Struts In Action

使用领先的Java框架构建Web应用

中文版

Ted Husted 著

Eric Liu(铁手) 译

目 录

第一部分

Struts入门

第一部分是 Struts 入门。我们将介绍 Java web 应用，剖析框架的结构，构建两个简单的应用，并讨论 Struts 的配置组件。

1.介绍

本章内容

- *应用框架介绍*
- *理解 HTTP, CGI, servlet, 和 JSP*
- *使用 Model 2 架构*
- *构建一个简单的 Web 应用*

The only stupid question is the one you never ask.

—佚名

1.1. 关于本书

欢迎你阅读《*Struts In Action*》。本书的目的是帮助 Web 应用开发者能够最好的使用 Struts web 应用框架。

Struts 是一个开源软件，有助于开发者更加快速和容易地建立 Web 应用程序。Struts 依靠绝大多数开发者已熟知的标准技术—比如 JavaBeans, Java servlet, 以及 JavaServer Page (JSP)。通过基于标准的技术，“填空式”的软件开发方法，Struts 可以减轻在创建新项目时那些令人抱怨的极费时间的工作。

1.1.1. 谁创建了Struts?

Struts 是 Apache 软件基金下 Jakarta 项目子项目。除 Struts 之外, Jakarta 还有其他成功的开源产品，包括 Tomcat, Ant, 和 Velocity。

开始的代码基础从 2000 年 5 月开始开发，直到 2001 年 6 月，1.0 版本发布。有 30 多个开发者参与进来，并有数千人参与到讨论组中。Struts 代码基础由一个志愿者团队来管理。到 2002 年，Struts 小组共有 9 个志愿者参与。

Struts 框架的主要架构设计和开发者是 Craig R. McClanahan。Craig 也是 Tomcat 4 的主要架构师，以及 Java Web Services Developer Pack 的主要架构师和实现者。他现在是 Sun 的 JavaServer Faces (JSR-127) 的规范领导以及 J2EE 平台的 Web 层架构师。

Struts 在 Apache 软件许可 [ASF, License]下对公众是免费的。使用此软件没有任何获得和再现成本。不象其他一些开源许可协议，Apache 软件许可对商业用途是友好的。你可以在你的商业项目中使用 Struts，并自由分发 Struts 库。你也可以将 Struts 组件集成到你的框架中，就像他们是你自己编写的一样。详细情况，参见 Apache Software License，www.apache.org/LICENSE。

1.1.2. 为什么Struts 要开源?

现在有许多非常优秀 Java 程序和框架都是开源项目。有许多的开发人员为这些项目工作，他们同时又在诸如 IBM， Sun Microsystems， 以及 Apple 这样的公司从事其日常工作。这类软件的开发式协作有利于整个软件市场。今天，许多开源组件都集成到了商业产品之中。公司可以向其客户出售其专业的文档，保证支持服务水平，以及其他有价值的售后服务和增值服务。

当软件是自由的时候，对市场来说它更容易得到支持。Struts 就是个典型例子。虽然它还只是个很新的产品，也已经有很多文章和教程涉及到它，但却还没有什么象样的书籍。

许多开发团队不喜欢使用不是自己内部开发的软件。开源组件提供了所有自行开发的软件的优点，但绝不会将你锁定在一个只有你们团队才懂的专有解决方案上。

开源软件对所有人都是双赢的。

1.1.3. 为什么叫Struts?

这个框架之所以叫“Struts”，是为了提醒我们记住那些支撑我们房屋，建筑，桥梁，甚至我们踩高跷时候的支撑。 这也是一个对 Struts 在开发 Web 应用程序中所扮演的角色的精彩描述。当建立一个物理建筑时，建筑工程师使用支柱为建筑的每一层提供支持。同样，软件工程师使用

Struts 为业务应用的每一层提供支持。

什么是应用框架?

框架（framework）是可重用的，半成品的应用程序，可以用来产生专门的定制程序[Johnson]。象人一样，软件应用的相似性比不同点要多。它们运行在相似的机器上，期望从相同的设备输入信息，输出到相同的显示设备，并且将数据存储到相同的硬盘设备。开发传统桌面应用的开发人员更习惯于那些可以涵盖应用开发同一性的工具包和开发环境。构架在这些公共基础上的应用框架可以为开发人员提供可以为他们的产品提供可重用服务的基础架构。

框架向开发人员提供一系列具有以下特征的骨架组件：

- 已经知道它们在其它程序上工作的很好；
- 它们随时可以在下一个项目中使用；
- 它们可以被组织的其它团队使用；

对于框架是典型的**构建还是购买**命题。如果你自己构建它，在你完成时你就会理解它，但是在你被融入之前又将花费多长时间呢？如果要购买，你必须得克服学习曲线，同样，在你可以用它进行工作之前又得花多长时间？这里没有所谓正确答案，但许多观察者都会同意，象 Struts 这样的框架能提供比从头开始开发更显著的投资回报，特别是对于大型项目来说。

其它类型的框架

框架的概念不仅用于应用程序也可用于组件。通过此书，我们也介绍其它可以和 Struts 一起使用的框架。这些包括 Lucene 搜索引擎，Scaffold 工具包，Struts 验证器，以及 Tiles 标签库。与应用框架一样，这些工具也提供了一些半完成的版本，可以用在用户的定制组件之中。

某些框架被限制于专门的开发环境中。Struts 以及本书中涉及的组件却不是这样。你可以在很多环境中来开发 Struts: Visual Age for Java, JBuilder, Eclipse, Emacs, 甚至使用 Textpad 。

对于你的工具，如果你可以用来开发 Java，你就可以用它来开发 Struts[1]。

使用的技术

使用 Struts 的应用开发使用了大量的其他基础技术。这些技术并不是专门针对 Struts ，而是所有 Java web 应用都可以使用的。开发者使用 Struts 之类的框架是为了隐藏在诸如 HTTP，CGI，以及 JSP 之类技术后面的繁琐的细节。作为一个 Struts 开发者，你并不需要知晓所有的相关知识，但是这些基本技术的工作原理可能有助于你针对棘手问题设计出创造性的方案。如果你已经非常熟悉这些技术，你可以跳过这些章节到 1.4 节。

超文本传输协议（HTTP）

当两个国家之间进行调解时，外交官们总是遵循一定的正式***协议***。外交协议主要设计来避免误解，以及防止谈判破裂。同样，当计算机间需要对话，它们也遵循一个正式的协议。这个协议定义数据是如何传输，以及它们到达后如何进行解码。Web 应用程序就是使用 HTTP 协议在运行浏览器的计算机和运行的服务器的程序间传输数据。

很多服务器应用程序使用 HTTP 之外的其他协议。他们在计算机之间维护一个持久性的连接。

---

[1] 译者注：目前很多大型公司也重视到它，它们的工具也提供相应的Struts开发支持。比如IBM WSAD，BEA WorkShop等。另外，一些公司还提供可视化的Struts集成开发环境。

应用服务器可以清楚的知道是谁连接上来，而且何时中断连接。因为它们知道每一个连接的状态，以及每一个使用它的人。这称之为状态协议。

相反， HTTP 是一个无状态协议。HTTP Server 可以接受来自于各种客户的各种请求，并提供各种响应，即使是这个响应仅仅是说 No。没有大量的协商和连接持久性，无状态协议可以处理大量的请求。这也是 Internet 可以扩展到很多计算机的原因。

HTTP 成为通用标准的原因是其简单性。HTTP 请求看起来就像一个平常的文本文档。这使应用程序很容易创建 HTTP 请求。你甚至可以通过标准的程序如 Telnet 来手动传递一个 HTTP 请求。当 HTTP 响应返回时，它也是一个开发者可以直接阅读的平面文本。

HTTP 请求的第一行包含方法，其后是请求的来源地址和 HTTP 版本。HTTP 请求头跟在首行后面，可以没有也可以有多个。HTTP 头向服务器提供额外的信息。可以包括浏览器的种类和版本，可接受的文档类型，浏览器的 cookies 等等。7 种请求方法中， GET 和 POST 是用得最多的。

一旦服务器接收到请求，它就要产生一个 HTTP 响应。响应的第一行称为状态行，包含了 HTTP 协议的版本，数字型状态，以及状态的简短描述。状态行后，服务器将返回一个 HTTP 响应头，类似于 HTTP 请求头。

如上所述， HTTP 并不在请求间保持状态信息。服务器接受请求，发出响应，并且继续愉快地处理文本请求。

因为简单和效率，无状态协议不适合于需要跟踪用户状态的动态应用。

Cookies 和 URL 重写是两个在请求间跟踪用户状态的方式。cookie 是一种特殊的信息包，存储于用户的计算机中。URL 重写是在页面地址中存储一个特殊的标记，Java 服务器可以用它来跟踪用户。这两种方法都不是无缝的，是用哪一个都意味着在开发时都要进行额外的工作。对其本身来说，标准的 HTTP web 服务器并不传输***动态内容***。它主要是使用请求来定位文件资源，并在响应中返回此资源。通常这里的文件使用 Hypertext Markup Language (HTML) [W3C, HTML] 格式化，以使浏览器可以显示它们。HTML 页面通常包含一些到其他页面的超文本连接，也可以显示其他一些内容比如图像和视频等等。用户点击连接将产生另一个请求，就开始一个新的处理过程。

标准 web 服务器处理静态内容处理得很好，但处理动态内容时则需要额外的帮助手段了。

定义 **静态内容直接来自于文本或数据文件，比如 HTML 或者 JPEG 文件。这些文件可以随时改变，但通过浏览器请求时，却不能自动改变。相反，动态内容是临时产生的，典型地，它是针对浏览器的个别请求的响应。**

公共网关接口(CGI)

第一个普遍用来产生动态内容的标准是通用网关接口（Common Gateway Interface (CGI)）。CGI 使用标准的操作系统特征，比如环境变量和标准输入输出，在 Web 服务器间以及和主机系统间创建桥接和网关。其他程序可以看到 web server 传递过来的请求，并创建一个定制的响应。

当 web 服务器接收到一个对 CGI 程序的请求时，它便运行这个程序并向其提供它请求里面所包含的信息。CGI 程序运行，并将输出返回给 Web server，web server 则将输出响应给浏览器。

CGI 定义了一套关于什么信息将作为环境变量传递，以及它希望怎样使用标准输入和输出的惯例。与 HTTP 一样，CGI 是灵活和易于实现的，并且已经有大量现成的 CGI 程序。

CGI 的主要缺点是它必须为每个请求运行一个程序。这是一个相对昂贵的处理方法，对大容量

站点来说，每分钟有数千个请求，有可能使站点瘫痪。CGI 程序的另一个缺点是平台依赖性，一个平台上开发的程序不一定在另一个平台上能运行。

Java servlet

Sun 公司的 Java Servlet 平台直接解决了 CGI 程序的两个主要缺点。

首先，servlet 比常规 CGI 程序提供更好的性能和资源利用。其次，一次编写，随处运行的 JAVA 特性意味着 servlet 在有 JVM 的操作系统间是轻便的可移动的。

Servlet 看起来好像是一个微小的 web server。它接受请求并产生响应。但，和常规 web server 不同，servlet API 是专门设计来帮助 Java 开发人员创建动态应用的。

Servlet 本身是要编译成字节码的 Java 类，就像其他 Java 对象一样。Servlet 访问 HTTP 特定服务的 API，但它仍然是一个运行于程序之中的 Java 对象，并可以利用所有的 Java 资产。

为了使常规 web servers 能访问 servlet，servlet 被安插在一个容器之中。Servlet 容器连接到 Web 服务器。每个 servlet 都可以声明它可以处理何种样式的 URL。当符合所注册样式的请求到达，web server 将请求传递给容器，容器则调用响应的 servlet。

但和 CGI 程序不同，并不是针对每个请求都要创建一个新的 servlet。一旦容器实例化了一个 servlet，它就仅为每个新的请求创建一个新的线程。Java 线程可比使用 CGI 程序的服务器处理开销小多了。

一旦 servlet 被创建，使用它处理额外的请求仅带来很小的额外开销。Servlet 开发人员可以使用 `init()` 方法保持对昂贵资源的引用，比如到数据库或者 EJB Home 接口的连接,以便它们可以在不同的请求之间进行共享。获得这些资源要耗费数秒时间，这比大多数冲浪者愿意等的时间要长些。

Servlet 的另一个好处是，它是多线程的， servlet 开发人员必须特别注意确保它们的 servlet 是线程安全的。学习 servlet 编程，我们推荐 *Java Servlets by Example*, 作者 Alan R. Williamson [Williamson]。

JavaServer Pages

虽然 servlets 对 CGI 程序来说前进了一大步，但它也不是万能灵药。为了产生响应，开发人员不得不使用大量的 `println` 语句来生成 HTML。比如这样的代码

```
out.println("<P>One line of HTML.</P>");
out.println("<P>Another line of HTML.</P>");
```

在产生 HTTP 响应的 Servlet 中是很普遍的。也有一些库有助于你产生 HTML。随着应用越来越复杂，Java 开发人员将不再扮演 HTML 页面设计的角色。

同时，大多数项目经理更喜欢将团队分成不同的小组。 它们喜欢 HTML 设计人员处理表现层的工作，而 Java 工程师则专注于业务逻辑。单独使用 servlet 的做法鼓励混合标记和业务逻辑，很难区分团队人员的专业工作。

为解决这个问题，Sun 提出了一个将脚本和模板技术结合到一个组件中的服务器页面技术（JavaServer Pages）。为创建 JSP 页面，开发者按创建 HTML 页面类似的方式创建页面，使用相同的 HTML 语法。为将动态内容引入页面，开发人员可以将脚本元素置入页面之中。脚本元素是一些标记，封装了可以被 JSP 识别的逻辑。你可以在 JSP 页面中很容易的识别出脚本元素，他们被封装在一对`<%` 和 `%>`标记中。例如，要显示页面的最后修改日期，开发人员可以将以下代码放入页面中：

```
<B>This page was accessed at <%= new Date() %></B>
```

有三种不同的脚本元素：

表达式，脚本小程序和声明。如表1.1所示：

表格 1.1.1 JSP 脚本元素

元素	目 的
表达式	Java代码，封装在<%= 和 %>之中，用来计算JAVA语句的值，并将结果插入Servlet的输出之中
脚本程序	Java代码，封装在<% 和 %>之中，常用来创建动态内容
声明	Java代码，封装在<%！和 %>之中，常用来添加代码到Servlet类的body之中

为了识别 JSP 页面，文件需要保存为扩展名.jsp。当一个客户请求 JSP 页面时，容器将页面翻译成 Java servlet 源代码文件，并将它编译成 Java 类文件--就象你写的 servlet 文件一样。在运行时，容器也能检测 JSP 文件和相应的类的最后更新时间。如果，JSP 文件自上次编译以来被修改了，容器将重新翻译和编译 JSP 文件。

项目经理现在可以将表现层分派给 HTML 开发人员，将业务逻辑工作分派给 JAVA 开发人员。重要的是记住， JSP 页面事实上是一个 servlet。你可以在 servlet 做的，也可以在 JSP 中做。

JSP标签

脚本元素仅是两种产生动态内容的方式之一。Scriptlet 是快捷、简单、强大的手段但要求开发者在 HTML 中混合 Java 代码。经验告诉我们，混合业务逻辑到 JSP 页面中将导致难以维护的应用和最小的可重用性。一个可选的方案是使用 JSP 标签（tags）。JSP 标签可以和 HTML 标记混合使用，就如同它们是原生 HTML 标记一样。一个 JSP 标签可以代表许多 Java 语句，但是所有的开发者都需要了解如何在页面中插入标记。源代码隐藏在 Java 类文件之中。

JSP 和 ASP

Microsoft 和 Sun 都提供它们各自品牌的服务器页面。Sun提供JavaServer Pages (JSP)而 Microsoft 提供 Active Server Pages (ASP)。JSP 和 ASP 都设计来能够使开发者能从后端系统产生动态页面。

虽然表面看起来相似，ASP 和JSP仍有一些不同之处：

- JSP 是平台独立性的，一次编写，随处运行；
- 开发者通过Java Community Process(JCP)指引方向；
- JSP 开发者可以通过定制标签扩展JSP标标签库；
- JavaBeans 和 Enterprise JavaBeans (EJB) 可以和JSP一起使用，增强可重用性和减小维护。
- JSP 可以存取其他一些Java 库，包括Java 数据库连接(JDBC)，Java Mail，Java Message Service(JMS)，以及JNDI。
- JSP 编译成二进制类文件，不需要在每次请求时进行解释；
- JSP 有广泛的支持，包括工具，容器和服务器；

为在其他页面中使用同一代码，只需要在该页面中重新插入相同的标签。如果标签代表的代码改变了，所有的标签都将使用更新的版本。而使用标签的 JSP 页面并不需要进行修订。

JSP 标记比 scriplet 提供了更好的可重用性，也更易被页面设计者使用，因为它们看起来很象 HTML 标记。

有大量的现成的 JSP 标签库（tags libraries）可用，他们完成很多有用的功能。其中就有新的 JSP 标准标签库(JSTL)。这是一个新的标准，提供丰富的可重用的 JSP 标签库。

关于 JSTL 的详细情况，我们高度推荐《*JSTL in Action*》，*作者* Shawn Bayern [Bayern]。Struts 可以很好的和 JSTL 以及其他公开标签库一起使用，甚至是你自己写的标签库。

关于 JSP 的详细内容，我们强烈推荐《*Web Development with JavaServer Pages*》，作者 Duane K. Fields，Mark A. Kolb, 和 Shawn Bayern[Fields]。

JSP 是 Struts 开发者工具箱的一部分。大多数 Struts 开发者使用 JSP 和定制标记来创建应用的动态内容。

JavaBean

JavaBean 是一种 Java 类，它遵从一定的设计模式，使它们易于和其他开发工具和组件一起使用。

定义

JAVABEAN 是一种 JAVA 语言写成的可重用组件。要编写 JAVABEAN，类必须是具体类和公共类，并且具有无参数的构造器（NON-ARGS CONSTRUCTOR）。JAVABEAN 通过提供符合一致

该你做的

整本书都在说你需要编写你自己的 JSP 标签，这里是这个过程的一个快速总揽：

1 编写一个类，通过实现 doStart()或者 doEnd()方法来实现 javax.servlet.jsp.tagext.TagSupport 或者 javax.servlet.jsp.tagext.BodyTagSupport 接口。这些方法获得一个 JspWriter 对象，你可以用它来输出你需要的 HTML 内容。

2 创建一个标签库描述文件(TLD)来将你的新建的类，映射到一个标签名称。

3 在你的 Web 应用描述符(web.xml)中定义你的<taglib> 元素。通过在 JSP 页面的顶部放置下面的语句：<%@taglib uri="/tags/app.tld prefix="app" %> 来告诉 JSP 页面你将使用你自己的标签库。

4 这个语句导入将在本页中使用的标签库，并分配给它一个前缀。关于更多细节，请参考 JSP 标签库技术页面。

性设计模式的公共访问方法将内部字段暴露称为属性。众所周知，属性名称也符合这种模式，其他 JAVA 类可以通过自省机制发现和操作这些 JAVABEAN 属性。

JavaBean 设计模式提供两种类型的方式来访问 bean 的内部状态：访问器（*accessor*） 用来读 JavaBean 的状态，修改器（*mutator* ）*用来改变* JavaBean 的状态。

Mutator 通常以小写的 *set* 前缀开始，后跟*属性名*。属性名的第一个字母必须大写。返回值通常是 void，因为 mutator 仅仅改变属性的值，而不返回它们。简单属性的 mutator 在其方法体中可能只有一个参数，该参数可以是各种类型。Mutator 也可根据其前缀称为设置器 *setters* 。*例如，*对 `Double` 类型的属性 `weight` 的 mutator 方法体可能是：

```
public void setWeight(Double weight)
```

相似的设计模式也用于访问器方法的创建。Accessor 通常以小写的 *get* 为前缀，后跟属性名。属性名的第一个字母必须大写。返回值必须匹配相应的修改器方法的参数。简单属性的 Accessor 在其方法体中不能接受参数。同样，访问器 accessor 也经常称为获取器 *getter*。

属性 weight 的访问器方法体可能是：

```
public Double getWeight()
```

如果访问器返回一个逻辑值，这种情况下有个变体模式。不使用小写的 get，逻辑属性的访问器可以使用小写的 *is* 前缀，后跟属性名。属性名的首字母必须大写。返回值肯定是逻辑值，不管是 boolean 还是 Boolean。逻辑访问器在其方法体中不能接受参数。

On属性的逻辑访问器的方法体可能是：

```
public boolean isOn()
```

在使用 JavaBean 时，规范的方法体签名扮演了极为重要的角色。其他组件可以使用 Java 的反射 API 通过查找前缀为 *set, is,* 或者 *get* 的方法来发现 JavaBean 的属性。如果一个组件在一个 JavaBean 中发现一个这样的方法，它就知道这个方法可以用来访问或者改变 JavaBean 的属性。

Sun 引入 JavaBean 是为了用于 GUI 组件，但它们已经用在 Java 开发的各个方面，包括 Web 应用。Sun 的工程师在开发 JSP 标签的扩展类时，也被设计来可以和 JavaBean 一起工作。一个页面的动态数据可以使用一个 JavaBean 来传递，并且 JSP 标记可以随后使用 bean 的属性来定制页面的输出。

Model 2

Servlet/JSP 规范的 0.92 版描述了在一个应用中使用 servlet 和 JSP 的架构。在其后的规范中，***Model 2*** 这个叫法消失了，但它已经在 Java web 开发人员中非常通用了。

根据 Model 2，servlet 处理数据存取和导航流， JSP 处理表现。Model 2 使 Java 工程师和 HTML 设计者分别工作于它们所擅长和负责的部分。Model 2 应用的一部分发生改变并不强求其他部分也跟着发生改变。HTML 开发人员可以改变程序的外观和感觉，并不需要改变后端 servlet 的工作方式。

Struts 框架是基于 Model 2 的架构。它提供一个控制器 servlet 来处理导航流和一些特殊类来帮助数据访问。随框架也提供一个丰富的标签库，以使 Struts 易于和 JSP 一起使用。

Struts开始于三万英尺

抓紧你的帽子！

既然我们已经讲了一些基本知识，我们现在可以进行一个 Struts 的飞速之旅了。在我们打开盒子吃到果子之前，我们先了解一个大概模样。

Struts 使用 Model 2 架构。Struts 的 *ActionServlet* 控制导航流。其他 Struts 类，比如 *Action*, 用来访问业务逻辑类。当 ActionServlet 从容器接收到一个请求，它使用 URI (或者路径“path”) 来决定那个 Action 将用来处理请求。一个 Action 可以校验输入，并且访问业务层以从数据库或其他数据服务中检索信息。

为校验输入或者使用输入来更新数据库， Action 需要知道什么值被提交上来。它并不是强制每个 Action 都要从请求中抓取这些值，而是由 ActionServlet 将输入绑定到 JavaBean 中。

输入 bean 是 Struts *ActionForm* 类的子类。ActionServlet 通过查找请求的路径可以决定使用哪个 ActionForm， Action 也是通过同样的方法选取的。ActionForm 扩展了 `org.apache.struts.action.ActionForm` 类。

每个请求都必须以 HTTP 响应进行应答。 通常，Struts Action 并不自行渲染响应信息，而是将请求转发到其他资源，比如 JSP 页面。Struts 提供一个 *ActionForward* 类，用来将一个页面的路径保存为逻辑名称。当完成业务逻辑后， Action 选择并向 Servlet 返回一个 ActionForward。Servlet 然后使用保存在 ActionForward 对象中的路径来调用页面完成响应。

Struts 将这些细节都绑定在一个 *ActionMapping* 对象中。每个 ActionMapping 相对于一个特定的路径。当某个路径被请求时，Servlet 就查询 ActionMapping 对象。ActionMapping 对象告诉 servlet，哪些个 Action，ActionForm，和 ActionForward 将要被本次请求使用。

所有这些细节，关于 Action， ActionForm， ActionForward， ActionMapping，以及其它一些东西，都在 *struts-config.xml* 文件中定义。 ActionServlet 在启动时读取这个配置文件，并创建一个配置对象数据库。在运行时，Struts 应用根据文件创建的配置对象，而不是文件本身。

图 1.1 显示了这些组件是如何一起工作的。

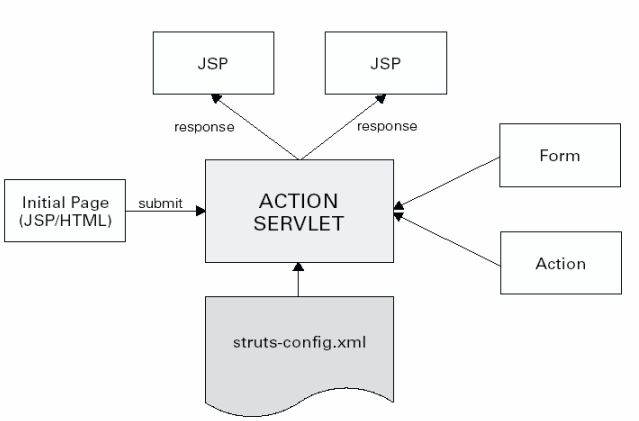


图 1-1 Struts 组件

不管你相信与否，你已经知道了足够的东西，可以构造一个简单的 Struts 应用了。它并不需要做很多工作，但它显示了 Struts 实际是如何工作的。

建立简单的Struts应用

开发者进行开发，我们大多数则从例子中学习最好的方法。虽然我们花了几乎一页不到的篇幅来讲 Struts 是如何工作的，我们也需要建立一些什么，以便你可以看到如何完成这个工作。完成了这章，我们将建立一个非常简单但完整的 web 应用。这个程序将用来注册用户和用户密码。一旦你完成它，你将接触到部署你的 Web 应用所需的所有 Struts 组件。

开始开发

当我们渴望要创建点什么时，接下来再设置环境并遇到障碍的时候就可能让所有人都觉得受挫。根据这章所述，你所需要的是一个 Java Development Kit (JDK)，一个 web 容器（比如 Tomcat），以及一个简单的文本编辑器。如果你还没有一个 Java 开发环境和 web 容器，下面是你该做的：

下载并安装 JDK 1.4.

下在并安装 Tomcat 4 .

