捷(Porsches)每季度的销售量(见代码清单 22-3)。

代码清单22-3 以折线图显示不同汽车的销售量

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:XML id="salesDataXML" source="data.xml"/>
  <mx:XMLListCollection source="{salesDataXML.all.data}"
                id="salesData"/>

  <mx:Panel title="FlexMoto Car Sales" width="500" height="250">

    <mx:LineChart id="salesChart" dataProvider="{salesData}"
              height="100%" width="100%" >
      <mx:horizontalAxis>
        <mx:CategoryAxis categoryField="@interval"/>
      </mx:horizontalAxis>

      <mx:series>
        <mx:LineSeries yField="@ferrariUnits"/>
        <mx:LineSeries yField="@porscheUnits"/>
      </mx:series>
    </mx:LineChart>

    </mx:Panel>
</mx:Application>
```

检索并建立数据

建立水平轴以便显示季度

为每个汽车制造商建立系列

编译并运行这个应用程序，将看到与图 22-2 所示类似的包含两条折线的图表。

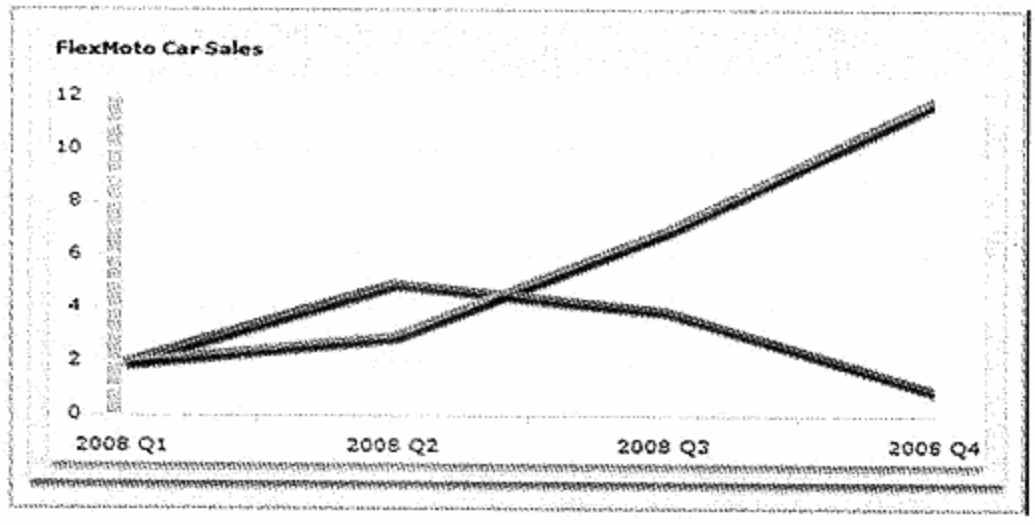


图 22-2　以折线图显示的 FlexMoto 公司每种汽车的销售量

有关这个示例的说明如下。

- Y 轴表示系列中的值，有时候也称为值轴。
- Y 轴默认从 0 开始到完整系列中最大的值结束。
- X 轴表示类别，也称为类别轴。
- X 轴的标签默认会顺着图表宽度均匀分布。

想一想，只用 9 行代码就可以创建一个图表，实际上还是非常简单的。不过，图表还不完整：用户怎么知道每条线的颜色表示什么？显然还需要一个图例。

22.3.2　添加图例

图例用于说明图表中显示的每个系列代表什么意思。如果说图例映射了系列的颜色，在某种程度上是正确的；但是系列不一定只有颜色上的差别。例如，除了颜色之外，系列可能还会以不同透明度、样式(如虚线)甚至纹理(如渐变色)显示。

代码清单 22-4 示范了如何添加图例。

代码清单22-4　添加图例，以便用户了解每条线的含义

```
<mx:Legend dataProvider="{salesChart}"/>
<mx:LineChart id="salesChart" dataProvider="{salesData}"
          height="100%" width="100%" >
  <mx:horizontalAxis>
    <mx:CategoryAxis categoryField="@interval"/>
  </mx:horizontalAxis>
  <mx:series>
    <mx:LineSeries yField="@ferrariUnits"
              displayName="Ferrari Count"/>
    <mx:LineSeries yField="@porscheUnits"
              displayName="Porsche Count"/>
  </mx:series>
</mx:LineChart>
```

以图表作为数据源的图例

displayName字段告知图例为每个系列添加标签

以上代码会在原先的图表上方添加一个图例。

这里应该注意如下问题。

- Legend 组件在图表标签外部。
- 图例使用图表的 id 属性作为其数据源。

- 图例会通过引用检查图表的所有子系列，从中提取每个系列的设计样式(如颜色)和 displayName 属性值。

此时的图表就比较完整了。不过，要是想换成其他图表类型呢？这也不难。

22.3.3 改变图表类型

改变图表类型容易与否，取决于具体实现。不过，即使是最麻烦的情况，也不会有太多工作量。另外，我们只想让读者对转换过程有所了解，因此不会列举所有转换。

想象一下，你已经把 FlexMoto 的销售图表板提交给老板。老板是一个看什么也不顺眼，又特别不喜欢折线图的家伙。于是，他立即要求你换一种图表类型。没问题，换成面积图怎么样，如代码清单 22-5 所示。

代码清单22-5 改用面积图显示FlexMoto的汽车销售量

```
<mx:AreaChart id="salesChart" dataProvider="{salesData}"
          height="100%" width="100%" >
  <mx:horizontalAxis>
    <mx:CategoryAxis categoryField="@interval"/>
  </mx:horizontalAxis>
  <mx:series>
    <mx:AreaSeries yField="@ferrariUnits"
              displayName="Ferrari Count"
              alpha="0.5"/>
    <mx:AreaSeries yField="@porscheUnits"
              displayName="Porsche Count"
              alpha="0.5"/>
  </mx:series>
</mx:AreaChart>
```

alpha属性控制透明度

我们要做的就是把代码中的 line 修改为 area，仅此而已。图 22-3 和图 22-4 所示为面积图的效果。

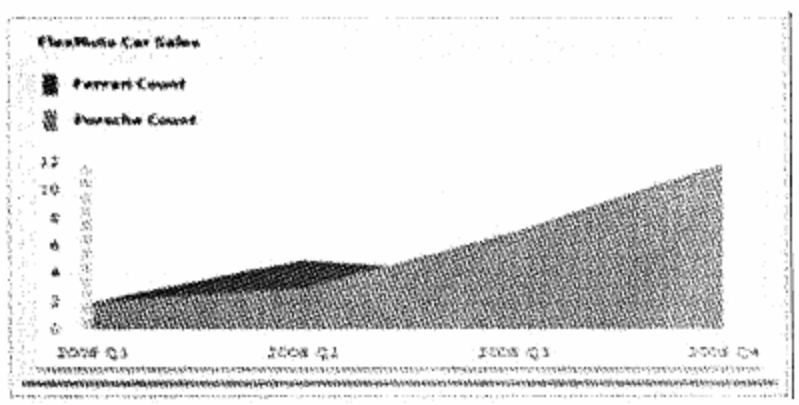


图22-3 由于面积图默认把一个系列放在另一个系列之上，因此只能看清最上层的系列

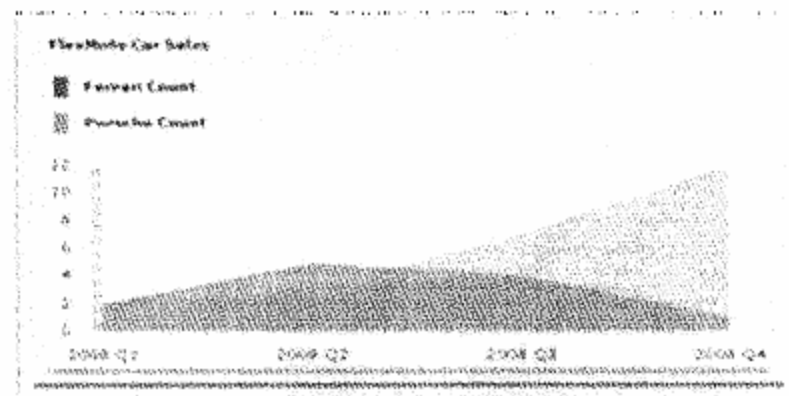


图22-4 通过设置透明度可以看清系列之间的覆盖情况

如图 22-3 所示，一个系列挡住了另一个系列。解决方案是自定义图表，控制系列的透明度(使用 alpha 属性)，结果如图 22-4 所示。本章后面将详细介绍自定义图表。

提示：

这个示例中使用了@字符，因为访问的是 XML 节点的属性。如果数据源是

ArrayCollection，则无需使用@。

倒霉的是，你们老板还看不上眼。他觉得面积图不如条形图——改成条形图虽然也不难，但工作量却要稍微大一些(见代码清单 22-6)。

代码清单22-6　使用垂直轴的条形图

```
<mx:BarChart id="salesChart" dataProvider="{salesData}"
         height="100%" width="100%" >
  <mx:verticalAxis>
     <mx:CategoryAxis categoryField="@interval"/>
  </mx:verticalAxis>
 <mx:series>
   <mx:BarSeries xField="@ferrariUnits"
             displayName="Ferrari Count"/>
   <mx:BarSeries xField="@porscheUnits"
             displayName="Porsche Count"/>
  </mx:series>
</mx:BarChart>
```

必需。因为条形图以Y轴作为类别轴

代码清单 22-6 在创建条形图时同样使用了<BarChart>和<BarSeries>，但条形图使用垂直(Y)轴显示分组，结果如图 22-5 所示。

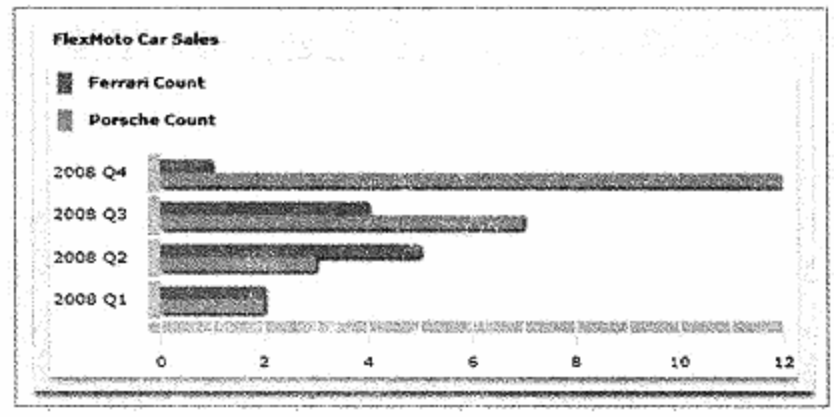


图 22-5　转换为条形图必须把垂直轴作为类别轴

前面示例中使用的图表都支持一种功能，叫做堆叠(stacking)。不过，在图表中堆叠或聚焦(cluster)系列的方式也有很多种。

22.4　堆叠图表

许多图表都支持堆叠系列的功能。在以分类方式展示累积数据时，堆叠功能很有用。下列图表支持堆叠：

- 面积图
- 条形图
- 柱状图

通过设置这些图表的 type 属性可以决定它们支持何种堆叠方式。这个字段可能的值包括如下。

- clustered：按类别聚焦系列中的数据点。这是条形图和柱状图的默认方式，如图 22-4 所示。注意，面积图不支持这种类型。
- overlaid：按照代码中定义的系列顺序以分层方式将一个系列置于另一个系列之上(想象一下洋葱皮)。这是面积图的默认堆叠方式(图 22-3)。用于对比不同的系列。
- stacked：将一个系列堆叠在另一个系列上(就像搭积木)。用于(通过累积)演示每个系列占总体的比例。所有系列累积为一个整体。
- 100%：与 stacked 相同，但每个系列都累积为 100%(类似饼图的原理)。

代码清单 22-5 使用面积图显示销售量，但其默认行为是使用 overlaid 值将每个系列逐层叠加起来。如果想显示每种汽车累积为总销售量的情况，可以把 type 属性修改为 stacked。

```
<mx:AreaChart type="stacked" id="salesChart"
          dataProvider="{salesData}">
. . .
</mx:AreaChart>
```

图所示为修改后的结果。可见，虽然与图 22-6 所示为数据相同，但这次提供的观察角度却不一样。

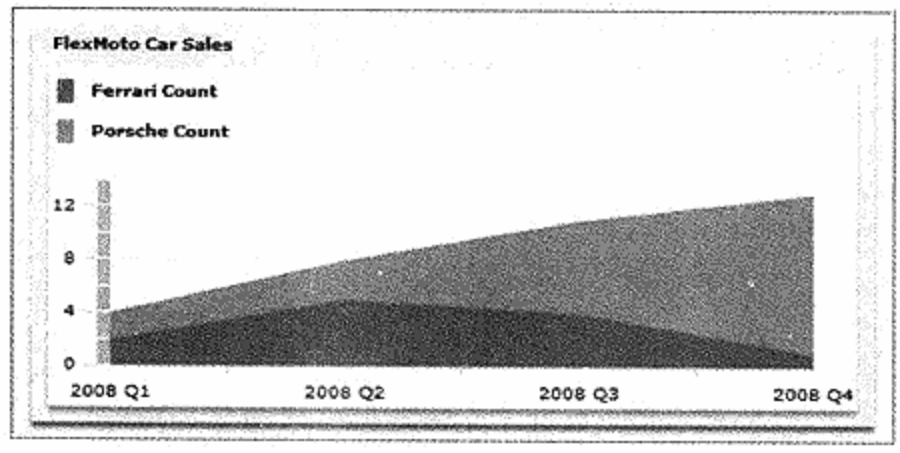


图 22-6 修改面积图的堆叠方式，把每个系列累积为一个整体

类似地，如果把 type 属性修改为其他值，则会转换为另一种观察角度。每种角度都能突出问题的一个方面。

在了解了如何创建图表之后，下面我们来深入讨论 Flex 内置的图表类型。

22.5 探索图表类型

从目前我们讨论过的内容来看，不同类型的图表使用起来并没有多大差别(代码都类似)。但是，有一种情况(条形图)下的差别比较明显，需要关注更多细节。

本节将讨论各种类型的图表及其创建方式，同时会就某些典型情况给出有用的提示。

22.5.1 面积图

面积图与折线图类似，只不过折线下方填充了颜色。通过控制填充区域的透明度，可以让图表以分层方式显示(从视觉上可以看出差别和交迭)，或者以堆叠方式显示(一个系列堆叠在另一个系列上)。虽然面积图与折线图类似，但面积图特别适合表现数量变化的趋势。

在使用 AreaSeries 组件时，可以通过很多参数来控制系列的显示。

其中一个比较有趣的属性是 minField。面积图用于表现数量，而数量可能是某个范围内的值。

注意：

面积图支持堆叠，参见 22.4 节。

假设有一组表示每天最高最低气温的数据。可以像下面这样定义 AreaSeries 组件。

```
<mx:AreaSeries yField="@high" minField="@low" displayName="Temperature"/>
```

如图 22-7 所示，结果填充的是最低值和最高值之间的气温变化范围。

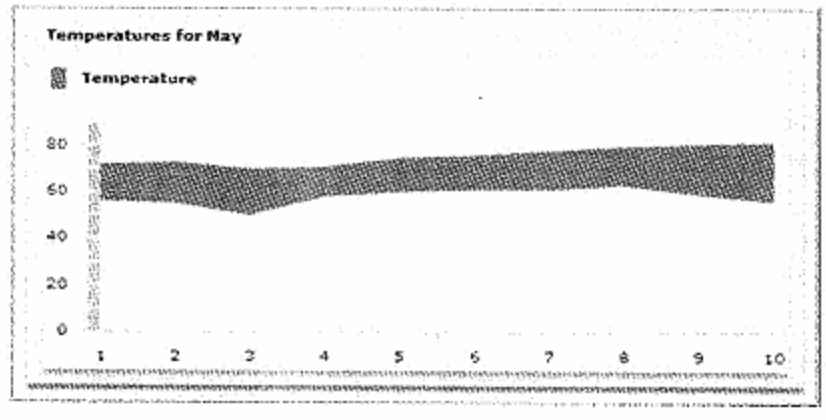


图 22-7　通过 minField 属性可以设置数据点的起始值(起点可为任意值)

这是一种观察数据的独特视角——用户无需关注数值，仅通过形状变化就能了解情况。宽度扩张，表明差异显现；而宽度变窄，表明变化趋小。

利用面积图的堆叠功能，可以通过不同视角来表现数据。例如，overlaid 方式能够突出和对比数据中的差异与共同点，而 stacked 和 100%则可以展示所有系列累积后的效果。

面积图虽然支持上述堆叠方式，但不支持 clustered 系列，而条形图和柱状图支持此系列。

22.5.2　条形图和柱状图

条形图和柱状图实际上是一回事，它们唯一的区别是：条形图水平绘制系列，而柱状图垂直绘制系列。你说奇怪不奇怪，当你对某人说“条形图”时，他的第一反应肯定是 Flex 中的柱状图。

虽然这两种图表可以用于显示趋势信息(随时间改变的信息)，但大多数情况下，人们都使用它们显示分组信息，即按照类别分组的数据。

注意：

条形图和柱状图支持堆叠，参见 22.3 节。

如图 22-5 所示，可以看到以条形图表现的每季度每种汽车的销售情况。条形图非常适合显示这种与时间绑定的数据，但也不止如此。例如代码清单 22-7 所示的数据。

代码清单22-7　candidates.xml：按年龄统计的候选人支持率