校验 Tomcat 是否工作正常。

1.1.4. 落到实处

我们的第一个 Struts 程序将是一个用户注册程序。用户将看到一个注册屏幕，包含 3 个字段：用户名，密码和密码确认。成功的注册要求两次密码相符。如果注册成功，控制将转向一个页面，显示注册成功 *successful!*.。如果两次输入密码不同，控制流将转向一个显示失败的页面。

这个简单的练习将展示以下内容：

- 创建 HTML 表单；
- 从 HTML 表单获取输入；
- 处理输入（业务逻辑）；
- 根据动态输入改变控制流；

为完成这个程序，你需要建立：

- 一个 ActionForm

- 一个 Action
- struts-config.xml 文件
- 三个页面

就这些！

创建ActionForm

ActionForm 是一个 JavaBean ，它扩展了 `org.apache.struts.ActionForm` 类。这个对象捕获通过请求传送的输入。当浏览器提交一个表单，它在请求中为每个表单中的字段创建一个参数。ActionForm 针对每个 HTML 表单中的字段具有一个对应的属性。ActionServlet 匹配请求中的参数和 ActionForm 中的属性。当匹配好后，ActionServlet 为属性调用 setter 方法，并将请求中的值传入 ActionForm。在我们的练习中，表单中的 username 字段需要一个 `setUsername(String)` 方法。password 字段需要 `setPassword1(String)` 和 `setPassword2(String)` 方法。这些方法负责组装隐藏在 RegisterForm JavaBean 中的实例变量。RegisterForm 的源代码显示在清单 1.1 中。

```
package app;

import org.apache.struts.action.ActionForm;
public class RegisterForm extends ActionForm {
    private String password2;
    private String password1;
    private String username;
    public String getUsername() {
        return username;
    }
    public void setUsername(String i) {
        username = i;
    }
    public String getPassword1() {
        return password1;
    }
    public void setPassword1(String i) {
        password1 = i;
    }
    public String getPassword2() {
        return password2;
    }
    public void setPassword2(String i) {
        password2 = i;
    }

}
```

代码清单 1.1 RegistrationForm

创建一个文件，取名为 RegisterForm.java，内容示于代码清单 1.1。存储在 *<Base Directory>*/webapps/register/WEB-INF/classes/app 下。默认情况下，*<Base Directory>* 可能是 C:/PROGRAM FILES/APACHE TOMCAT 4.0。对于其他容器，使用其 classes 目录的路径来部署我们的 Register 程序。

创建 RegisterAction

Action 是一个 Java 类，扩展了 `org.apache.struts.Action`。ActionServlet 负责组装 ActionForm ，然后将其传递给 Action。Action 通常负责输入校验，访问业务信息，以及决定向 Servlet 返回哪个 ActionForward 。

现在，创建一个文件，命名为 RegisterAction.java，其内容为代码清单 1.2 的内容：

```
package app;
import org.apache.struts.action.*;
import javax.servlet.http.*;
import java.io.*;
public class RegisterAction extends Action {
    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest req,
        HttpServletResponse res) {
        // ① 将form 转型为RegisterForm
        RegisterForm rf = (RegisterForm) form;
        String username = rf.getUsername();
        String password1 = rf.getPassword1();
        String password2 = rf.getPassword2();
        ActionForward forward = new ActionForward();
        // ② 应用业务逻辑

                    if (password1.equals(password2)) {
                        try {
                            // ③ 如成功，则返回针对success 的ActionForward

    UserDirectory.getInstance().setUser(username,password1);
                            forward = mapping.findForward("success");
                        } catch (UserDirectoryException e) {
                            forward = mapping.findForward("failure");
                        }
                    } else{
                        forward = mapping.findForward("failure");
                    }

                    // ④ 返回针对failure的ActionForward

        return (forward);
    }
}
```

代码清单 1.2 RegisterAction.Java

将文件存放在<*Base Directory*>/webapps/register/WEB-INF/classes/app 目录下。

虽然很简单，但是我们的 RegisterAction 却做了 Action 的典型事情。在①处，输入 ActionForm 被转换为 RegisterForm。我们就可以获取 `username`, `password1`, 和 `password2` 的内容。

如果两次密码匹配②，我们将用户添加到 *UserDirectory* 中③，并返回与 *success* 对应的 ActionForward 。UserDirectory 是一个 helper 类，它记录 usernames 和 passwords 到一个标准的属性文件之中。否则，返回与 *failure 对应的* ActionForward 。

当我们在下一步创建 struts-config 文件时，我们将标识代表 success 和 failure 的 ActionForward 对象。

注 **STRUTS 1.1 提供另外一个进入方法，名为 EXECUTE。这个方法提供更好的异常处理，但和 STRUTS 1.0 的 PERFORM 方法不一样。 在这里我们将使用 PERFORM 方法，以使我们的例子可以运行在两个版本之下。**

创建Struts 配置文件 (struts-config.xml)

struts-config.xml 文件包含了 ActionServlet 需要用来处理对应用请求的详细信息。为了练习，我们创建一个空壳的 struts-config.xml 文件。你需要做的是填入一些细节。

文件存储在<*BaseDirectory*>/webapps/register/WEB-INF/目录下，需要改变的是：

首先，添加/register 到<action>元素的 path 属性。ActionServlet 使用 Web 容器转发给它的 URI 来选择正确的 Action 类。URI 和 ActionMapping 的 path 属性匹配。这里，请求给出的路径必须在去除前缀和后缀后和/register 匹配。前缀或后缀通常是*/do/ 或者 .do*。我们的练习中，将后缀设置为.do。当 URI 具有一个.do 扩展名，容器就知道将请求转发给 ActionServlet。Struts 会自动去除扩展名，所以我们在配置时不必加上它们。

下一步添加 registerForm 到<action> 元素的 name 属性。<action> 元素使用 name 属性来识别哪个 ActionForm 将被创建，并将提交的表单组装给他。

然后，添加 app.RegisterAction 到<action> 元素的 type 属性。ActionServlet 使用这个属性来识别将用来处理请求的 Action 类。

接下来，在<forward> 元素下，添加 success 到 name 属性，并且添加/success.html 到 path 属性。最后，再在另一个<forward>下添加 failure 到 name 属性，/failure.html 到 path 属性。

这些元素将创建 ActionForward 对象，我们将用它来选择程序的控制流。<forward> 元素定义了在 RegisterAction 中使用的逻辑名称之间的关联。

Struts-config.xml 源代码见代码 1.3。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.1//EN"

"http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<struts-config>

    <!-- 数据源 -->
    <data-sources>
    </data-sources>
```

```
    <!-- 表单 Bean -->
    <form-beans>
        <form-bean name="RegisterForm" type="app.RegisterForm">
        </form-bean>
    </form-beans>

    <!-- 全局异常 -->
    <global-exceptions>
    </global-exceptions>

    <!-- 全局转发 -->
    <global-forwards>
    </global-forwards>

    <!-- 操作映射 -->
    <action-mappings>
        <action path="/register" type="app.RegisterAction"
name="RegisterForm" scope="request" input="/register.jsp">
            <forward name="success" path="/success.jsp">
            </forward>
            <forward name="failure" path="/failure.jsp">
            </forward>
        </action>
    </action-mappings>

    <!-- 消息资源 -->
    <message-resources parameter="test.resources.ApplicationResources"/>

</struts-config>
```

代码清单 1.3 Struts-Config.XML

Struts 框架将 struts-config.xml 文件视为部署描述符使用。它使我们可以创建和改变 ActionMapping 和路径的关联而不用重新编译 java 类。我们也可以改变页面之间的连接，而不改变 JSP 模板。

创建页面

最后的步骤是创建 success.html, failure.html, 以及 register.jsp 页面。

3 个文件的源代码如下,见代码清单 1. 4,1. 5,1.6。存放在<*Base Directory*>/webapps/register 目录下。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ page
language="java"
contentType="text/html; charset=GB18030"
```

```
pageEncoding="GB18030"
%>
<TITLE>注册成功</TITLE>
</HEAD>
<BODY>
<P>注册成功！</P>
<P><BR>
<BR>
<A href="register.jsp">再试一次</A>？
</P>
</BODY>
</HTML>
```

代码清单 1.4 Success HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ page
language="java"
contentType="text/html; charset=GB18030"
pageEncoding="GB18030"
%>
<TITLE>注册失败</TITLE>
</HEAD>
<BODY>
<P>注册失败！</P>
<P><BR>
<BR>
<A href="register.jsp">再试一次</A>？
</P>
</BODY>
</HTML>
```

代码清单 1.5 Failure.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ page
language="java"
```

```
contentType="text/html; charset=GB18030"
pageEncoding="GB18030"
%>
<HTML>
  <HEAD>
    <TITLE>register</TITLE>
  </HEAD>
  <BODY>
    <P><html:form action="/register.do">


    用户名：<html:text property="username"></html:text><BR>
    密码1： <html:text property="password1"></html:text><BR>
    密码2： <html:text property="password2"></html:text><BR>

    <html:submit value="Register"></html:submit>
    </html:form></P>
  </BODY>
</HTML>
```

代码清单 1.6 Register.jsp

这时，所有构建一个简单 Struts 应用的工作都做完了。

现在，试一下运行如何。

如果 Tomcat 没有运行，启动它。

在浏览器中输入以下地址：

```
http://localhost:8080/register/register.html
```

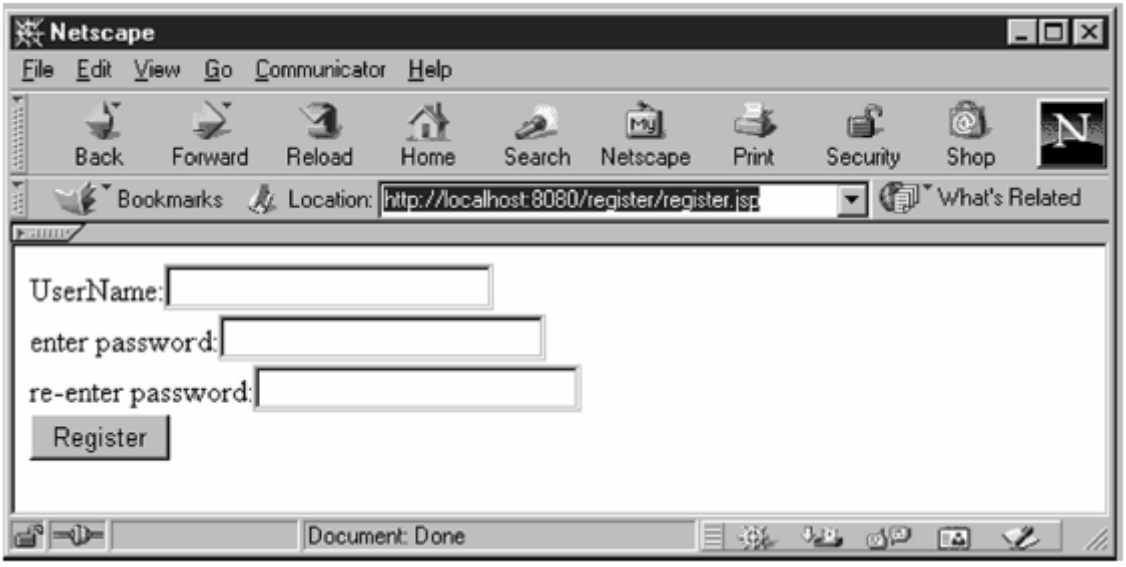


图 1-2 Registration 程序主界面

1.1.5. 再看看

让我们再回头看看，我们做了什么，它如何工作的，以及我们还需要做些什么。

做了什么

建立 Register 应用我们实际上完成了以下内容：

- RegisterForm ActionForm
- RegisterAction Action
- 3个页面

它如何工作

当你将浏览器指向地址 http://localhost:8080/register/register.html，Tomcat 按通常情况渲染这个页面。输入 username 和 password，点击 Register 提交页面。浏览器在请求中 post 表单的内容。容器检查请求需要送到哪一个注册的路径去处理。然后请求被转发到 ActionServlet ，并由 RegisterAction 来处理。在返回成功或失败之前，RegisterAction 校验输入的有效性。最后 servlet 将控制根据返回的 ActionForward 转发到响应页面。图 1.3 是程序结构。

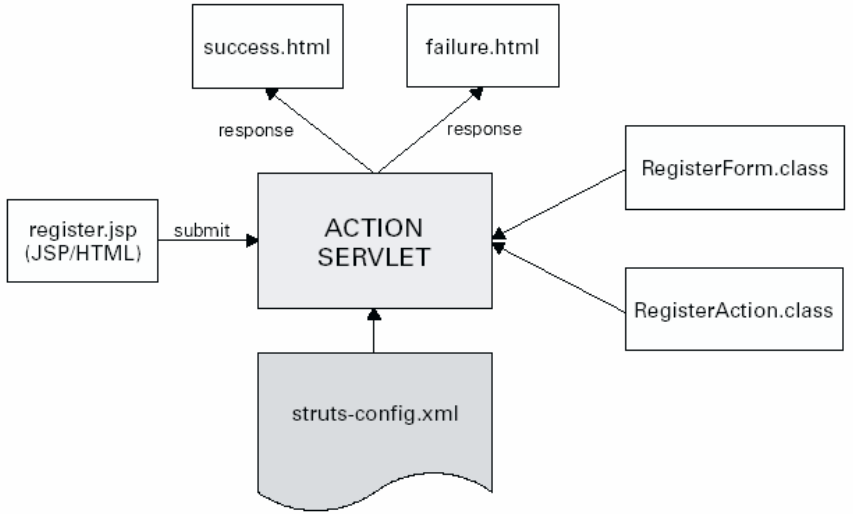


图 1-3 Registration 程序结构

再看看代码 1.6 中的 Register.jsp，可以看到表单是提交给 URI /register。但是如果你观察正提交的页面，会发现是提交给 register.do。Struts 表单标标签自动加上.do 前缀。当我们设置程序骨架时，我们要求所有匹配*.do 的请求都传递给 ActionServlet。

当接收到一个请求，ActionServlet 做的第一件事情就是查找 ActionMapping 来匹配请求的路径。ActionMapping 是 Struts 根据 struts-config.xml 文件创建的 JavaBean.。我们给出了 XML 的具体文件，但运行时，Struts 引用的是对象，而不是 XML 文档。

从代码 1.3 中可以看到，我们使用这个元素创建了一个到 path /register 的映射：

```
<action path="/register"
        input="register.html"
        name="RegisterForm"
```

```
            type="app.RegisterAction">
```

然后，ActionServlet 检查是否有 name 属性表明的 FormBean 和这个映射相关：

```
        <action path="/Register"
            input="register.html"
            name="RegisterForm"
            type="app.RegisterAction">
```

这里 /register 映射通过 registerForm 的名称标识了一个 form bean。ActionServlet 使用这个 name 属性来查找相应的 ActionFormBean 对象。由 Form Bean 标识的类型（type）用来创建 ActionForm 对象：

```
    <!-- 表单 Bean -->
    <form-beans>
        <form-bean name="RegisterForm"
            type="app.RegisterForm" />
    </form-beans>

    <!-- 全局异常 -->
    <global-exceptions></global-exceptions>

    <!-- 全局转发 -->
    <global-forwards></global-forwards>

    <!-- 操作映射 -->
    <action-mappings>
        <action path="/register"
            input="register.html"
            name="RegisterForm"
            type="app.RegisterAction">
        </action>
    </action-mappings>
```

这里，servlet 将使用 RegisterForm 类:

```
    <form-beans>
        <form-bean name="RegisterForm"
            type="app.RegisterForm" />
    </form-beans>
```

一旦 RegisterForm 被实例化， ActionServlet 就试图为请求中的输入字段调用 RegisterForm 的 setter 方法。在例子中，它们是 setUsername, setPassword1, 和 setPassword2。如果某个 setter 方法不存在，该参数将会被忽略。

ActionMapping 对象的 type 属性是 ActionServlet 用来实例化 ActionAction 的类名。这里，将使用你创建的 RegisterAction 对象。RegisterAction 对象的 perform 方法被调用，并传递一个对在前一步中所创建和组装的 RegisterForm 的引用：

```
    <!-- 操作映射 -->
```

```
    <action-mappings>
    <action path="/register" input="register.html" name="RegisterForm"
  type="app.RegisterAction">
            <forward name="success" path="/success.html" />
            <forward name="failure" path="/failure.html" />
        </action>
    </action-mappings>
```

依赖于 perform 方法的执行结果，将返回两个 ActionForward 之一。findForward() 方法使用一个 String 参数来查找 与 name 属性相匹配的 forward 对象。而 path 属性则由 ActionServlet 用来决定用哪个页面来完成响应：

```
<forward name="success" path="/success.html" />
<forward name="failure" path="/failure.html" />
```

还有什么没做

为了尽快入门，我们省略了一些内容。我们没有编译源代码，而依赖于这个起始程序一起绑定的已经编译好了的类文件。我们想给你一个机会来开始开发 Struts 程序，而不被诸如 Ant 构建文件之类的扰乱心神。

在第 3 章，我们开发另外一个程序来展示框架的其他特征。在那里，我们也对 Ant 和一个编辑器 jEdit 的做一些入门介绍。我们还要开始深入介绍 Struts 组件。

本书的第 2 部分非常详细的讨论了框架组件。在第 4 部分，我们将它们和起来开发一个实际的应用 Artimus。

1.2. 小结

在本章，我们介绍了 Struts 应用框架。并介绍了一些基本知识，关于 HTTP, CGI, Java servlet, JSP, 以及 JavaBean。我们也说明了 Model 2 应用架构，以及它如何用来将 servlets 和 JSP 在结合在同一个应用之中。

本章末尾，我们快速建立第一个简单的 Struts 应用。现在你已经有关于 Strtus Web 应用程序模样的初步印象，下一章我们将更深入的讨论 Strtuts 框架的理论和具体实践。

2.深入 Struts 架构

本章内容

- 介绍 MVC和 Model 2应用框架
- 理解 Struts 原理
- 使用 Struts 控制流
- 讨论Struts 的优缺点

A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools.

—Douglas Adams, *Mostly Harmless*

2.1. 随便谈谈

本章深入探讨 Struts 框架，以及它能给你的应用开发所带来的诸多好处。我们相信，一旦你也能“随便谈谈”web 架构和设计，你就可以很好的在你的应用中使用 Struts。

为了能对 Struts 架构有个充分的全面印象，我们将总体介绍 Struts 的控制流和它处理请求-响应事件循环的方式。只有彻底理解这个处理原理才能最好的在应用中使用这个框架。

选择一个 web 应用框架不应该是个漫不经心的决定。很多人都可以使用这本书，特别是用这章的内容，来评价 Struts 是否适合它们的项目。因此，我们在这章的最后部分将有一个关于 Struts 优缺点的客观评价，并阐明其总体性能。

Struts 设计来针对专业开发人员。为做出正确的决策，专业人员应该知晓工具的能力和限制。

2.2. 为什么我们需要Struts

今天的 web 应用基本上都是代表共同理念的关键组件。通常，开发团队需要在有限的时间里创建应用，然后它们不得不正确的构建，并能持续构建它。

Java web 开发人员已经有一些工具可用来建立表现层，比如 JavaServer Pages 和 Velocity 模板。也有一些机制来处理数据库—如 JDBC 和 Enterprise JavaBean (EJB)。但我们用什么来将它们集合在一起？ 我们已经有了型材和砖墙…还缺什么呢？

2.2.1. 退一进三

在上世纪 80 年代，当 GUI 被发明时，软件架构认为，应用具有 3 个主要部件：

管理数据的部件，创建屏幕和报表的部件，以及处理用户交互和子系统交互的部件[Ooram]。

在 90 年代早期，ObjectWorks/Smalltalk 编程环境将这个三角结构引入为一个开发框架。按 Smalltalk 80 的说法，数据系统称为模型 *Model*，表现系统称为视图 *View*，而交互系统称为控制器 *Controller*。许多现代开发环境，包括 Java 的 Swing，都使用 *Model/View/Controller (MVC) 架构*作为它们的基础架构。

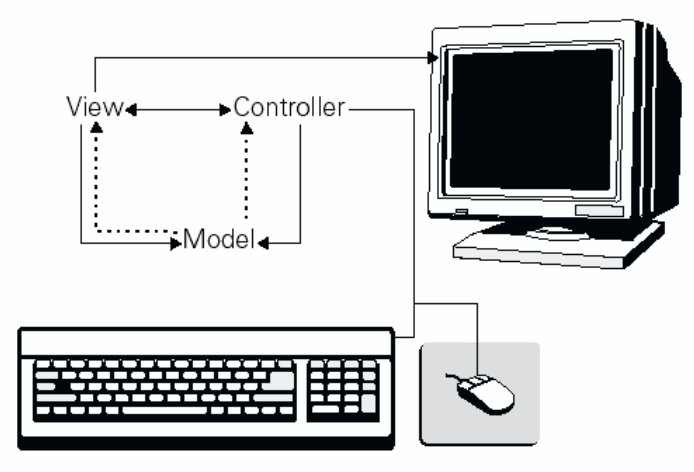


图 2-1 MVC 架构

Java web 开发者已经有很多有用的工具，比如 JDBC 和 JSP，作为 Model 和 View 的手段，但作为控制器的工具和组件在哪？

2.2.2. 进入 Struts

Struts 的核心是一个 MVC 风格的控制器。Struts 控制器搭起了 Model 和 View 之间的桥梁。框架也包括开发人员想用来开发可伸缩的、先进的应用系统的其他组件。Struts 是一个“隐藏支柱”的集合，帮助开发人员将分散的材料，如数据库和页面，结合成一个整体的应用系统。

2.2.3. Struts控制器组件

Struts 控制器组件是一个可编程的组件集，允许开发人员定义它们的应用如何准确地和用户进行交互。这些组件在逻辑名称后面隐藏了令人讨厌的、繁琐的实现细节。开发人员可以一次性编写这些实现细节，然后转头考虑它们的应用应该做什么，而不是考虑应用应该如何做。

用户通过超链接和 HTML form 与 Web 应用程序进行交互。超链接引导页面显示数据和其他内容，如文本和图像。表单通常通过一些定制动作向应用提交数据。

图 2-2 主要 Struts 组件

如图 5 中所示， Struts 提供了开发人员可用来定义超链接，表单，和定制动作这些交互的相关组件。我们已经使用这些组件在第 1 章创建了一个入门程序。第 3 章，我们还要用它们来创建另一个程序。然后，在第 4 章，我们将讨论这些组件的详细配置。随后的章节，将详细讨论如何将每个组件以及如何用在你的程序之中。在第 4 章，我们将展示如何在运行的程序上下文中使用这些组件。但是，因为这一章是架构性的总体介绍，所以我们继续介绍 Struts 的主要部件。

注 Struts 组件是通过一个XML文件进行配置的。实践中，配置项是Struts框架的有机组成部分。为了将它们糅合在一起，我们在讲述它们的时候，会展示每个组件的XML配置项。

超链接

对应用开发人员来说， 超链接是指向应用中某些资源的路径。这些资源可能是 web 页面，或者是定制动作。超链接中也可以包含特殊的参数。在 Struts 中，开发人员可以将超链接定义为一个 *ActionForward*。

这些对象都有个逻辑名称和一个 `path` 属性。这使得开发人员可以设置 path，然后通过名称来引用 ActionForward。ActionForward 通常在一个 XML 文件中定义，这个配置文件在 Struts 启动时载入。Struts 使用 XML 定义来创建 Struts 配置，包括一个 ActionForward 的列表。可用来创建到欢迎页面链接的 ActionForward 对象的 XML 元素看起来可能像：

```
<forward
      name="welcome"
      path="/pages/index.jsp"/>
```

这个元素事实上是创建了一个 ActionForward JavaBean ，其 name 属性设置为 `welcome`，`path` 属性设置为`/pages/index.jsp`。

JSP 页面和其它组件就可以引用这里定义的 `welcome` 转发。Struts 框架将查找 `welcome` ActionForward bean 并获取其 `path` 属性以完成这个超链接。这样开发人员可以改变链接的目标而不用改变所有引用该链接的相关组件。在很多 `Web` 应用中，象这样的细节被硬编码到 JSP 或 Java code 中，使维护变得困难并且容易发生错误。在 Struts 应用中，这些细节可以通过应用配置来改变，而不用触及到具体的页面和 Java 类。

HTML表单

Web 协议，即 HTTP 和 HTML，提供了一个从表单中提交数据的机制，但却把数据的接收作为一个难题留给了开发人员。为此，Struts 框架提供了 *ActionForm* 类。ActionForm 设计来就是处理来自 HTML 表单的输入：校验输入，重新显示表单以供用户进行修订（如果需要），以及伴随着相应的提示和错误信息。ActionForm 其实是具有一些用来来管理校验和修订循环的标准方法的 JavaBean。Struts 自动匹配 JavaBean 属性和 HTML 表单控件的属性。开发者只需定义 ActionForm 类，余下的就交给 Struts 。

例如，这个类将自动用 HTML 表单中同名的属性来组装 username 域：

```
public final class LogonForm extends ActionForm
{
  private String username = null;
  public String getUsername() {
  return (this.username);
  }
  public void setUsername(String username) {
  this.username = username;
```

```
  }
}
```

其它属性也会根据表单中的每个输入域被自动加入。这使得其它组件可以从标准的 JavaBean 取得它们想要的属性。所以，完全不需要对 HTTP 请求进行详细解析。

ActionForm 是按通常的 JavaBean 类创建的。Struts 配置通过一系列描述符引用 ActionForm 类： `<form-beans>` 和 `<form-bean>` 元素。`<form-bean>` 元素是框架用来识别和实例化 ActionForm 对象的描述符：

```
<form-bean
       name="articleForm"
       type="org.apache.artimus.struts.Form"/>
```

Struts 配置中要列出它使用的 ActionForm bean 的清单，并给每个 bean 一个在应用中被引用时的逻辑名（name 属性）。

1.0 和 1.1

Struts 1.1中，ActionForm 可以使用Map (java.util.Map)来存储属性名，而不是一个个单独定义它们。一种新的JavaBean，DynaBean，也可以在Struts 1.1和后来的版本中使用。

你可以使用XML元素来配置DynaActionForm的属性。这使你可以使用Struts配置文件来定义ActionForm。

定制动作

HTML 表单使用 `action` 参数来告诉浏览器将数据送到何处。Struts 框架提供相应的 *Action* 类来接收数据。框架会自动创建、组装、校验和最后处理 Action 对象所对应的 ActionForm。这样，Action 就可以直接从 ActionForm bean 取得它需要的数据。比如下例：

```
public final class LogonAction extends Action {

   public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
              throws IOException, ServletException {
        MyForm myForm = (MyForm) form;
        // ...
        return mapping.findForward("continue");
      }
}
```

Action 根据返回到控制器的 ActionForward 对象作出控制流的决定。这允许 Action 选择一个逻辑名称，比如 *continue* 或者 *cancel*，而不是具体的系统路径。

为保证可扩展性，控制器也传递当前的请求和响应对象。实际上， Action 可以做所有 Java

Servlet 可以做的事情。

1.0 vs 1.1	在Struts 1.1中，新的 execute 方法应该是首选，而不是我们例子中的perform方法。perform 已经不被推荐，但为保持向后兼容，予以支持。execute 方法体允许更好的异常处理。新的ExceptionHandler 在第9章讲述。 关于Action 对象的详细信息，见第8章。

除了 ActionForward， ActionForm，和 Action 对象，Struts 控制器层还提供了几个特殊的组件，包括 *ActionMapping* 和 *ActionServlet*。 Struts 也提供在控制器层的应用程序的本地化。

ActionMapping

在一个 web 应用中，每个资源都必须通过 URI 来进行引用。 资源束括 HTML 页面，JSP 页面，和定制动作。为了给定制动作一个 URI， 或者说路径，Struts 框架提供了一个 *ActionMapping 对象*。 象 ActionForward 和 ActionForm 一样， ActionMapping 通常也在 XML 配置文件中定义：

```
<action-mappings>
   <action path="/logonSubmit"
          type="app.LogonAction"
          name="logonForm"
          scope="request"
          validate="true"
          input="/pages/logon.jsp" />
</action-mappings>
```

这也允许将同一个 Action 对象定义为不同的 ActionMapping。例如，其中一个映射要求校验而另一个映射不要求校验。

ActionServlet

Struts *ActionServlet* 完全在幕后工作，它将其他组件绑定在一起。虽然它也可以子类化，但大多数 Struts 1.0 的开发人员将 ActionServlet 处理为一个黑盒：他们只是配置它，然后让它自己工作。

在 Struts 1.1 中，ActionServlet 是比较易于扩展的。第 9 章将讨论 Struts 1.1 ActionServlet 新的扩展点和配置选项。

本地化

Web 应用也通过各种提示和信息与用户进行交互。Struts 组件均有内建的本地化特征，以便 Struts 应用可以为国际化用户使用。我们在此书中贯穿使用本地化特征。本地化的详细讨论见第 13 章。

2.2.4. 用Struts开发Web应用

要使用 Struts 开发 web 应用，开发人员将需要的超链接定义为 ActionForward，HTML 表单定义为 ActionForm，定制的服务器端动作定义为 Action 类。

需要访问 JDBC 和 EJB 的开发人员也可通过 Action 对象进行他们的工作。这样，表现层不需要和 Model 层打交道。Struts Action 对象将收集 View 需要的数据，然后将它们转发到表现页面。Struts 提供 JSP 标记库，它们将和 JSP 页面一起使用，简化 HTML 表单和访问 Action 要转发的其它数据。其它表现机制，比如 Velocity templates, 也可用来访问 Struts 框架，以创建动态的 web 页面。这种处理流程入下图：

图 2-3 数据回传给视图

关于和 Struts 中如何使用各种数据系统，见第 14 章。第 10 章和第 11 章学习如何创建 Struts 的表现页面。

在深入 Struts 架构前，让我们看看一个 Web 应用框架必须说明的问题。

2.3. 为什么需要框架

第 1 章，我们介绍了应用框架，简短讨论了为什么框架很重要。但为了真正理解一个解决方案，我们需要了解问题所在。为 web 开发应用虽然是值得的，但也要迎接各种挑战。让我们快速看看是什么使 web 开发富有挑战。

2.3.1. Web—永无休止的修补

Web 开发者受到两种 web 缺陷的影响。首先，我们希望使用浏览器作为客户端。其次，我们必须使用 web 协议进行通讯。

Web 浏览器通过 HTTP 协议通信，并用 HTML 显示页面。Web 浏览器发送 HTTP 请求，并渲染和显示它收到的响应。在处理很少改变的预先编好的页面时，这是个很好的平台。但我们大多都是编写动态程序，页面针对不同的用户是不同的。虽然有一些现成的动态特征的手段，web 仍然受到 HTTP/HTML 的制约。

如下表所示： web 协议和客户端的限制，确定了如何编写 Web 程序。

表格 2.1 HTTP/HTML 的限制

限制	导致的困难
协议	缺省情况下，HTTP接收来自于网络上各种客户端的连接。但各种服务器间的行为是不一样的；
	首先，HTTP用简单的文本域来传输数据。传输二进制文件需要复杂的协议扩展
	HTTP协议是无状态的，需要额外的努力来跟踪程序的用户
	HTTP信赖并期望客户能提供正确的信息
客户端	浏览器是个独立的程序，处于应用的控制之外

	所有的浏览器都是不一样的，都只支持官方标准的一个子集
	从浏览器的输入可能是不正确或者不完整的。甚至可能是敌意的，和对程序有害的信息
	HTML不能够建立一些在桌面环境中有的接口元素
	用缺省数据建立HTML控件对应用来说是个考验

很不幸，这种状况现在并没有些许改变。Web 开发人员在想战胜挑战时必须首先看到这些缺陷。因为这对编写强壮的 Web 应用有太多障碍，使用框架便显得至关重要，免得你的应用陷入无休止的工作和改进之中。

在开发 Web 应用时我们面临的挑战是很巨大的。但同时也是值得的。HTTP 协议和 HTML 客户端使所有的人都可以访问你的应用。没有其他哪个平台能声称这样。

2.3.2. Servlet 解决方案

如第一章所述，Java Servlet 平台[Sun, JST] 扮演了一个基本框架，为 Java web 应用提供了大量的能力。Servlet 提供了一个处理 HTTP 请求和确保响应的基本接口。它在 HTTP 之上构建了一个“会话” 上下文，帮助跟踪应用程序的用户。当然它也提供其他的上下文，帮助应用传输数据到浏览器或者应用中的其他 servlet。Java web 应用也具有对基本安全特性的统一访问，而这些安全特性在不同的服务器上的管理是不一样的。

为了将这些内容集成在一起，Servlet 规范引入了一个容器来管理它们。容器也可以提供其他服务，比如作为一个 JSP 的处理器。Servlet 容器可以包含它自己的 web server，也可以简单的作为一个现有服务器的附属组件。

对数据库访问， Java 应用在其建议中有另外一个通用的框架： JDBC。开发者可以编写标准的 SQL 接口，而将烦人的细节留给适配器来处理。这使得可以很容易的改变数据库厂商，而不用重写源代码。

为了取得远程服务器的高性能数据库访问， web 开发人员可以使用 EJB 平台。大多数 Java 应用框架，包括 Struts, 都可以和 EJB 一起使用。

总之，这使得基于 Servlet 的 web 应用非常轻便，并相对易于编写和维护。Servlet 和 JSP 在编写应用中扮演了完全不同的角色。 象 Strtus 这样的 Java web 应用框架就构架于 Servlet 之上，给开发者提供一个无缝的集成环境。

2.3.3. Servlet 框架

大多数，不是全部， Java web 框架使用 Sun Servlet 平台为基础。这些框架都绑定一些预制的 servlet，你可以插入到你的应用中去。框架本身也包括一个类层次结构，这些类（或接口）你可以在你的应用中实现或者扩展。通常， 应用框架的目标是帮助你将你需要的数据从浏览器发出，进入到编程结构之中，这样你的应用就可以使用它—或者从编程结构中发出，进入到浏览器之中，这样你就可以看到。

一些框架，如 Turbine [ASF, Turbine]，也提供 helper 类来使用 JDBC 数据库。其他框架，如 Struts，则是模型中立的。它们既不阻碍数据库访问，也没有提供帮助。而某些框架, 如 dbForms [dbForms]，则专注于数据库访问，而将其它任务留给开发人员或者其他框架。

通用框架策略

如下图所示， Java web应用框架使用一些通用技术来帮助产品易于设计、编写和维护，这些技术包括：

外部配置文件

提供开发人员不想嵌入源代码中的实现细节。

中心控制器

提供一种方式，将 HTTP 请求排入一个易于管理的队列。这种设计有时叫前端控制器模式（*Front Controller* [Go3]）

外部表现系统

让不同的人同时工作在同一应用的不同部分。如， Java 工程师可以工作在和中心控制器相关的类，而页面设计者则专注于 JSP。除了 JSP，其他表现系统，如 Velocity Templates 或者 XSLT, 都可以和 Struts 一起使用。

框架通常有各种组件，但基本上都共享这个特性。这些公共策略早已在一些书象 *Design Patterns* [Go4] 和 *Core J2EE Patterns* [Go3]给出的范例中根深蒂固了。许多开发者在讨论并使用这些模式，但也许还没有第一次在 Web 环境中实现它们。使用设计模式，如 MVC，使你可以容易的通过做正确的事情来构建你的应用。在桌面环境中使用设计模式的优点已经众所周知了，但在 Web 环境部署这些模式却对大多数开发者来说还是不确定的。

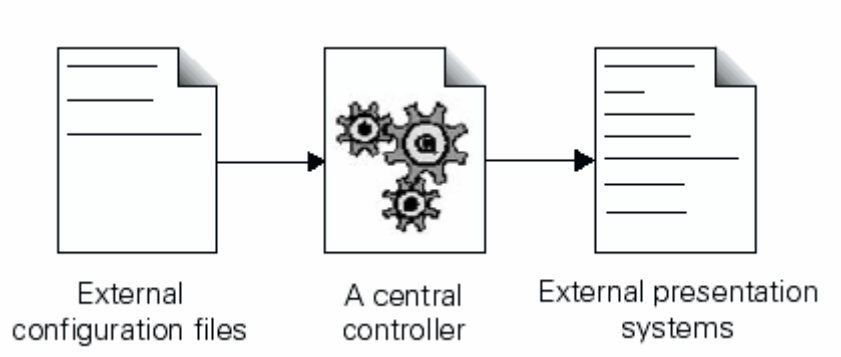


图 2-4 通常使用配置文件，控制器和表现系统的框架

2.3.4. 黑盒-白盒统一体

框架有时分为两极标有白盒和黑盒的统一体[Fayad]。白盒框架严重依赖于面向对象的语言的特征，如继承和动态绑定。黑盒框架则注重定义可插入组件的接口，然后基于这些接口提供基本的起始组件。接口和基本组件通常提供热点（*hotspot*）方法，这个方法可以直接使用或者重写后提供特别的行为。

定义 热点有时也称为是灵活点或者扩展点，它其实是框架中的一些位置，在这里可以加入一些代码来定制框架。热点（热点系统）描述了每个应用中可以被框架所支持的不同特征。本质上，它们代表框架所解决的问题。许多面向对象框架有一个核心系统和一些热点子系统组成[Braga, et al]

像许多正在使用的框架， Struts 使用混合的黑盒和白盒技术。但总体上，框架将偏向统一体的黑盒一端。

黑盒框架通常依赖于设计模式。Struts 也不异常。事实上，设计模式通常被用来作为框架的总体描述 [Johnson]。为保持这个趋势，我们先介绍一下设计模式，以及它们是如何用在 Struts

框架之中。

2.4. Struts, Model 2,以及 MVC

Struts 关于自己要说的第一句话就是：框架

*…**鼓励应用架构基于 Model 2 方法，即经典的 MVC 设计模式的变体***

这句话打消了一些 web 开发者的疑虑，对那些还没深入 Model 2 或 MVC 的人来说却更加疑惑了。 事实上，没有深入了解 MVC 和 Sun Model 2，要理解很多 Struts 的文章都很困难。

2.4.1. MVC的演化

如 2.1 节所说， Model/View/Controller 原本是建立 Smalltalk 应用的框架。框架支持代表应用状态、屏幕表现和控制流的 3 个类，分别叫做 Model, View, 和 Controller。

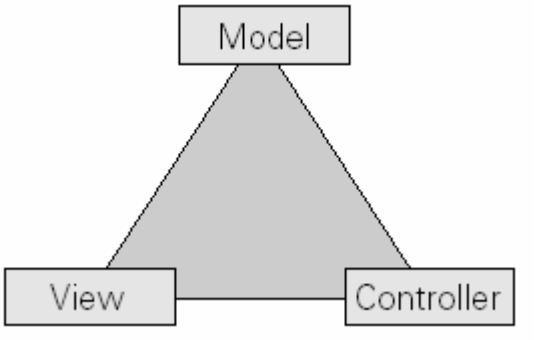


图 2-5 MVC 三角

Smalltalk MVC 框架在颇为流行的书 Design Patterns: Elements of Reusable Object-Oriented Software[Go4]是作为案例来研究的。Design Patterns 这本书有四个作者，被称为“四人帮 GoF”。

Design Patterns 中的 MVC 例子称为**通知/订阅者**(notify/subscribe)协议和观察者(Observer)模式的使用。例子的基础是，对同一数据，系统可能需要不同的显示视图，比如条形图、饼图、数据表格等等。这是一个划分应用的精彩理由，经常被重复引用。

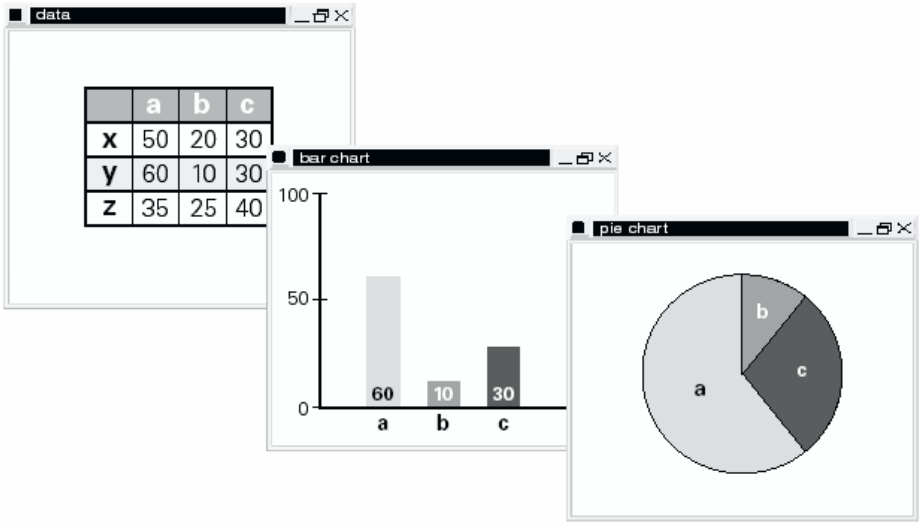


图 2-6 模型数据可以被用于不同的视图中

在图 2-6 所示的例子中，每种视图可能在同一时间显示给不同的用户。应用必须保证在其下面的数据或者模型改变时视图的更新。为改变模型，用户提交一个请求给控制器，由控制起来配合改变模型。数据视图必须跟着改变，以反映最近的模型改变状态。

Smalltalk MVC 方案使用观察者通知模式。在这种模式下，每个视图注册为一个模型数据的观察者。然后模型可以通过发送消息给所有这册观察者，通知它们相关的改变。其为 Smalltalk MVC 框架已经通用化了，他也可以将它应用到其他平台上面。

2.4.2. Model 2的出现

JSP 的意图是使创建动态页面更容易。JSP 首先是作为 servlet 的替代引入的，还有就是 MS 的 ASP。Servlet 的强大功能当作易于创建服务器页面的工具提供给开发者。但强大的功能伴随着巨大的责任。很多团队发现，如果他们一不小心，他们的项目就会因为纠缠如麻的页面变的容易崩溃。进一步的特性需要使用复杂的脚本程序。但脚本程序是非常难于重用的—除非你在页面间把代码“拷贝粘贴”。

工具页面也可以包括进来，但它们很难被组织在一起，并且造成非常丑陋的“资源”树。有些东西会出错。

很多开发人员很快意识到， JSP 和 servlet 可以一起使用来部署 web 应用。Servlet 可以应付控制流，而 JSP 则可专注于讨厌的编写 HTML 的任务。在这种情况下，结合使用 JSP 和 servlet 开始被称为 Model 2 (单独使用 JSP 称为 Model 1)。

当然，从 Sun 那里仍然没什么新东西... 而且很多人很快指出 JSP Model 2 类似于经典的 Model-View-Controller 架构。

在很多场合，现在交互使用 Model 2 和 MVC 这两个词已经很平常了，虽然还有一些争论，即一个应用是否是 MVC，以及是否支持经典的观察者通知模式。没有观察者通知的 Model-View-Controller 有时被称为 MVC2 或 Web MVC。.

2.4.3. 应用层—视图之间的去耦合

Model 2 被认为区别于 MVC 的一个原因是，观察者/通知模式不能在 web 环境内工作的很好。HTTP 是一个“拉”的协议: 客户请求然后服务器响应。没有请求就没有响应。观察者模式需要一种“推”协议来进行通知，以便服务器能在模型改变时将信息推送到客户端。虽然也有一些方法能模拟将数据推送到客户端，但这和基本情况相悖，并且会视为是个权宜

之计的修补。

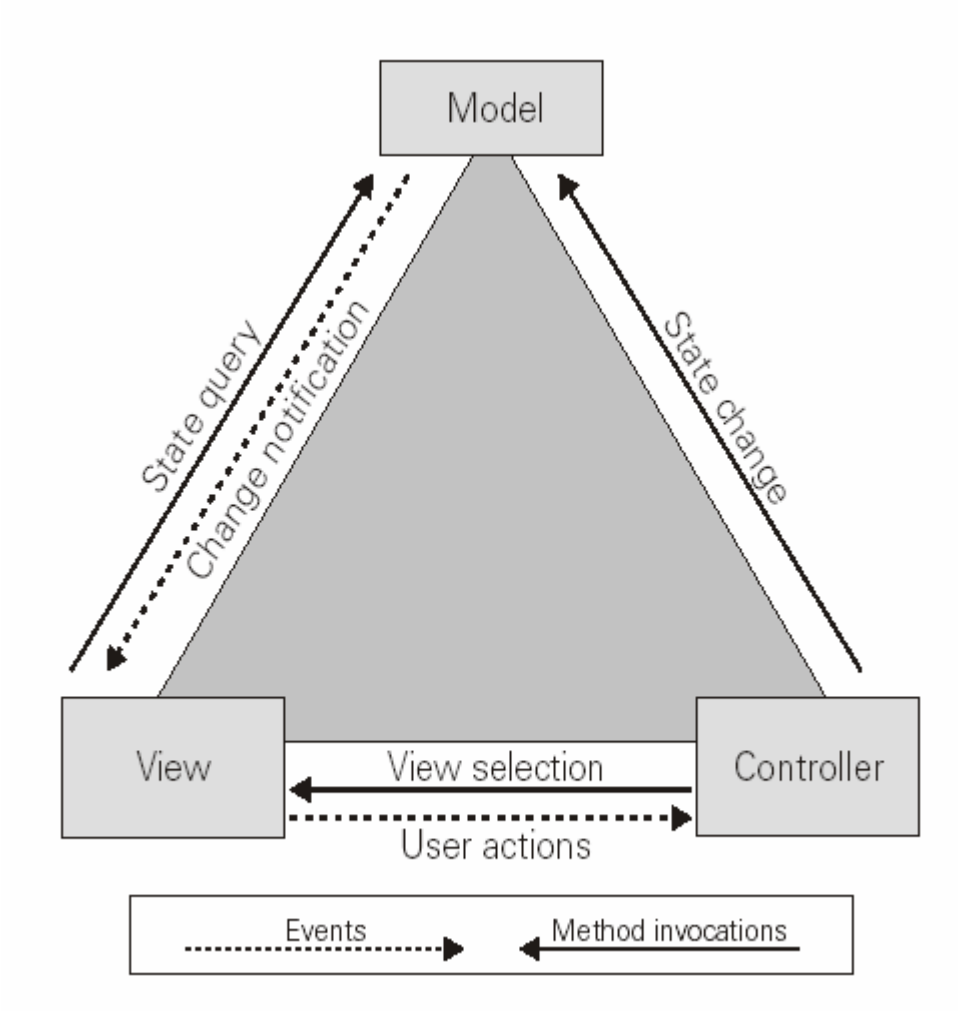


图 2-7 MVC 通常表示为 3 个互相连接的组件

图 2-7 是典型的 Model-View-Controller 范式，经常被表示为：一个互相连接的三角形。在 web 应用中维护范式中的“通知改变”部分是非常困难的。

这些东西在所有资源都在一台服务器上，而且客户端保持一个开放连接的情况下工作得非常好。如果资源分布在不同的服务器上，并且客户端不能维护一个开放的连接情况下，工作的并不理想。

许多分布式系统架构，包括 web 应用，在视图进行状态查询的概念时退缩了。绝大多数情况下，远程应用是按***层模式***[POSA]设计的。基本上，层模式下，层内的对象可以和同一层或者相邻层的对象进行通信。在一个复杂应用中，这可以在添加组件时，防止依赖关系呈指数增长。在设计远程应用时，分层是一个核心模式。

从 MVC 上下文中，引入层模式将状态改变和状态查询的职责加于控制器之上，并伴随着改变通知。

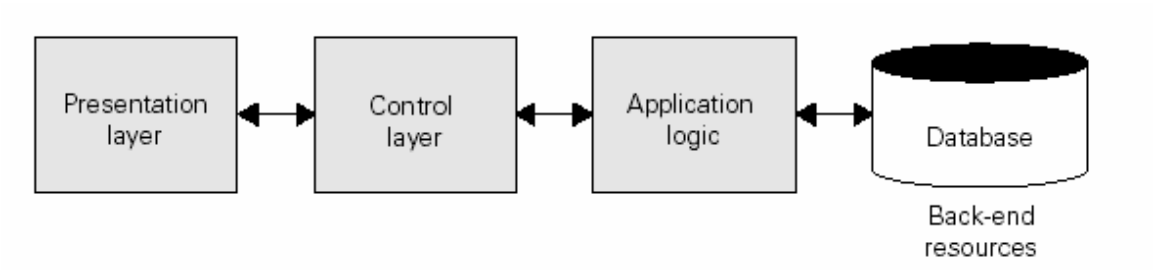


图 2-8 Web 应用的层模式

如图 2-8 ，分层的 web 应用使用一种比传统 MVC 模式更加“扁平”的模式。控制器被夹在表现层(View) 和 应用逻辑 (Model)之间。

每个组件的主要职责并没有改变。流程有轻微改变，即查询状态和改变通知都必须通过控制器。另一个改变是，当视图，或者表现层需要渲染动态页面时，它使用从控制器传递的数据而不是直接来自于模型层。这种改变去除了 View 和 Model 的耦合，允许控制器选择数据和显示这些数据的视图。

2.4.4. Struts如何实现Model 2, MVC, 和分层

Struts 通过提供一个控制器 Servlet 实现了 Sun 的 Model 2 架构，这个控制器可以用来管理JSP 页面和其他表现设备之间的流程。Struts 通过使用 ActionForward 和 ActionMapping 来保证表现层之外的控制流决策来实现 MVC/层 模式。JSP 可以引用一个逻辑目标。控制器组件在运行时提供准确的 URI。

表列出了 Struts 的核心类，即对应的经典的 MVC 组件职责。

表格 2.2 核心 Struts 类和 MVC 的对应

类	描述
ActionForward	用户指向或者视图选择
ActionForm	状态改变的数据
ActionMapping	状态改变事件
ActionServlet	控制器，接受用户请求和状态改变，以及发出视图选择
Action	控制器的一部分，与模型交互，执行状态改变或状态查询，以及告诉ActionServlet 下一个选择的视图

除了这些核心类， Struts 使用一些配置文件和视图助手（view helpers）来沟通控制器和模型。下表列出了 Struts 配置文件和描述了他们在架构中的角色。

表格 2.3 Strtuts 配置文件

文件	目的
ApplicationResources.properties	存储本地化信息和标签，以使应用可以国际化
struts-config.xml	存储控制器对象的缺省配置，包括模型支持的用户指向，状态改变，状态查询

为将 Struts 配置数据暴露给视图，框架以 JSP 标签的形式提供了大量的助手类，如表：

表格 2.4 Strtus 视图助手

标记库描述符	目的
struts-html.tld	扩展HTML Form的JSP标记
struts-bean.tld	扩展处理JavaBean的JSP标记
struts-logic.tld	扩展测试属性值的JSP标记

将以上内容放在一起，下表按层列出了 Struts 组件：

表格 2.5 Struts 组件，按层索引

视图层	控制器层	模型层
JSP 标签扩展	ActionForward ActionForm 类 ActionMapping ActionServlet Action 类 ActionErrors MessageResources	GenericDataSource
JSP, Velocity 模板, 以及其他表现系统	各种工具类, 比如 CommonsDigester和 CommonsBeanUtil	开发者提供的其他数据服务和API

注意 根据层模式（2.4.3)，组件应该只能和相同层和相邻层的组件交互。因此， Model 组件不能直接和View组件进行交互。

实践中，控制器与视图的交互通过请求，会话以及 Servlet 平台提供的应用上下文进行。(2.3.2)。

控制器和模型的交互通过文件和数据存储系统完成 (比如装入 XML 文档或者属性文件)，或者通过其他服务，如 TCP, 创建一个到 JDBC 数据库的连接。

2.5. Struts 控制流

因为 web 应用是动态的，所以很难表现“一个真正固定的控制流”。取决于环境，不同的方式下有很多不同的事情发生—特别是在 web 应用中。但是事情仍然有一个通用的秩序。

如果你是个 Struts，应用框架，甚至 web 应用的新手，这些流程刚开始可能难以跟得上（理

解）。亟待解决的各种问题不一定那么明显。我们将在本书中慢慢详细涉及。首先，在介绍树木之前我们先认识这片森林。你读完此书后，我们建议你再次回来，看看每一部分是如何切合进这个总图的。

2.5.1. 总图

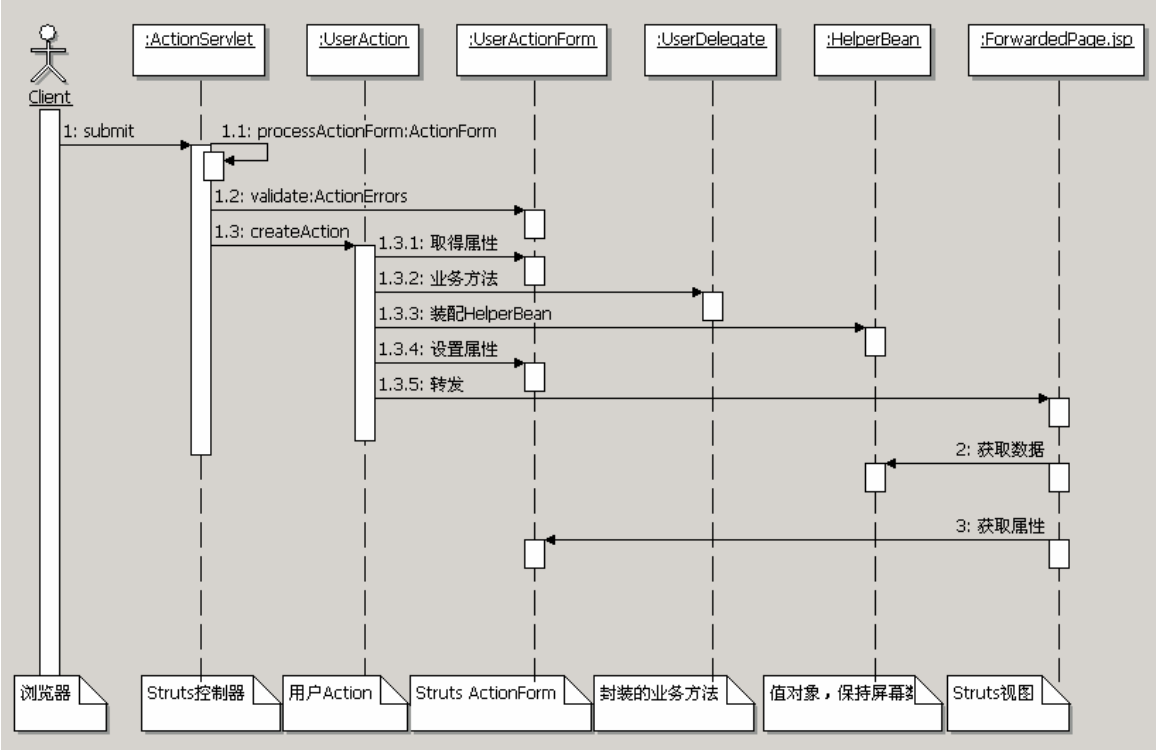


图 2-9 Struts 请求-相应流程

图2-9 以UML以次序图的方式展示了Struts 请求-响应流程。我们来按这个请求-响应流程走一遍。 括号内的数字请参照图11中的相关地方：

- 客户请求匹配 Action URI 样式的路径 (1).
- 容器将请求传递给 ActionServlet.
- 如果这个是模块化应用，ActionServlet 选择响应的模块。
- ActionServlet 查询路径的映射。（来自于配置文件）
- 如果映射标明了一个 form bean，ActionServlet 看是否已经有一个实例，或者创建一个新的实例 (1.1)。如果已经有一个 form bean，ActionServlet 重设它，并根据 HTTP 请求重新组装它。
- 如果 mapping 的 `validate` 属性设置为 true，它将调用 form bean 的 `validate` 方法(1.2)。
- 如果失败，Servlet 将控制转发到 `input` 属性标明的路径，控制流终止。
- 如果 mapping 标明一个 Action 类型，如果它已经存在或已经实例化，它将被重用(1.3)。
- Action 的 `perform` 或 `execute` 方法被调用，并传递一个实例化的 form bean (或者 null)。
- Action 组装 form bean，调用业务对象，以及其他需要做的事情。 (1.3.1-1.3.4)。

- Action 返回一个 ActionForward 给 ActionServlet (1.3.5).
- 如果 ActionForward 指向另一个 Action URI，重新开始； 否则，显示页面或者其他资源，流程结束。通常，结果是一个 JSP 页面，或者 Jasper，或其它类似技术 (非 Struts)渲染的页面。 (2, 3).
- 如果 JSP 中使用了 Struts HTML 标记，并且在请求中看到正确的 ActionForm (1.1)，他们会从 ActionForm 中组装 HTML 控件。否则， `<html:form>` 标记将创建一个。从 Struts 1.1 开始， 如果 form 标记自行创建一个 ActionForm ，它将调用 ActionForm 的 Reset 方法。如果你只是想创建一个空白的表单 (1.1)，你可以使用标准的 ForwardAction(见第 8 章) 来通过 Action 传递控制，然后离开页面。

2.5.2. 出色的细节

他们说，恶魔总是藏在最隐秘的地方。

前面章节的大纲和图示很好的显示了 Struts 的概揽，但忽略了很多重要的细节。让我们深入到更好的地方。因为这里是 HTTP，所有东西都是从请求开始。

请求由容器接收

Struts 框架的核心组件是 ActionServlet。象所有的 servlets，它生存在容器中，比如 Tomcat，Resin，或者 WebLogic 等。当容器启动时，读入部署描述符 (web.xml)，告诉容器要装入哪些个 servlet。

一个标准的 servlet 设定是 *servlet mapping*。 容器使用这个设定来决定哪个请求将被送到哪个 servlet:

```
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>/do/*</url-pattern>
</servlet-mapping>
```

这里，我们让容器将ActionServlet映射到那些符合/do/*样式的请求。这些请求可以是：

/do/This

/do/That

/do/something/Whatever.