```
<candidates>
  <all>
    <data category="18-25" clinton="50" mccain="30" obama="20"/>
    <data category="26-35" clinton="40" mccain="20" obama="40"/>
    <data category="36+" clinton="10" mccain="60" obama="30"/>
  </all>
</candidates>
```

代码清单 22-7 所示为按年龄组(类别)统计的信息，针对每个候选人列出相应的支持率

(百分比)。为了形象地展示这些数据，可以使用条形图，如代码清单 22-8 所示。

代码清单22-8 使用条形图显示分组的信息

```
<mx:Legend dataProvider="{candidateChart}"/>
<mx:BarChart  id="candidateChart" dataProvider="{tempData}"
          height="100%" width="100%" >
  <mx:verticalAxis>
    <mx:CategoryAxis categoryField="@category"/>
  </mx:verticalAxis>
  <mx:series>
    <mx:BarSeries xField="@clinton" displayName="Clinton"/>
    <mx:BarSeries xField="@mccain"  displayName="McCain"/>
    <mx:BarSeries xField="@obama"   displayName="Obama"/>
  </mx:series>
</mx:BarChart>
```

使用xField，因为要水平绘制系列

编译并运行这个应用程序后，将看到如图 22-8 所示的结果。

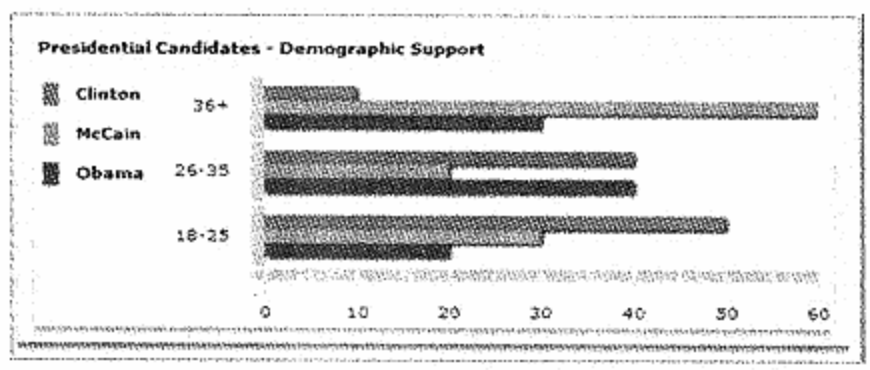


图22-8 显示按年龄分组的统计信息的条形图

没有时间段。图 22-8 所示为按照每个年龄组统计的分类信息。条形图和柱状图都支持堆叠，如前所述，它们的默认方式是按类别分组每个系列。

可以把 type 属性修改为使用其他堆叠值。代码清单 22-9 改用了柱状图，并将 type 属性设置为 stacked。

代码清单22-9 使用堆叠柱状图

```
<mx:Panel title="Presidential Candidates - Demographic Support"
   layout="horizontal" width="500" height="200">
  <mx:Legend dataProvider="{candidateChart}"/>
    <mx:ColumnChart type="stacked"  id="candidateChart"
              dataProvider="{tempData}" height="100%" width="100%" >
      <mx:horizontalAxis>
        <mx:CategoryAxis categoryField="@category"/>
      </mx:horizontalAxis>
      <mx:series>
        <mx:ColumnSeries yField="@clinton" displayName="Clinton"/>
        <mx:ColumnSeries yField="@mccain"  displayName="McCain"/>
        <mx:ColumnSeries yField="@obama"   displayName="Obama"/>
      </mx:series>
    </mx:ColumnChart>
</mx:Panel>
```

在X轴定义类别

定义值

这里必须改为使用 yField，因为柱状图使用垂直方式——每个数据点的高度需要 Y 轴的值。与此类似，由于水平轴显示每个类别，因此要改为使用 horizontalAxis。

图表中的值累积到 100——这是因为数据本身表示的就是百分比。不过，假如这些数

据表示每位总统候选人希望得到的选票数，则可以将 type 属性设置为 100%，也可以看到相同的结果(Flex 会自动计算百分比值)。

不仅可以使用条形图和柱状图显示时间系列，还能通过它们显示按任意类别分组的信息。但是，如果想表现数据随时间变化的趋势，折线图才是最合适的选择。

22.5.3　折线图

折线图是显示随时间变化的数据点(数据趋势)的经典方式。代码清单 22-3 和代码清单 22-4 中已经使用了折线图，而且当时已经介绍了相关的基础知识。

不过，在此我们要讨论一个新知识点——interpolateValues 属性。这个属性用于告诉 Flex 在必要时填充缺失数据点造成的空白。

为了演示这个属性的作用，我们有意拿掉一个数据点。然后，Flex 就只能从具有相邻数据点的位置开始绘制线条，如图 22-9 所示。

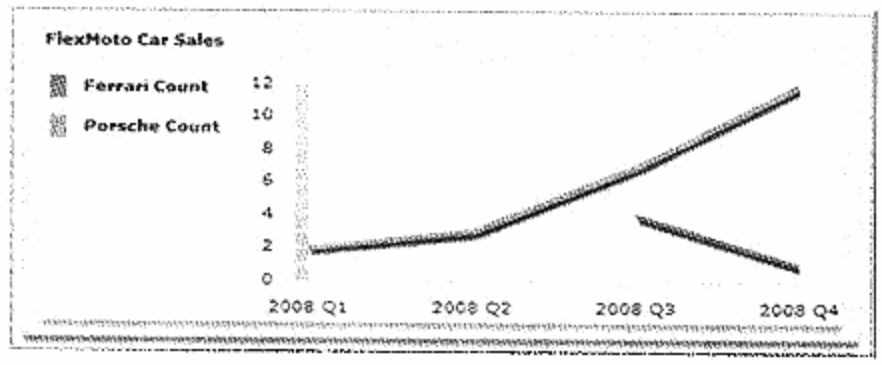


图 22-9　由于没有相邻的数据点，Ferrari 系列中断了

将 interpolateValues 属性设置为 true 即可解决这个问题。

```
<mx:LineSeries interpolateValues="true" yField="@ferrariUnits"
          displayName="Ferrari Count"/>
```

设置之后，再次运行这个应用程序，将看到如图 22-9 所示的折线会因 Flex 填充了空白而连续起来。

很少有 Flex 开发人员在自己的职业生涯中不使用折线图，毕竟折线图具有广泛的适用范围，而且非常适合表现历史趋势。折线图与条形/柱状图一样，还有另外一个优势，即它们都为人们所熟悉，因此便于挖掘数据的内涵。如果读者想暂时抛开传统图表的话，就看一看气泡图吧。

22.5.4　气泡图

气泡图也不是什么新鲜事物了，有读者可能看到过它的例子。不过，要说有谁经常会使用气泡图，那倒真是件新鲜事儿。在商务智能领域中，气泡图逐渐得到人们的认可。不过也有人说气泡图的用处有限。

气泡图可以添加第三个维度，即除了绘制 X 和 Y 坐标外，还可以通过气泡的大小来表示某种规模(如数量、强度、重量或尺寸)。Flex 只支持二维图表，因此气泡图可以突破这种局限性。

BubbleChart 组件支持一些特有的属性，应该格外注意(见表 22-4)。

表 22-4 气泡图的特有属性

属 性	类 型	说 明
minRadius*	Number	数据点的最小值。默认为 0
maxRadius*	Number	数据点的最大值。默认为 50

* 这些值用于计算比例，以确定气泡的大小。

作为 BubbleChart 组件的补充，BubbleSeries 组件也有一些独特的属性，如表 22-5 所示。

表 22-5 BubbleSeries 属性

属 性	类 型	说 明
xField	String	可选。包含相关类别值的字段。如果不指定，系列会在 CategoryAxis 组件中查找指定的类别
yField	String	包含类别/值的字段
radiusField	String	包含气泡大小值的字段

目前，我们只绘制了 FlexMoto 销售数据中给出的汽车销量(见代码清单 22-2)。要添加另一个维度，可以再绘制出销售收入。代码清单 22-10 通过 BubbleChart 组件显示了销量，同时以销售收入作为决定气泡大小的依据。

代码清单22-10 通过气泡图形象地展示销售收入

```
<mx:Legend dataProvider="{salesChart}"/>
<mx:BubbleChart maxRadius="10" id="salesChart" dataProvider="{salesData}"
            height="100%" width="100%" >
  <mx:horizontalAxis>
    <mx:CategoryAxis categoryField="@interval"/>
  </mx:horizontalAxis>
  <mx:series>
    <mx:BubbleSeries yField="@ferrariUnits" radiusField="@ferrariSales"
                displayName="Ferrari Count"/>
    <mx:BubbleSeries yField="@porscheUnits" radiusField="@porscheSales"
                displayName="Porsche Count"/>
  </mx:series>
</mx:BubbleChart>
```

图 22-10 形象地表达了销量和销售收入。

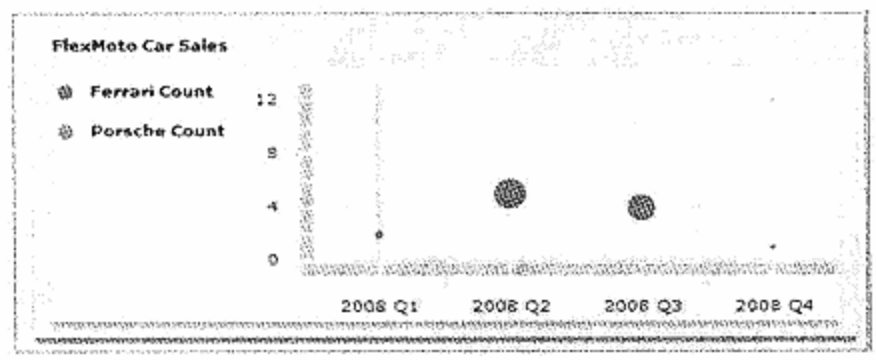


图22-10 气泡图通过第三个维度表现出更多信息

通过添加第三个维度，增加了一个观察相同数据的新视角。从折线图来看，似乎管理层应该更多地关注销售保时捷，但气泡图(图 22-10)却显示，保时捷的销售收入不如法拉利可观。

现在又回到了气泡图是否有用的问题上，我们的看法是肯定有用，它毕竟提供了第三个表现信息的维度。商务智能关注的重点是理解数字背后的故事，能够使用的维度越多，故事也就越圆满。

气泡图唯一真正的问题，就是由于它们不常见，用户并不熟悉它们，因此不会马上读懂每个圆点的含义。如果读者在自己的项目中使用气泡图，一定要为最终用户提供到位的培训或相应的帮助信息。

看懂气泡图并不是用户唯一要面对的挑战。看懂烛台图和 HLOC 图的情况也大致相似。

22.5.5　烛台图和 HLOC 图

在行色匆匆的生活中，读者可能见过烛台图和 HLOC(High-Low-Open-Close，即高-低-开-收)图，金融行业频繁地使用这两种图表展示小粒度(granular level)信息。如果没有这两种图表，恐怕就得使用很多单个图表，既会造成屏幕拥塞，也会导致难以理解。

烛台图和 HLOC 图都是 4 合 1 的图表。金融信息的用户通过它们很容易理解 4 组数据，同时确定这些数据之间的关系。

烛台图的含义如下。

- 垂直线段表示最高价和最低价(上高下低)。对于金融数据，这条线段表示一天(或任意时间段)内价格的变动范围。
- 矩形框(也称主体)浮动在线段上。
- 如果矩形框是空心的，则框的顶部表示收盘价，底部表示开盘价，即收盘价高于开盘价，也就是金融产品(如股票)升值了。
- 如果矩形框被填充了，则框的顶部表示开盘价，底部表示收盘价，即金融产品贬值了。

如图 22-11 所示。

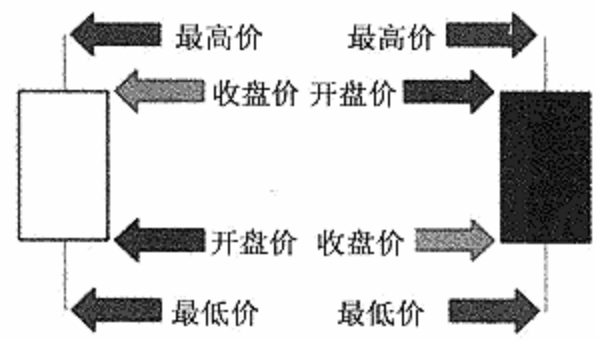


图 22-11　烛台图各部分的含义

主体(即矩形框)的大小不同，提供给用户的信息也不一样。例如，较长的主体表示买入或卖出压力较大，较短的主体表示价格区间较窄。长而空心的主体，表明买入压力很大(牛市)，长而实心的主体表明卖出压力很大(熊市)。

HLOC 图的工作方式与烛台图相同，只不过是以左、右侧的杠表示开盘和收盘价。工作原理如下。

- 垂直线段表示最高价和最低价(上高下低)。这条线段表示一天(或任意时间段)内价格的变动范围。
- 左侧的杠表示开盘价。
- 右侧的杠表示收盘价。

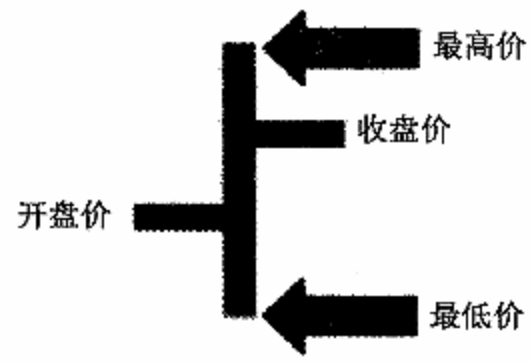


图 22-12　HLOC 图的含义

HLOC 图与烛台图的一个重要区别是，烛台图要求提供 4 个数据点。而不管 HLOC 图的名称如何(HLOC)，其开盘价是可选的。图 22-12 所示为 HLOC 图的含义。

在使用这两种图表时，需要 4 个数据点来创建 CandlestickSeries 和 HLOCSeries，如表所示。

表 22-6 CandlestickSeries 和 HLOCSeries 的属性

属 性	类 型	说 明
highField	String	包含最高价的字段
lowField	String	包含最低价的字段
openField	String	包含开盘价的字段；对 HLOCSeries 是可选的
closeField	String	包含收盘价的字段
xField	String	包含在 X 轴显示的类别值的字段。可选

假设 FlexMoto 公司 CEO 想通过烛台图展示股票价格。在代码清单 22-11 所示的历史股价数据基础上，代码清单 22-12 使用 CandlestickChart 创建了烛台图。

代码清单22-11 hloc.xml：FlexMoto公司的历史股价数据

```
<stockdata>
 <ticker>
  <data interval="09/01/2008" high="54" low="40" open="53" close="43"/>
  <data interval="09/02/2008" high="69" low="37" open="62" close="40"/>
  <data interval="09/03/2008" high="62" low="52" open="55" close="60"/>
  <data interval="09/04/2008" high="65" low="30" open="60" close="52"/>
  <data interval="09/05/2008" high="60" low="20" open="42" close="48"/>
 </ticker>
</stockdata>
```

代码清单22-12 显示股票价格的烛台图

```
<mx:XML id="DataXML" source="hloc.xml"/>
<mx:XMLListCollection source="{DataXML.ticker.data}" id="stockData"/>
<mx:Panel title="FlexMoto Stock" width="500" height="200"
       layout="horizontal">
 <mx:CandlestickChart id="hlocChart" dataProvider="{stockData}"
                 height="100%" width="100%" >
  <mx:horizontalAxis>
   <mx:CategoryAxis categoryField="@interval"/>
  </mx:horizontalAxis>
  <mx:series>
   <mx:CandlestickSeries highField="@high" lowField="@low"
                   openField="@open" closeField="@close"/>
  </mx:series>
 </mx:CandlestickChart>
</mx:Panel>
```

运行后的结果如图 22-13 所示。

类似地，如果把<mx:CandlestickChart>修改为<mx:HLOCChart>(同时修改相关系列)，可以看到如图 22-14 所示的结果。

烛台图和 HLOC 图虽然广泛应用于金融领域，但如果能够创造性地使用数据，相信读者也可以通过它们为企业管理人员展示非金融数据。

下面介绍另一种更加普及的图表——饼图。

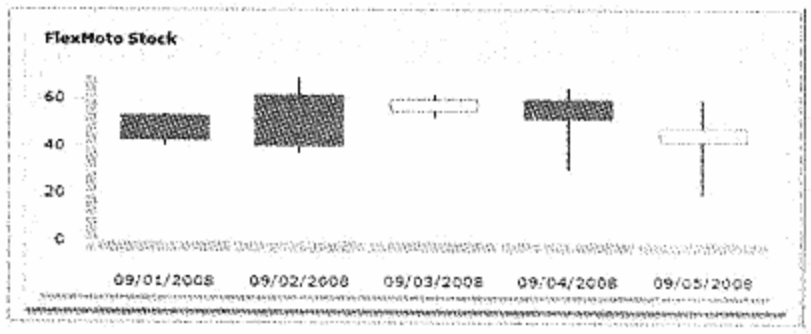


图 22-13　显示股票价格的烛台图

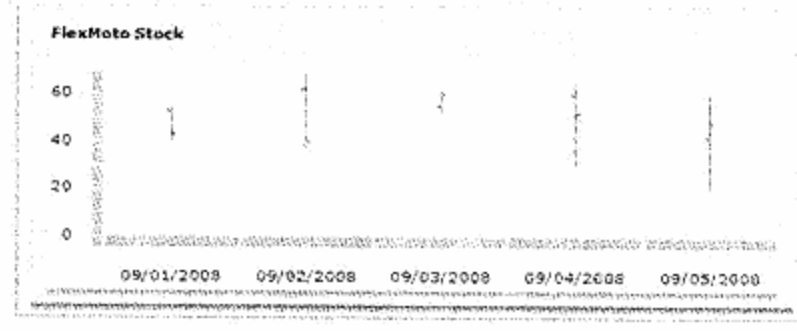


图 22-14　显示相同数据的 HLOC 图

22.5.6　饼图

饼图可谓是尽人皆知的一种图表——我们经常用它来把一个整体分割成几个片段。每个片段的大小表明该片段的权重、价值或影响。

由于饼图表现的是构成一个整体的不同组成部分，因此提供的数据应该能够表示一个整体(显然趋势数据不适合饼图)。在下面的示例中，将使用饼图显示 FlexMoto 的年度销售收入，按品牌分类。代码清单 22-13 所示为原始数据。

代码清单22-13　pie.xml：制造商的年度销售收入