许多应用喜欢使用前缀：

```
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

URL 样式也可以使用象 this.do 或 that.do 或 /something/whatever.do 的样式。可以用任何有效的扩展名，但.do 是比较简单和常用的选择。

当一个请求随着符合 Servlet 上下文的路径组件提交时，容器将其转发给 ActionServlet。不匹配的请求则不转发到 ActionServlet。比如，匹配*.jsp 的请求将直接转发给容器的 JSP 服务，比如 Jasper（如果你使用 Tomcat 或者 WebSphere 的话）。在应用中可以有其他的 servlet 来处理其他格式的请求路径。不匹配所有 servlet mapping 的请求将直接送给容器默认的 web server。

请求被ActionServlet接收

当 ActionServlet 收到一个请求，它通过一系列的流程处理 locale, mapping, form bean, 最后是 Action 来运行这个请求。这里某些步骤只在 Struts 1.1 应用才有:

- *处理多部分请求.* 如果请求是个多部分（multipart）请求 (比如，一个表单加上一个MIME附件)， Servlet 用一个特殊的句柄包装这个请求，避免处理错误。
- *处理路径* ActionServlet 首先检查这个请求路径是否是一个应用模块。如果是，相应模块的配置被选中。[Struts 1.1]
- *处理场所.(local)* 默认下，ActionServlet 会检查是否有一个标准的locale 对象在用户会话的上下文当中。如果没有， ActionServlet将放入一个。这个对象将为每个用户提供本地化表现。
- *处理内容和缓存* 默认的MIME 类型和可选的请求头将加在响应之上。
- *处理映射（ Mapping）* ActionServlet检查ActionMapping，是否有注册的路径符合正处理的请求。如果没找到，ActionServlet 转发到默认(或 “unknown”) Action（如果设定有默认或未知Action）,否则，产生一个 “bad request” 错。如果找到相关映射，将被放入请求之中，供后面引用。
- *处理角色.* ActionServlet 检查是否用户被授权可以访问action. [Struts 1.1]。
- *处理ActionForm.* ActionServlet 检查是否mapping 中指定了一个ActionForm.。如果是，servlet 检查是否已经有一个存在于特定的范围之内(默认是会话)。如果不存在，ActionServlet 创建一个。
- *处理组装.* ActionForm的 reset 方法被调用，然后通过反射机制被自动组装。匹配ActionForm属性的参数将被使用。其他参数和属性被忽略。
- *处理验证* ActionForm的 validate 方法被调用。如果方法返回false, 控制传递给ActionMapping的input 属性标明的路径， Action 未被处理。
- *处理转发和包含* 如果ActionMapping 标明forward 或include 属性，控制被传递给其他资源。否则ActionServlet 将请求委托给一个Action 对象。
- *处理 Action.* 如果mapping 标明一个Action 类型, ActionServlet检查是否已经有一个被实例化了。如果没发现， Action 对象将被实例化。每个类只能有一个Action 对象 (Singleton 模式)，它通过多线程处理所有对它的请求。Servlet调用 Action的 perform 或 execute 方法，传递请求, 响应, mapping, 以及form bean。

Action 执行要求的行为，可以包括：

- 访问数据系统，比如JDBC 数据库
- 在请求中创建被视图使用的对象
- 如果需要，在会话中创建对象
- 根据需要更新会话对象，比如用户的场所
- 执行需要的业务功能

- 处理以外和其他错误条件
- 发送一个直接发送一个响应，或者返回一个ActionForward 给servlet

这里某些行为，象访问数据库，通常由 Action 调用的业务对象来处理(Business Delegate 模式)。 Action 处理一些 web 特定的任务，可以放在业务对象中的代码都应该放入业务对象。Action 是一个控制器类，不应该用来处理业务的核心逻辑。

Action返回ActionForward

Action 完成后，它返回一个 ActionForward。如果 ActionForward 为 null, ActionServlet 假定响应产生了，但不做任何事情。否则， ActionServlet 读入 ActionForward ，重定向或者转发请求到相应的资源。

如果请求是另一个 Action URI, 容器将请求返回给 ActionServlet。否则容器发送请求到其它 servlet 或 service。

如果 ActionForward 设为重定向（redirect），请求被发送回客户端，并提示重新提交一个新请求到特定的地址。

由Jasper (或类似的东西) 渲染JSP页面

ActionServlet 发送一个请求 到 JSP, 请求是被另外的服务处理, 如 Jasper。典型地, Struts 和其他标签扩展用来编写页面的动态部分。有时，也使用 JSP 模板，以便页面可以从其它组件进行构建。

通常， 动态数据在 JavaBean 中传递到请求上下文中的页面。这就是熟知的视图助手（*View Helper* ）*模式* [Go3]。 标签扩展简单的调用 JavaBean 的方法，并返回格式化的数据。而数据如何被放入页面中那是表现逻辑的事情。数据本身的格式通常是业务逻辑的一部分，所以委托给了 bean。

Struts 标记也可以访问框架提供的视图助手。这些包括本地化标签和提示，错误信息，以及超链接路径。另外，Struts 标记可以计算表达式，通过列表反复，以及在 HTML 表单中组装控件。

其他 servlet 渲染响应

处理完 Action 后，请求可以被送到应用中的其他 servlet 或服务。其他表现系统，如 Velocity templates, 可以通过 servlet 上下文访问框架资源。

2.5.3. Struts 是富有效率的吗?

详细描述完 Struts 处理流程后，你可能会想知道这些的花多长时间。通常，Struts 应该能提升大部分正确设计的 Web 应用的性能。在本节中，我们检查一些关系到框架效率的特殊设计点。

定义 Performant 是一个法语词，意思是有效率（efficient）。软件工程常用这词来表示一个流程或者设备在实际中执行得很好。

Struts不仅是线程安全（thread-safe）而且是线程依赖（thread-dependent）的

Struts 使用轻量的 Action 对象，而不是各个单独的 servlet，来对请求处理响应。 Struts 实

例化每个 Action 类一次，并允许其他请求通过原有的对象线程化。这种核心策略节省了资源，并提供最大的吞吐性。一个正确设计的应用将通过使用一个单独的 Action 来路由各种相关操作来发挥这种特征。

ActionForm bean最小化子类代码并缩短子类层次

Struts 框架的一个关键点是可以从请求中自动组装 ActionForm bean 。没有这个组件，用户不得不自行编写代码并实例化来组装每个 bean 类。小心使用反射机制会节省不少资源，资源是有限的，并允许它们更好的使用。

Struts 标签库提供通用功能

Struts 一起提供的 bean 和 logic 标记库符合大部分 JSP 标记的需要。它们减少甚至消除了编写额外标签的需要。JSP 规范就包含了 JSP 中的标签重用。使用相同的通用标签 3 次比使用 3 次不同的标签来的有效率。

Struts 组件对应用来说都是可重用的

框架绑定的工具可以在大部分应用中使用。 `BeanUtil.populate` 方法就是个例子。这个方法用来从 HTTP 请求组装一个 ActionForm Bean，但也可以用来从其它类型的映射中组装一个 FormBean。重用组件可以减少开销和节省资源。

Struts本地化策略减少了大量冗余JSP

通过允许本地化页面在运行时才获取，国际化应用可以为每种可能需要的语言只提供一个单独的页面。同时，相同的消息系统也可以用于处理错误信息。同一对象提供了双重用途。

Struts设计为一个开放架构

Struts 组件设计来是可以被应用子类化的，以便可以提供其它的服务功能。这使得开发人员可以扩展存在的类而不是重新编写新类。而且，Struts 也和应用共享资源。这时开发人员可以使用存在的组件，而不用编写和实例化它们自己的类。

Struts是轻量型架构

类似的框架也许提供数百个类和几十个包。整个 Struts 框架 由 5 个标记库和 5 个核心包组成。

Struts是标准兼容的

Strtus 在许多运行标准组件的容器上都工作的非常之好。

Struts是开源的，具良好的文档

这意味着开发人员可以检查源代码，找出一些潜在的瓶颈。而且 *Struts 是模型中立的*。因为 Struts 并没有对后端模型做任何假定，一个应用可以按其最有效率的方式实现模型层。Struts Actions 可以调用一系列助手类来访问需要的数据。一旦数据被检索到，对 JavaBean 的依赖，使 Struts 更容易保持值对象，这样来减少了大量的模型层调用。

2.6. Struts的长处和弱点

象一些复杂系统一样， Struts 是个解决方案包。它也有其长处和弱点。这里的某些观点具有主观性，但也希望能对你有些帮助。

2.6.1. 弱点

不管有多么喜爱 Struts，重要的是要看到框架是干什么的，它的瑕疵和所有。有些弱点已经在 Struts 1.1 release 中解决。下表列出了 Struts 1.0 的弱点，并在 Struts 1.1 中解决了。如果你以前曾使用过 Struts，这里某些问题是很精彩的，现在 Struts 1.1 可以更加符合你的要求。

表格 2.6 Struts 1.0 的弱点，在 Struts 1.1 中解决了

弱点	说明
Logging（记录）	Struts 使用容器的缺省记录系统，没有提供一个自己的记录包来为应用提供记录机制。 (Struts 1.1 实现了 Commons-Logging 接口包)
每个应用装入一个单独的配置文件	大型项目可能需要使用多个不被整个团队共享的配置文件 (Struts 1.1 支持多配置文件)
每个场所装入一个单独的资源文件	大型项目可能需要使用多个不在整个团队共享的资源文件。 (Struts 1.1 支持多资源文件)
没有服务管理器	ActionServlet 必须被子类化来提供附加服务，比如用户记录器或者身份认证系统 (Struts 1.1 提供一些新的扩展点和组件)
ActionForm 官样文章	Struts 1.0 希望开发人员创建定制JavaBeans 与HTML 输入表单一起使用 (Struts 1.1 支持 Map和 DynaBean 来代替定制JavaBean.)

表 2-6 列出了框架当前的弱点。在下一节，我们将详细的讨论它们，并象资产一样描述。“Goofus”或者“Gallant,”选择在你。

表格 2.7 Struts1.1 的弱点

弱点	说明
没有事件模型	Struts 紧密地和HTTP的请求-响应模型结合，这限制了开发人员更好地处理事件
调试	不支持自动调试（除错），开发人员不得不手工创建“断点”，并向容器的记录系统写标准输出
没有缺省的数据模型或者具体的推荐	访问持久数据模型的任务留给了开发人员
单一ActionServlet	在一个应用中只可以使用一个单一的ActionServlet，这个限制可能导致配置冲突

需要理解Struts组件	开发人员需要理解一些特殊的类以及他们如何交互。
不能提供优先技术支持	ASF是个志愿者组织，没有全职人员提供可担保的响应
Mailing list已经成为知识的障碍	Struts 有一个日益增长的邮件列表，但要在其中找到最好的建议非常困难。
正式发布版本并不快速	Struts正式发布版相对于其它项目来说显得慢了。开发人员必须经常检查“每日构件”或的最新改进。
i18n 限制	Struts 的消息资源对建立国际化的资源和错误信息非常好，但不适合于处理大文本块
JSP mindset	因为使用MVC架构，使得资源对所有表现层都是有效的。这对JSP来说是个长期的Struts弊病。
JSP异常的本地化	很多系统级消息，象JSP异常，都没有本地化，通常显示为英语
标记属性冗长	许多标记扩展要求很多参数，对编程来说显得很笨
perform 和 execute 方法签名	Struts 架构的关键是将请求委托给一个 Action类或者叫分发者dispatcher。Action 类是Struts支持的唯一分发者，并且只有通过其perform 方法来调用。这将应用限制在只能和perform 方法传递的数据一起工作。即使有办法超出这个限制， perform 方法也是个瓶颈。 一个通常的请求要求ActionServlet组装几个ActionForm。但是因为 perform 接受单个ActionForm 参数，如果不经过较大的框架革新是不可行的。 Struts 1.1 添加了一个execute方法，它有助于改善perform的其它主要缺陷：因为会返回异常。然而，其它后果仍然存在。
模糊的术语	Struts 框架在明显的增长。而给与一些应用选项和类的名称却容易让人混淆。例如，web.xml“ 中的validate” 选项却和Action对象的validate 方法无关，而和如何解析配置文件相关。同样，神秘的“ null”选项则表示当消息关键字未找到时，是否返回一个错误信息。 有一个趋势是在类层次体系中使用复合名称。在Action包中的每个类都有个前缀为“ Action,”这却是多余和容易混淆的。同时，在Struts配置文件中， ActionMappings 定义的元素名是“ Action” 而不是“ ActionMapping” 。如果开发人员引用一个“ action,” 很难区别它们是指Action 类或是配置类的ActionMapping。 在 Struts 配置中，“ name” 字段标识ActionForward 和ActionForms。“ path” 字段标识 ActionMapping。action-mapping 元素的 “ name” 属性则指出使用哪个ActionForm 。 ActionForward 的URI字段也称为“ path”，但可以包括伴随path的查询组件。到ActionMapping 的“ path”

	不包括servlet 样式，象 *.do，但是ActionForward 的path 却包括 *.do 扩展名。 应用资源其实是真正的消息资源。 等等。 凡此种种，这些小矛盾可以把一些新手搞糊涂，并且使框架难以学习

2.6.2. Struts的强项

强项	说明
以HTTP为中心	Struts设计围绕标准 HTTP 请求-响应模式，为许多Web开发人员所熟悉
标准记录	Struts 可以使用容器的缺省记录系统，而不需要配置和理解其他包
可选的调试记录	Struts 可选记录大量状态处理时的信息，它们可以有助于进行调试
模型中立	Struts 并不倾向于哪个特定的持久层
在一个中心配置中汇集实现细节	Struts 配置封装了应用，或者应用模块[Struts 1.1]的实现细节。所以它们可以作为一个整体评价和管理
允许为每个场所配置不同的消息资源	不同的语言翻译可以工作在他们自己的消息资源文件版本上。添加一个新场所的支持仅需简单地添加一个资源文件
轻量	Struts 仅有几个核心类要学习
开源	全部源代码在自由的 Apache 软件许可下，所有的选择都在你
强大的开发人员团体	有一个强大的开发人员团体使用Struts。邮件列表时非常活跃的。许多开发人员的扩展是很成功的
强大的供应商团体	Struts 已经和其他一些产品合在一起，包括Jcorporate的s Expresso 和 IBM的 WebSphere。一些厂商也提供Struts专用工具。
强大的产品支持	Struts 有其自身的专业管理的JGuru 论坛。Struts邮件列表至少可以通过两种支持门户进行访问。 Struts 已经被许多文章和书籍涉及，并有一些组织提供专业的教程
强大的开发团队	超过30个开发人员为Struts 1.1做贡献。Struts团队现在有九

	个活跃的志愿者，他们全为源代码负责
稳定发布版本	Struts正式发布版本要经过长期的测试，并没有最后期限，所以团队可以提供高质量的产品
i18n 支持	支持内建的国际化
高度兼容	Struts专注于提供公共标准100%兼容的产品
全方位的标记扩展	Struts包括一系列通用标记扩展。他们一起可以符合你所有的 JSP 需要，而不用编写脚本程序
良好文档的源代码	Struts JavaDoc非常详细，以使你几乎不需要参考源代码。这其实是个高级别的用户指南。
建立在设计模式之上	Struts 框架在其架构中实现了一些经典的设计模式，这些模式为许多开发人员所熟知。
可扩展性	所有默认的设置都可以配置。核心Struts可以被重写，和子类化。开发人员可以定制关键类如ActionForm 和Action.

2.7. 小结

今天的开发人员需要建立完整的应用，并且可以随时维护。Web 应用框架，如 Struts，解决了这个普遍问题，所以开发人员可以专注于它们应用的特定功能。在开发 Web 应用时，框架特别重要，因为 HTTP 和 HTML 要创建动态应用非常困难。

Struts 使用了大多数标准的 Java servlet API 并成为一些 servlet 容器的兼容性测试。Struts 也构建于通用的设计模式，特别是 MVC 架构。框架鼓励应用采取分层设计。这种设计使应用具有强壮性和伸缩性。

架构的一个关键之处是它扩展了 HTTP 请求-响应循环的流程。Struts 控制器管理着应用使用的路径，帮助安全的收集用户输入，并可以本地化应用消息，特别是错误消息。

Struts 是一个富有效率的解决方案。它绝不会抑制你的应用，并随处有很多免费资源可以使用。

当然， Struts 还有一些缺陷。许多类名的选择在开发时显得比较轻率，容易引起混淆。其他一些地方也值得改进。

尽管有一些阻碍， Struts 也很容易地成为现今最流行的 Web 应用框架。

在下一章，我们回过头来运行 Struts，并创建另一个应用。

3. 构建一个简单应用

本章内容

- *创建一个简单应用*
- *扩展一个应用*
- *修改一个应用*
- *在应用中使用JSP页面*

A human being should be able to change a diaper, plan an invasion, butcher a hog, conn a ship, design a building, write a sonnet, balance accounts, build a wall, set a bone, comfort the dying, take orders, give orders, cooperate, act alone, solve equations, analyze a new problem, pitch manure, program a computer, cook a tasty meal, fight efficiently, die gallantly. Specialization is for insects.

—Excerpt from the notebooks of Lazarus Long, from Robert Heinlein's *Time Enough for Love*

3.1. 被支柱支撑的支柱

现在，各种团队都在编写各种各样的 web 应用。在应用中使用分层架构 [POSA] ，如第 1 章和第 2 章所述，可以很容易地让团队的成员分别专注于应用的不同部分。但是，仍然很有必要让团队的每个人都从头到尾理解整个处理流程。在我们深入探究 Struts 的各部分是如何优雅的相互结合之前，让我们先从头开始构建一个简单但有用的程序。在这一章，我们将通过教程、逐步解剖来和 Struts 来一个亲密接触，然后构建一个供用户登入和登出的程序。

虽然没有第一章的例子显得细微，我们仍然暂时保持例子的简单性。在第 4 章才会涉及到一个实际的应用。

在本章，我们将从用户的角度来讨论一个经典的登录 logon 应用。第一章的练习只是对从注册页面输入的密码信息进行比较，根据是否匹配来控制页面流的分支跳转。该应用让您使用第 1 章的练习所创建的账户来进行实际登录。控制流和页面内容的改变取决于你的状态。

引入这个应用后，我们将对它进行分解来仔细探讨其各个部分。如果你的机器上安装了 Struts 开发环境，你可以跟着来做。当然，你也可以舒服地靠在你的椅子上，喝着卡普契洛咖啡，来看它是如何工作的。

然后，打好基础后，我们就开始一步步构建这个应用。

每部分的表述都力图使你可以根据它自行完成，目的是你可以编写之（而不是单调的重写工作）。如果你在你的计算机前面，你可以跟着我们来输入源代码。如果不，每部分中的足够的细节也使你可以只看书就跟上我们。

3.1.1. 为什么选择 logon应用?

我们的例子程序允许用户登录到应用中去。一旦用户登入，页面的改变将反映出用户已经得到授权。一般情况下，这都是大型应用的第一步，授权用户进入，进行它们感兴趣的操作。但我们现在的目的，只是登录进一个用户，这已经足够用来演示 Struts 应用是如何实际工作的了。

如表 3.1 所示，我们选择 logon 应用仅仅因为它易于理解，简单，自包含，而且在许多应用中都需要。

表格 3.1 为什么选择登录应用

原因	解释

好理解	我们几乎都需要登录到应用的共享部分，所以这个流程很好理解
简单，且自包含	一个接受用户登录的应用可以写得很简单并且是自包含的，不需要复杂的模型
被许多应用需要	我们几乎都需要编写使用某种登录流程的应用，所以这就是我们需要的代码

3.2. 漫游logon应用

在开始我们的 Struts 的漫游之旅之前，我们先讨论这个应用的范围，以及你如何才能跟得上我们的节奏。然后我们来看看应用中使用的屏幕，请注意在登录以后是如何变化的。我们完成这个漫游之后，会再回头过来看一看。

3.2.1. 从这里开始

我们创建 logon 应用的目的是给你一个关于 Struts 的各个部件间是如何配合工作的印象。为了不至于跑题，这个应用仅包含我们需要用来展示整个框架的那些组件。它没有包含实际的业务逻辑，单元测试，或者漂亮的对话框。虽然这些东西对一个要交付的成型产品来说是很重要的，但在会跑之前我们得先学会走。

Logon 应用也有意搞得很刻板。它没有修饰 HTML 来愉悦眼睛—仅仅是一些我们需要来接受登录的初步功能。

当然，你的 Struts 应用可以编写得很漂亮，只要你高兴。

如果你愿意在你的机器上运行这个应用，请访问本书网站的下载页面 [Husted]。 这是一个可以自动部署的 WAR 包。

我们并不要求你保持打开这个应用，但在某些时候是很有趣的。你需要跟着做的东西都在本章列出来了。

首先，我们从用户的角度来漫游一下各个屏幕。然后，再回过头来看看如何编写实际的代码。

3.2.2. 我们看到的屏幕

如表 5 所示，logon 应用有两个屏幕: 欢迎屏幕和登录屏幕。

如果你跟着做，并将应用部署在你的本地计算机上，你可以在浏览器中用以下地址访问欢迎页面：

```
http://localhost:8080/logon/
```

表格 3.2 Logon 应用的屏幕

屏幕	目的
欢迎屏幕	欢迎用户并提供到应用的连接

登录屏幕	允许输入用户名和密码

3.2.3. 欢迎屏幕

第一次访问欢迎屏幕时，它只有一个链接， “Sign in”(见图 3-1)。如果点击这个链接，就会出现登录屏幕。

图 3-1 欢迎屏幕

3.2.4. 登录屏幕

登录屏幕输入和提交用户名和密码，如图 3-2 所示。

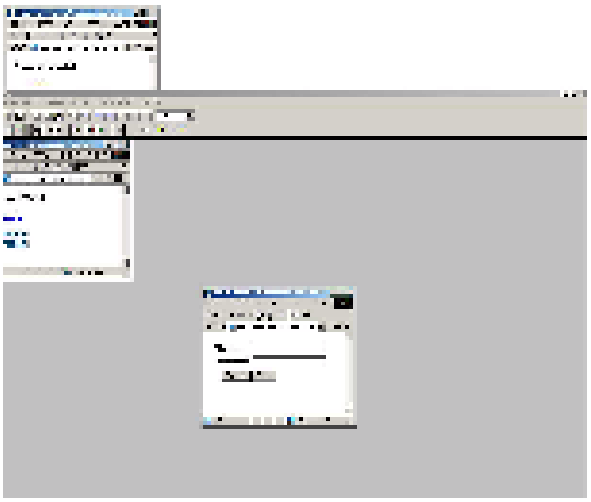


图 3-2 登录屏幕

为了看看实际动作的登录表单，什么都不输入，点击提交。登录屏幕将返回，但伴随了错误提示消息，如图 3-3。

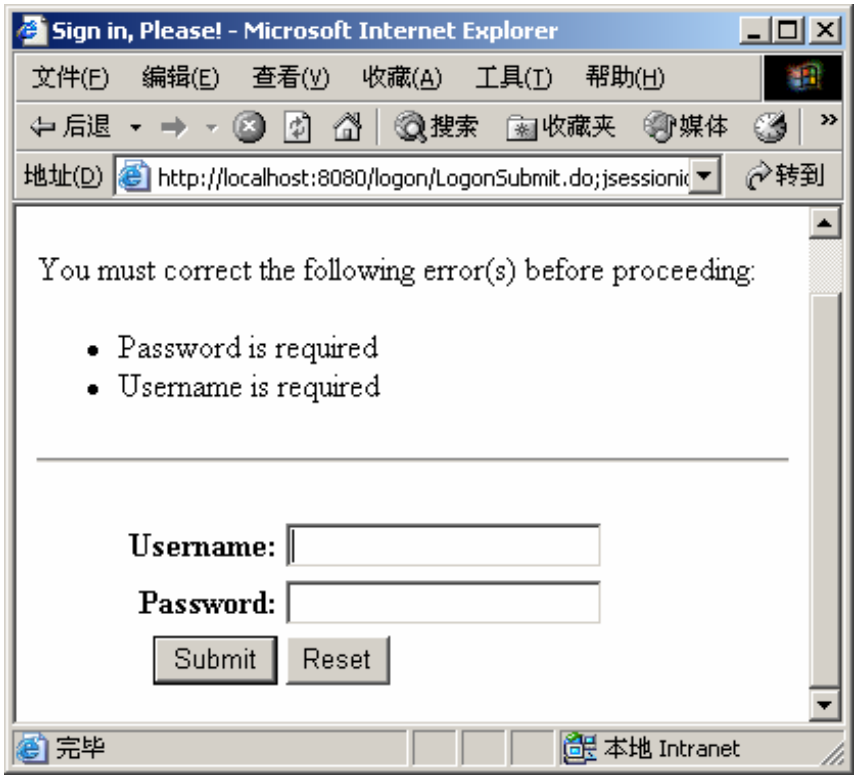


图 3-3 登陆屏幕告诉缺少用户名和密码

如果你输入用户名但是忘记输入密码提交后，消息变为图 3-4 所示：

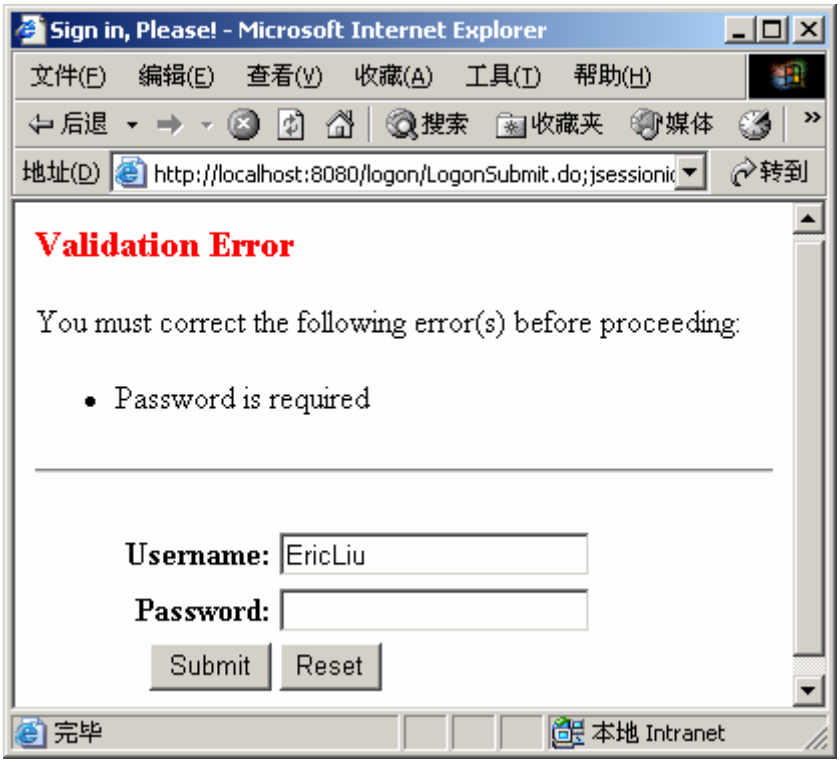


图 3-4 缺少密码

这里有关工作流过程中重要的是，从用户的角度来看：

- 它马上告诉用户缺失所有信息
- 如果用户只是提交一个信息，它会提醒用户还缺另外一个信息；
- 它重新显示用户进入的屏幕，而不是要求用户按下Back键
- 用户设法输入用户名和密码，表单被接受，并显示下一个屏幕。

logon 应用根据一个属性文件来校验用户的输入，就象第 1 章所用的那个文件。如果你从本书的网站上下载了这个 logon 应用，你可以用本书的作者名字登陆，如表所示：

表格 3.3 缺省的登录用户

用户名	密码
Ted	Husted
Cedric	Dumoulin
George	Franciscus
David	Winterfeldt

3.2.5. 重新显示欢迎屏幕

成功登录后，将重新显示欢迎屏幕---不过有两点不同。

图 3-5 用户登录后的欢迎屏幕

首先，它根据登录的用户做了调整。不再显示 “Welcome World!” 而是显示欢迎已经登录的具体的用户名字。

另外，你会注意到，加入了另一个链接，除了 sign in 外你会还看到 Sign out。

3.2.6. 欢迎屏幕，再见

为结束这个循环，点击 sign-out 链接，我们将返回最初的欢迎屏幕，如图 3-1 所示。

3.2.7. 所使用的特征

虽然简单，我们的这个应用却展示了以下的技术：

- 编写链接
- 编写表单
- 校验输入
- 显示错误消息
- 重装表单
- 显示代替文本

虽然不明显，也展示了：

- 从动态页面引用图象
- 重写超链接

在下一节，我们将深入应用的源代码去看看其核心特征是如何实现的。

3.3. 解剖 logon 应用

既然我们已经和 Struts logon 应用打过招呼了，我们现在回过头去仔细打量一下它。

现在，我们将展示每个页面的代码以及它们的相关组件，并解释每一部分是干什么的。待介绍完所有的部分后，再来演示如何将它们组装在一起。

3.3.1. 欢迎屏幕的浏览器代码

你可以回头看看，我们的应用开始于一个欢迎屏幕。让我们来看一下这个页面在浏览器中的代码：黑体部分是显示在屏幕上的内容：