```
<salesdata>
 <revenues>
  <data brand="Ferrari" rev="987265"/>
  <data brand="Porsche" rev="678293"/>
  <data brand="BMW" rev="567987"/>
  <data brand="Mercedes" rev="210697"/>
 </revenues>
</salesdata>
```

以这些数据为基础，可以使用一个 PieChart 组件和一个查看 brand 及 rev 字段的 PieSeries 组件，创建相应的饼图(见代码清单 22-14)。

代码清单22-14　创建按品牌显示年度销售收入的饼图

```
<mx:Legend dataProvider="{salesChart}"/>
 <mx:PieChart id="salesChart" dataProvider="{salesData}"
          height="100%" width="100%" >
  <mx:series>
    <mx:PieSeries field="@rev" nameField="@brand"
              labelPosition="outside"/>
   </mx:series>

 </mx:PieChart>
```

结果将得到一个按品牌收入划分的饼图，而且带一个图例，如图 22-15 所示。

饼图的应用不止如此。通常，一个饼图只有一个系列(见图 22-15)。可是，如果提供多个饼图系列会得到什么结果呢？下面就来试一试。

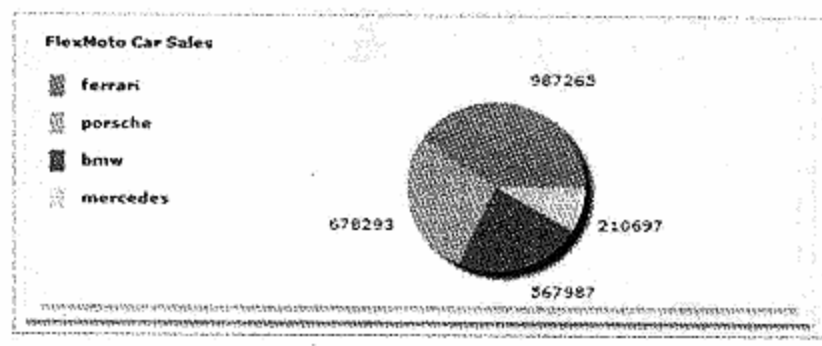


图 22-15　通过饼图显示各个品牌的年度销售收入

代码清单 22-15 所示为添加利润数据

后的文件。

代码清单22-15 pie.xml：包含销售收入和利润数据

```
<salesdata>
 <revenues>
  <data brand="Ferrari" rev="987265"/>
  <data brand="Porsche" rev="678293"/>
  <data brand="BMW" rev="567987"/>
  <data brand="Mercedes" rev="210697"/>
 </revenues>
 <profits>
  <data brand="Ferrari" profit="87265"/>
  <data brand="Porsche" profit="78293"/>
  <data brand="BMW" profit="67987"/>
  <data brand="Mercedes" profit="10697"/>
 </profits>
</salesdata>
```

现在修改代码清单 22-15，以便包含两个数据集并为每一个创建 PieSeries 组件(见代码清单 22-16)。

代码清单22-16 扩展原有代码，创建包含两个系列的饼图

```
<mx:XML id="salesDataXML" source="pie.xml"/>
<mx:XMLListCollection source="{salesDataXML.revenues.data}"
                id="salesData"/>
<mx:XMLListCollection source="{salesDataXML.profits.data}"
                id="profitData"/>

<mx:Panel title="FlexMoto Car Sales" width="500" height="400"
       layout="horizontal">
 <mx:Legend dataProvider="{salesChart}"/>
 <mx:PieChart id="salesChart" dataProvider="{salesData}"
          height="100%" width="100%" >
  <mx:series>
   <mx:PieSeries field="@rev" nameField="@brand"
            labelPosition="callout"/>
   <mx:PieSeries dataProvider="{profitData}" field="@profit"
            nameField="@brand" labelPosition="inside"/>

  </mx:series>
 </mx:PieChart>
 </mx:Panel>
```

创建第二个数据集

创建第二个PieSeries
(注意dataProvider属性)

注意为第二个饼图系列指定的 dataProvider 值。如果不指定这个值，则默认使用饼图正在使用的数据。

解决问题的办法当然不止一个。代码清单 22-16 所示为如何处理两组完整的数据，而同样的数据也可以按照如下结构来组织。

```
<data brand="ferrari" rev="987265" profit="23424"/>
<data brand="porsche" rev="678293" profit="42323"/>
```

你无需指定 dataProvider，相关代码也可以简化如下。

```
<mx:PieSeries field="@rev" nameField="@brand"
  labelPosition="callout"/>
<mx:PieSeries field="@profit" nameField="@brand"
  labelPosition="inside"/>
```

结果仍然是嵌套的饼图。

组合饼图系列的最大好处就是节省桌面空间。另外，它允许比较相关数据集的相关部分。

Flex 内置的图表类型就快介绍完了。最后一个是散点图。

22.5.7　散点图

最后介绍不代表不重要，但散点图确实很简单。前面讨论的图表在计算要绘制的数据点位置时，需要分析数据并确定如何将数据转换为图表。而散点图实际上就是一个网格(由我们为它指定具体的数据点位置)。在下面的示例中，将使用 X 和 Y 坐标数据绘制两组不同的数据，如代码清单 22-17 所示。

代码清单22-17　plot.xml：要绘制的数据

```
<plotdata>
 <one>
  <data x="1" y="5"/>
  <data x="3" y="8"/>
  <data x="3" y="9"/>
  <data x="5" y="9"/>
 </one>
 <two>
  <data x="2" y="1"/>
  <data x="2" y="3"/>
  <data x="4" y="7"/>
  <data x="4" y="4"/>
 </two>
</plotdata>
```

代码清单 22-18 使用以上数据创建了包含两个散点系列的散点图。

代码清单22-18　显示两个系列的散点图

```
<mx:XML id="plotDataXML" source="plot.xml"/>
<mx:XMLListCollection source="{plotDataXML.one.data}" id="plotData1"/>
<mx:XMLListCollection source="{plotDataXML.two.data}" id="plotData2"/>
<mx:Panel title="Plot Data" width="500" height="200"
       layout="horizontal">
 <mx:Legend dataProvider="{plotChart}"/>
  <mx:PlotChart id="plotChart" height="100%" width="100%" >
   <mx:series>
     <mx:PlotSeries dataProvider="{plotData1}" displayName="One"
               xField="@x" yField="@y" radius="4"/>
     <mx:PlotSeries dataProvider="{plotData2}" displayName="Two"
               xField="@x" yField="@y" radius="4"/>

   </mx:series>
  </mx:PlotChart>
  </mx:Panel>
```

创建两组数据 ❶

每个系列都指向相关的数据集 ❷

首先把 XML 文件中的两个数据集分成两个独立变量❶。每个散点系列分别指向其中一个变量❷。如图 22-16 中的结果所示，Flex 为每个系列分别赋予颜色和形状。

散点图适合显示什么数据？由于可以实际地控制 X 和 Y 坐标，因此可以将 X 轴和 Y 轴作为度量两者关系的手段。例如，用 X 轴表示人的年龄，Y 轴表示车祸的平均值，就可以直观地看出这两类数据之间的关系。

从统计学的角度看，如果有一组无关的数据点，通过散点图可以看出这些数据的聚集

倾向。而这种特征是通过计算数据的平均值无法知悉的。例如，图 22-17 所示为一组聚集的数据点，从 Y 轴上看，它们在标记了周围。然而，计算这些数的平均值，结果是 5。掌握了数据的分布特征，有助于更好地对信息做出解释。

下一节，将讨论如何自定义图表。

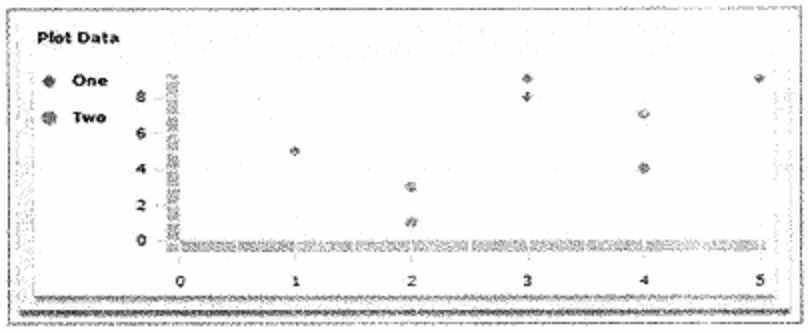


图 22-16 显示两组数据的散点图

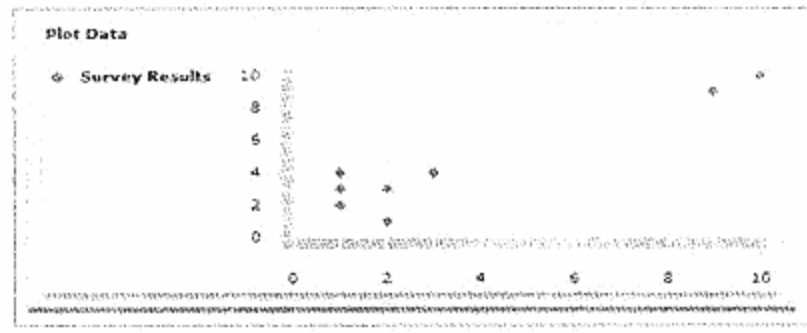


图 22-17 聚集的数据点能够揭示出比平均值更丰富的信息

22.6 自定义图表

如果所有图表看起来都一样，就会令人厌烦。况且，为了与应用程序的整体风格保持一致，自定义外观也是必要的。本节将高度概括与自定义图表有关的内容。

许多图表都支持某些独有的特征，而通过 Flex API 参考可以查得如何控制这些特征。当然，一些公共的自定义特性也是要考虑的内容。

22.6.1 系列笔触

笔触(Stroke)能够起到增强系列线条或边界的作用。要设置笔触，首先必须找到给定系列的相关属性。下面就以面积图为例，步骤如下：

(1) 在 Flex API 参考中找到相关系列，即 AreaSeries 组件。最快的方式是单击代码中的<mx:AreaSeries>，然后按 Shift+F2。

(2) 滚动到 Styles 部分。

(3) 查找包含 stroke 的样式。在此，找到了 areaStroke。

要设置笔触样式，需要先定义笔触。

```
<mx:Stroke id="stroke1" color="#6600CC" weight="4" alpha="0.8"/>
```

然后将笔触相关的样式绑定到这个实例。

```
<mx:AreaSeries yField="@ferrariUnits" displayName="Ferrari Count"
          areaStroke="{stroke1}"/>
```

这样就可以在面积图上绘制出相对较粗的线条。

笔触不会影响线条下面的区域，这些区域由填充负责。

22.6.2 系列填充

填充(fill)负责为系列添加背景颜色。有几种使用填充的方式，下面我们通过示例来说明。

1. 单色填充

首先按照与设置笔触相同的步骤，找到给定系列支持的填充属性。

(1) 在 Flex API 参考中找到相关系列，即 AreaSeries 组件。最快的方式是单击代码中的 <mx:AreaSeries>，然后按 Shift+F2。

(2) 滚动到 Styles 部分。

(3) 查找包含 fill 的样式。在此找到了 areaFill。

提示：

大多数图表系列都支持 fill 和 fills 属性，但 AreaSeries 和 LineSeries 会忽略这两个属性。

要设置填充样式，需要声明一个 SolidColor 组件实例。然后再将系列的填充属性绑定到相应用的 SolidColor。例如：

```
<mx:SolidColor id="solidcolor1" color="#33FF66" alpha="0.5"/>
```

在 AreaSeries 组件中，像下面这样绑定 areaFill。

```
<mx:AreaSeries yField="@ferrariUnits" displayName="Ferrari Count"
       areaStroke="{stroke1}" areaFill="{solidcolor1}"/>
```

这样就创建了半透明的绿色填充。

这个示例适用于表现趋势的图表。数据点未连接在一起(视觉上)的图表支持颜色数组。

2. 颜色数组填充

分离每个数据点的图表(就像是一个系列，如条形图和饼图)使用 fills 属性，该属性的值是颜色数组。这样就可以使用颜色来编码所有数据项(因为它们被包含在一个系列中)。

例如，如果不想使用饼图的默认颜色，可以如代码清单 22-19 所示将其覆盖。

代码清单22-19　为饼图中的每个片段指定一种颜色

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundColor="white">
  <mx:XML id="salesDataXML" source="pie.xml"/>
  <mx:XMLListCollection source="{salesDataXML.revenues.data}"
                id="salesData"/>

  <mx:Script>
   <![CDATA[
    import mx.graphics.SolidColor;
    [Bindable]
    private var clrArray:Array = new Array(
      new SolidColor(0x6666CC,0.5),
      new SolidColor(0xCC6666,0.6),
      new SolidColor(0x66CC66,0.7),
      new SolidColor(0xB8B83D,0.8)
    );
   ]]>
  </mx:Script>

  <mx:Panel title="FlexMoto Car Sales" width="500" height="400"
         layout="horizontal">
   <mx:Legend dataProvider="{salesChart}"/>
    <mx:PieChart id="salesChart" dataProvider="{salesData}" height="100%"
            width="100%" >
```

导入ActionScript类SolidColor

创建SolidColor数组

```
    <mx:series>
      <mx:PieSeries field="@rev" nameField="@brand"
                fills="{clrArray}"/>
     </mx:series>
    </mx:PieChart>
    </mx:Panel>
</mx:Application>
```

将fills属性绑定到数组

结果饼图中的每个片段分别呈现为对应的颜色。如果饼图的片段(数据点)增多，超过了定义的颜色数目，Flex 会为多出的片段应用默认颜色。

尽管自定义 Flex 图表的方式非常多，但设置笔触和填充是最常见的。

22.7 小结

Flex 的图表特性非常强大，也非常受欢迎。有些开发团队使用 Flex，就是为了创建图表(特别是在构建图表板时)。图表由许多部分组成，如系列、标签、轴和元素，当然还有图表本身。

Flex 为满足大多数企业常见的需求提供了 9 种内置图表。有些图表(如折线图和面积图)，适合表现趋势信息；另一些图表(如条形图)，则侧重于展现不同类别数据的对比。

虽然本章只是大致介绍自定义图表的内容，但归根结底，都是通过 API 参考找到相关的样式属性，然后再根据需要加以利用。而且修改笔触和填充也是自定义图表最常用的手段。

本章是讨论编码内容的最后一章。换句话说，本书至此已经介绍完与编程有关的所有内容。接下来，要讨论的是开发周期的收尾阶段——测试和调试。

第 23 章 调试与测试

23

本章内容：

- 使用 Flex 调试器及第三方工具调试应用程序
- 在发布前使用自动测试工具查找并修复故障
- 部署应用程序

本章将从实用的角度向读者推荐一些有助于提高 Flex 应用程序开发效率的工具，包括 Flex 内置工具和第三方工具。Flex Builder 内置的功能完善的调试器，支持单步调试和对变量值的实时监控。Flex 3 新引入的分析程序，可以用来分析内存及资源占用情况。

除了 Flex 内置的工具之外，Flex 和 Flash 技术社区也提供了很多工具，包括低成本和开源解决方案，乃至成熟的企业级测试自动化程序。这些工具可以用来调试和测试 Flex 应用程序。尽管读者暂时还用不到其中比较高级的工具，但至少从一开始就要有工具帮助我们方便地找出程序中问题的起因。首先我们来探讨如何调试 Flex 应用程序。

23.1 调试

可以在不同层面上使用调试。本节将介绍各种调试手段，首先从最简单的开始。

23.1.1 使用 Flash Debug Player

在安装 Flex Builder 时，会自动安装 Flash Debug Player。Flash Debug Player 可以生成诊断日志，从而能够为外部工具所用，以便实现调试。

从 Adobe 网站页面 http://www.adobe.com/support/flashplayer/downloads.html 可以下载到最新版的播放器请留意 Debugger Versions 部分。

要测试当前播放器的版本，可以访问 http://kb.adobe.com/selfservice/viewContent.do?externalId =tn_15507。请确定结果中的 Debug Player 项显示为 Yes。

即使先前安装过 Debug Player，后来再安装某些软件时，有可能会一起安装 Flash Player 的生产版。无论如何，如果经过测试，发现当前版本不是 Debug Player，都应该安装它。

在确保当前运行的是 Debug Player 之后，接下来需要配置日志。

23.1.2 配置日志

为了正常生成日志，需要设置 Flash Debug Player 的配置文件。如果还没有这个文件，可以创建一个名为 mm.cfg 的文本文件，然后把它保存到相应目录中。具体目录取决于操作系统，如表 23-1 所示。

表 23-1 Flash Player 配置文件在不同操作系统中的保存位置

操 作 系 统	路 径
MAC OS X	/Library/Application Support/Macromedia
Windows 95/98/ME	%HOMEDRIVE%\%HOMEPATH%
Windows 2000/XP	C:\Documents and Settings\[username]
Windows Vista	C:\Users\[username]
Linux	/home/username

在这个文件中要定义 3 个配置属性，表 23-2 所示为这 3 个属性。

表 23-2 Flash Player 的配置文件设置

配 置 属 性	类 型	说 明
TraceOutputFileEnable	Boolean	0 表示 false(默认值)，1 表示 true。是否开启日志
ErrorReportingEnable	Boolean	0 表示 false(默认值)，1 表示 true。是否开启错误消息日志
MaxWarnings	Number	最大警告记录数。默认为 100，0 表示无限制

mm.cfg 文件的内容大致如下所示。

```
TraceOutputFileEnable=1
ErrorReportingEnable=1
MaxWarnings=500
```

在启用日志后，具体消息将被保存到名为 flashlog.txt 的文件中，这个文件的位置也因操作系统而异(见表 23-3)。

表 23-3 Flash Player 日志文件的输出位置

操 作 系 统	路 径
MAC OS X	/Users/[username]/Library/Preferences/Macromedia/Flash Player/Logs/
Windows 95/98/ME/ 2000/XP	C:\Documents and Settings\[username]\Application Data\Macromedia\Flash Player\Logs
Windows Vista	C:\Users\[username]\AppData\Roaming\Macromedia\Flash Player\Logs
Linux	/home/[username]/.macromedia/Flash_Player/Logs/

以上配置就绪之后，就可以通过 trace()函数使用它了。

23.1.3　使用 trace()函数

trace()是一个简单的全局函数，在代码中的任何地方都可以调用它。这个函数的作用是向 flashlog.txt 日志文件中写入内容。在设置好 mm.cfg 之后，即可使用 trace()函数，如代码清单 23-1 所示。

代码清单23-1　使用trace()向Flash日志文件中写入信息

```
<mx:Label text="Type in something:"/>
<mx:TextInput id="something"/>
<mx:Button click="trace('Something is currently: ' + something.text)"
   label="Record Something"/>
```

在文本框中输入一些内容(“show this please”)，然后单击 Record Something 按钮，输入的内容就会被写入 flashlog.txt 中(代码清单 23-2 展示了其中的文本)。

代码清单23-2　Flash日志文件示例

```
*** WARNING: Please use gotoAndPlay(1) in the last frame of your Motif
   creative if you intend to loop. ***
*** Please use 'stop();' at the end of your Motif creative if you do not
   intend for it to loop. ***
showing end copy
hiding end copy
showing end copy
Something is currently: show this please
```

如果想记录不同变量或组件的值，这是一种高效的方式。

不过，如果没有适当的工具，查看这个日志文件似乎有点麻烦。好在可以使用跟踪日志查看器。

23.1.4　跟踪日志查看器

生成日志是一回事，而查看日志又是另一回事。目前来看，不断地重新加载 flashlog.txt 文件查看日志内容，的确是件令人讨厌的事。

别急，有几个第三方工具能够提供查看面板，面板在日志文件被修改时可以自动刷新。在众多日志查看工具中，我们推荐以下几个。

- Flash Tracer：这是 Firefox Web 浏览器的一个扩展，能够在 Firefox 中添加一个内置面板。使用它的方便之处是可以不必在两个独立程序间来回切换。价格：免费。https://addons.mozilla.org/ enUS/firefox/addon/3469。
- ABLogFile Viewer：这是一个开源查看器，而且经常被维护。这是一个通用工具，但它能记住最近打开的一些文件，因此可以方便地找回日志文件。在 Windows 中，Application Data 文件夹是隐藏的，需要通过资源管理器找到这个目录，然后把日志文件拖放到 ABLogFile Viewer 中。价格：免费。http://www.amleth.com/ablogfile/。
- Afterthought：专门为 Flash 开发的一个工具，但它没有及时更新，因而可能会找不到当前 Flash Player 升级后保存 flashlog.txt 文件的位置。为此就需要编辑其选项，手工指定日志文件的路径。这个工具好在它的窗口可以浮动在前面，能够筛选警

告消息，还有便捷的 Clear Screen 和 Clear Log 按钮。价格：免费。http://www.blazepdf.com/downloads/afterthought_v2_1.zip。

这些日志文件只能记录文本，而 Flex 中的很多东西并不是以文本形式存在的。因此，还需要某些工具能够将这些对象转换成字符串。

23.1.5 把对象转换成字符串

无论是把复杂数据类型写入日志文件还是显示在屏幕上，都要首先把它们转换成文本。在 Flex 中实现这种转换很方便，因为大多数对象都支持 toString()函数。请看下面的示例。

```
var myNumber:Number = 32;
trace(myNumber.toString());
```

对于更复杂的对象，如 ArrayCollection，ActionScript 的 ObjectUtil 类也提供了一个 toString()函数，该函数可以接收其他对象的引用并将其转换为字符串。

代码清单 23-3 使用 ObjectUtil 类显示了 ArrayCollection 中的所有项。

代码清单23-3 使用ObjectUtil转储对象的属性

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   creationComplete="trace(ObjectUtil.toString(myAC))">

<mx:Script>
   <![CDATA[
      import mx.collections.ArrayCollection;
      import mx.utils.ObjectUtil;
      public var myAC:ArrayCollection = new ArrayCollection([
      {label:"Jon Hirschi", data:"jhirschi"},
      {label:"Tariq Ahmed", data:"tahmed"},
      {label:"Frank Krul", data:"fkrul"}
      ]);
   ]]>
</mx:Script>
</mx:Application>
```

以上代码会在日志文件中写入代码清单 23-4 所示的内容。

代码清单23-4 ObjectUtil的示例输出内容

```
(mx.collections::ArrayCollection)#0
 filterFunction = (null)
 length = 3
 list = (mx.collections::ArrayList)#1
  length = 3
  source = (Array)#2
   [0] (Object)#3
    data = "jhirschi"
    label = "Jon Hirschi"
   [1] (Object)#4
    data = "tahmed"
    label = "Tariq Ahmed"
   [2] (Object)#5
    data = "fkrul"
    label = "Frank Krul"
  uid = "70528450-89C6-9120-D5D4-847C1D8C3B70"
 sort = (null)
 source = (Array)#2
```

通过 Alert.show()把值显示在屏幕上(而不是转储到日志文件)，也是一种显示对象值的常见方式。例如：

```
Alert.show(ObjectUtil.toString(myAC))
```

使用这种技术和 Flash 日志文件，在某些情况下是可以的。不过，要把调试工作提升到新的层次，则必须使用更高级的工具。

23.1.6　FxSpy

FxSpy(也称为 Flex Spy)是一个开源工具，可以使用它来检查组件及其属性和样式，同时还可以操作其中的许多项(见图 23-1)。

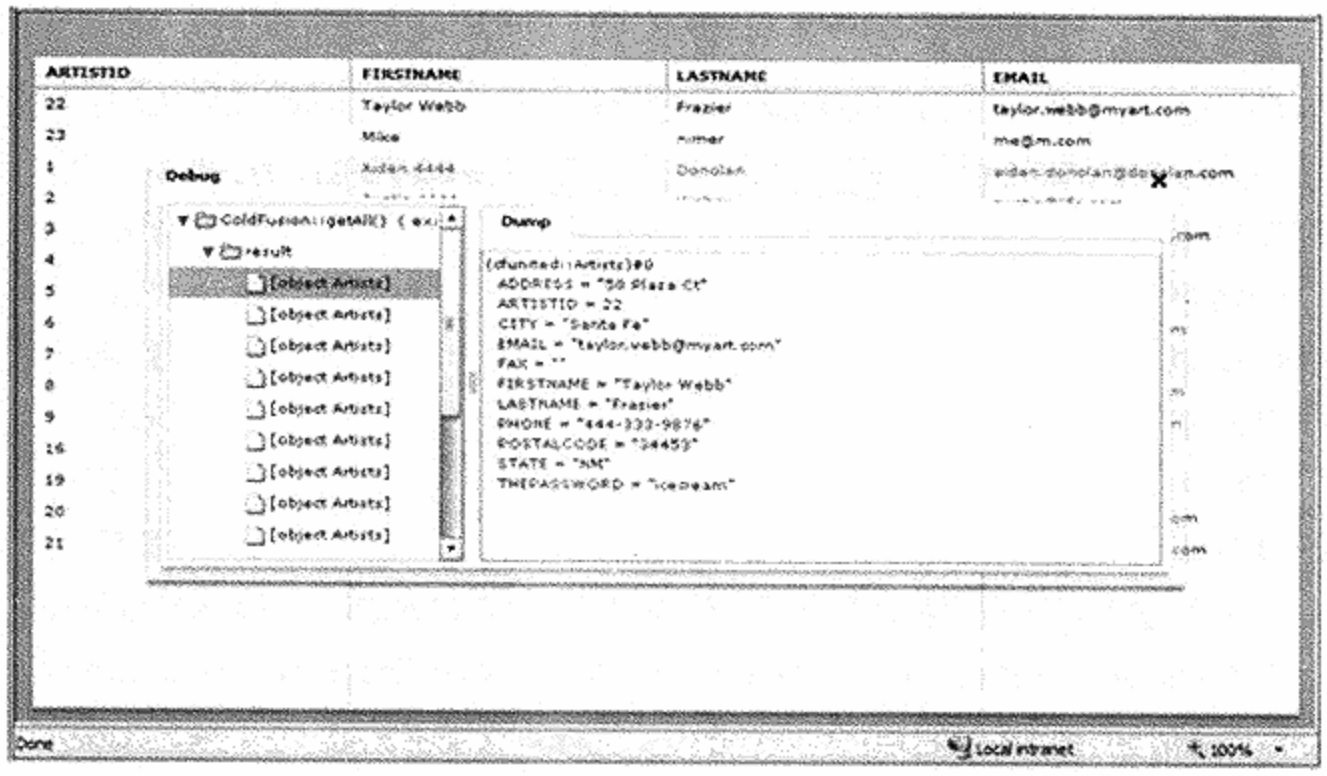


图 23-1　使用 FxSpy 可以浏览检查组件及其属性

在想调整那些影响应用程序外观的属性时，这个工具非常方便。与时常修改代码并重新编译才能看到结果相比，使用 FxSpy 可以节省不少时间。

这个免费工具的下载地址为 http://code.google.com/p/fxspy/。

此前介绍的都是前端工具。实际上，在需要了解网络通信情况时，监控后端的工具也很重要。

23.1.7　监视网络活动

在开发 Flex 应用程序时，最具挑战性的难题，莫过于查出与后端数据服务(如 Web 服务或远程对象)通信失败的原因了。没错，预定义的错误处理程序可以处理异常，不过只处理异常显然还不够。

Flex Builder 内置支持一种监视网络活动的特性。如代码清单 23-5 所示，要使用这个特性，需要在代码中添加<mx:TraceTarget/>组件。

代码清单23-5 使用TraceTarget组件监视网络活动

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:TraceTarget/>
   ...
   ...
</mx:Application>
```

在单击 Debug 按钮(见图 23-2)启动应用程序后，Console 视图中会显示所有网络通信的细节。

图 23-2 Debug 按钮位于 Flex Builder 工具栏上

不过，这个内置特性经常不能满足需要。好在，有很多第三方工具可以用来监视网络，从而加快调试过程。

- ServiceCapture：最好的工具之一。它不仅理解 Flash 的本地 AMF 远程对象协议，而且能解释 Web 服务 SOAP 和 JSON-RPC。使用 ServiceCapture 可以实时、交互式地查看传递给远程函数的参数以及返回的数据。价格：35 美元。http://kevinlangdon.com/serviceCapture/。
- Charles Web Debugging Proxy(CWDP)：一个设计独到、功能丰富的产品。它支持的协议与 ServiceCapture 类似，但通过它能节制带宽以模拟网络连接速度，而且可以把结果导出为 CSV 文件。有多种平台的版本可供选择，但需要 Sun Java JDK 支持。价格：50 美元。http://xk72.com/charles/index.php。

说到真正重量级的调试工具，还要数 Flex Builder 内置的调试器。而且 Flex 3 内置的调试器功能较以前更加完备。

23.1.8 使用调试器

如果读者希望了解变化情况，而且还希望加以精细控制，那么就应该考虑使用调试器。通过调试器以受控方式单步调试代码，既可以帮助我们迅速查出问题，又可以看到代码执行过程中变量值的变化。

使用调试器时，需要先启动调试模式，然后设置暂停执行的断点并添加要监视的变量。

1. 启动调试模式

使用调试器与运行应用程序示例有些类似，只不过这次要单击 Debug 按钮(参见图 23-2)。

另外，在项目文件夹中，除有 bin 文件夹外，还有一个 bin-debug 目录——这是 Flex Builder 保存调试版应用程序的地方。单击 Debug 按钮，调试版应用程序就会启动。

如果尚未在 Flex Builder 中以调试模式运行过应用程序，此时应该切换到调试视角，方法是选择 Window | Perspective | Flex Debugging。然后，就会看到一些视图，其中包括：

- Debug：调试应用程序的主视图。
- Variables：在此观察 Flex 控制的所有变量值。
- Breakpoints：希望应用程序暂时中断执行的点，以便分析情况。
- Expressions：所有要监视的变量。

启用调试器后，下一步就是设置断点。

2. 添加/删除断点

可以在 ActionScript 代码的任意位置添加断点。例如，可以在 MXML 标签中的行内 ActionScript 代码中添加断点。

```
<mx:Button click="myVar=myText.text"/>
```

或者在<mx:Script>块或 ActionScript(.as)文件中加入断点。

添加断点的过程如下。

(1) 在 Flex Builder 中，找到想要暂停执行的代码行。

(2) 右击记号栏(显示行号的地方)。

(3) 选择 Toggle Breakpoint。

删除断点的步骤与添加断点相同。

如果一切顺利，可以在 Breakpoints 视图中看到添加的断点，如图 23-3 所示。

启动调试器并设置断点之后，就可以控制应用程序的执行了。

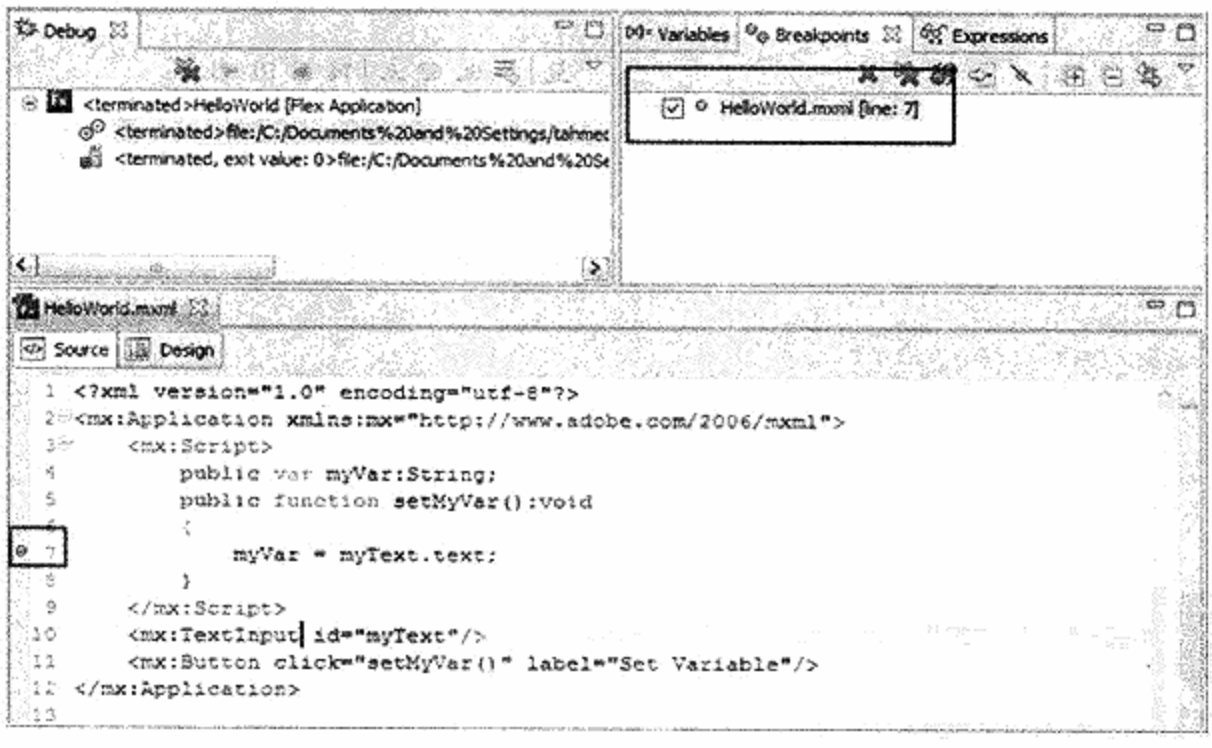


图 23-3　设置的断点出现在 Breakpoints 视图中

3. 控制执行

设置了断点后，在调试模式下启动应用程序。当执行到设置了断点的行时，执行会停止，Flex Builder 将等待开发人员的指令。

此时，通过使用表 23-4 列出的功能图标，可以在 Debug 视图中控制程序执行。

表 23-4 Flex Builder 调试器中的功能图标及说明

图 标	功 能	说 明
	恢复	恢复执行应用程序
	暂停	暂停执行应用程序
	停止	停止执行应用程序
	断开	断开调试器与应用程序的连接
	单步进入	单步进入代码中的下一行
	单步跳过	跳过代码中的下一行
	单步退出	继续执行完当前函数