```
<HTML>
  <HEAD>
    <TITLE>Welcome World!!</TITLE>
    <base
href="http://localhost:8080/logon/pages/Welcome.jsp">
  </HEAD>
  <BODY>
    <H3>Welcome World!</H3>
    <UL>
      <LI><a href="/logon/logon.do">Sign in</a></LI>
    </UL>
    <IMG src='struts-power.gif' alt='Powered by Struts'>
```

```
  </BODY>
</HTML>
```

清单 3.1 欢迎页面的浏览器代码

如果你是个 web 应用的新手，一个重要的事情是，请注意这里除了标准的 HTML 外绝对不能有其它东西。事实上， 除了浏览器能理解的通用标记外，页面中也绝不可能有其它任何东西。所有的 web 应用都基于 HTML 的限制，并不能做一些 HTML 不能做的事情。Struts 可以很容易的使用 Velocity 模板， JSP，以及其它表现系统来编写你想要的 HTML，但是你所必须做的所有事情都必须使用浏览器能理解的标记。

JSESSIONID 关键字

在这个浏览器的源代码中可能有一个你不认识的非标准HTML。当首次访问这个页面时，sign-in链接实际上可能是这样：

```
<LI><a
href="/logon.do;jsessionid=aa6XkGuY8qc">Sign
in</a></LI>
```

大多数web应用都需要保持对使用该应用的用户的跟踪。HTTP有一些基本的对维护用户登录的支持，但其方法非常有限并且不安全。Java servlet 框架对维护用户会话提供了强大的支持，但需要有一个机制来通过HTTP维护会话。

Jsessionid是一个容器维护的关键字，用来通过HTTP跟踪用户。在一个超链接中包含一个会话关键字称为URL重写。Servlet规范 [Sun, JST]鼓励使用 cookie来维护会话。当这种方法行不通时，可使用 URL 重写来代替。浏览器在第一次请求容器时，容器并不知道浏览器是否接受cookie。所以容器可以先向浏览器发送一个cookie，但并不告诉它下次请求时是否还被接受(HTTP不进行 “握手.”)同时，对当前请求的响应必须输出。所以，面对浏览器的第一个页面通常需要使用URL 重写。如果后来的请求中，容器发现了浏览器可以接受cookie，它可以跳过URL重写。

3.3.2. 欢迎页面的JSP源代码

现在让我们看一下产生图 3-1 页面的 JSP 源代码。JSP 标记以黑体形式显示。

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<HTML>
  <HEAD>
    <TITLE>Welcome!</TITLE>
    <html:base/>
  </HEAD>
```

```
  <BODY>
    <logic:present name="user">
      <H3>Welcome <bean:write name="user"
          property="username"/>!</H3>
    </logic:present>
    <logic:notPresent scope="session" name="user">
      <H3>Welcome World!</H3>
    </logic:notPresent>
    <html:errors/>
    <UL>
      <LI><html:link forward="logon">Sign in</html:link></LI>
      <logic:present name="user">
        <LI><html:link forward="logoff">Sign out</html:link></LI>
      </logic:present>
    </UL>
    <IMG src='struts-power.gif' alt='Powered by Struts'>
  </BODY>
</HTML>
```

清单 3.2 欢迎页面的 JSP 代码

现在我们来看看，黑体部分做些什么

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
```

这是 JSP 中相当于 import 语句的东西，它使标明的标签扩展在接下来的页面中有效。

而代码

```
<html:base/>
```

则产生一个标准的 HTML base 标记，以便对如图片这样的引用要相对于这个 JSP 页面的地址。你可能会注意到有时 logon 应用引用到.do 页面。这不是实际的服务器上的页面文件，而是对开发人员编写的 Java 类，或者 Action 的引用。这些 Action 然后转发到产生响应的 JSP 页面。

JSP 通常包括到 HTML 资源如图片和样式表的引用。最方便的方法是通过相对于 JSP 模版的路径来引用。但当 Action 转发控制时，它并不事先通知浏览器。如果浏览器被给定一些相对路径，它就会根据 Action URI 来解析它们，而不是 JSP 文件模版的位置。

依赖于你何时访问欢迎页面，其地址可能被浏览器显示为：

```
http://localhost:8080/logon/
http://localhost:8080/logon/LogonSubmit.do
http://localhost:8080/logon/Logoff.do
```

这对动态应用来说是个很常见的问题。HTML 规范[W3C, HTML] 提供了一个标记作为这个问题的解决方案。Struts 也提供了一个类似的 html-base 标记，可以插入 JSP 之中。如果你查看 logon 页面的 HTML 源代码，查询其外观地址，你就可以看到该标记被被渲染成：

```
<base
href="http://localhost:8080/logon/pages/Welcome.jsp">
```

这可以让浏览器找到“Powered by Struts”图片，该图片也存放在 pages 文件夹下面。

现在，让我们来看这段代码：

```
<logic:present name="user">
<H3>Welcome <bean:write name="user" roperty="username"/>!</H3>
</logic:present>
```

你可能记得 welcome 页面会根据用户是否登录进去来定制页面显示。这一段就是检查是否在客户会话中存储了一个“user” bean。如果这个 bean 存在，则显示欢迎这个用户。

下面的代码显示了为什么维护用户会话是这么重要。(第 3.3.1 节)。庆幸的是，Struts 标签和 servlet 容器一起来自动维护会话(不管浏览器是否设置为使用 cookie)。对开发人员来说，感到就象会话是内建在 HTTP 里面一样——这就是框架的作用。框架扩展了基础环境，所以开发人员可以专注于更高级的任务。

```
<logic:notPresent scope="session" name="user">
<H3>Welcome World!</H3>
</logic:notPresent>
```

相反，如果 user bean 不存在，我们便使用一个通用的欢迎信息。所有的 Struts 逻辑标签都使用“this”和“notThis”格式。其它标签则并未提供。

虽然这意味着要进行一些重复测试，但它简化了总体语法和标签的实现。当然， 你也可以使用其它标签扩展。

你并未被局限于 Struts 软件包中所提供的东西。其它志愿者贡献的标签扩展列于 Struts 资源面 [ASF, Struts]，甚至有提供 if/then/else 语法的，你可以用它们作为替代。

如 3.3.1 所述， Struts 自动重写超链接来维护用户会话。它也使你可以给链接取个逻辑名字并将实际的链接存放在配置文件之中。这就象通过关键字引用数据库一样，记录中的名字和地址可以根据需要进行更改，其它表格可以通过关键字来查找更新的版本。

这里：

```
<LI><html:link forward="logon">Sign in</html:link></LI>
```

使用 logon 作为关键字来查询存放到用户登录页面的记录。如果我们想改变这个链接，只需要在配置文件里面改写就行了。页面在其下次被渲染的时候将以新的链接开始。

这段代码结合了<logic:present> 和 <html:link> 标记，仅当用户已经登录进去时显示登出链接。

```
<logic:present name="user">
  <LI>
     <html:link forward="logoff">Sign out</html:link>
  </LI>
</logic:present>
```

3.3.3. Welcome屏幕的配置源代码

Struts 用配置文件来定义应用的一些东西，包括链接的逻辑名称。这是一个 XML 文档，Struts 在启动时读入，用来创建一个所需的对象数据库。各种 Struts 组件都引用这个 数据库 来提供框架的各种服务。配置文件的缺省名称是 *struts-config.xml.*。

因为配置文件要被各个不同的组件使用，全部展示这个文件可能会让我们昏头。现在我们仅提供与我们这个阶段相关的部分。后面，当我们全部构建应用时，我们会展示整个配置文件。

在最初的 welcome 屏幕中，我们引用了一个 logon forward。它在配置文件中是这样定义的：

```
  <forward
     name="logon"
     path="/Logon.do"/>
```

这儿，logon 是个关键字，用来查找链接的实际路径。在这里引用了 Struts action，但 path 也可以引用 JSP 页面，Velocity 模板，HTML 页面，或其它任何具有 URI 的资源 [W3C, URI]。

定义

统一资源标识符(URI) 是一个简短的字符串，用来标识在Internet 或者其他网络中的资源。资源可以是一个文档，图像，下载文件，电子邮箱，或者其它东西。一个 URI可以对应一个服务器文件系统的路径，但通常也有一个别名。Strtuts应用中的许多URI都是Java类或者Action的别名。

因为路径是在配置文件中定义的，你可以随时改变你的主意而不用触及到JSP源代码。如果 你更改并重载了配置文件，改变会在页面下次渲染时生效。

3.3.4. logon 屏幕的浏览器代码

如果我们进入 welcome 页面的 Signin 链接，它将我们带到 logon 屏幕（图 3-2）。下面是 logon 屏幕的浏览器代码。同样，黑体部分是显示在屏幕上的内容：

```
<HTML>
 <HEAD>
  <TITLE>Sign in, Please!</TITLE>
 </HEAD>
 <BODY>
  <form name="logonForm" method="POST"
        action="/logon/LogonSubmit.do">
    <table border="0" width="100%">
     <TR>
       <TH align="right">Username:</TH>
```

```
      <TD align="left"><input type="text" name="username"
          value=""></TD>
    </TR>
    <TR>
      <TH align="right">Password:</TH>
     <TD align="left"><input type="password" name="password"
         value=""></TD>
    </TR>
    <TR>
      <TD align="right"><input type="submit" name="submit"
          value="Submit"></TD>
      <TD align="left"><input type="reset" name="reset"
           value="Reset"></TD>
    </TR>
   </table>
 </form>
 <script language="JavaScript" type="text/javascript">
 <!--
  document.forms["logonForm"].elements["username"].focus()
  // -->
 </script>
</BODY>
</HTML>
```

清单 3.3　登录屏幕的浏览器代码

而下面是相应的 JSP 代码：

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<HTML>
  <HEAD>
    <TITLE>Sign in, Please!</TITLE>
  </HEAD>
  <BODY>
    <html:errors/>
    <html:form action="/LogonSubmit" focus="username">
    <TABLE border="0" width="100%">
      <TR>
        <TH align="right">Username:</TH>
        <TD align="left"><html:text property="username"/></TD>
      </TR>
      <TR>
        <TH align="right">Password:</TH>
        <TD align="left"><html:password property="password"/></TD>
```

```
      </TR>
      <TR>
        <TD align="right"><html:submit/></TD>
        <TD align="left"><html:reset/></TD>
      </TR>
    </TABLE>
    </html:form>
  </BODY>
</HTML>

<%--

Allow user to submit username and password to logon action.


--%>
```

清单 3.4 登录屏幕的 JSP 代码

让我们象前面一样一步步来看这个页面的各个部分。

首先，象前面一样，使 Struts `html` 标签扩展在后面的页面中有效：

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
```

象 Struts action 一样， 标记库 URI 是一个逻辑引用。标签库描述符(TLD)的位置在 web.xml 文件中给出。

你还记得，我们如果没有输入信息就提交的话，将会显示一个错误信息。下面的标签就是渲染此错误消息的。如果没有错误消息，标记则什么都不输出，就象从页面消失了一样：

```
<html:errors/>
```

<html:form> 标签产生一个HTML 表单供数据输入。它也产生一个简单的JavaScript 来将光标移到第一个输入域上。

`Action` 属性是对配置文件中的 ActionMapping 的引用。它告诉表单哪个 JavaBean helper 类将用来组装 HTML 控件。Java-Bean helper 基于 Struts 框架类 ActionForm:

```
<html:form action="/LogonSubmit" focus="username">
```

<html:text> 标签创建一个HTML输入控件供文本域输入。它也用JavaBean helper 的 username 属性来组装这个输入域：

```
<TR><TH align="right">Username: </TH><TD align="left">
<html:text property="username"/></TD>
```

所以，如果这个表单被返回来做校验，并且最后提交的username是`Ted`, 这个标记就会输出：

```
<input type="text" name="username" value="Ted">
```

否则，标签将使用 JavaBean helper 类中标明的 `username` 属性的缺省值初始化。通常，那是个 `null`，当然也可以是任何值。

同样 <html:password> 标签也创建一个 HTML 输入控件:

```
<TR><TH align="right">Password: </TH>
<TD align="left"><html:password property="password"/></TD>
```

Password 控件象一个文本域，但是显示文本为*而不是输入的字符。如果表单被返回校验，缺省下，password 标记会重写先前的值，而不用重新输入。如果你想每次重新输入 password ，可以将 redisplay 属性关闭。

如果初次登录失败，这段代码将清除浏览器的缓存，并以重新输入 password，而不管是否通过验证。

```
<html:password property="password" redisplay="false"/>
```

下面的标签创建标准的 HTML Submit 和 Reset 按钮:

```
<TD align="right"><html:submit/></TD>
<TD align="left"><html:reset/></TD>
```

当表单提交时，将涉及两个框架对象：ActionForm 和 Action。这两个对象都必须由开发人员创建并包含应用细节。如图 3-6，ActionServlet 使用 Struts 配置来决定使用哪个 ActionForm 和 Action 。

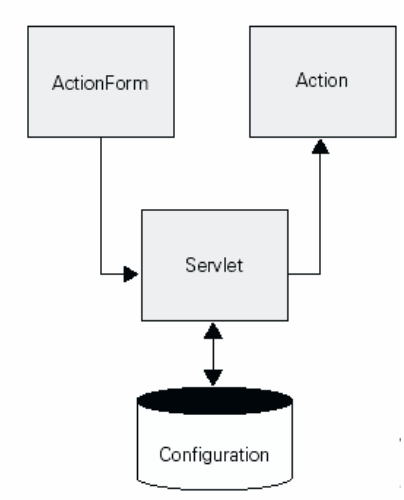


图 3-6 配置决定使用哪个 Actionform 和 Action

下面我们看看 Struts 配置中的 logon 屏幕的 ActionForm 和 Action 配置。然后再看看这些类的源代码。

3.3.5. logon 的配置源代码

logon 屏幕自身在配置文件中仅仅引用一个元素：/LogonSubmit ActionMapping。这个元素再依次引用其他两个对象，app.LogonForm 和 app.LogonAction。全部 3 个对象如下表所以：

元素	描述

/LogonSubmit	ActionMapping，封装了构建并提交一个HTML表单给Struts 框架所需要的各种细节
app.LogonForm	描述了HTML form所需的属性
app.LogonAction	处理提交的表单

3.3.6. LogonSubmit配置

上一节，我们说过 <html:form> 标签紧密配合Struts配置，使HTML表单更加有用：

```
<html:form
    action="/LogonSubmit"
    focus="username">
```

action 参数告诉 <html:form> 标签究竟使用哪一个 ActionMapping。

这里， Struts 配置中的 mapping 可能会像这样：

```
<action
      path="/LogonSubmit"
      type="app.LogonAction"
      name="logonForm"
      scope="request"
      validate="true"
      input="/pages/Logon.jsp">
```

表 3-4 提供了提供一个关于 mapping 中每个属性的意义的索引。

就象在第 2 章所述，Struts 框架使用的许多对象和属性名字都是很模糊的。比如，name 属性并不是 mapping 的名字；而是 JavaBean helper 的名字，或者是和这个 mapping 一起使用的 ActionForm bean 的名字。

相应的 form bean 在配置中是这样设置的:

```
      <form-bean
          name="logonForm"
          type="app.LogonForm"/>
```

这个元素使 logonForm 的逻辑名字和特定的 Java 类 app.Logonform 相关。这是一个 Struts ActionForm 类的子类。ActionForm 类提供标准的方法给框架使用，包括 validate 校验方法。

让我们先看看 LogonForm 的源代码再回过头看 LogonAction。

表格 3.4 Actionmapping 设置属性

属性	目的
path	这个属性是mapping的唯一标识。它可以包括在Web地址中，就象 *http://localhost:8080/logon*/LogonSubmit.*do*.

type	路径被请求时调用的Action 对象
name	和HTMl表单一起使用的JavaBean helper (ActionForm)类
scope	属性，标明是否将助手类存储在请求或者会话中的属性
validate	属性，标明form bean helper (由name属性标明的) 在调用 Action 对象 (由type属性标明)前是否调用标准的校验方法
input	属性，标明在校验方法返回False时，将控制发送到哪里

3.3.7. LogonForm 源代码

虽然 HTML 给用户一个地方来输入数据，但却没有给应用一个地方来放置数据。

当用户点击提交时，浏览器收集表单中的数据，并按一个名-值对列表的方式发送给服务器。所以，如果用户在 logon 页面输入 username 和 passwaord 并点击提交，应用所看见的是：

```
username=Ted
password=LetMeIn
```

浏览器将所有的东西都按字符串提交。你可以使用JavaScript校验来强迫用户在某个域里面只能输入数字，或者使用固定的数据格式，但是这也仅是镜花水月。所有的东西仍然以字符串的方式提交给服务器—而不象准备传递给Java方法的二进制对象。

重要的是要记住，这是浏览器和 HTML 工作的方式。

Web 应用无法控制这些。Struts 之类的框架的存在是使我们必须做的事情做的最好。Struts 对 HTTP 数据输入难题的解决方法是使用 ActionForm。

在象 Swing 之类的环境中，数据输入控件有一个内建的文本缓冲区，可以校验所输入的字符。当用户离开控件，缓冲区可以转换为二进制类型，可以传递给业务层。

不幸的是，HTTP/HTML 平台不提供可以缓冲、校验和输入转换的组件。所以 Struts 框架提供了一个 ActionForm (`org.apache.struts.action.ActionForm`)类来沟通 web 浏览器和业务对象。ActionForm 提供了想要的缓冲/校验/转换机制，我们可以用来保证用户输入它们想要输入的东西。

当 HTML 表单提交时，名-值对被 Struts 控制器获取，并应用到 ActionForm。ActionForm 是一个 JavaBean，有属性和 HTML 表单控件中的域相对应。Struts 比较 ActionForm 属性的名称和输入名-值对的名称。当匹配时，控制器设置属性值为相关的输入域的值。其它的属性会被忽略。错过的属性会保持它们的缺省值（通常是 null 或者 false)。

这里是 LogonForm 的公共属性:

```
// ------------------------------------------------- Properties
    /**
     * Return the password.
     */
    public String getPassword() {
        return (this.password);
```

```
    }

    /**
     * Set the password.
     *
     * @param password The new password
     */
    public void setPassword(String password) {
        this.password = password;
    }

    /**
     * Return the username.
     */
    public String getUsername() {
         return (this.username);
    }

    /**
     * Set the username.
     *
     * @param username The new username
     */
    public void setUsername(String username) {
        this.username = username;
    }
```

大部分 Struts ActionForm 的属性看起来就象如此。节约的开发人员可以当作一个宏来创建它们的属性并简单地替换属性名称。其他人则可以使用这个代码骨架，并使用查找替换的方法进行改写。Struts 代码生成器可以通过解析 HTML 和 JSP 来自动生成 ActionForm。

注意 在 Struts 1.1中，如果你使用DynaActionForm或者向后映射ActionForm，创建 ActionForms更加简单。第5章有详细信息。

基本 ActionForm 也包括两个标准方法—`reset` 和 `validate`。

当使用 ActionForm 作为向导工作流的一部分时，`reset` 方法非常有用。 如果 mapping 设置为请求范围，这个方法就没必要实现。

当 mapping 设置为 `validate=true`, `validate` 方法就会在 formbean 从 HTTP 请求中组装完成后被调用。 `validate` 方法经常用于主要的外观校验。它仅仅检查数据 “看起来”正确，并且所有要求的域都提交上来。再说一下，这些都是 Swing 控件在数据传输到应用之前内部做的事情。

你也可以手工作这些检查，或者使用象 *ValidatorForm*的机制(第12章)，这种Form可以从配置文件中创建。

下面是 LogonForm 的 validate 方法。它检查是否两个域都输入了一些数据。应用有一些关于用户名和密码的长度之类的要求，都可以在这里进行检验。

```
    public ActionErrors validate(ActionMapping mapping,
                                 HttpServletRequest request) {
```

```
        ActionErrors errors = new ActionErrors();

        if ((username == null) || (username.length() < 1))
            errors.add("username",
                new ActionError("error.username.required"));

        if ((password == null) || (password.length() < 1))
            errors.add("password",
                new ActionError("error.password.required"));

        return errors;
    }
```

validate 返回的 ActionErrors 对象是另一个框架类。如果 validate 没有返回 null，控制器对象将在请求上下文中的一个关键字下存储 ActionErrors 对象。

<html:errors> 标记知道关键字，将会在它们存在时渲染出错误消息。或者如果错误信息不存在，就什么都不做。

记号 error.username.required 和 error.password.required 也都是关键字。它们用于从 Struts 消息资源文件中查找要显示的实际消息。每个场所可以有其自己的资源文件，这使得消息容易被本地化。

Struts 消息资源文件也是用名-值对格式。我们所用的消息条目是：

```
error.username.required=<li>Username is required</li>
error.password.required=<li>Password is required</li>
```

注意

在Struts 1.1 中有一个方法将标记保持在消息之外。一个新的 errors.prefix/error.suffix 特征可以用于表示包围每个消息的 和 标记。一个新的消息标记集也可用于代替原有的 <html:error>标记。message 标记使其保持在其属于的表现页面中变得相对容易。

参见第10章查看更详细的Struts JSP 标记说明。

使没有使用本地化， Struts 应用资源文件也将集所有的消息收集到一个单独的地方，在此你可以对它们进行修改和修订，而不用触及任何 Java 源代码。

3.3.8. LogonAction 源代码

收集数据项到 ActionForm 并且执行了一些初始化校验后，控制器将 FormBean 传递给 Mapping 标明的 Action 类。

Struts 架构期望你能用自己的 Java 类来完成大部分的请求处理。JSP 页面可以渲染结果，但 Action 则是获得结果的方式。

就象第 2 章中看到，这就是熟知的 MVC 或 Model 2 方式， Action 充当了一个请求分派器 dispatcher。

当对一个 Action 的请求被送到 Struts servlet，它通过调用 perform (或 execute)方法来

调用(分派) Action。

注意 在Struts 1.1中有另外一个替换方法，叫做execute。这个方法提供了更好的异常处理，其它都和 Struts 1.0的 perform 方法一样。我们在这章仍然调用 perform 方法，使代码可以运行在两种版本之下。本书中的其他应用都是基于 Struts 1.1，可使用更好的特性。

下面是 logonAction 的全部源代码：

```
package app;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;


/**
 * Implementation of <strong>Action</strong> that validates a user
logon.
 *
 * @author Craig R. McClanahan
 * @author Ted Husted
 * @version $Revision: 1.1.1.1 $ $Date: 2002/08/15 15:50:55 $
 */
public final class LogonAction extends Action {


    /**
     * Validate credentials with business tier.
     *
     * @param username The username credential
     * @param password The password credential
     * @returns true if credentials can be validated
     * @exception UserDirectoryException if cannot access directory
     */
public boolean isUserLogon(String username,
                           String password)
                           throws UserDirectoryException {
  return
      (UserDirectory.getInstance().isValidPassword(username,password
      ));
       // return true;
    }
```

```
    /**
     * Login the user.
     * The event is logged if the debug level is >= Constants.DEBUG.
     *
     * @param mapping The ActionMapping used to select this instance
     * @param actionForm The ActionForm bean for this request (if any)
     * @param request The HTTP request we are processing
     * @param response The HTTP response we are creating
     *
     * @exception IOException if an input/output error occurs
     * @exception ServletException if a servlet exception occurs
     */
    public ActionForward perform(ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response)
        throws IOException, ServletException {

        // Obtain username and password from web tier
        String username = ((LogonForm) form).getUsername();
        String password = ((LogonForm) form).getPassword();

        // Validate credentials with business tier
        boolean validated = false;
        try {

            validated = isUserLogon(username,password);
        }

        catch (UserDirectoryException ude) {
            // couldn't connect to user directory
            ActionErrors errors = new ActionErrors();
            errors.add(ActionErrors.GLOBAL_ERROR,
            new ActionError("error.logon.connect"));
            saveErrors(request,errors);
            // return to input page
            return (new ActionForward(mapping.getInput()));
        }

        if (!validated) {
            // credentials don't match
            ActionErrors errors = new ActionErrors();
            errors.add(ActionErrors.GLOBAL_ERROR,
            new ActionError("error.logon.invalid"));
            saveErrors(request,errors);
            // return to input page
            return (new ActionForward(mapping.getInput()));
        }

        // Save our logged-in user in the session,
        // because we use it again later.
        HttpSession session = request.getSession();
        session.setAttribute(Constants.USER_KEY, form);

        // Log this event, if appropriate
```

```
        if (servlet.getDebug() >= Constants.DEBUG) {
            StringBuffer message =
                new StringBuffer("LogonAction: User '");
            message.append(username);
            message.append("' logged on in session ");
            message.append(session.getId());
            servlet.log(message.toString());
        }

        // Return success
        return (mapping.findForward(Constants.SUCCESS));