在知道了如何控制执行后，还需要经常在调试器中监视变量。

4. 监视变量

监视变量可以实时跟踪赋给变量的值。如果想看一看给变量赋其他值会产生什么结果，也可以手工修改变量值。

无论是否启动了应用程序，都可以通过下列操作设置要监视的变量。

(1) 右击变量名。

(2) 选择 Watch。

添加要监视的变量后，就可以在 Expressions 视图中看到该变量。下面就来试一试。

(1) 按照代码清单 23-6 所示在行中设置断点。

代码清单23-6 在设置变量的行上使用断点

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      public var myVar:String;
      public function setMyVar():void
      {
         myVar = myText.text;          <-- 在此处设置断点
      }
   </mx:Script>
   <mx:TextInput id="myText"/>
   <mx:Button click="setMyVar()" label="Set Variable"/>
</mx:Application>
```

(2) 监视 myVar 和 myText.text 变量。

(3) 以调试模式启动应用程序。

(4) 在文本框中输入一些文本，单击按钮，应用程序会暂停执行。

(5) 切换到 Flex Builder，可以看到所列出的变量。

(6) 单击几次 Step Into 按钮，观察变量的变化(见图 23-4)。

对习惯于使用高级语言(如 ColdFusion 和 PHP)的开发人员来说，以这种方式使用调试器是一种新鲜的体验。Flex 应用程序的异步特性就决定了在事件驱动环境下需要使用步进式的调试。多使用几次，自然就会熟悉起来。

与调试类似的过程是测试。然而，调试的目的是为了修复问题，而测试的目的则是确保问题不存在。

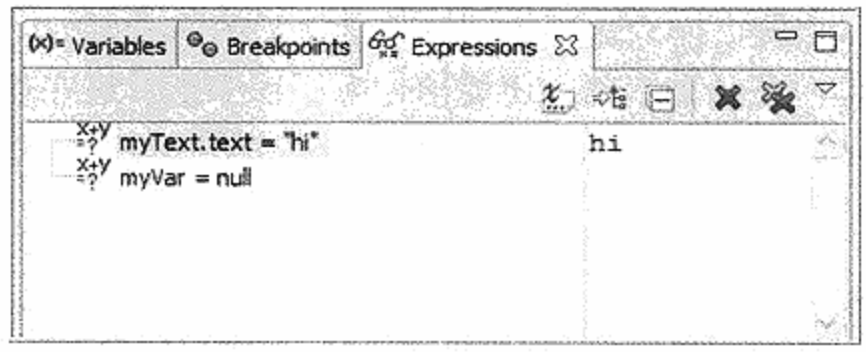


图 23-4　Expressions 视图中显示了被监控变量及它们的值

23.2　测试

在 SDLC(Systems Development Life Cycle，系统开发过程)中，应该尽可能多地安排对应用程序进行各种测试，以确保产品质量。如果使用 ASP、PHP、ColdFusion 及其他同类语言开发，在发生问题时，会告知代码行和文件名——Flex 则无法享受这种奢侈的待遇，因为 Flex 应用程序是编译执行的。

在结束了调试之后，测试是确保新功能达到设计要求，而原功能仍然有效的最佳途径。从另一个角度讲，测试也有利于推动应用程序的改进和完善。

23.2.1　测试的类型

有 3 种测试类型：

- 性能测试(Profiling)
- 单元测试(Unit testing)
- 功能测试(Functional testing)

下面分别给出大致定义，先从性能测试开始。

1．性能测试

性能测试的目的不是验证应用程序是否实现了某种行为，而是查找有没有优化应用程序使用资源的方式。使用性能测试工具，可以发现内存泄漏、不必要的处理器开销，暴露有待优化的问题。

2．单元测试

单元测试是以编程方式自动测试组件、函数和类的一种方式。从编程角度看，单元测试属于后端测试，因为它可以保证后端代码为调用者提供适当的结果和行为。

3．功能测试

功能测试着眼于质量保证过程的另一端，也就是前端用户的体验。简言之，功能测试就是模拟用户运行应用程序，以验证各种行为和信息是否准确。功能测试可以手工进行，

但实践表明，手工测试常伴有人的主观错误，也比较费时间。

功能测试的过程主要是记录对应用程序的实际使用，包括跟踪输入和应用程序行为。每个记录就是一个测试用例，而测试用例则事先定义了预期结果。

功能测试不仅仅要保证结果正确，还要检查应用程序是否遵守业务规则？是否恰当地验证了字段？总体使用感受是否合乎用户预期？除了自动测试新功能，通过功能测试还可以确保原有功能继续有效。

23.2.2 Flex 分析程序

Flex 3 新增了分析程序，该程序能够识别应用程序中的资源(处理能力和内存利用方面的)瓶颈，从而帮助开发人员确定和解决问题。

分析程序会对应用程序定时采样。形象地说，有点类似慢速摄影(或分段拍摄)，即它会按照固定时间间隔记录应用程序在某个时点的运行状况。这样一来，就能够确定某一时刻存在多少对象，这些对象占用了多大内存空间，调用了多少函数以及这些函数被调用了多少次等信息。

开发人员应该关注如下指标。

- **函数调用频率**：多长时间调用一次函数？是否存在不必要的调用？看看能否把调用大型函数的次数降至最少；还是应该把大型函数分解为小型函数，然后多调用几次其中的一个小函数。
- **函数执行时间**：函数执行需要多长时间？这是同前一个指标密切相关的，通过监控特定函数，能够知悉该函数的平均执行时间。
- **函数调用关系**：观察调用堆栈和函数与函数的调用链，能够揭示很多有用的信息。
- **对象数目/对象配置**：实例化了多少个对象？是不是为同一个对象创建了多个实例？显然，创建的实例越少，开销越小。
- **对象大小**：这些对象会占用多少空间？对那些有多个实例的对象，是否可以通过合并来缩减它们的大小？
- **垃圾收集**：默认情况下，Flex 会在对象没有必要存在时，负责销毁它们。垃圾收集就是把原先被占用的内存释放出来。有时候，某些闲置对象在被销毁前，还会在内存中继续存在一段时间(这是导致内存泄漏的原因)。如果发生了这种情况，可能就需要添加一些逻辑，以便明确销毁这些对象。

使用分析程序与使用调试器类似，它们都通过 Flex Builder 启动。

1. 启动分析程序

在 Flex Builder 工具栏中的 Debug 图标旁边，可以找到 Profile 按钮。

单击 Profile 按钮，将打开如图 23-5 所示的窗口，从中可以选择想要监视的项目。

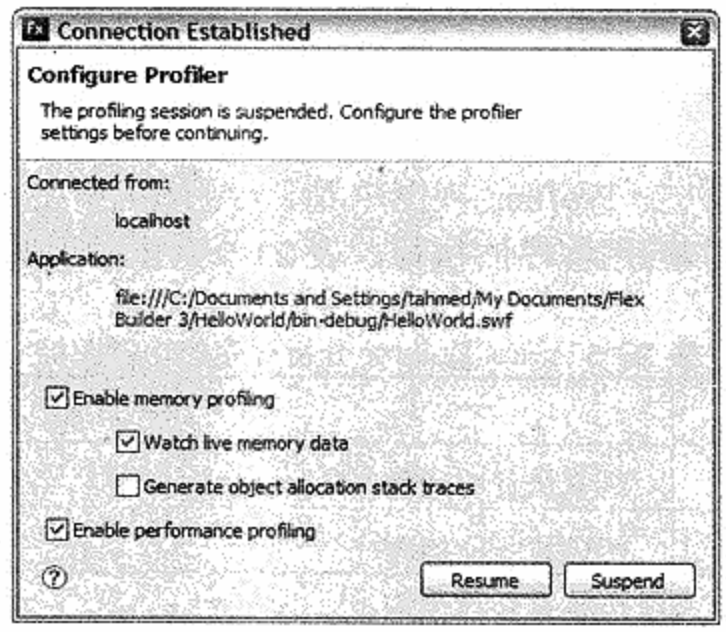


图 23-5　启动分析程序，会出现配置选项窗口

对话框中的设置如下。

- Connected from：想要连接的服务器(如果是同一台计算机，则为 localhost)。
- Appliction：Flex 应用程序中 SWF 文件的路径。
- Enable Memory Profiling：启用对内存占用情况的跟踪。
- Watch Live Memory Data：选择 Enable Memory Profiling，会跟踪内存状态。而 Watch Live Memory Data 表示是否在 Live Object 视图中监视内存状态。
- Generate Object Allocation Stack Traces：用于跟踪创建对象的方式。
- Enable Performance Profiling：可以监视哪些代码或函数占用处理时间。

可以在任何情况下都全选这些项，唯一的差别是分析过程中生成分析数据的多少以及分析过程中对性能产生的影响。

启动分析程序，将看到如图 23-6 所示的界面。

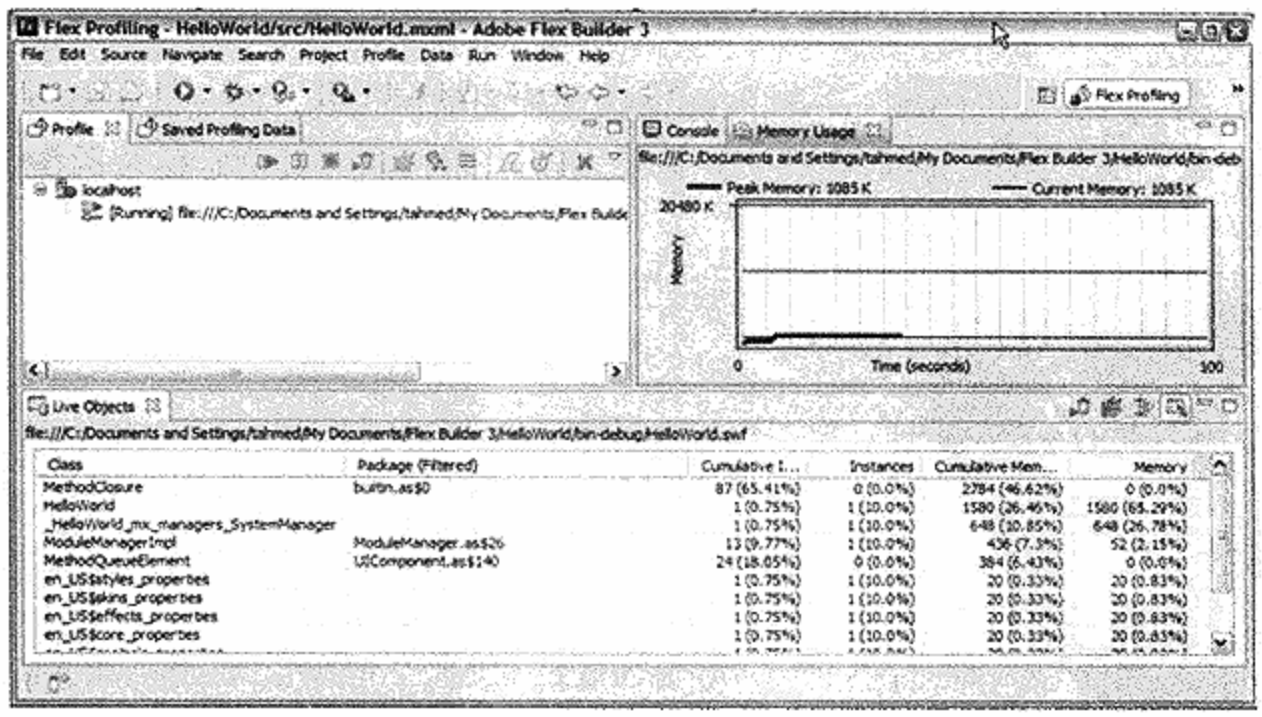


图 23-6　分析程序提供有关内存和 CPU 使用情况的汇总统计信息

分析程序与调试器在功能上有些类似。运行分析程序后，可以切换回 Flex Builder 开

发模式，以便控制应用程序的执行。

2. 控制执行

运行分析程序后，可以在 Profiler 视图中看到应用程序。选择一个应用程序，会激活一组图标，然后就可以控制应用程序的执行。表 23-5 列出并说明了这些功能图标的用途。

表 23-5 Flex Builder 分析程序的功能图标

图 标	功 能	说 明
	恢复	恢复执行应用程序
	暂停	暂停执行应用程序
	停止	停止执行应用程序
	运行垃圾收集	立即执行垃圾收集，而不是再等下去。这样会销毁闲置对象，释放它们占用的内存空间
	抓取内存快照	获取内存使用情况的快照并按用途分类。如果怀疑发生了主要内存操作，就可以抓取一个快照，以便将来分析使用
	查找闲置对象	与“抓取内存快照”相互配合，可以比较不同快照间的区别
	查看分配跟踪	在选择两个内存快照后方可使用。用于查看两次快照都会调用的函数以及在此过程中占用的内存数量
	重置性能数据	重置记录信息
	捕获性能分析	与“重置性能数据”结合使用。单击“重置性能数据”图标后，单击此按钮可以捕获对一组函数的调用及其执行时间的数据。这一过程可以反复多次执行
	删除	删除选择的项(如内存快照)

分析程序是 Flex 提供的唯一一种能够这样实施的测试工具，而且该工具确实很棒。使用分析程序不仅可以全面了解内存和资源占用情况，而且能够准确定位占用资源的对象和过程。

接下来将介绍单元测试，也就是验证对象是否具有预期行为的自动化测试。Flex 社区创建了许多可资利用的单元测试框架。

23.2.3 FlexUnit(单元测试)

如果读者熟悉 Java，可能使用过 JUnit——FlexUnit 就是 Flex 中的 JUnit。FlexUnit 是针对 Flex 和 ActionScript 3 的单元测试框架，由几位 Adobe 和非 Adobe 开发人员协作开发。这个工具可以免费下载，地址为：http://opensource.adobe.com/wiki/display/flexunit/。

需要创建一个测试运行程序(test runner)，该程序负责执行我们创建的一组 ActionScript 函数，通过执行这些函数实现对其他各种函数和类的测试。FlexUnit 有文档可查，不过，我们在此也会给出一个简单的使用说明。首先是设置 FlexUnit。

1. 设置 FlexUnit

测试运行程序是使用 FlexUnit 的 TestSuite 类运行所有测试的一个小程序。可以把它添加到主 MXML 项目文件中；但由于涉及 GUI 组件，因此将其作为一个独立的小程序会更方便。测试运行程序可以作为一个新项目，也可以作为应用程序的辅助性单元存在。请读者按照下列步骤操作。

(1) 从 FlexUnit 站点下载最新的 zip 文件：http://opensource.adobe.com/wiki/display/flexunit/。

(2) 把下载到的文件解压缩到一个本地目录(如 c:\flexunit)中。

(3) 在 Flex Builder 中打开要测试的项目。

(4) 右击项目的顶级文件夹，选择 Properties。

(5) 选择 Flex Build Path 下的 Library Path 标签页。

(6) 单击 Add SWC 按钮。

(7) 导航到 FlexUnit 目录，打开 bin 目录，选择 flexunit.swc 文件。

(8) 下一步就是创建执行测试的测试运行程序。

2. 创建测试运行程序

测试运行程序可以添加任意数量的测试用例集合并逐个运行它们。FlexUnit 把这些测试用例集合称为测试套件，这样会方便大量测试用例的执行。代码清单 23-7 所示为一个测试运行程序。

代码清单23-7　创建FlexUnit测试运行程序

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
            xmlns:flexunit="flexunit.flexui.*" >

    <mx:Script>
      <![CDATA[
         import flexunit.framework.TestSuite;

         private function runTests():void
         {
             var ts:TestSuite = new TestSuite();
             ts.addTest( myTest.suite() );

             testRunner.test = ts;
             testRunner.startTest();
         }

      ]]>
   </mx:Script>
   <mx:Button label="Begin Tests" click="runTests()"/>
   <flexunit:TestRunnerBase id="testRunner"
      width="100%" height="100%" />
</mx:Application>
```