    }

} // End LogonAction
```

清单 3.5 LogonAction 类的 Java 代码(/WEB-INF/src/java/app/LogonAction.java)

Action 位于 Struts 食物链的最顶层，所以会导入好些类。 我们将每个类在此说明，以便你可以知道他们来自于何处。

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;
```

如果我们比较懒，这个语句块可以写成：

```
import java.io.*;
import javax.servlet.*;
import org.apache.struts.action.*;
```

但是这恐怕是没什么好处的。像许多 Apache 产品一样，Struts 源代码遵循最好的实践经验并且是很全面的。我们在源代码中也要遵循这一套。虽然我们在这儿忽略了 JavaDoc，但我们的源代码和 Struts 源代码都是文档齐全的。

接下来，我们使用了一个 helper 方法来调用业务层方法。我们也可以将同一段代码放在 Action 的 `perform` 方法内，但是，我们强烈建议将通用业务方法和 Struts 控制器代码分开。

如果你加入了一行（业务代码），很快就会是三行，五行，…然后你的 Action 就象一个大球陷入泥浆 [Foote]。避免 “代码蔓延” 的方式就是在从 Action 调用之前，将业务层代码封装到 helper 方法中：

```
public boolean isUserLogon(String username,
                           String password)
                           throws UserDirectoryException {
  return
      (UserDirectory.getInstance().isValidPassword(username,password
      ));
       // return true;
    }
```

如我们在别处所说， Struts 1.1 使用新的 execute 方法来替代原来的 perform 方法，但是它们两个都可以工作。在这里我们使用 perform 方法，以使代码可以运行在两个版本之中：

```
    public ActionForward perform(ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response)

        throws IOException, ServletException {

  }
```

Action 的目的是将输出从 web 层带到业务层，即应用的其它部分生长之地。在这里，我们从 ActionForm (JavaBean helper)中获取 username 和 password 并经它们存为平面的 Strings:

```
 // Obtain username and password from web tier
        String username = ((LogonForm) form).getUsername();
        String password = ((LogonForm) form).getPassword();
```

然后我们就可以将 username 和 password Strings 传递给业务层功能，看是否通过校验。这儿，我们小心地将调用封装进一个单独的方法，并没有将代码直接放入 Action 的 perform 方法之中。这个调用是对类中另一个方法的调用，但也可以容易的对其它 Java 类的方法进行调用:

```
        // 在业务层校验用户凭证
        boolean validated = false;
        try {

            validated = isUserLogon(username,password);
        }

        catch (UserDirectoryException ude) {
            // 无法连接到用户目录
            ActionErrors errors = new ActionErrors();
            errors.add(ActionErrors.GLOBAL_ERROR,
            new ActionError("error.logon.connect"));
            saveErrors(request,errors);
            // 返回输入页面
            return (new ActionForward(mapping.getInput()));

        }
```

isUserLogon 方法的 API 说明，如果校验符合，它将返回 true，如果不符合则返回 false，以及如果它不知道结果 (比如，连接不到目录)，则抛出一个异常。如果异常发生，Action 就会捕获它，将这个事件转换成 ActionError，并将控制转发回 input 页面。

如果业务层返回说 logon 没有成功， Action 将送出错误消息，并将控制路由到输入页面。当 ActionForm 的 validate 方法失败时，也会做同样的事情。(第 3.3.7 节):

```
        if (!validated) {
            // 校验不匹配
            ActionErrors errors = new ActionErrors();
            errors.add(ActionErrors.GLOBAL_ERROR,
            new ActionError("error.logon.invalid"));
            saveErrors(request,errors);
            // 返回输入页面
            return (new ActionForward(mapping.getInput()));
         }
```

因为错误并不适合于一个具体的属性，我们在 ActionErrors.GLOBAL_ERROR 关键字下记录此错误，而不是一个属性名。为了指出 logon 本身无效，我们也在 validate 外定义了一个错误信息。在 Struts 应用资源中，它是：

```
error.logon.invalid=<li>Username/password combination is
invalid</li>
```

显示时，表现层用适当的信息代替 error.login.invalid 标记。如果有一个单独的信息针对用户的当前场所，则用户会收到一个消息的本地化版本。

如果业务层说登录成功，我们就会告诉浏览器保留用户的认证信用。Java Servlet 框架为此目的提供了一个用户会话。这儿我们存储用户的 logonForm 到会话上下文之中。

每个用户都有一个上下文，由 servlet 容器负责维护。那些在上下文中存储有 logonForm 的用户都是登录用户。否则，便是未登录用户：

```
 // 在会话上下文中记录登陆用户，因为我们随后会用到
        HttpSession session = request.getSession();
        session.setAttribute(Constants.USER_KEY, form);
```

这种策略称之为基于应用的安全。许多 Java web 应用都使用这个方法，因为它轻便且易于实现。其它身份认证方法也可以用在我们的应用中。

Struts 框架依靠容器的默认记录系统。这儿，我们仅在当 Web.xml 文件中 Servlet 的 debug level 设置为足够高时记录这个事件。

```
    // Log this event, if appropriate
        if (servlet.getDebug() >= Constants.DEBUG) {
            StringBuffer message =
                new StringBuffer("LogonAction: User '");
```

```
            message.append(username);
            message.append("' logged on in session ");
            message.append(session.getId());
            servlet.log(message.toString());
        }
```

在 web.xml 中设置为：

```
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
```

在生产应用中，你可以将 debug 设置为 0，这些项目将不会出现。如果要插入其它记录系统，开发者可以继承 Struts ActionServlet 类，并重写 log 方法。

注意 在 Struts 1.1中,通过支持Jakarta Commons Logging 组件 [ASF Commons]使用另一个注册包。

当所有的信息发出，所有的工作干完， `perform` 方法返回一个 ActionForward 给控制器 (ActionServlet)。这里，我们发送控制到 `success` forward:

```
 // Return success
        return (mapping.findForward(Constants.SUCCESS));
```

这个转发是在Struts 配置中定义:

```
            <forward
                name="success"
                path="/pages/Welcome.jsp" />
```

既然我们已经登录，表现页面就会应该有点变化。所以将显示一个登出连接。

3.3.9. LogoffAction 源代码

请看图 3-5，当用户登录进去后，欢迎页面是如何改变的。在 Welcome.jsp 中，<logic:present> 标签首先查看是否在会话上下文中被 LogonAction 放入一个 user bean：

```
<logic:present name="user">
```

然后置入一个 <html:link> 标记，引用 forward:

```
<html:link forward="logoff">Sign out</html:link>
```

在 Struts 配置中, logoff 是这样定义的：

```
<forward
name="logoff"
path="/logoff.do"/>
```

这里的路径引用到.do 动作。这应该是个相应的/logoff ActionMapping，也在 Struts 配置中定义:

```
<action
path="/logoff"
type="app.LogoffAction"/>
```

如你所见，/logoff 是个极其简单的 mapping; 它仅仅是将控制传递给 LogoffAction 类，没有其他特殊参数和设置。LogoffAction 类的工作也非常简单。只是将用户的 logonForm 对象从会话上下文移除。 如果没有 logonForm 在会话上下文中，用户就被认为是已经登出了。

我们来看看 LogoffAction 的源代码：

```
    public ActionForward perform(ActionMapping mapping,
         ActionForm form,
         HttpServletRequest request,
         HttpServletResponse response)
            throws IOException, ServletException {

      // Extract attributes we will need
      HttpSession session = request.getSession();
      LogonForm user = (LogonForm)
        session.getAttribute(Constants.USER_KEY);

      // Log this user logoff
      if (user != null) {

        if (servlet.getDebug() >= Constants.DEBUG) {
            StringBuffer message =
                new StringBuffer("LogoffAction: User '");
            message.append(user.getUsername());
            message.append("' logged off in session ");
            message.append(session.getId());
            servlet.log(message.toString());
        }
      }

      else {

        if (servlet.getDebug() >= Constants.DEBUG) {
            StringBuffer message =
                new StringBuffer("LogoffAction: User '");
            message.append(session.getId());
            servlet.log(message.toString());
        }
      }
```

```
        // Remove user login
        session.removeAttribute(Constants.USER_KEY);

        // Return success
        return (mapping.findForward(Constants.SUCCESS));


    }
```

清单 3.6 LogoffAction 的源代码

首先，我们获得 logon 对象。本应用的做法是存放用户的 logon 对象到会话的上下文，并以 Constants.USER_KEY 作为关键字，所以我们会看到：

```
        // Extract attributes we will need
        HttpSession session = request.getSession();
        LogonForm user = (LogonForm)
          session.getAttribute(Constants.USER_KEY);

        // Log this user logoff
        if (user != null) {

          if (servlet.getDebug() >= Constants.DEBUG) {
              StringBuffer message =
                  new StringBuffer("LogoffAction: User '");
              message.append(user.getUsername());
              message.append("' logged off in session ");
              message.append(session.getId());
              servlet.log(message.toString());
          }
        }
```

之前，如果 debug level 设为为高，我们记录了一些细节。

这是该类的核心行为。我们移除了存放在USER_KEY下面的对象，就意味着,用户登出了。

```
  // Remove user login
        session.removeAttribute(Constants.USER_KEY);
```

如果想要移除为用户存储在会话中的所有东西，可以简单的使用 invalidate 方法：

```
// Remove user login
         Session.invalidate();
```

但这也消灭了所有用户对象，比如用户场所 locale，它要用来进行本地化。

当登录操作完成后，返回到 welcome 页面:

```
// Return success
        return (mapping.findForward(Constants.SUCCESS));
```

在下一节，我们要看看应用是如何从头构造的。到此，我们已经讨论了页面和类的内部细节。现在我们将眼光放在 Struts 配置文件和源代码结构树。

3.4. 构造应用

我们已经详细探讨了应用的驱动，踢了踢轮胎，并检查了引擎盖下面的东西，准备上路了。我们已经知道了要做什么，并且怎样做，但是从那里开始构建的你应用呢？

在这一节，我们回到开头并向你展示可以如何从头至尾建立你的应用。因为我们已经有一个很好的关于我们想要应用干什么的印象，我们从一个实际的需求集开始。这样我们可以创建一个白板计划，包括那些明显的对象以及我们准备叫它们什么名称。然后我们就可以开始为对象编码，精化和扩展所作的计划。这种 计划/编码,再精化-计划/再精化-编码的方法称之为 “螺旋式” 方法。

当然，有其它方式来进行软件开发。其它方法也可以很好的用于 Struts。但本节的目的并不是节是软件开发方法论。我们想要的是展示，构架一个简单的 Struts 应用需要做些什么。所以，让我们开始上路喽...

3.4.1. 定义需求

Requirements are the effects that the computer is to exert in the problem domain, by virtue of the computer's programming

—*Practical Software Requirements*, by Benjamin L. Kovitz

既然我们已经有一个关于应用需要做些什么的充分理解，从一个需求集开始是一个很好的实践方法。下面的章节，我们将汲取最简单有用的需求集。

我们的简单需求文档包括 3 个主要部分：目标、需求、规约。

表格 3.5 需求文档的主要标题

标题	目的
目标	我们在问题领域中需要达到的结果
域需求	为达到目标，我们需要实现的东西
编程规约	我们要实现需求需要做的事情

目标I

- 允许有权限的用户向应用标识自己

领域需求

- 允许用户递交他们的身份信用信息(username 和password)
- 检验递交的信息是否有效

- 允许纠正无效的信用信息
- 身份凭证信息有效时，通知用户
- 允许校验有效的用户访问需要权限的特征
- 允许用户在需要时，使访问无效

程序规约

- 可以从标准的web 浏览器访问，可以使用或不使用JavaScript
- 为新访问者从welcome 页面提供登陆链接
- 允许在logon 页面输入用户信用（身份）信息
- 要求每个身份信息包含1-20个字符
- 要求输入username 和password
- 提交信息给业务层方法来进行校验
- 返回无效的身份信息给用户以便进行纠正
- 如果身份有效，使用户登入
- 登陆后，用用户名定制 welcome 页面
- 允许登陆的用户从页面登出

当然,这是一个非常简单的规约。许多规约要消耗大量的纸张，并用很多图，数据表，屏幕定义和问题域的详细描述来进行修饰。

3.4.2. 规划应用

根据手头的这些需求，我们可以开始勾画应用并规划要用哪些对象来实现这些程序规范。一个方法是列出规约的列表，以及有助于实现他们的组件清单。通常，团队做这种事情是在一个大白板上进行，作为一个初始项目会议的一部分。

注意	这个过程中非常重要的一点是要注意到，规约项目和实现它们的组件之间并不是严格的1：1对应关系。虽然规约和程序都是为了一个相同的目标，它们却是从不同的方面的接近目标。 所以，象这样的列表并不是一个“规范化”的东西。某些规约可能已不止一个组件的方式出现。

视图

实际上，大多数应用都是以一个故事板（情节串联板）开始的。JSP 定义应用的可见部分。

下面是表现层的大概要求。

表格 3.6 白板视图规划

规约	JSP页面
向新访问者提供从欢迎页面开始的登陆	Welcome.jsp

允许在登陆页面输入用户身份信息(username 和 password)	Welcome.jsp
返回无效的身份信息给用户纠正	Logon.jsp
当用户登陆进去时，根据用户名定制欢迎页面	Welcome.jsp
允许将校验的用户从welcome 页面登出	Welcome.jsp
可以从标准浏览器中访问，有或者没用 JavaScript	Logon.jsp; Welcome.jsp
导引用户到欢迎页面	index.jsp

请注意我们在规约中加了一个自定的规约。这是个小技巧，在 Struts 应用中使应用的 welcome 页面重定向到一个 Struts action。这可以尽可能将控制流纳入框架，以便有助于在应用增长时将变更最小化。

控制器

在一个强壮的分层应用中(见第 2 章), 所有对页面和数据的请求请求都传递给控制层。表 8 是我们的控制器(“前端控制器” [Go3])的需求。

表格 3.7 控制器规划

规约	**ActionForm**
允许在登录页面中输入身份信息 (username and password)	LogonForm
	Action
用业务层方法校验身份信息	LogonAction
向用户返回无效身份信息供纠正	LogonForm;LogonAction
如果身份信息有效，则登入用户	LogonAction
允许经过校验的用户从欢迎页面登出	LogoffAction
	ActionForwards
向新访问用户提供从欢迎页面的登录	Welcome， logon
允许经过校验的用户从欢迎页面登出	Logoff
	ActionMappings

提交身份信息给业务方法校验	LogonSubmit
	Utility
记录所有内部常数	Constants

请注意我们自己添加了一个规约 “记录所有内部常数”。这在多层应用中也非常重要，因为这些常数可以“松散绑定”。Java 编译器不能验证我们在 XML 配置 文件中使用的标记，所以必须仔细跟踪我们所使用的标记。

模型

我们对数据访问层仅有一个需求。

表格 3.8 model 规划

规约	方法接口
提交身份信息给业务方法校验	boolean isUserLogon(String username,String password);

3.4.3. 规划源代码树

现在有了一个基线应用计划，我们可以规划一下应用的代码结构。因为应用很简单，我们可以为页面使用一个单独的子目录，为 Java 类使用一个单独的包。

我们的结构如图 3-6：

注意

Struts希望在classpath中有一个应用资源束。logon 应用将它的属性文件方在其自己的资源束中。国际化应用可能具有多个属性文件。将他们放入包中可以方便进行组织。我们的构建文件（build file ）将这些文件拷贝到类文件夹，以便在运行时它们可以在classpath中找到。在进行某些资源改变的时候，请记住要重新构建应用。

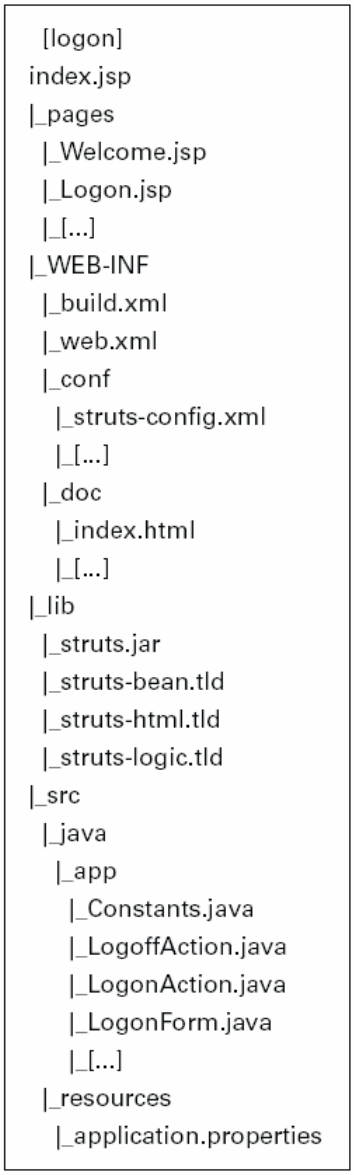


图 3-7 代码结构

如果你想跟随我们并建立了你自己的 logon 应用，跳到代码结构和 Struts classes 得好办法是部署一个空白的应用：

下载一个空白应用。

将 blank.war 拷贝为 logon.war。

将 WAR 放入容器的自动部署文件夹 (通常是 webapps).

这既是为什么要提供一个 Blank 应用。它们其实是一个通用的应用模板。我们将在第 3.4.4 到 3.4.8 节讨论基本结构。然后再来关注 logon 应用。

为了修改和重新部署应用，需要安装一些开发工具。

3.4.4. 设置开发工具

略

安装 Ant

略

安装jEdit

略

3.4.5. 设置 build.xml文件

象现今的其他一些 Java 产品，Struts 也希望使用 Jakarta Ant 工具 [ASF, Ant] 作为构建过程的一部分。Ant 也使用一个 XML 配置文件，名字叫 build.xml。 通常，你可以为你的应用设置一个固定的 build 文件，自始至终不变。在第 4 章，我们再表述 logon 应用的构建文件。

3.4.6. 设置web.xml文件

Java 2 Servlet 框架使用一个配置文件来帮助设置应用。

web 部署描述符， web.xml，标明需要的 servlets，以及其他应用设定参数。其格式在 servlet 规范[Sun, JST]有描述。大多数 Struts 应用仅需要部署一个单一的 servlet 和几个标记库，以便保持相对简单。我们在第 4 章表述 logon 应用的 web.xml 文件。

3.4.7. 设置 struts-config.xml 文件

非常象 web 部署描述符，Struts 也有一个 XML 配置 文件。你的应用在此注册其 ActionForm, ActionForward, 和 ActionMapping。每个类在文件中都有其自己的一个配置段，在这里你可以定义在启动时需要创建的缺省对象。下面是我们的起始 Struts 配置文件：

表格 3.9 配置文件

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration
1.0//EN"

"http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">

<struts-config>

<!-- ======================== Form Bean Definitions -->

    <form-beans>

        <form-bean
            name="logonForm"
            type="app.LogonForm" />

    </form-beans>


<!-- ========== Global Forward Definitions -->
```

```
    <global-forwards>
        <forward
            name="logoff"
            path="/Logoff.do"/>
        <forward
            name="logon"
            path="/Logon.do"/>
        <forward
            name="welcome"
            path="/Welcome.do"/>
    </global-forwards>


<!-- ================ Action Mapping Definitions -->

    <action-mappings>

        <action
            path="/Welcome"
            type="org.apache.struts.actions.ForwardAction"
            parameter="/pages/Welcome.jsp"/>

        <action
            path="/Logon"
            type="org.apache.struts.actions.ForwardAction"
            parameter="/pages/Logon.jsp"/>

        <action
            path="/LogonSubmit"
            type="app.LogonAction"
            name="logonForm"
            scope="request"
            validate="true"
            input="/pages/Logon.jsp">
            <forward
                name="success"
                path="/pages/Welcome.jsp"/>
        </action>

        <action
            path="/Logoff"
            type="app.LogoffAction">
            <forward
                name="success"
                path="/pages/Welcome.jsp"/>
        </action>

    </action-mappings>
</struts-config>
```

在你配置你的应用时，你可以从一个空白配置文件开始，就像这个一样，并一步步加入你需要的对象。我们将在本章余下的内容做这个事情，以便你可以看到是如何实际配置 Struts 配

置文件。第 4 章将更深入讨论 Struts 配置文件。你可能会注意到我们的起始并不是完全空白的。为方便已经提供了一个缺省 welcome forward 。

welcome action

通常，路由页面流要尽可能的通过 Struts 控制器。这样将应用总体设计都保持在 Struts 配置之中。这样你可以从一个单一的地方来调整应用的控制流。不幸的是，容器要求一个物理的 welcome 页面。在 web 部署 描述符 (web.xml) 中列出一个 Struts action URI 作为 welcome 页面是不允许的。

最好的方案是在一个根 index.jsp 文件中将控制重定向到你的 welcome action。struts-blank 就提供了这样一个根。这是一个极其简单页面仅有两行的页面：

```
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<logic:redirect forward="welcome"/>
```

空白应用提供 index.jsp 转发页面和一个缺省的 welcome 页面。我们将照原样使用 index.jsp 但是要对 welcome 页面做些改变。然而在开始之前，我们测试一下部署情况。

3.4.8. 测试部署情况

在测试一个新的应用之前确保一切 OK，应该打开一个运行的应用作为基线。Struts Blank 应用是一个基线应用的好选择。其缺省 welcome 页面包括了一些基本的系统检测，以看看配置文件是否被正确装入，标签扩展是否能找到，以及消息资源文件是否有效。

Struts Blank 应用的 WAR 文件可以在本书的站点或者 Struts 发布包中找到。将 blank.war 文件放入容器的自动部署文件夹并重新启动容器。然后可以使用这个 url 访问应用的 welcome 页面 ：

```
http://localhost:8080/blank
```

如果一切 OK，就会看到页面图 3-8 一样的页面：

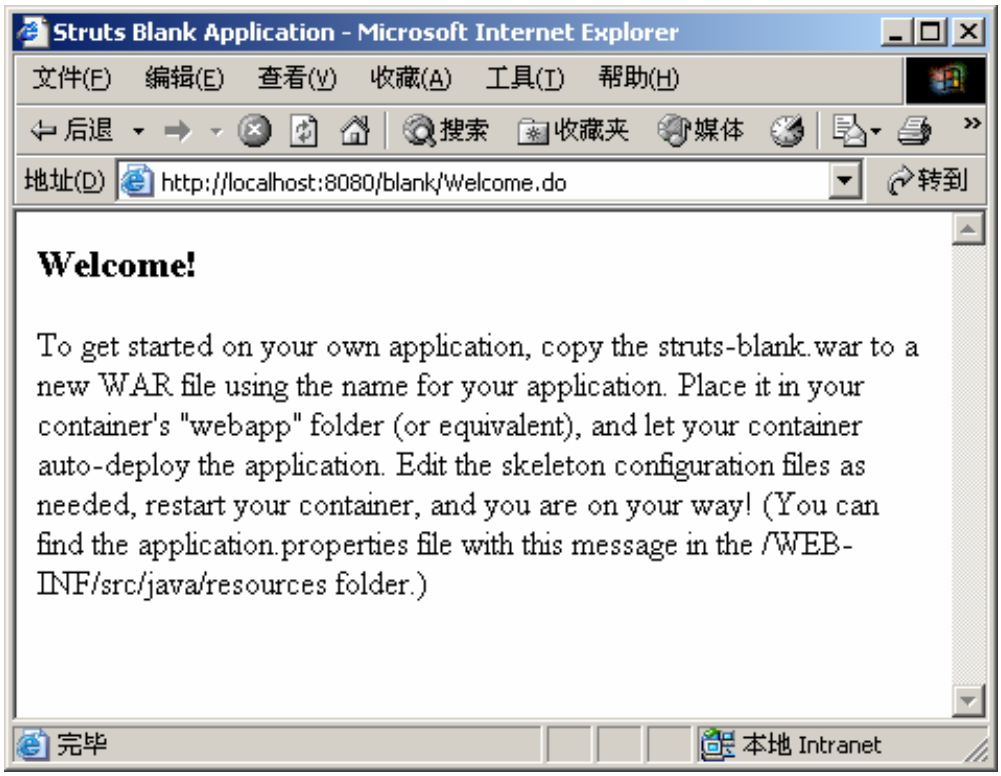


图 3-8 空白应用的欢迎页面

下面是它的源代码：

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<html:html locale="true">
<head>
<title><bean:message key="index.title"/></title>
<html:base/>
</head>
<body bgcolor="white">
<logic:notPresent name="org.apache.struts.action.MESSAGE"
scope="application">
  <font color="red">
    ERROR:  Application resources not loaded -- check servlet
container
    logs for error messages.
  </font>
```

```
</logic:notPresent>
<h3><bean:message key="index.heading"/></h3>
<p><bean:message key="index.message"/></p>
</body>
</html:html>
```

清单 3.7 空白应用的缺省欢迎页面源代码

3.4.9. 构造欢迎页面

大多数软件开发方法的一个基本原则是尽快的得到一个可以工作的原型。如果你同意这点。那么我们应该做的第一件事就是根据我们的规约构造我们的 welcome 页面。这个 welcome 页面早期版本，没有逻辑条件，可能看起来像：

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<html>
  <head>
    <title>Welcome World!!</title>
    <html:base/>
  </head>
  <body>
    <ul>
      <li><html:link forward="logon">Sign
in</html:link></li>
    </ul>
    <img src='struts-power.gif' alt='Powered by Struts'>
  </body>
</html>
```

清单 3.8 欢迎页面的早期版本

因为这里引用到logon ActionForward，我们需要将它添加到我们的Struts配置中去。我们也可以改变缺省 welcome 页面从Index.jsp 到Welcome.jsp:

```
<global-forwards>
  <forward
     name="logon"
     path /Logon.do"/>
  <forward
     name="welcome"
```

```
    path /Welcome.do"/>
  <!-- ... -->
</global-forwards>
```

这时，我们可以重新启动容器装入新的 Struts 配置。

某些容器,象 Tomcat，让你重新装入一个单独的应用。

1.0 VS 1.1

在Struts 1.0中,有大量的管理Action，包括重新装入Struts 配置。在Struts 1.1被取消了，因为他们和支持多应用模块相冲突.

一旦新的配置装入，我们可以打开新的 welcome 页面:

```
http://localhost:8080/logon/
```

你将看到图 3-9 的屏幕。

图 3-9 登录前的欢迎屏幕

然而，如果你点击连接，你将无法前进，会得到图 3-10 的信息。

为修正这个错误，我们需要看看下一个目标，构造 logon 页面。

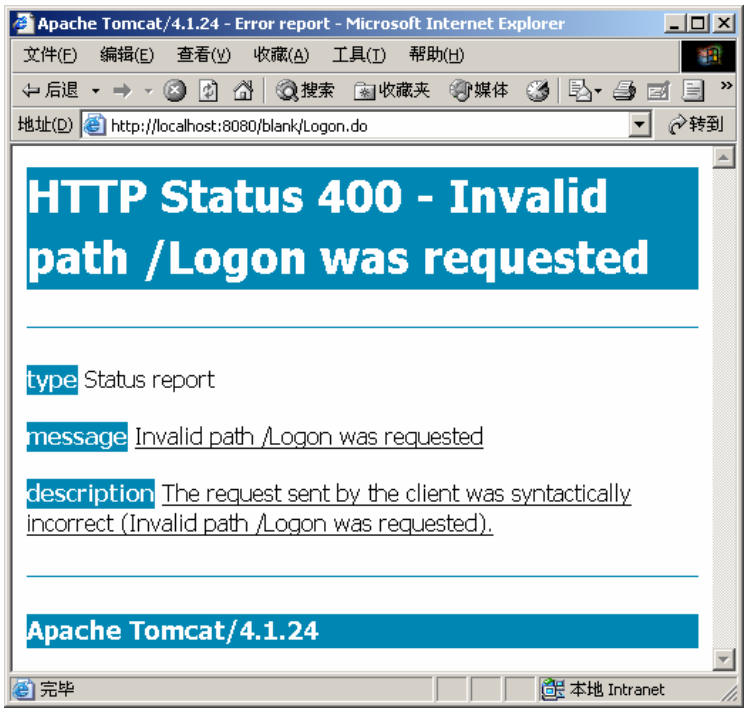


图 3-10 文件未找到错误

3.4.10. 构造logon 页面

回去看看 3.4.2 的白板，我们看到 logon 页面需要收集 username 和 password，并提交到一个名为/LogonSubmit 的 Mapping。这意味着我们需要创建一个 Struts form 来标明/LogonSubmit action，作为一个文本域和密码域的输入控件。下面是 logon.jsp 的源代码：

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<html><head><title>Sign in, Please!</title></head>
  <body>
    <html:errors/>
    <html:form action="/LogonSubmit" focus="username">
      <table border="0" width="100%">
        <tr>
          <th align="right">"Username: </th>
          <td align="left"><html:text
property="username"/></td>
        </tr>
        <tr><th align="right">Password: </th>
          <td align="left"><html:password
property="password"/></td>
        </tr>
        <tr>
```

```
        <td align="right"><html:submit property="submit"
          value="Submit"/></td
         <td align="left"><html:reset/></td>
       </tr>
     </table>
   </html:form>
  </body>
</html>
```