添加FlexUnit命名空间

导入TestSuite类

如有必要调用addTest()函数以便添加测试套件

执行测试

添加FlexUnit的MXML视觉组件

然后我们要做的就是创建测试用例。

3. 创建测试用例

最后一件事是编写测试用例，也就是创建 ActionScript 类，让它返回一个包含全部必

要测试的测试套件。此时可以创建一个包含许多测试用例的大型测试套件，也可以创建多个包含少量测试用例的小型测试套件。

在前面的测试运行程序中，调用了一个名为 myTest 的测试套件，如代码清单 23-8 所示。

代码清单23-8 myTest.as：包含多个测试用例的FlexUnit测试套件示例

```
package {

    import flexunit.framework.TestCase;
    import flexunit.framework.TestSuite;

    public class myTest extends TestCase {

        public function myTest( methodName:String ) {
            super( methodName );
        }

        public static function suite():TestSuite {
            var ts:TestSuite = new TestSuite();
            ts.addTest( new myTest( "testMilesToKm" ) );
            ts.addTest( new myTest( "testKmToMiles" ) );
            return ts;
        }

        public function testMilesToKm():void {
            var mph:Number = 65;
            var kmh:Number = speedConverter.toKmh( mph );
            assertTrue( "We want to see 104km/h", kmh == 104 );
        }

        public function testKmToMiles():void {
            var kmh:Number =104;
            var mph:Number = speedConverter.toMph( kmh );
            assertTrue( "We want to see 65mph", mph==65 );
        }

    }
}
```

要测试的 ActionScript 类名叫 speedConverter(见代码清单 23-9)，该类用于把速度值从 km/h(公里/小时)转换为 mph(英里/小时)或相反。

代码清单23-9 speedConverter.as：测试套件要对代码进行测试

```
package
{
    public class speedConverter
    {
        public static function toKmh(i:Number):Number
        {
            return (i*1.6);
        }
        public static function toMph(i:Number):Number
        {
            return (i/1.6);
        }
    }
}
```

测试运行程序已经创建了，测试用例也已准备就绪。剩下就是执行它们了。

4. 执行测试

由于涉及好几个文件，因此我们通过图 23-7 来说明一下。

图 23-7　FlexUnit 测试用例涉及的文件

测试运行程序不是项目的主 MXML 文件，因此它不会自动运行。别急！加载 TestRunner.mxml 文件再单击 Run/Launch 应用程序按钮即可解决这个问题。

在这个示例中，是通过单击按钮开始测试的。如果测试出现问题，右侧将出现一堆诊断信息(见图 23-8)。单击一条失败的测试用例，会显示描述测试失败情况的栈跟踪信息。

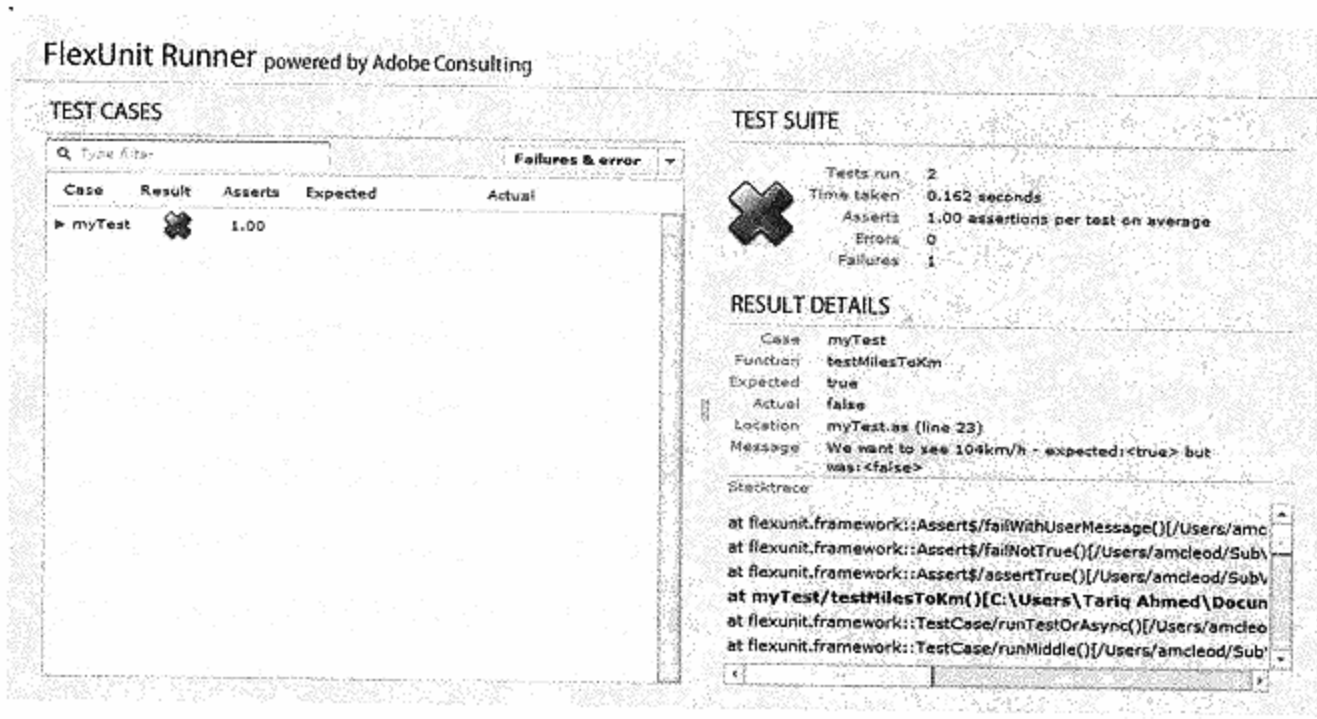


图 23-8　FlexUnit 显示了它遇到的所有问题

Fluint 是另一个 Flex 单元测试框架，它与 FlexUnit 的工作原理类似。

23.2.4　Fluint(单元测试)

Fluint 最初的名字有点古怪，叫做 dpUInt(没有拼错)，它应该是这个领域的后来者。Fluint 是一个开源项目，由 Digital Primates(www.digitalprimates.net)创建并发布。免费下载地址为 http://code.google.com/p/fluint。

Fluint 与 FlexUnit 类似，但它被认为是一个单元和集成测试框架(这就是它名字的由来)。除了与 FlexUnit 具有相同的功能外，Fluint 还提供了可靠的异步支持、测试序列、XML 输出、对测试 UI 组件及 AIR(Adobe Integrated Runtime)的支持。

*测试方法(test metod)*是实际执行测试的函数。测试方法包含在测试用例中，而测试用例

又由测试套件来使用。

类似于 FlexUnit，Fluint 也有文档。下面只是简单地介绍它的用法。

1. 设置 Fluint

Fluint 同样使用一个测试运行程序执行测试，可以把测试运行程序作为独立的项目，也可以将其作为主项目中的一个辅助应用程序运行。本节的示例将在一个现有项目中运行测试。

(1) 从 http://code.google.com/p/fluint/downloads 中下载最新的 SWC 文件。
(2) 在 Flex Builder 中打开要测试的项目。
(3) 右击项目的顶级文件夹，选择 Properties。
(4) 单击 Library Path 选项卡。
(5) 单击 Add SWC 按钮。
(6) 找到 SWC 文件下载到的目录，选中该文件。

安装 Fluint 后，接下来查看如何创建测试运行程序。

2. 创建测试运行程序

测试运行程序就是一个“迷你型”Flex 应用程序，它可以加载并运行所有测试用例。以下是创建 Fluint 测试运行程序的步骤。

(1) 从 http://code.google.com/p/Fluint/downloads/list 中下载示例测试运行程序。
(2) 把 SampleTestRunnerFlex.mxml 文件保存到项目文件夹中。
(3) 打开这个文件，按照代码清单 23-10 所示修改调用 suiteArray.push()的代码行。

代码清单23-10 修改Fluint测试运行程序

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:fluint="http://www.digitalprimates.net/2008/fluint"
  layout="absolute"
  creationComplete="startTestProcess(event)"
  width="100%" height="100%">

  <mx:Script>
    <![CDATA[
    import
    net.digitalprimates.fluint.unitTests.frameworkSuite.FrameworkSuite;

    protected function startTestProcess( event:Event ) : void
    {
      var suiteArray:Array = new Array();
      suiteArray.push( new testSuite() );          <- 添加测试套件
      testRunner.startTests( suiteArray );
    }
    ]]>
  </mx:Script>

  <fluint:TestResultDisplay width="100%" height="100%" />
  <fluint:TestRunner id="testRunner"/>
</mx:Application>
```

然后需要为测试运行程序定义测试用例。

3. 创建测试用例

可以创建任意多个测试用例文件。这些是 ActionScript 类，每个类都有执行某种测试的函数/方法。每个函数的名称必须以小写的 test 开头。

Fluint 会自动搜索以 test 开头的所有函数，因此无需明确调用它们。代码清单 23-11 所示为一个测试用例，用于测试前面 FlexUnit 示例中的 speedConverter 类。

代码清单23-11　testCase.as：创建Fluint测试用例

```
package
{
 import net.digitalprimates.fluint.tests.TestCase;

 public class testCase
 extends net.digitalprimates.fluint.tests.TestCase
 {
  public function testMilesToKm():void
  {
   var mph:Number = 65;
   var kmh:Number = speedConverter.toKmh( mph );
   assertEquals( 104,kmh);
  }

  public function testKmToMiles():void
  {
   var kmh:Number =104;
   var mph:Number = speedConverter.toMph( kmh );
   assertEquals( 65,mph);
  }
 }
}
```

创建所有测试用例

现在有了测试运行程序，它是执行测试的基础；也编写完了测试用例，接下来要把测试用例添加到测试套件中。

4. 创建测试套件

Fluint 测试套件很简单，就是包装所有测试用例。测试套件也是一个 ActionScript 类。

注意看一下代码清单 23-10，在创建测试运行程序时，我们添加了一个名为 testSuite 的测试套件。

```
suiteArray.push( new testSuite() );
```

代码清单 23-12 所示为 testSuite 类的代码。

代码清单23-12　testSuite.as：把测试用例添加到Fluint测试套件中

```
package
{
 import net.digitalprimates.fluint.tests.TestSuite;
```

```
public class testSuite
extends net.digitalprimates.fluint.tests.TestSuite
{
  public function testSuite()
  {
    addTestCase( new testCase() );        <-- 添加所有测试用例
  }
 }
}
```

至此，一切准备就绪。测试用例已经加入测试套件，测试运行程序随时可以执行测试套件。

5. 执行测试

虽然在准备这个示例时创建了很多文件，但执行测试时只需启动 SampleTestRunnerFlex 应用程序即可。图 23-9 所示为执行测试后的结果。

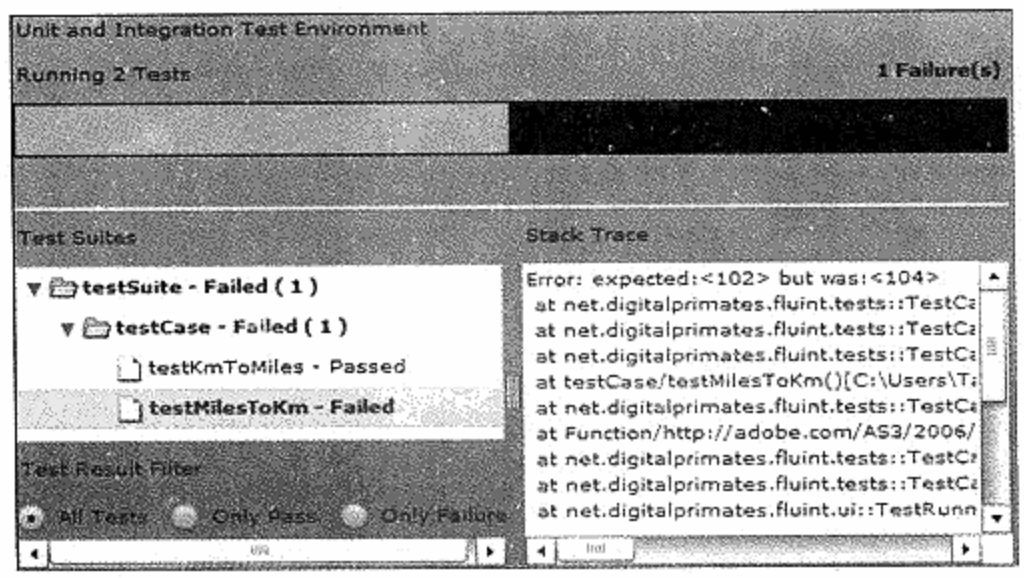


图 23-9 Fluint 在测试结果中突出显示了通过和失败的用例

界面上方是显示每个测试的彩色图表——绿色表示通过，红色表示失败。左侧是包含测试套件及每个测试用例的文件夹结构。右侧是描述测试失败的诊断信息。左侧下方提供了筛选功能，可以选择显示所有测试用例，或者只显示通过的和失败的测试用例。

前面介绍的性能和单元测试，有助于开发人员了解应用程序内部的工作状态。接下来要讨论最后一种测试——功能测试，其重点是从用户角度来测试界面。

23.2.5 FunFX(功能测试)

FunFX 是一个任何人都可以使用的开源项目。就其本身而言，FunFX 更像是一个框架(让用户来创建自动化测试工具)，而非自动化功能测试工具。FunFX 是免费的，该项目的站点是 http://funfx.rubyforge.org/。下载 FunFX 的地址是 http://files.rubyforge.vm.bytemark.co.uk/funfx/。

FunFX 可以为开发人员节省花在功能测试上的时间。FunFX 是使用 Ruby 编写的，利用了其脚本特性。但是应该注意，FunFX 还是一个新项目，目前还没有达到“即插即用”的状态。

23.2.6　RIATest(功能测试)

与售价几千美元的竞争产品相比，RIATest 是一个能够在质量保证阶段为开发团队节省大量时间的低成本解决方案。要使用 RIATest，可以在编译时嵌入一个代理，或者使用运行时加载程序。RIATest 的价格是 499 美元，产品站点为 http://www.riatest.com。

使用这个产品简单直观，表现如下。

- 可以在编译应用程序时嵌入 RIATest 模块，也可以使用其加载器特性动态包装应用程序的代理。
- 通过应用程序记录运行结果(如果愿意，也可以手工编写测试脚本)。
- 能够回放脚本以验证结果。

RIATest 的主要功能如下。

- 通过 Action Recorder 简化了记录操作(action)的过程，而且生成的脚本容易看懂。
- 容易看懂的脚本加上语法加亮生成的彩色代码，更加容易理解。
- RIATest 引擎以自有脚本语言开发，该脚本以 ActionScript 为原型，因此有助于利用 ActionScript 编程经验并实施很多控制。
- 提供了组件查看器(Component Inspector)，鼠标悬停在 UI 组件上即可检查其属性。
- 内置的脚本调试功能可以辅助用户发现测试脚本中的问题。

RIATest 有许多支持者。它提供了一种公开的渠道，方便用户提交改进要求和 bug 报告，因此具有社区推动的性质。

23.2.7　HP QuickTest Pro(功能测试)

QuickTest Pro(简称为 QTP)是由惠普公司(在并购 Mercury Interactive 后)推出的一款行业级产品。这是一款强大而通用的产品，因此售价也高达 6 000～9 000 美元(具体价格取决于许可选项)。要了解有关信息，可以登录 http://www.hp.com，搜索 QuickTest Professional。QTP 是针对 Flex 的第一个功能测试工具，过去很长一段时间内都是用户唯一的选择。

QuickTest Pro 的主要功能如下。

- 缺陷复制。
- 关键字驱动的测试用例开发。
- 通过共享的对象仓库快速更新多个测试脚本。
- 容易使用，只需最低限度培训。
- 简单的界面。

由于测试用例涉及业务流程(而不是纯粹的技术问题)，因此 QTP 对质量保证人员而言更容易使用。

测试用例的创建比较先进。例如，上下文相关或基于条件的测试用例，根据结果可以向下分解为多个分支；可以从数据库中查询处于当前状态的数据，然后使用这些数据执行测试，等等。

23.2.8 IBM Rational Functional Tester(功能测试)

知道 Rational Functional Tester，特别是知道它支持 Flex 的 Flex 开发人员并不多。这款企业级产品主要针对 Java 和.NET 社区，但支持所有应用程序测试，从独立项目到基于 Web 的各种应用程序。Rational Functional Tester 的售价是 5 400～10 500 美元(具体价格取决于许可选项)，产品页面为 http://www-306.ibm.com/software/awdtools/tester/functional/index.html。

Rational Functional Tester 的主要功能如下。

- 数据及关键字驱动的测试用例。
- 支持多种语言编写的测试脚本(包括 Java 和 Visual Basic)。
- 版本控制。
- 动态数据验证。

世界级的 Rational Functional Tester 也有世界级的价位。这款极为稳健的软件产品甚至支持哑终端(dumb terminal clients)的测试自动化，就像 Siebel 和 SAP 的应用程序测试一样。

23.3 小结

在 Flex 问世以来的短短几年内，测试与调试曾一度是其薄弱环节。不过，Adobe 及 Flex 社区也在较短时间内陆续推出了各种工具和先进手段。

说到调试，可以通过一些简单的方式获悉应用程序的运行情况。此外，Flex 的调试器也是一个可靠的工具，使用它能够直观地控制查找问题的过程。

性能分析是对调试的补充，而 Flex 的分析程序(Flex 3 的新特性)为开发人员透视应用程序内部运行细节提供了一种高级手段。通过它能够掌握应用程序占用内存和处理器资源的情况。

测试的首要目标是确保不引入新故障。Flex 应用程序的测试工具包括免费开源的项目，乃至大公司推出的价格不菲的企业级工具。

在调试和测试完应用程序之后，就到达了开发过程的最后一个阶段——项目收尾和部署。在第 24 章，我们将介绍如何为部署应用程序做好准备。

第 24 章 项目收尾

24

本章内容：

- 添加打印功能
- 使用包装器加载应用程序
- 部署应用程序

看看自己吧，满怀激情的 Flex 开发人员：在如此之短的时间内，你已经实现了这么大的飞跃。不仅知道了如何通过数据服务连接到数据库，明白了怎样布局应用程序，而且还懂得了基于事先验证增强可用性，甚至掌握了利用效果创造引人注目的用户体验。

管理层信任你绝对能创建出更加漂亮的具有商务智能的图表，而你的那些 AJAX 好友暗地里都对你羡慕不已。目前来看，还需要了解的只有项目的收尾和发布了。

24.1 打印

请克制一下自己内心的兴奋！还有一个问题不容忽视；打印虽然不是 Flex 应用程序开发中可圈可点的特性(特别是与交互式且具有穿透力的图表相比)，但对用户却非常重要。

在第 1 章中，我们曾谈到过 RIA 与 RWA 的区别，也解释了为什么最初的浏览器和 Web 是一种迥异的文档分发机制。换句话说，Web 浏览器天生就不是独立于平台的功能完备的应用程序，但它的另一面却是天生就擅长表现文档，而打印则更是强项(打开文档，打印文档)。

Flex 恰恰相反。作为一个令人惊叹的应用程序框架，说实话，打印对 Flex 而言确实有点勉为其难(打开实时的应用程序，打印实时的应用程序？)

本节将讨论如何尽量实现最好的打印功能。

24.1.1 在 Flex 中实现打印

Flex 不是由文档构成的(它是实时运行的程序)，因此它在打印时需要创建一个对象，然后再将所有要打印的内容关联到这个对象。Flex 会渲染出打印结果，经过内部屏幕捕获后创建一幅打印图像，最后把图像发送到打印机。

24.1.2 打印工具

在 Flex 中实现打印，要用到几个 ActionScript 类。这些核心类包含在 mx.printing 包中。

- FlexPrintJob：Flex 打印的中坚力量。把打印内容交给它，它负责完成剩下的工作。
- FlexPrintJobScaleType：FlexPrintJob 的辅助类，用于缩放和调整大小。

为了便于实现打印，还有一些专门针对打印做了优化的基于列表的类。它们没有滚动条，但增加了为打印而缩放自身的逻辑。这些类同样包含在 mx.printing 包中。

- PrintDataGrid：经过优化以支持打印的 DataGrid 版本。
- PrintAdvancedDataGrid：经过优化以支持打印的 AdvancedDataGrid 版本。
- PrintOLAPDataGrid：经过优化以支持打印的 OLAPDataGrid 版本。

接下来，我们通过打印示例介绍这些类的用法。

24.1.3 打印示例

了解了相关工具后，我们就来使用 FlexPrintJob 实现打印。以下是实现步骤。

(1) 创建 FlexPrintJob 的实例。

(2) 启动打印过程。操作系统会显示打印对话框，让用户选择打印机、页面方向、要打印的页面等相关选项。

(3) 此时打印池(print spooler)中有了一个打开的流，因此下一步是向池中添加要打印的对象。

(4) 把池数据发送给打印机，打印完成。

注意：

只能打印可视的组件(即 UIComponent 类的子类)。

代码清单 24-1 按照上述步骤实现了一个打印示例。

代码清单24-1 打印示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="vertical" backgroundColor="white">
  <mx:Script>
      <![CDATA[
     import mx.printing.*;
     public function printStuff():void
     {
       var printJob:FlexPrintJob = new FlexPrintJob();
       printJob.start();
       printJob.addObject(stuff);
       printJob.send();
     }
   ]]>
  </mx:Script>
  <mx:TextArea text="This is some Stuff." id="stuff"/>
  <mx:Button click="printStuff()" label="Print Stuff"/>
</mx:Application>
```

不要忘记导入类

实例化FlexPrintJob对象，
启动打印池，
添加对象，
发送数据

编译运行这个应用程序，单击 Print Stuff 按钮，看看结果跟你期望的有没有出入？

不用绕弯子，你一定会感到惊讶的：因为打印出来的文本并不是想象中的小号字体。默认情况下，TextArea 组件会放大到与页面同宽，所以就打出了“巨型”文本。

24.1.4 控制缩放

下面来解决大小不符合期望的问题。为此，需要使用辅助的 FlexPrintJobScaleType 类，该类拥有表 24-1 所示的属性；这些属性都是常量(不能修改的固定值)。如果想在前面的示例中实现原样打印，可以将这些属性作为可选参数传递给 Flex PrintJob 的 addObject()函数：

```
printJob.addObject(stuff,FlexPrintJobScaleType.NONE);
```

表 24-1 FlexPrintJob 用以控制如何缩放的属性

属 性	值	说 明
FILL_PAGE	fillPage	告诉 FlexPrintJob 把对象缩放到 MATCH_WIDTH 表示的宽度或 MATCH_HEIGHT 表示的高度——选择其中较大的值
MATCH_WIDTH	matchWidth	告诉 FlexPrintJob 把对象水平缩放到与页面同宽。如果这样会导致对象高度超过页面高度，则将对象分页打印。这是 FlexPrintJob 默认使用的属性
MATCH_HEIGHT	matchHeight	与 MATCH_WIDTH 类似，不过是把对象垂直缩放到与页面同高。如果这样会导致对象宽度溢出边界，则将溢出部分打印在后续页面中
NONE	none	不缩放——原样打印
SHOW_ALL	showAll	与 FILL_PAGE 相反，即为 FlexPrintJob 应用 MATCH_WIDTH 和 MATCH_HEIGHT 中较小的值

在为应用程序添加打印功能时，要考虑的第一件事就是使用哪个缩放常量(如有必要)最合适。

24.1.5 添加对象的艺术

想成为打印专家，那么要考虑的第二个问题就是如何添加对象。可以反复调用 addObject()方法，但每次调用都会把添加的对象打印到一个新页面上。

解决方法是使用一个容器包装要打印的对象。例如，考虑代码清单 24-2 中的代码。

代码清单24-2 在单独页面上打印对象

```
<mx:Script>
  . . .
  printJob.addObject(text1,FlexPrintJobScaleType.NONE);
  printJob.addObject(text2,FlexPrintJobScaleType.NONE);
  . . .
</mx:Script>

<mx:VBox id="myVBox" width="100%">
  <mx:Text id="text1" text="This is a bunch of text..."/>
  <mx:Text id="text2" text="This is more text text..."/>
</mx:VBox>
```

如果不想把每个对象都打印到独立的页面上，可以添加一个容器，把要打印的对象包装起来。在此，可以使用 VBox 作为容器：

```
printJob.addObject(myVBox,FlexPrintJobScaleType.NONE);
```

容器会被当做一个对象，因而它包装的所有对象会作为一个整体打印出来。有时候，把所有对象都包装起来并不现实，但也有解决方案。

1. 要打印的对象不在同一个容器内

在现实的开发中，不可能把所有要打印的组件都放到一个容器内。就算有这种情况，那如果不想打印其中的全部组件又该如何呢？

还需要考虑想在一页上打印的对象没有被包装在一个容器内的情况。解决方案是使用 ActionScript 临时创建一个容器，然后打印该容器，如代码清单 24-3 所示。

代码清单24-3 使用ActionScript创建临时的打印容器

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="vertical" backgroundColor="white">
<mx:Script>
 <![CDATA[
   import mx.containers.VBox;
   import mx.printing.*;
   public function printStuff():void
   {
     var printJob:FlexPrintJob = new FlexPrintJob();
     var   tmpVBox:VBox = new VBox();
     printJob.start();
     addChild(tmpVBox);
     tmpVBox.addChild(text1);
     tmpVBox.addChild(text2);
     printJob.addObject(tmpVBox,FlexPrintJobScaleType.NONE);
     printJob.send();
     removeChild(tmpVBox);
   }
 ]]>
</mx:Script>
<mx:HBox id="myHBox1" width="100%">
  <mx:Text id="text1" text="Lorem ipsum dolor sit amet"/>
  <mx:Text id="text2" text="Nunc dui purus, egestas id"/>
</mx:HBox>
<mx:HBox id="myHBox2" width="100%">
  <mx:Text id="text3" text="porttitor mi sit amet mi."/>
  <mx:Text id="text4" text="consectetuer adipiscing elit."/>
</mx:HBox>
<mx:Button click="printStuff()" label="Print Stuff"/>
</mx:Application>
```

创建临时的容器

把容器添加到当前视图中

添加要打印的组件

销毁容器

这里必须使用应用程序的 addChild()函数向当前视图中添加临时对象。同时要注意，只能在用户确定打印之后再添加(即在调用 FlexPrintJob 的 start()函数之后)；否则临时容器会出现在用户屏幕上。

2. 使用打印自定义组件

扩展一下前面的解决方案，就可以创建一个专门用于打印的自定义组件，步骤如下。

(1) 创建一个自定义组件，其布局专门为打印而设计。

(2) 创建一个自定义组件的临时实例。

(3) 向自定义组件中传入信息。

(4) 添加自定义组件以完成打印。

(5) 销毁自定义组件。

参见代码清单 24-4 和代码清单 24-5。

代码清单24-4 调用自定义组件解决打印布局问题

```
<mx:Script>
 <![CDATA[
  import mx.printing.*;
  import components.*;
  public function printStuff():void
  {
    var printJob:FlexPrintJob = new FlexPrintJob();
    var   printComp:printComponent = new printComponent();
    addChild(printComp);
    printComp.dAppointment = "05/07/2009";
    printComp.location = "San Jose, CA";
    printJob.start();
    printJob.addObject(printComp,FlexPrintJobScaleType.NONE);
    printJob.send();
    removeChild(printComp);
  }
 ]]>
</mx:Script>
```

如果自定义组件不在根目录下，则需要导入

创建实例

设置数据

代码清单24-5 printComponent.mxml：专门为打印而设计的自定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml">
 <mx:Script>
  <![CDATA[
   [Bindable]
    public var dAppointment:String = "";
   [Bindable]
    public var location:String = "";
  ]]>
 </mx:Script>
 <mx:Label text="Appointment" fontSize="20" fontWeight="bold"/>
 <mx:HBox>
     <mx:Label text="Date: {dAppointment}"/>
   <mx:Label text="Location: {location}"/>
  </mx:HBox>
</mx:VBox>
```

运行以上示例将得到与图 24-1 所示的格式完好的打印结果。

图24-1 使用打印专用组件获得的输出

但是在遇到基于列表的组件时，则需要使用专门的打印版本。

24.1.6 打印列表

常规的基于列表的组件(如 DataGrid)，都是基于屏幕显示设计的；这些组件会在必要时调用滚动条，而且允许用户按照标题排序。显然，这些特性对于打印没有多大用处；在

需要输出基于列表的组件时，应该查看对应的打印版本。适合打印的列表组件针对打印布局进行优化，因此会显示所有记录行以保证打印需要的所有数据。

为适应打印需要，适合打印的列表组件支持对打印页数的控制(因为基于列表的组件可能会包含数千个记录行)，这些控制通过以下的项实现。

- validNextPage：布尔值，表示是否存在下一页。
- nextPage()：切换至下一页数据的函数。

1. 使用自定义组件

为了方便起见，我们以前面的示例为基础，用自定义打印组件包装适合打印的基于列表的控制版本(见代码清单 24-6 和代码清单 24-7)。

代码清单24-6 使用包含PrintDataGrid的自定义组件进行打印

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="vertical" backgroundColor="white">
  <mx:Script>
  <![CDATA[
    import mx.printing.*;
    import components.*;
    public function printPages():void
    {
      var printJob:FlexPrintJob = new FlexPrintJob();
      var printComp:printComponent = new printComponent();

      printJob.start();
      printComp.dataProvider=myAC;
      addChild(printComp);

      printJob.addObject(printComp);
      while(printComp.validNextPage)
      {
        printComp.nextPage();
        printJob.addObject(printComp);
      }
      printJob.send();
      removeChild(printComp);
    }
  ]]>
  </mx:Script>

  <mx:ArrayCollection id="myAC">
    <mx:Object firstName="Jon" lastName="Hirschi"/>
    <mx:Object firstName="Faisal" lastName="Abid"/>
    <mx:Object firstName="Frank" lastName="Krul"/>
    <mx:Object firstName="Abdul" lastName="Qabiz"/>
  </mx:ArrayCollection>
  <mx:Button click="printPages()" label="Print"/>
</mx:Application>
```

传递数据

打印第一个数据页

循环添加剩余页面(如果有)

代码清单24-7 printComponent.mxml：基于PrintDataGrid的自定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:PrintDataGrid xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:columns>
    <mx:DataGridColumn dataField="firstName" headerText="First Name" />
    <mx:DataGridColumn dataField="lastName" headerText="Last Name"/>
```

```
  </mx:columns>
</mx:PrintDataGrid>
```

使用这种方法，可以方便地设置列的布局、指定必需的项渲染器以及完成其他任务(与完全使用 ActionScript 相比而言)。

2. 使用复合自定义组件

虽然前面的示例以最简单的方式实现了打印，但使用复合自定义组件当然也没有问题。例如，需要在打印的每个页面上都显示公司标志；对于这种情况，通过容器来实现也很正常。代码清单 24-8 经过修改使用了代码清单 24-9 中定义的复合组件。

代码清单24-8 使用包含PrintDataGrid的复合组件

```
public function printPages():void
{
 var printJob:FlexPrintJob = new FlexPrintJob();
 var printComp:printComponent = new printComponent();
 printJob.start();
 addChild(printComp);
 printComp.width=printJob.pageWidth;
 printComp.height=printJob.pageHeight;
 printComp.printAC=myAC;

 printJob.addObject(printComp);

 while(printComp.pdg.validNextPage)
 {
  printComp.pdg.nextPage();
  printJob.addObject(printComp);
 }

  printJob.send();
  removeChild(printComp);
}
```

❶ 将组件大小设置为与页面大小相同

这种情况下，有必要注意的是：容器不知道自己应该是多大，因此基于 printJob 的 pageWidth 和 pageHeight 属性设置它的大小❶很关键。

代码清单24-9 printComponent.mxml：包含PrintDataGrid的复合组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml">
 <mx:Script>
  <![CDATA[
   import mx.collections.ArrayCollection;
   [Bindable]
   public var printAC:ArrayCollection;
  ]]>
 </mx:Script>
 <mx:Image source="@Embed(source='assets/cflex.png')"/>
 <mx:PrintDataGrid id="pdg" dataProvider="{printAC}">
  <mx:columns>
   <mx:DataGridColumn dataField="firstName" headerText="First Name" />
   <mx:DataGridColumn dataField="lastName" headerText="Last Name"/>
  </mx:columns>
 </mx:PrintDataGrid>
</mx:VBox>
```

这种方法还有一个不错的用途，即只在起初和最后的页面中根据情况添加页眉和页脚。读者不妨自己试一试。

24.1.7 捕获用户执行取消操作的时间

检测用户何时单击了打印弹出窗口中的 Cancel(取消)按钮也很简单(见代码清单24-10)。FlexPrintJob 的 start()函数会根据用户的操作返回 true 或 false(true 表示打印，false 表示取消)，因此可以检测这个函数的返回值。

代码清单24-10 区分开始和取消操作

```
var printJob:FlexPrintJob = new FlexPrintJob();
if(printJob.start())
{
 printJob.addObject(myObj);
 printJob.send();
}
else
{
 mx.controls.Alert("User cancelled printing.");
}
```

这里没有什么新内容，但可以把利用这种技术作为一种最佳实践(即只在“绿灯亮”的时候才添加对象)。

24.1.8 FlexReport

FlexReport 是由 Frederico Garcia 领导的一个开源项目，旨在改进 Flex 应用程序的打印体验(特别是在报表的层面上)。这个项目主要针对以下领域：

- 报表布局；
- 增强的多页支持；
- 打印预览。

FlexReport 本质上是一组 ActionScript 和 MXML 组件，用户可以根据需要调用。要了解更多信息，请访问 http://www.kemelyon.com/bts。

24.2 自定义包装器

包装器(wrapper)就是加载 Flex 应用程序的东西。

在构建和启动应用程序时，Flex Builder 会自动创建包装器。包装器是一个 HTML 文件，其中包含启动 Flex 应用程序(SWF 文件)的必要代码。

Flex Builder 默认生成的包装器作用如下。

- 加载 JavaScript 以支持“后退/前进”的历史导航功能和为应用程序视图添加书签(称为深层链接)的功能。
- 检查并确保用户安装了最低要求的 Flash Player。
- 创建加载 Flex 应用程序的 HTML。
- 支持快速安装(Express Install)，减少用户在升级 Flash Player 时的麻烦，确保用户安装了最低要求的 Flash Player。

通过包装器非常适合进行如下操作。

- 传入发送给 Flex 应用程序的外部参数。
- 嵌入 Flex 应用程序需要使用的 JavaScript 函数。

接下来，看一看包装器文件的位置及它们的形式。

24.2.1 包装器文件

打开 Flex Builder 项目文件中的 bin-debug 文件夹，估计读者早就对这些不知从哪里冒出来的文件感到莫名其妙了。下面我们解释这些文件的用途。

- <filename>.html：主包装器文件。所有操作都通过这个文件实现。
- AC_OETags.js：支持 JavaScript 函数(例如检查 Flash Player 版本)以及启动 Flex 应用程序的函数。
- playerProductInstall.swf：当用户的 Flash Player 低于最低要求时会被调用。通过一种简单的机制提醒用户安装最新版本。
- history/*：包含支持 Flex 应用程序实现历史导航功能的文件。

不过，开发人员无需自己从头创建这些文件，因为 Flex 提供了相应的模板。

24.2.2 包装器模板

如果需要自定义包装器，首先应该找到本示例中将会用到的 Flex 包装器模板。包装器模板保存在下面目录中。

```
<installdir>\sdks\3.0.0\templates
```

在 Windows 中，如果把 Flex Builder 安装在默认位置，则模板目录应该为：

```
C:\Program Files\Adobe\Flex Builder 3\sdks\3.0.0\templates
```

其中包含如下模板文件。

- client-side-detection：确定用户是否安装了适当版本的 Flash Player，只有在满足条件的情况下才加载应用程序。
- client-side-detection-with-history：验证 Flash Player 的版本及支持浏览器历史导航功能。
- express-installation：如果用户需要升级，则启动快速安装。
- express-installation-with-history：启动快速安装以及浏览器历史管理。
- no-player-detection：一个基本的示例。
- no-player-detection-with-history：基本的示例以及历史管理。

要使用任何一种模板，都可以按照下面这样操作。

(1) 打开所需的模板目录。

(2) 选择所有文件。

(3) 把它们复制到项目的 html-template 目录中。

此时，如果想添加自定义的内容，请打开 index.template.html。

下面先介绍包装器的最低配置——必需的组成部分。

24.2.3 最基本的要求

包装器中必须包含两个主标签。

- <object>：针对 Internet Explorer。
- <embed>：针对基于 Mozilla 的浏览器(如 FireFox)。

要想知道最最基本的包装器是什么样子，参见代码清单 24-11，其中只包含了启动 Flex 应用程序必需的部分。

代码清单24-11 最基本的包装器

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    width="${width}" height="${height}"
   codebase="http://fpdownload.macromedia.com/get/flashplayer/
     ➥ current/swflash.cab">
  <param name="movie" value="${swf}.swf" />
    <embed src="${swf}.swf"
      width="${width}" height="${height}"
      type="application/x-shockwave-flash"
      pluginspage="http://www.adobe.com/go/getflashplayer">
     </embed>
 </object>
```

注意：

在编译应用程序时，Flex Builder 会在占位符${...}的位置上插入适当的值。

当然，如果知道用户只会使用特定的浏览器(例如，在内部网中，公司可能只允许安装一种浏览器)，那么还可以再精减。

24.2.4 嵌入到 Web 应用程序中

读者如果使用 ColdFusion、PHP 或 JSP 等 Web 技术开发了应用程序，也可以将 Flex 应用程序嵌入 Web 应用程序中。为此，最简便的方式就是重命名生成的包装器文件(.html)的扩展名，将其修改为与服务器端 Web 应用程序文件一致。这样可以立即把包装器转换到相应的 Web 应用程序环境下。

例如，对于 ColdFusion 应用程序，把应用程序的扩展名改为.cfm，然后它就能够像其他任何 ColdFusion 模板一样执行——可以访问各种作用域(应用程序、会话、请求和 CGI 等)，而且也可以嵌入任何必要的逻辑。

通常需要添加哪些逻辑呢？以下为两种可能。

- 向 Flex 应用程序中传递参数，例如当前的 UserID 或配置信息。
- 添加安全代码，预先检查用户是否具备启动应用程序的权力。

利用这种能力也可以把 Flex 应用程序嵌入到 Web 应用程序文件中。

把 Flex 应用程序作为 Portlet

Portlet 是指存在于另一个应用程序中的应用程序。在上述思路基础上发挥一下，可以

在 Web 应用程序文件中使用包含机制(或者生成包装器启动代码)，实现在 Web 应用程序中启动 Flex 应用程序。

例如，在 ColdFusion 中，如果把包装器文件由 MyProject.html 重命名为 MyProject.cfm，就可以编写以下代码：

```
<cfinclude template="/path/to/MyProject.cfm">
```

这样就实现了在 ColdFusion 文件中调用 Flex 应用程序。

24.2.5 在包装器中传递参数

无论将 Flex 应用程序作为整个 Web 应用程序中的 portlet 启动，还是通过某种服务器端技术执行包装器，都会涉及向 Flex 应用程序传入启动参数。传入参数不难——只需在包装器模板(即 Flex 项目中的 html-template/index.template.html 文件)或生成的包装器中添加叫做 flashvars 的设置即可。

打开包装器文件(在这种情况下，是 MyProject.html)，可以找到与代码清单 24-12 所示类似的内容。

代码清单24-12 包装器文件中调用应用程序的代码

```
} else if (hasRequestedVersion) {
 AC_FL_RunContent(
  "src", "CH24",
  "width", "100%",
  "height", "100%",
  "align", "middle",
  "id", "CH24",
  "quality", "high",
  "bgcolor", "#ffffff",
  "name", "CH24",
  "allowScriptAccess","sameDomain",
  "type", "application/x-shockwave-flash",
  "pluginspage", "http://www.adobe.com/go/getflashplayer"
);
```

这是一个调用 Flex 应用程序的关键 JavaScript 函数，也是添加其他 flashvars 变量的地方。如果想传入用户的安全凭证，可以参见代码清单 24-13。

代码清单24-13 通过添加flashvars属性来为应用程序传入信息

```
} else if (hasRequestedVersion) {
 AC_FL_RunContent(
  "src", "CH24",
  "width", "100%",
  "height", "100%",
  "align", "middle",
  "id", "CH24",
  "quality", "high",
  "bgcolor", "#ffffff",
  "name", "CH24",
  "allowScriptAccess","sameDomain",
  "flashvars","UserID=12321&SessionKey=BAE23424A",
  "type", "application/x-shockwave-flash",
  "pluginspage", "http://www.adobe.com/go/getflashplayer"
);
```

如果已经把包装器转换为可以执行代码的包装器，则可以动态传递信息。例如，使用ColdFusion 可以像下面这样实现：

```
   "allowScriptAccess","sameDomain",
<cfoutput>
   "flashvars","UserID=#Session.UserID#&SessionKey=#Session.Key#",
</cfoutput>
   "type", "application/x-shockwave-flash",
```

在 Flex 应用程序中，则要通过下列语法访问这两个值。

```
Application.application.parameters.UserID;
Application.application.parameters.SessionKey;
```

最终的完整实现如代码清单 24-14 所示。在初始化应用程序期间，接收所有参数并将它们保存为本地变量。

代码清单24-14 在Flex中访问包装器flashvars

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
            backgroundColor="white" creationComplete="initApp()">
<mx:Script>
 <![CDATA[
   import mx.controls.Alert;
   [Bindable]
   public var UserID:int;

   [Bindable]
   public var SessionKey:String;

   public function initApp():void
   {
      UserID = Application.application.parameters.UserID;
      SessionKey = Application.application.parameters.SessionKey;
         mx.controls.Alert.show("UserID="+UserID+" Key="+SessionKey);
   }
 ]]>
</mx:Script>
</mx:Application>
```

这就是全部代码。如果还有其他需求，可以再添加相应代码。在此基础上，下一步考虑 UserID 和 SessionKey，并使用后端数据服务验证是否匹配，以确定用户的身份是否属实。

24.3 部署

这是项目收尾前的最后一项工作——该让它发挥作用了！现在应用程序已经过测试，得到质量保证部门和用户的认可，下一步就是把它部署到生产环境中去。

由于应用程序的实现千差万别，所以并没有一成不变的解决方案。不过本节将向读者介绍一些通用的原则和过程。

24.3.1 创建生产版本

在构建 Flex 应用程序的整个过程中，构建生成的文件都默认保存在 bin-debug 文件夹中。好好猜猜，其中都是些什么文件呢？

虽然问题的答案显而易见，但如果你说“是调试文件”，也值得赞赏。通过 Flex 应用程序的调试版，可以进行内存分析和单步调试等。

然而对于生产版本来说，就没有必要考虑那么多了。要创建应用程序的生产版本，请按照以下步骤操作：

(1) 选择 File | Export | Release Build，将弹出一个对话框，可以通过它配置任何应用程序的生产版本。

(2) 选择需要的应用程序。

(3) 选择输出文件夹(默认创建 bin-release 文件夹，如图 24-2 所示)，然后单击 Finish。

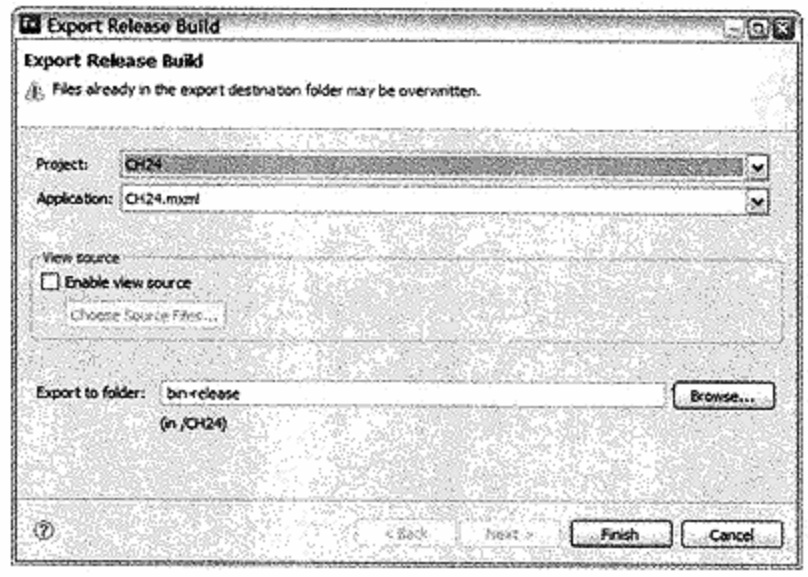


图 24-2　通过 Export Release Build 对话框可以创建应用程序的生产版本

然后 Flex Builder 会创建输出文件夹并把生成的各种应用程序文件放入其中。

注意：

选中 Enable View Source 选项，可以允许用户查看应用程序的源代码。

有了应用程序的生产版本仍然没有结束，因为在部署时不能丢下任何客户端文件。

24.3.2　放置客户端文件

下一步需要把所有客户端的 Flex 文件复制到适当的地方去。具体需要复制哪些文件因应用程序而异，不过我们在此可以提供一份参考清单。

- 发布版本过程生成的所有文件
  - 包装器文件
  - JavaScript 文件(如 AC_OETags.js)
  - history 文件夹
  - playerProductInstall.swf
  - SWF 文件
- 外部资产(没有直接嵌入到应用程序中的文件)
  - 图像和图标

  - 音频/MP3 文件
  - 视频文件
  - 样式表
- 配置文件
  - XML 文件
- 库文件
  - RSL
  - SWC
  - SWF 模块

前端文件复制就位后，接下来应该处理位于服务器上的后端文件。

24.3.3 放置服务器端文件

如果应用程序的运行依赖于后端文件，则需要把相关的文件也布置到生产环境中。服务器端文件的部署与服务器端技术有关。不过，如果已经在某种 Web 技术基础上创建了自定义的包装器，则相关文件都必须各就各位。

其他相关的服务器端配置工作可能涉及如下。

- 远程对象、Web 服务和数据服务。
- 数据库更改。
- 服务器端配置。
- Web 服务器配置。

最后验证应用程序能否正常运行。

24.3.4 测试效果

这一步是不言而喻的。但是，无论采取分段部署，还是直接投入生产，最终的运行环境终究有别于本地开发环境。现在是通过 Web 服务器访问应用程序，就不能再像本地那样直接访问资源了。例如，访问 URL 应该从：

```
file:///C:/Documents%20and%20Settings/tahmed/My%20Documents/
  Flex%20Builder%203/CH24/bin-debug/CH24.html
```

变为基于 HTTP 的 URL：

```
http://www.cflex.net/myApplication.cfm
```

而且，由于要通过网络通信，还必须掌握如下因素。

- 网络访问。
- 防火墙。
- Web 服务的配置/身份验证。
- Flash Player 能否使用浏览器中的 cookie。

如果应用程序不能马上运行，以上是需要首先考虑的问题。

24.4 小结

这就是本书的结尾了！读者已经掌握了一名高效 Flex 开发人员必备的知识。当然，今后要学习的东西还有很多。不过也用不着为此担心，随着对这门技术掌握得越来越熟练，你的视野自然会越来越开阔，领悟和掌握的知识也将越来越丰富。

合上本书之后，希望读者尽快把所学知识付诸实践，使用 Flex 开发几个真正的应用程序。可能会有一些困难，不过相信读者一定能够克服，而且你随时可以寻求帮助。

作为 Flex 开发人员，最令人感到惬意的莫过于它富于互助精神的社区(你也将成为其中一员)了。在本章后面的“在线资源”部分查一查，加入你感兴趣的 Flex 用户组，积极地参加有关 Flex 的活动吧。

随着对 Flex 技术的日臻熟练，你可以通过参与论坛讨论、撰写博客、向媒体投稿等方式，推动社区的发展。我们衷心希望本书能够成为你永远怀念的启蒙教材，更盼望你能在 Flex 技术圈子里脱颖而出。

附录 A　在线资源

开发人员资源

- Adobe Flex Developer Center：http://www.adobe.com/devnet/flex/。
- AYFABTU 聚合博客：http://www.allyourflexarebelongtous.com/。
- Community Flex：http://www.cflex.net/。
- Flex.org：http://www.flex.org/。
- Flex 开发人员中心：http://www.adobe.com/devnet/flex/。
- JAM: Just ActionScript and MXML：http://www.onflex.org/code/。
- FlexBox(第三方 Flex 组件目录)：http://flexbox.mrinalwadhwa.com/。
- MyFlex.org(商业性 Flex 组件)：http://www.myflex.org/。
- ScaleNine(主题和皮肤)：http://www.scalenine.com/。
- Adobe MXNA 聚合博客：http://weblogs.macromedia.com/mxna/。
- Feed-squirrel.com 聚合博客：http://www.feed-squirrel.com/。
- Flex Search：http://flexsearch.org/。
- Flex Cookbook：http://www.adobe.com/go/flex_cookbook/。

论坛

- FlexCoders：http://www.adobe.com/go/flexcoders。
- Flex 组件开发：http://tech.groups.yahoo.com/group/flexcomponents/。
- Flex 支持论坛：http://www.adobe.com/cfusion/webforums/forum/index.cfm? forumid=60。
- FlexCoders.ru(俄罗斯)：http://www.flexcoders.ru/。
- FlexForum.de(德国)：http://www.flexforum.de/。
- FlexComponents：http://www.adobe.com/go/flexcomponents。
- House of Fusion(针对 ColdFusion 开发人员的 Flex)：http://www.houseoffusion.com/groups/flex/。
- 欧洲 Flex 开发人员论坛：http://www.flexdeveloper.eu/。

开源项目

FlexLib(开源 Flex 2 组件库)：http://code.google.com/p/flexlib/。
Cairngorm 微型架构(轻量级框架)：http://www.adobe.com/go/cairngorm/。
eBay 的 ActionScript 3 库：http://code.google.com/p/as3ebaylib/。
Flickr 的 ActionScript 3 库：http://code.google.com/p/as3flickrlib/。
Mappr 的 ActionScript 3 库：http://code.google.com/p/as3mapprlib/。
Odeo 的 ActionScript 3 库：http://code.google.com/p/as3odeolib/。
RSS/ATOM 的 ActionScript 3 库：http://code.google.com/p/as3syndicationlib/。
YouTube 的 ActionScript 3 库：http://code.google.com/p/as3youtubelib/。
RIA Forge：http://www.riaforge.org/。

Flex 技术圈

Adobe Labs(实验阶段的产品)：http://labs.adobe.com/。
The Flex Show 播客：http://www.theflexshow.com/blog/。
FlexCursion(Flex 爱好者的社区网络)：http://flexcursion.ning.com/。
Flex Authority 杂志：http://www.flex-authority.com/。

用户组

北美洲的用户组

硅谷 Flex 用户组：http://www.silvafug.org/。
西雅图 Flex 用户组：http://www.flexilicio.us/。
爱达荷州 Adobe 开发人员社区：http://www.adidaho.org/。
犹他州 Flex 用户组：http://www.flexusergroup.org/。
丹佛 RIA 开发人员组：http://www.ria5280.org/。
明尼苏达州 Flash/Flex/AIR 编程人员组：http://www.mnswf.com/。
密歇根州 Flex 用户组：http://www.theflexgroup.org/。
印第安纳波利斯 Adobe Flex 用户组：http://www.indyflex.com/。
多伦多 Flex 用户组：http://www.torontoflex.org/。
纽约 Flex 用户组：http://www.nyflex.org/。
费城 Flash 及 Flex 用户组：http://www.pfpaug.org/。
华盛顿首都地区 Flex 用户组：http://www.dc-flex.org/。
亚特兰大 Flash 及 Flex 用户组：http://www.affug.com/。

其他部分国家的用户组

法国 Adobe Flex 用户组：http://fr.groups.yahoo.com/group/flexeurs/。
意大利 Flex 用户组：http://www.augitaly.com/。
东京 Flex 用户组：http://www.fxug.net/。
韩国 Flex 用户组：http://www.fxug.co.kr/。
新西兰 Adobe Flex 用户组：http://fxug.org.nz/。
西班牙 Flex 用户组：http://www.madeinflex.com/。