清单 3.9 Logon 页面的源代码

<html:form> 标签记引用一个 ActionMapping 对象，它再引用其他对象(org.apache.struts.action.ActionMapping)。我们先配置ActionMapping，然后是它使用的对象:

```
<action-mappings
  <action
    path="/LogonSubmit"
    name="logonForm"
    scope="Request"
    validate="true"
    input="/pages/Logon.jsp"/>
<!-- ... -->
</action-mappings>
```

两个相关的对象是 logonForm form bean 和 LogonAction。我们也需要在 Struts 配置中注册 ActionForm Bean。我们使用的名称成为对象在请求上下文中创建时的缺省属性名称。

```
<form-beans
    <form-bean
       name="logonForm"
       type="app.LogonForm"/>
     <!-- ... -->
</form-beans
```

这时我们需要加入两个特别的 Java 类, LogonForm 和 LogonAction。

3.4.11. 构造Constants类

虽然不是严格要求,但是将 ActionForward 名称和其他标记记录在文档中则是强烈推荐的。这其实很简单，而且可以使你的代码易于管理。我们展示代码时，通常忽略了 JavaDoc 说明。但是，在这里我们将留下他们。为什么？因为整个类都是记录常数的类。所以，在这里，

文档就是代码。下面是整个类的代码：

```
package app;
public final class Constants {
/**
* The session scope attribute under which the Username
* for the currently logged in user is stored.
*/
public static final String USER_KEY = "user";
/**
* The value to indicate debug logging.
*/
public static final int DEBUG = 1;
/**
* The value to indicate normal logging.
*/
public static final int NORMAL = 0;
/**
* The token that represents a nominal outcome
* in an ActionForward.
*/
public static final String SUCCESS= "success";
/**
* The token that represents the logon activity
* in an ActionForward.
*/
public static final String LOGON = "logon";
/**
* The token that represents the welcome activity
* in an ActionForward.
*/
public static final String WELCOME = "welcome";
}
```

清单 3.10 Constants 类的源代码

3.4.12. 构造其他类

我们在 3.3.8 和 3.3.9 展示了 logonForm 和 logonAction 的源代码。我们也需要在第 1 章中讲述过的 UserDirectory 和 UserDirectoryException 类。我们可以将他们不经改变的加入到我们的新应用中。在我们的源代码结构中将他们放在/WEB-INF/src/java/app/。

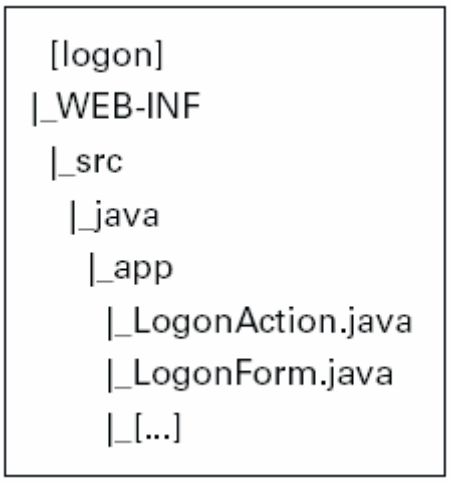


图 3-11 LogonAction,LogonForrm 以及其它 Java 类的位置

LogonAction 也引用了 Constants 类。在编译前我们要先加入它。

3.4.13. 创建user directory

在第 1 章，我们引入了一个简单的 registration 应用来存储 user ID 和 password。这些登陆账号纪录在一个标准的属性文件中，名为 user.properties。它可以从那个应用中引入，或者在 WEB-INF/src/java/resources 下重新创建，如图 3-12：

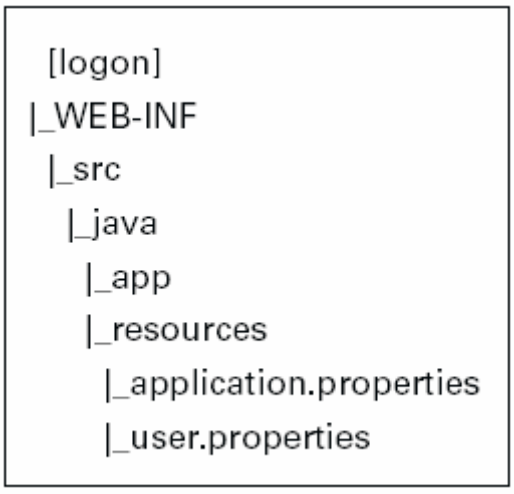


图 3-12 用户属性文件的位置

属性文件是一个简单的文本文件。这里是一个简单的例子，使用本书作者的名为用户名，姓为密码。

```
TED=Husted
CEDRIC=Dumoulin
GEORGE=Franciscus
DAVID=Winterfeldt
CRAIG=McClanahan
```

如果你喜欢，你可以只输入这些，或者你愿意输入的内容，并将他们存放在/WEB-INF/src/java/resources/user.properties。只是保证 user ID 全部大写，因为这是业务逻辑中要求的。

当然，你的应用也可以容易的根据一个 JNDI 服务，或者数据库来进行校验，或者使用容器的安全领域。我们在第 14 章说明在 Struts 中使用数据服务。

3.4.14. 配置ActionErrors

可能记得，LogonForm 和 LogonAction 都要产生错误信息。ActionError 系统是集成在应用消息之中的。

在测试 LogonForm 和 LogonAction 之前，我们需要添加这些消息到 Application.properties 文档中：

```
errors.header=<H3><font color="red">Validation
Error</font></H3>You must correct the following error(s)
before proceeding:<UL>
errors.footer=</UL><HR>
error.username.required=<LI>Username is required</LI>
error.password.required=<LI>Password is required</LI>
error.logon.invalid=<LI>Username and password provided not
found in user
directory. Password must match exactly, including any lower
or upper case
characters.</LI>
```

1.0 VS 1.1	Struts 1.1中的新标记允许从消息中忽略标记。见第10章的详细内容。

作为构建过程的一部分，我们将应用资源文档从/WEB-INF/src/java/resource 拷贝到 classes 文件夹下的资源束，这里 ActionServlet 才能找到它们。

3.4.15. 编译并测试logon 页面

在 3.4.8 中，我们为 logonForm 创建了 JSP 页面。但为了使他们运行，我们得添加相应的配置元素和 JAVA 类，如表 10。

表格 3.10 Logon 页面配置元素

配置元素	Java 类
LogonSubmit action-mapping	*LogonAction*, subclass of Action
logonForm form-bean	*LogonForm*, subclass of ActionForm

现在我们可以编译应用并测试 logon 页面了。有一个存好的 build.xml 文件放在 WEB-INF 目录，你可以在 Ant 使用它。

按缺省方式构建目标，编译它，将从 java 源代码构建 Java 类，并将应用资源消息文件拷贝到 classes 目录。

当构建成功，你可以使用下面的链接进到应用中的 logon 页面。 (根据你的容器如何重新装入 Java 类，你可能需要重新启动容器。如果不确定，就重启它。)

logon 应用应该可以工作得象我们原来设想的一样 (从 3.3.3 到 3.3.6)。所不同的是 welcome 页面在用户登录后不会发生变化，或者不提供登出的机会。下面我们将修正它，然后我们的应用就完成了。

3.4.16. 修改welcome页面

我们先前的 welcome 页面忽略关于判断用户是否登陆的条件逻辑。既然，用户已经可以登陆，我们就可以将这些语句加入，以符合 3.3.2 中的版本 。黑体部分是我们加入的内容：

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic"%>

<HTML>
<HEAD>
<TITLE>Welcome!</TITLE>
<html:base />
</HEAD>
<BODY>
<logic:present name="user">
   <H3>Welcome <bean:write name="user" property="username"
/>!</H3>
</logic:present>
<logic:notPresent scope="session" name="user">
   <H3>Welcome World!</H3>
</logic:notPresent>
<html:errors />
<UL>
   <LI><html:link forward="logon">Sign in</html:link></LI>
   <logic:present name="user">
      <LI><html:link forward="logoff">Sign
out</html:link></LI>
   </logic:present>
</UL>
<IMG src='struts-power.gif' alt='Powered by Struts'>
</BODY>
</HTML>

<%--

If user is logged in, display "Welcome ${username}!"
Else display "Welcome World!"
```

```
Display link to log in page; maintain session id if needed.
If user is logged in, display a link to the sign-out page.

Note: Only the minimum required html or Struts custom tags
are used in this example.

--%>
```

清单 3.11 welcome 页面 (/pages/Welcome.jsp)的修订版本

如图 3-13, 这使我们回到了原来的情形。当用户到来时，他被邀请登录进去。一旦登录进去，则按名字欢迎它并提供登出的机会。这是你的进步！

3.4.17. Struts ForwardAction Action

如果你注意到你浏览器的地址栏，你可能会注意到我们一直没有暴露 JSP 页面的地址。 许多应用可能不注意这个方式，但如果你想要严格的 MVC 架构(见第 2 章), 你可能不希望暴露一些关于你的视图的实现，包括你是否使用 JSP 页面或者你如和存储它们。理想情况下，所有的导航都应该通过.do Action 进行，因为他们都通过控制器管理。

当然，并不是所有的事情控制器都能干的很好。这也是为何使用 logon 和 welcome 页面的原因。它们并不需要任何来自于 model 的信息，直接提供显示 JSP 页面的连接。

但这也使用户可以将页面的位置存为书签。接下来，你可能在显示 logon 页面之前需要执行一些背景动作，或者你可能需要移动或者将 JSP 页面改名。如果用户将这些页面存为书签了,他们就会回到旧的地址，绕开你的逻辑，并返回一个 File not found 错误。在实践中，通常导致将遗留检查放入服务器页面并重定向到 web server—事情发生错误的方式更多了，并有更多代码需要维护。

道德？我们必须尽可能虚拟化详细的导航。否则，我们就会不断的为浏览器是否进行缓存或者存储进行补偿。不要直接连接到物理的 JSP, 我们应该总是连接到一个虚拟的 Struts Action, 然后由它提供相应的页面。

当然，不管是否需要，为每个页面编写一个 Action, 会是一个非常繁琐的工作。一个更有效的办法是，部署一个有用的 Action，它可以在 Struts 配置中定制，并在需要的地方可以重用。因为 Struts 为每个 Action 创建一个多线程实例，这是一个非常有效的办法，确保控制保留在控制器之中。我们需要做的就是将 Action 和 path 传递给页面。

高兴的是，你可以使用 struts.jar 中的标准的 ForwardAction 类。你可以简单地将目标路径作为 ActionMappin 的 parameter 属性传递给 ForwardAction:

```
<action
  path=" /Welcome"
  type="org.apache.struts.action.ForwardAction"
  parameter="/pages/Welcome.jsp"/>
```

ForwardAction 将会从 mapping 中查找目标路径，并使用它来返回一个 ActionForward 给 servlet。

实际结果是 `http://localhost:8080/logon/pages/welcome.jsp` 不会出现在浏览

器的地址栏中，那样的话就可以被保存为书签直接访问，浏览器中的 Action URI 会出现:

```
http://localhost:8080/logon/Welcome.do
```

用户仍然可以把这个地址记为书签，但是现在你有更多的控制了。你可以修改 logon 活动的具体实现而不用担心有什么被浏览器记为书签了。之前，你可能不得不考虑如果它们直接访问原来的服务器地址的话，会发生什么事情。

在 MVC 架构中，action 就是你的 API。服务器页面是一个实现细节。如果 JSP 作为导航系统的一部分暴露给了浏览器，控制器和视图层就混合在一起了，那么 MVC 的优势就被削弱。

在需要直接导航至其他页面时，我们也可以加入其他 ForwardAction 实例。因为应用系统实际上仅仅实例化一个 ForwardAction，我们所添加的实际上是一个 ActionMapping 对象。如果要求使用 ForwardAction 通常的 MVC 架构原因还不充分，Struts 1.1 引入的模块化特征要求所有的 JSP 请求都要通过 Action。这是因为每个模块有其自己的配置上下文并且控制必须通过 ActionServlet 控制器传递，以便为 JSP 页面选择配置。当然，如果你使用单个的缺省应用，这并不是必须的。但是如果从一开始便遵循这个实践，那么你便可以在不改变任何东西的情况下将你的应用变成一个模块。

3.5. 小结

不管你在开发团队中担任何种角色——工程师，设计员，架构师，QA——对整个应用的总体运行有个总体印象总是有帮助的。在这一章，我们综合的解剖了一个小而有用的应用。通过漫游相关场景，对其解剖，然后构造一个 logon 应用，我们向你展示了 Struts 框架实际上是如何工作的。作为构造阶段的一部分，我们还创建了一个设计文档，来规划我们的目标、客户需求和程序规约。要使一个应用能够运行，我们还构建了应用的 web.xml，Ant 的 build.xml 脚本，以及 Struts 配置文件。

为了使正确的构建部件到位，我们按需要的顺序构建了每个组件。 在此过程中，我们也指出了最好的实践方法，并强调分离模型、视图和控制器的重要性。

在第 4 章，我们会详细探讨 Struts 配置文件。如我们在此所见，配置扮演了一个强有力的角色，使应用易于设计和维护。

4.配置 Struts 组件

本章内容

- *Web 应用部署描述符*
- *Struts 配置文件*
- *应用资源文件*
- *Ant 的构建文件*

Change alone is unchanging.
—Heraclitus (c 535–c 475 B.C.)

4.1. 三个 XML文件和一个属性文件

除了 Java 类和 JSP 页面之外，开发人员必须创建或者修改几个配置文件以便能够使 Struts 应用能运转起来，这些文件包括：

- web.xml. 这是 Java Servlet 要求的 web 应用部署描述符。Servlet/JSP 容器使用这个文件来载入和配置你的应用。
- struts-config.xml. Struts 框架的部署描述符。它用来载入和配置 Struts 框架使用的各种组件。
- Build.xml. Jakarta Ant 构建工具使用它来编译和部署你的应用。使用 Ant 不是必需的，但它在 Struts 开发人员中很流行。
- Application.properties. 该文件为你的 Struts 应用提供资源。像 build.xml 文件一样，它不是严格要求的，但是大多数 Struts 应用都要用到。

尽管处理这些文件看起来也许不象是在进行“Java 开发”，但是正确的使用它们却是使你的 web 应用能拿得出手的基本要求。在这一章，我们会仔细讨论这些文件的工作原理，以及它们能对你的应用的开发和部署起什么作用。

4.1.1. 家族的其他人员

除了每个 Struts 应用都需要的配置文件外，还有其它一些 Struts 应用也可能要用的东西。如果使用可选的组件，可能还需要另外的 XML 配置文件，比如 Tiles 框架和 Struts Validator 。如果你想要把你的应用分成多个模块，每个模块也要有其自己的 Struts 配置和资源文件。

在本章，我们将首先关注核心配置文件，然后来配置 Struts 1.1 中的标准配置项。

1.0 vs 1.1 在本书写作时， Struts 1.1 beta release 2 还在开发之中。在最终发布版中有些细节可能会改变。请参考本书的刊误站点。

4.2. Web应用部署描述符

框架的核心是 ActionServlet，Struts 把它当作是一个控制器。虽然它也可以被子类化，但大多数开发人员都将它看成是一个黑盒。他们总是在 web 应用部署描述符 (web.xml) 中配置它，然后让它自己工作。

ActionServlet 可以接受多个初始化参数。大多数都有合理的缺省值，不需要重新设定。但有一些却必须设置，以便使你的应用能正常工作。

在这一节，我们将考察一个典型的 Struts web 部署描述符，并且详细讨论 ActionServlet 的初始化参数。

4.2.1. Web.xml 文件

Web 应用部署描述符的目的和格式在 Sun Servlet 规范[Sun, JST]中定义。基本上，它告诉 servlet 容器如何配置 servlet 和应用需要的其它高层次对象。

Struts 框架有两个组件需要从应用部署描述符中配置：ActionServlet 和标签库（可选）。虽然大多数 Struts 应用的确需要使用标签库，但它也不是严格要求的。使用 XSLT 或者 Velocity 的应用根本不需要配置标签库。

清单 4.1 是我们的 logon 应用的 web.xml 文件(见第 3 章)。

```
<!—① -->
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
    <!—② -->
    <servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>
          org.apache.struts.action.ActionServlet
        </servlet-class>
        <init-param>
            <param-name>application</param-name>
            <param-value>Application</param-value>
        </init-param>
        <init-param>
            <param-name>config</param-name>

    <param-value>/WEB-INF/conf/struts-config.xml</param-value>
        </init-param>
        <init-param>
            <param-name>debug</param-name>
            <param-value>2</param-value>
        </init-param>
        <init-param>
            <param-name>detail</param-name>
            <param-value>2</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <!—③-->
    <servlet-mapping>
```

```
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
    <!—④-->
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <!—⑤-->
    <taglib>
        <taglib-uri>/tags/struts-bean</taglib-uri>

    <taglib-location>/WEB-INF/lib/struts-bean.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>/tags/struts-html</taglib-uri>

    <taglib-location>/WEB-INF/lib/struts-html.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>/tags/struts-logic</taglib-uri>

    <taglib-location>/WEB-INF/lib/struts-logic.tld</taglib-location>
    </taglib>
</web-app>
```

清单 4.1 Logon 应用的 Web.XML

代码中编号的地方对应下面的注释。

①标识为 *web* 应用部署描述符——前两行将文件标识为一个 web 应用部署描述符。

②配置 *ActionServlet*— 这一段告诉容器装入 action 名称下的 ActionServlet 。有四个参数传递给 ActionServlet: 即 application,config, debug, 和 detail。(ActionServlet 也可以接受其它参数；我们将再下一节涉及)。这一段的最后一个设定，<load-on-startup>，给容器一个 action servlet 的权重。如果设置为 2，则允许其它 servlet 在需要时首先装入。这将在你子类化了 ActionServlet 之时显得很重要，以便你可以使用其它 servlet 先期装入的资源。对一个应用来说， 仅可以装入一个 ActionServlet 或者 ActionServlet 的子类。ActionServlet 设计为可以和应用中的其它组件共享资源。装入多个 ActionServlet 会造成冲突 ;因为一个 ActionServlet 可能会改写另一个 ActionServlet 提交的资源。Struts 1.1 支持模块化应用，但仍然只允许装入一个 ActionServlet 。

③标识 *Struts* 请求—这一段告诉容器将匹配*.do 格式的文件请求转发到 action servlet 。这就是我们在①中配置的 ActionServlet 。不匹配这种格式的文件请求将不被 Struts 处理。比如对*.html 或 *.jsp 文件的请求通常由容器内建的服务来处理。

④创建 *welcome* 文件—不幸的是，在这里设置一个 index.do 文件将不会工作。容器希望

welcome 文件也是一个物理文件。在第 3 章，我们展示了如何使用 welcome 文件来转发到一个 Struts action 。

⑤配置标签库—这里我们配置应用中使用的标签库。3 个核心的 Struts 标签库—bean, html, 和 logic— 将可用在大多数应用中。如果你的应用中使用了其它的标签库，它们也在此进行配置。第一个元素，<taglib-uri>，给出标签库的逻辑名称。这通常看起来像是一个文件路径， 但其实不是。JSP 在导入标签库时将引用这个 URI 。第 2 个元素，<taglib-location>，提供提供了一个上下文相关的标签库描述符 (*.tld)路径。TLD 标识了库的实际类型(Java 类)。当需要这个库时，容器会搜索标签库类文件的 classpath 。对 Struts 标签库来说，容器将在 struts.jar 文件中找到这些包。

关于 web 应用部署描述符的更多细节，请参见 Java Servlet 规范 [Sun, JST] 以及书籍 *Web Development with JavaServer Pages*[Fields] 。

4.2.2. ActionServlet 的参数

Struts ActionServlet 可接受多个参数，这些参数都总结在表 4.1 中。大多数参数在 Struts 1.1 中都不赞成了，这样有利于提供模块化支持的新的配置包中的组件。

表格 4.1 ActionServlet 参数

参数	缺省值	说明	备注
config	/WEB-INF/strutsconfig.xml	包含配置信息的 XML 文件的上下文相关路径	
config/${prefix}		使用指定的的前缀的应用模块的 XML 配置文件的上下文相关路径。在多模块应用中可以根据需要重复多次	1．1 以后
convertNull	false	一个参数，在组装表单时强制模拟 Struts 1.0 行为。如果设置为 true，数字的 Java 包装类类型 (如 java.lang.Integer) 将缺省为 null(而不是 0).	1．1 以后
debug	0	调试的详细级别，控制针对这个 servlet 将记录多少信息。接受的值为 0 (off) 和 1 (最不严格)直到 6 (最严格)。大多数 Struts 组件设置为级别 0 或者 2	
detail	0	用来处理应用配置文件的 Digester 的调试详细级别。. 接受值为 0 (off) 和 1(l 最不严格)到 6 (最严格).	

validating	true	标识是否使用一个检验 XML 的解析器来处理配置文件(强烈推荐)。	
application	无	应用资源束的名称,风格像是一个类名称。引用到位于名为 resources 的包中的一个名为 application.properties 的文件，这里使用 resources.application 。这种情况下，资源可以是 classes 下的子目录(或者 JAR 文件中的一个包)。	不推荐； 推荐使用 <messageresources> 元素的 parameter 属性进行配置
bufferSize	4096	处理文件上传时输入文件缓冲区的大小	不推荐； 推荐使用 <controller>元素的 buffer-Size 属性配置
content	text/html	每个响应的缺省内容类型和字符编码 ;可以被转发到的 servlet 或者 JSP 重写。	不推荐；使用 <controller> 元素的 contentType 属性配置
factory	org.apache.struts.util.propertyMessageResourcesFactory	MessageResourcesFactory 用来创建应用消息资源对象的类名	不推荐；使用 <messageresources> 元素的 factory 属性配置
formBean	org.apache.struts.action.ActionFormBean	ActionFormBean 实现使用的 Java 类名称	不推荐；使用每个 <form-bean> 元素的 class-Name 属性配置
forward	org.apache.struts.action.ActionForward	ActionForward 实现使用的 Java 类名。	不推荐；使用每个 <forward> 元素的 className 属性配置
locale	true	如果设置为 true，并且存在一个用户会话,在用户会话中存储一个合适的 java.util.Locale 对象(在 Action.LOCALE_KEY 标识的标准关键字下) (如果还没有 Locale 对象存在的情况下).	不推荐；使用 <controller> 元素的 locale 属性配置

mapping	org.apache.struts.action.ActionMapping	ActionMapping 实现使用的 Java 类名	不推荐；使用每个<action>元素的 className 属性配置，或者使用模块应用的<action-mappings>元素的 type 属性配置
maxFileSize	250M	文件上传时可以接收的最大文件尺寸（Byte）。可以表示为"K", "M", "G"。分别解释为 kilobytes,megabytes, 或者 gigabytes,	不推荐，使用<controller>元素的 maxFileSize 属性配置
multipartClass	org.apache.struts.upload.DiskMultipartRequestHandler	MultiPartRequestHandler 实现 l 类的全限定名称，用来处理文件上传。如果没有设置，禁止 Struts 多部分请求处理	
nocache	false	如果设置为 true，将在每个响应前加上 HTTP 头。这样可以使浏览器对我们产生和转发的响应 的缓存失效	不推荐；使用<controller>元素的 nocache 属性设置
null	True	如果设置为 true，那么如果使用 了未知的消息关键字，应用资源将返回 null。否则，将返回一个 包含不愉快信息的错误信息	不推荐；使用 <message-resources>元素的 null 属性配置
tempDir	提供给 web 应用作为 servlet 上下文属性的工作目录	处理文件上传时的工作目录	不推荐；使用<controller>元素的 tempDir 属性配置

4.3. Struts配置

Struts 配置文件 (struts-config.xml)用来装入多个关键的框架组件。这些对象一起构成了 Struts 配置。

Struts 配置和 Struts ActionServlet 一起工作，创建应用的控制层。在这一节，我们将研究

为什么需要 Struts 配置。下一节，我们将讨论 Struts 开发人员要如何来创建和维护配置。

注 从 Struts 1.1 开始，一个应用可以分成多个模块模块。每个模块都有其自己的 Struts 配置。每个 Struts 应用至少有一个缺省，或成为“根”模块。如果你没有使用多模块机制，或者正使用 Struts 1.0，那么当我们说模块时，你可以将它想象成应用。 我们在本章的末尾讨论 Struts 1.1 应用于多模块的配置。

4.3.1. 细节，更多细节

Struts 配置是你的应用的真实蓝图。它知道表单中有什么字段。它知道哪里可以找到 JSP 文件。它也知道应用执行的每一个 Action ，以及每个 action 需要的实际资源。

这看起来好像是把许多信息集中在了一个地方。实际上就是。但是通过将这些实现细节放在一起，许多开发人员会发现他们的应用更加易于创建和维护。

Struts 配置中的每个组件都是 Java 对象。ActionForm 对象知道字段和表单。ActionForward 对象知道何处可以找到 JSP 或其它资源。ActionMapping 则对象知道那个表单和转发将用于应用能理解的哪一个命令。

一个非常简单的应用可以在一个初始化内创建所有这些信息对象，然后设置需要的缺省值。例如:

```
ActionForwards globals = new ActionForwards();
ActionForward logoff = new ActionForward();
logoff.setName("logoff");
logoff.setPath("/Logoff.do");
globals.addForward (logoff);
ActionForward logon = new ActionForward();
logoff.setName("logon");
logoff.setPath("/Logon.do");
globals.addForward (logon);
```

但是，在实践应用中，初始化方法很快就会成为维护负担，并造成许多问题。具有讽刺意味的是，像这样的类并不涉及到多少编程问题。它只是从现有的类中实例化某个对象。它几乎用不着位于 Java 代码中。事实上，它也不该位于代码中。Java 语言可以通过名称来创建一些给定的类。Java 也支持一些诸如可以决定一个类在运行时支持哪些方法的反射特征。

定义 反射 告诉我们 **Java** 类提供什么方法。自省（**Introspection** ）则帮助我们推断出这些方法哪些是可以在运行时用来配置 **JavaBean** 的属性。**Java** 工具和框架 **(**如 **Struts)**使用反射和自省来自动化装入和配置 **JavaBean** 对象。这样就消除了那些因为粗心易导致错误的编码工作，以及装入仅仅为了装入其他对象的简单对象时的任务。

将这些特征结合在一起，其实并不需要一个 Java 类。你需要的是一个文档来描述如何实例化一个 Java 类使之成为一个全功能的对象。

当然，象 Struts 这样的框架并不是唯一具有这个问题的东西。Servlet 容器基于同一原因也需要同样的东西。开发人员不得不告诉容器应用中需要什么 servlet 以及其它一些对象。

他们不是编写一个 Java 类并插入到容器之中，Sun 的工程师却是选择了使用一个 XML 文

档来配置的方式。容器读入这个文档并使用它来实例化和配置应用需要的 servlet。

Struts 配置文件对 Struts 来说就像部署描述符对容器一样。Struts 控制器读入配置文件并使用它来创建和配置框架需要的那些对象。

每个编写过部署描述符 (web.xml) 文件的 web 开发人员都应该使用过 XML 元素来创建一个 Java 对象。例如，我们可以在 web.xml 中部署一个 servlet:

```
    <servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>
          org.apache.struts.action.ActionServlet
        </servlet-class>
        <init-param>
            <param-name>application</param-name>
            <param-value>Application</param-value>
        </init-param>
        <init-param>
            <param-name>config</param-name>
            <param-value>/WEB-INF/conf/struts-config.xml</param-value>
        </init-param>
        <init-param>
            <param-name>debug</param-name>
            <param-value>2</param-value>
        </init-param>
        <init-param>
            <param-name>detail</param-name>
            <param-value>2</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>

    </servlet>
```

而下面则是我们在前面的 Struts 配置文件中部署一个 forward 对象的代码片段：

```
<!-- 全局转发 -->
    <global-forwards>
        <forward name="logoff" path="/Logoff.do" />
        <forward name="logon" path="/Logon.do" />
    </global-forwards>
```

事实上， struts 配置并不是配置应用而是部署它。但大多数开发人员，还是很自然的将它这些对象视为是“配置,” 所以我们还是使用这个词汇。

4.3.2. 变更管理

按这种方式部署预配置的 Java 对象是个强大的特征。当然，强大的功能也带来更大的责任。因为 Struts 配置文件装入框架对象，所以它也对框架对象负责。

通过描述框架组件间如何交互，Struts 配置文件成为了一个管理应用变更的非常有效率的工具。实践中，配置文件要胜过一个简单的对象载入器，并被用作为动态的设计文档。

不难想象，可以用工具来读入 Struts 配置文件并用它来产生和创建一个 UML 图。现在有

好多 Struts GUI 都可以帮助你维护这个 XML (参见 4.4)。不久就会出现可视化的工具可以帮助你维护由 Struts 配置文件表达的架构设计[2]。

4.3.3. 受保护的变更原则

Struts 配置文件帮助你可以以最小的努力对应用变更作快速的反应。如果一个对象需要初始化为另一个值，你并不需要编辑，编译和部署一个 Java 类。许多配置细节都涉及到表现层。团队中工作于该层的人员可能不都是 Java 工程师。使用 XML 文档可以使配置被页面设计员和项目管理者都能访问到。需要 Java 工程师来创建和修改应用的基础对象，但配置这些对象却可以委派给其它人。实践中，我们常常将不经常的变更的东西—基础 Java 类—从经常变更的事物—Java 对象在运行的部署中分离开来。这就是受保护的变更原则（principle of *Protected Variation* [Larman]）.

定义

Protected Variation 受保护的变更是一个设计原则，它鼓励使用一个稳定的接口来封装变更的可以预知点。数据-驱动设计，服务查询，解释器驱动设计，反射设计都是这种机制的不同实现。

受保护的变更可以让我们记住一个的变更点至少可能产生一个的维护点。通常从基对象（不常改变）中分离出实现细节（经常改变），我们就可以减少维护应用要做的努力。

4.4. Struts配置元素

我们在第 4.3 节中所讨论过，Struts 配置文件是一个用来部署 Java 对象的 XML 文档。配置中的每个元素对应一个 Java 对象。当你在 Struts 配置文件中插入一个元素，你就是告诉 Struts 控制器在应用初始化时要创建一个 Java 对象。如果从一个有一些示例性注释的空白的配置文件开始，可以很容易得将它修改后为你的应用所用。但是如果你只是遵循一些通用示例的话，也可能容易丢掉一些重要的特征。

大多数 Java 开发人员都知道，如果他们需要关于 Java 类的更详细的信息时，他们会查找 JavaDoc。但是，你如何查找关于一个 XML 文档的更多信息呢？

每一个良构的 XML 文档，包括 Struts 配置文件，都包括一个描述该文档可用元素的指针。这就是文档类型定义 (DTD)。如果你看看 struts-config.xml 文件的顶部，你就会发现这个元素: 这是告诉我们这个文档的 DTD 的官方参考版本可以从所指定的 URL 找到。在内部，Struts 使用来自于 Jakarta Commons[ASF, Commons] 项目的 Digester 来解析 Struts 配置文件。Digester 使用 struts-config DTD 来校验文档的格式，并且创建文档所描述的 Java 对象。如果 XML 文件包含了非正式文档化的元素，或者以非正式文档化的方式使用了元素，Digester 将不会处理这个文件。

---

[2] 译者注：这些常用工具包括：

1 各大公司的 IDE， 比如 JB, WSAD, Jdeveloper 等等

2 如果使用 Eclipse 平台，有很多选择可用，包括 ObjectWeb 的 lomboz, Myeclipse, IBM WTP，Eclipse WTP(源于 IBM WTP)，Nitrox M7, Exadel Studio 等等。

你尽可以根据自己的需要进行选择。

```
<!DOCTYPE struts-config PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
```

如果基于某些原因,XML 校验产生了问题,你可以使用 4.3 节所描述的 validating servlet 参数将校验特性关闭。但我们不推荐这样做。

在内部，Struts 使用其自己的 DTD 拷贝来处理配置。并不是每次在你的应用程序载入的时候都要从 Internet 取回 DTD 的参考版本。 (基于某些神秘的原因，一小部分开发人员说，将 validating 设置为 false 好像可以在没有 Internet 连接的时候有助于应用的载入。通常，不管有没有 Internet 连接你都应该设置 validating=true。)

表 4.2 总结了可以在 Struts 1.1 中使用的配置元素。对 Struts 1.1 新加的元素我们会简要说明。

表格 4.2 Struts 配置元素

	元素	说明
	data-sources	包含 DataSource 对象(JDBC 2.0 Standard Extension)的集合.
	data-source	标识一个 DataSource 对象，它可以被实例化，和进行配置，并在 servlet 上下文中作为一个属性 (或者在 application-scope 的 bean 中).
	set-property	标识一个额外的 JavaBean 配置属性的方法名称和初始化值。
从 Struts 1.1	global-exceptions	描述一个可以被 Action 对象抛出的异常的集合
从 Struts 1.1	exceptions	为一个异常类型注册 ExceptionHandler
	form-beans	描述这个应用模块中的 form bean 描述符集合
	form-bean	描述一个可以被<action>元素引用的 ActionForm 子类
从 Struts 1.1	form-properties	描述一个 JavaBean 属性，可用来配置一个 DynaActionForm 实例或者其子类
	global-forwards	描述对所有 Action 对象都可以作为返回值的 ActionForward 对象集合
	forward	描述一个可以被 Action 作为返回值的 ActionForward 对象

	action-mappings	描述一个可以用来处理匹配 ActionServlet 注册到容器的 url-pattern 格式的请求的 ActionMappings 对象集合
	action	描述一个 ActionMapping 对象，可以用来处理一个对特定的模块相关的 URI 的请求
从 Struts 1.1	controller	描述一个封装了应用模块运行时配置的控制器配置 bean
从 Struts 1.1	message-resources	描述该模块的消息模板一起配合使用的消息资源 MessageResources 对象
从 Struts 1.1	plug-in	标识一个通用应用的 plug-in 模块的全限定类名，它接受应用的启动和退出事件的通知

为了更方便，附录 B 还以标准的 API 格式列出了 struts-config DTD。在这一节， 我们会讨论这些元素并提供一些使用示例。关于每个元素及其接收的属性的具体细节，请参考附录 B。

注意

如果你不喜欢手工编写*XML* ，那么你可以在一个*IDE* 环境中使用*Struts* ，可视化工具会帮助你维护*Struts* 配置。*Scioworks Camino [Camino]* 和 *Struts Console [Console]* 可以直接管理*Struts* 配置文件。

其他产品，比如 *Synthis [Adalon]* 的 *Adalon* 以及 *ObjectVenture [ObjectAssembler]* 的 *ObjectAssembler*, 都可以帮助你以可视化的方式设计应用，并为你编写初始的 *Struts* 配置，*Java* 类和 *JSP* 。关于当前最新的 *Struts* 相关产品，请参见 *Struts Resources* 页面 *[ASF, Struts]*.

如上所述，现今许多编程组件都以 XML 文件的方式进行配置，包括 Java servlet 容器。关于 Java 和 XML 的更多知识，我们推荐 *J2EE and XML Development* [Gabrick]这本书。

4.4.1. <global-exceptions>

在一个 Struts 应用中，ActionServlet 位于调用树的最顶层，但其实际工作会委托给 Action 对象。这种分而治之的策略在许多环境下都工作的很好，异常会被进行异常处理。许多应用喜欢用一致的方式来处理异常，但这会在多个 Action 中复制异常处理代码。

为了在全部 Action 对象中进行一致的异常处理，你可以在一个 Struts 配置文件中注册一个 ExceptionHandler 。框架提供了一个缺省的 ExceptionHandler (`org.apache.struts.action.ExceptionHandler`) ， 它可以在一个请求范围的属性下保存异常，为异常信息创建 ActionError 对象，并转发控制到 JSP 或其他你选定的 URI。Struts `<html:errors>` 标签会自动输出你的异常信息的本地化版本。所以，你可以使用同一个页面来显示你想用来显示的校验错误的异常错误信息。

如果你还需要做些其他的事情，ExceptionHandler 可以被子类化而加入新的行为。如果需要的话，每个异常都可以标识其自己的句柄类。

你可以为一个异常注册一个全局句柄以及针对某个 ActionMapping 的局部句柄。要注册一个异常，你需要提供 `Exception` 类型，消息关键字，以及响应路径，如下所示：

```
<exception
    type="org.apache.struts.webapp.example.ExpiredPasswordException"
    key="expired.password"
    path="/changePassword.jsp" />
```

请参见第 9 章，关于如何编写你自己的异常处理句柄。

4.4.2. <form-beans>

Struts ActionForm (`org.apache.struts.action.ActionForm`) 提供了一个方便的保存通过 HTTP 请求提交的输入属性的方法。但是为了保存输入属性，控制器必须首先创建一个 ActionForm 并将其保存在请求或者会话的上下文中，在这里，可以被其他框架组件—比如 JSP 标签—所找到。

如果在输入页面上有多个表单，那么每个表单都需要其 ActionForm 对象有不同的属性（Attribute）名称。因此，你不能仅使用标准的命名。因为 bean Attribute 名称是其公共 API 的一部分，我们应该能提供开发者友好的 ActionForm 命名，比如 *logonForm* 之类。

某些特别的 ActionForm ，比如 DynaActionForm (`org.apache.struts.action.DynaActionForm`)，需要在它们被创建时传递额外的属性（property）。所以，我们也需要有地方来放置这些元素。ActionFormBean (`org.apache.struts.action.ActionFormBean`)通过将之存储为一个 ActionForm 对象描述符来解决了所有的这些问题。每个 ActionFormBean 都有描述一个 ActionForm 特性名称和类型的属性。ActionFormBean 也包含 property 特性，可以在 DynaActionForm 中使用。

Struts 配置文件提供了一个`<form-bean>` 元素来归类一个模块所使用的各种 ActionFormBean 。每个 ActionFormBean 都由一个相应的`<form-bean>` 元素创建。在运行时， 控制器调用适当的 ActionFormBean 来找出哪一个 ActionForm 对象被创建，在哪里存储它，以及要使用什么属性（Attribute）名称。

下面是一个针对常规 ActionForm 的配置，以及一个 DynaActionForm 的 <form-bean> 元素配置：

```
<form-bean name="menuForm" type="org.apache.struts.scaffold.MenuForm" />

<form-bean name="logonForm"
            type="org.apache.struts.action.DynaActionForm">
    <form-property name="username" type="java.lang.String" />
    <form-property name="password" type="java.lang.String" />
</form-bean>
```

menuForm `<form-bean>` 表达了一个常规的 ActionForm 子类；它要求一个对应的 Java 类支持。logonForm `<formbean>`并不要求使用一个特别的子类，但可以使用 DynaActionForm。(DynamicActionForm 从 Struts 1.1 开始引入)

请参见第 5 章获取关于 ActionForm 的更多信息，包括 DynaActionForm 。

4.4.3. <global-forwards>

通过集中管理细节，Struts 配置将变更予以最小化。当环境改变时，多数实现细节可以通过配置来进行修改，而不用改动 Java 或者 JSP 源代码。

Web 应用中一个令人痛苦的细节处理就是 URI [W3C, URI]。多数 URI 都直接映射到应用目录树中的物理文件。这对常规的站点来说还比较容易。要将 “一个页面放到 web 上，”你只需要将页面存储到站点的某个目录。而目录已经映射到站点的公共 URI， 不需要配置其它地方。

发布 web 页面其实就是传输文件。这其实很简单，直到你想删除么某些页面或者使用不同的页面。当这些事情发生了，(实际上它们经常发生)， 你不得不更新应用中所有对该页面的引用之处。如果漏掉了一些，某些地方仍然引用到旧的页面，你就会遇到我们的数据库人员所称的“异常更新”（update anomaly）。两个假定为相同的情况现在都不同了。数据库对这个问题的解决方案是规格化（*normalization*.）我们将情况存放进一个表，而每个人都从这个表中进行查找。如果情况改变，我们仅仅需要更新情况表，每个人都会被带到相同的页面。

Struts 处理 URI 表的方法是 *ActionForward*。一个 ActionForward 对应一个 URI 的逻辑名称。其他组件可以引用这个名称，而不需要知道任何有关 URI 的情况。如果 URI 改变，我们只需要改变它的 ActionForward。

其他组件通过请求 ActionForward 的路径来得到更新过后的 URI。这样，通过封装实现细节到逻辑名称之后，我们最小化了变更并减少了潜在的错误。

ActionForward 的主要使用者是 Action 对象。当一个 Action 完成时，它返回一个 ActionForward 或者 null 。如果 Action 没有返回 null，ActionServlet 就将控制转发到返回的 ActionForward 的路径。典型地，Action 将通过名称查找 ActionForward ，而不需要知道关于 URI 的任何事情。你可以部署全局转发 Global ActionForward 在 `<global-forwards>` 元素中，像这样：

```
<global-forwards>
    <forward name="logoff" path="/Logoff.do" />
    <forward name="logon" path="/Logon.do" />

</global-forwards>
```

这些 forward 对应用中的每个 Action 都有效。你也可以部署一个局部 ActionForward 到 `<action>` 元素中。局部转发仅针对该 ActionMapping 有效。

关于 ActionForward 参见第 6 章。

4.4.4. <action-mappings>

ActionForm 负责保存应用需要收集的数据。ActionForward 归类那些应用需要引用的 URI。*ActionMapping* 则描述应用要采取的操作、命令。

Action 对象处理操作的实际工作。但大量的管理细节都关联到一个操作。ActionMapping 就是用来包装这些细节的。

一个重要的细节之处就是用来调用 Action 对象的 URI [W3C, URI]。Action 的 URI 被用作一个 ActionMapping 的逻辑标识符，或者路径。当 web 浏览器请求一个 Action 的 URI，ActionServlet 首先查找相应的 ActionMapping 。ActionMapping 则告诉 ActionServlet 哪个 Action 对象要用于这个 URI。

除了 URI 路径和 Action 类型以外，ActionMapping 还包含了几个可以用来在 Action 被调用时发生的行为的属性。改变这些属性会改变 Action 对象的行为。这可以帮助开发人员使同一个 Action 对象有更多的用途。如果没有 ActionMapping 对象，开发人员可能需要

创建比现在多得多的 Action 类。

你也可以使用 ActionMapping 来简化到另一个路径的转发或者重定向。但绝大多数情况下，它只是用来连接 Action 对象。

<action-mappings> 元素描述了我们的应用要用来处理请求的 ActionMapping 对象(org.apache.struts.action.ActionMapping) 的集合。请求要到达应用然后到达 ActionServlet ，它必须匹配上下文和我们在容器中注册的 url-pattern 格式。

因为所有的请求都匹配匹配这个格式，我们就不需要使用上下文或者 url-pattern 来标识一个 ActionMapping 。所以如果 URL 是

```
http://localhost/myApp/myAction.do
```

我们只需要引用/myAction 作为 ActionMapping 的路径。每个 ActionMapping 都由对应的嵌入到<action-mappings>元素中的<action> 元素创建，如下所示：

```
    <!-- 操作映射 -->

<action-mappings>
  <action  path="/logoff"
           type="app.LogoffAction" />

  <action  path="/logonSubmit"
           type="app.LogonAction"
           name="logonForm"
           scope="request"
           validate="true"
           input="/pages/Logon.jsp" />

  <action  path="/logon"
           type="app.ContinueAction">
    <forward name="continue"
             path="/pages/Logon.jsp" />
  </action>

  <action  path="/welcome"
           type="app.ContinueAction">
    <forward  name="continue"
              path="/pages/Welcome.jsp" />
  </action>

</action-mappings>
```

一个 ActionMapping 可以引用多个属性。紧随 Action 对象之后，ActionMapping 可能是 Struts 应用另一个最重要的对象。

关于 ActionMapping 的详细信息，参见第 7 章。

4.4.5. <Controller>

Struts 允许多个应用模块共享一个单一的控制器 servlet。每个模块有其自己的 Struts 配置并且可以相对于其他模块独立开发。<controler> 元素允许每个模块为 ActionServlet 标识一套不同的配置参数。它们大多数是部署描述符中的原始<init-params>设置。

<controller> 元素设置的属性值存储在一个控制器配置 bean (org.apache.struts.config.ControllerConfig)之中。每个应用模块要创建一个控制器配置，

包括缺省的根模块。如果一个模块的 struts-config 提供了一个<contrller> 元素，就是用来设置模块的控制器配置 bean 的属性。

因为各个模块共享 ActionServlet， 你也可以为每个模块插入不同的请求处理器。这可以使每个模块都按自己的方式处理请求，而不用子类化共享的 servlet。

下面是一个<controller> 元素的配置例子，它设置 nocache 和 null 配置属性为 true 并且装入一个定制的请求处理器:

```
<controller
  nocache="true"
  null="true"
  processorClass="com.myCompany.struts.RequestProcessor"/>
```

请求处理器是 ActionServlet 处理循环的核心。大多数情况下，你可以编写和装入一个请求处理器，来代替创建你自己的 ActionServlet 子类。关于 ActionServlet 和请求处理器的更多信息，参见第 9 章。

4.4.6. <message-resources>

每个模块都应该有其自己的缺省消息资源束。这是一个 Struts 组件，如 JSP 标签，在没有其他特别标明的情况下要使用的资源束。你也可以装入其它额外的资源束，连同特定的消息模板。例如，许多开发人员喜欢将图像相关的消息放在一个单独的资源中。

<message-resources> 元素用来部署应用需要使用的资源束。下面是一个<message-resources>元素的例子，它为模块部署了缺省的资源束，另一个则部署了一个图像消息的资源：

```
<message-resources
   parameter="resources.application"/>
<message-resources
   parameter="resources.image"/>
```

在需要时，框架会在名为 resources 的包中的名称为 application.properties 的文件中寻找缺省的消息资源束。包，或者文件文件夹，可以在容器的 classpath 路径的任何地方。典型地，资源束也可以以 JAR 文件的方式，或者放在 WEB-INF/classes 文件夹中。

如果 JSP 标签，或者其他组件，标识了 resources.image 资源束，框架会在 resources 包中查找名为 image.properties 的文件。

参见本章 4.5 节，获取关于消息资源束的更多信息，以及第 13 章如何本地化应用。

4.4.7. <plug-in>

对 Action 来说，需要特别的资源来完成其工作的情况并不常见。不过，它可能需要使用一个非数据源兼容的连接池。或者也许需要创建一个应用 bean 来为表单使用。也许需要读入自己的配置文件来创建一系列的对象，就像 struts-config 所做的一样。在一个常规 web 应用中，这些任务通常委托给一个特殊的 servlet。在 Struts 应用中，我们倾向于将这些任务

委托给 Action 。当一个 Action 需要初始化和销毁它自己的资源时，它可以实现 PlugIn 接口(org.apache.struts.action.PlugIn)。这个接口声明了 init 和 destroy 方法，控制器可以在适当的时候进行调用。

PlugIn Action 可以在 Struts 配置中通过<plug-in>元素进行注册。下面是一个标准的 plug-in ，用来初始化 Struts Validator:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
    property="pathname"
    value="/WEB-INF/validator-rules.xml"/>
  <set-property
    property="pathname"
    value="/WEB-INF/validation.xml"/>
</plug-in>
```

请参见第 9 章获取关于 ActionServlet 和 PlugIn Action 的更多信息。

4.4.8. <data-sources>

虽然 Struts 框架是模型中立的，它仍然需要和业务层甚至数据访问层进行交互。一个组件期望调用者给它传递一个活动的 SQL 连接(java.sql.Connection)的情况并不是不常见。这也使调用者(比如 Struts) 有责任管理连接的生命周期。为了向应用提供在连接到数据访问组件时更多的灵活性， JDBC 2.0 标准扩展包提供了一个基于工厂的方法来获取数据。对一个应用连接到数据库，或者其他数据服务的首选方法是使用一个实现数据源接口(javax.sql.DataSource)的对象。

在一个 web 应用中，一个数据源对象通常代表一个应用中所有用户都可以共享使用的连接池。获取一个数据库连接可能在实践和资源上都有很昂贵的代价。典型地，web 应用以一个单独的账户登录到数据库中，然后自己管理这个单独账户的安全性。

为了帮助开发人员使用连接， Struts 提供了一个数据源管理组件。你可以使用这个组件来实例化和配置一些实现数据源的对象，并且可以从 JavaBean 的属性进行整体配置。

如果你的数据库管理系统没有提供满足这两个要求的组件，你也可以使用 Jakarta Commons 数据库连接池基本数据源类(org.apache.commons.dbcp.BasicDataSource)。在 Struts 1.1 中，Struts 通用数据源(org.apache.struts.util.GenericDataSource) 是一个 BasicDataSource 类的包装类。(这个 Struts 类已经不推荐了，仅提供向后兼容之用)。

如果你的数据库管理系统提供了它自己的数据源可供 Struts 使用，你就应该考虑使用这个实现。在 Struts 中使用 BasicDataSource 或者 GenericDataSource 并不比使用其它类更有效率。要根据你的环境选择最好的实现。

你也可以配置不止一个数据源，然后根据名称进行选择。这个特征可以提供更好的安全型和扩展型，或者用一个数据源实现和另一个进行比较。

下面是一个数据源配置，使用 MySQL 数据库的 Struts 缺省配置:

```
<!-- 数据源 -->
<data-sources>
    <data-source>
        <set-property property="maxCount" value="4" />
        <set-property property="minCount" value="2" />
        <set-property property="description"
                      value="Artimus:MySQL Data Source Configuration" />
        <set-property property="driverClass" value="org.gjt.mm.mysql.Driver" />
        <set-property property="url" value="jdbc:mysql://localhost:3306/artimus" />
        <set-property property="autoCommit" value="true" />
        <set-property property="user" value="root" />
        <set-property property="password" value="" />
    </data-source>
</data-sources>
```

不像其他 struts 配置元素，<data-source> 元素非常依赖于<set-property> 元素。因为开发人员经常需要配置他们自己的 DataSource 子类，所以只有较少的属性内建到了<datasource>元素之中。

数据源对象是 Struts 和数据访问层之间的桥梁。而配置中的其他组件组成了 Struts 的控制层。

4.4.9. 该你了

如果你子类化了一些 Struts 配置对象，你可以使用<set-property>元素传递自己的属性到这些子类之中。这就使你可以扩展 Struts 框架类，而不用改变配置文件的解析方式。下面是一个例子，它传递一个 XSL 样式表引用到一个 (假定的) ActionForward 对象的定制实现：

```
<!-- 全局转发 -->
<global-forwards type="app.struts.XslForward">
    <forward name="logon">
        <set-property property="styleName" value="default" />
        <set-property property="stylePath" value=" /logon.xsl" />
    </forward>
</global-forwards>
```

当 logon 元素的 XslForward 对象被实例化时， Digester 相当于调用

```
logon.setStyleName("default");
logon.setStylePath("/logon.xsl");
```

你也可以在很多 Struts 配置元素中使用这种方法，使所有的对象都成为完全可插入的。

4.4.10. Struts config 骨架

清单 4.2 是一个 Struts 配置文件的骨架，展示了最常用的元素和属性。这个文件也和 Struts 空白应用中的配置文件非常相似：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```
<!DOCTYPE struts-config PUBLIC
"-//Apache Software Foundation//DTD Struts Config 1.1//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
  <data-sources>
    <data-source>
      <set-property
        name="${}"
        value="${}"/>
    </data-source>
  </data-sources>
  <form-beans>
    <form-bean
      name="${}"
      type="${}">
      <form-property
        name="${}"
        type="${}"/>
    </form-bean>
  </form-beans>
  <global-exceptions>
    <exception
      type="${}"
      key="${}"
      path="${}"/>
  </global-exceptions>
  <global-forwards>
    <forward
      name="${}"
      path="${}"/>
  </global-forwards>
  <action-mappings>
    <action
      path="${}"
      type="${}"
      name="${}"
      scope="${}"
      validate="${}"
      input="${}">
      <forward
        name="${}"
```

```
      path="${}"/>
    <exception
      type="${}"
      key="${}"
      path="${}"/>
  </action>
 </action-mappings>
 <controller processorClass="${}" />
 <message-resources
  parameter="${}"/>
 <plug-in
  className="${}">
  <set-property
    property="${}"
    value="${}"/>
 </plug-in>
</struts-config>
```

清单 4.2 Struts 配置骨架

完整的 Struts 配置元素和属性清单，参见附录 B，struts-config API 。

4.5. 应用资源文件

Struts 框架提供了好用和灵活的消息系统。我们在第 3 章接触过这个系统，在第 13 章还要详细讨论它。在这一节，我们需要用来使系统运转的配置文件。

在 4.4.6 节，我们看到了如何告诉 Struts 在何处找到资源束。在这一节，我们要讨论如何创建资源束。消息的文本被存储在一个标准的 Java 属性文件 (java.util.Properties) 之中。

在 Java 和 JSP 代码中，要给定一个消息的关键字；消息文本在运行时从 Property 文件中检索。框架文档将消息属性文件引用为 *application resources* 或者 *message resource bundle* 。

如果你想要本地化你的应用，你可以为你想要支持的场所创建一个额外的应用资源文件。这实际上是创建一个资源束(java.util.ResourceBundle)。框架会为每个用户维护一个标准的 Locale 对象(java.util.Locale) 。针对用户场所的合适的消息会自动从资源束中进行选取。关于本地化的更多信息，参见第 13 章

定义

场所（Local）对象是一个特定的语言和区域的综合标识符。

ResourceBundle 对象包含场所特定的对象。当需要一个场所特定的对象时，可以从资源束中取得，他返回与用户当前场所相匹配的对象。Struts 框架的消息文本使用基于字符串的资源束。

属性文件自身是一个普通的文本文件，每一行是一个关键字-值对。你可以使用任何文本编辑器进行编辑，包括 Windows Notepad 。

应用资源文件的缺省名称是通过在 web.xml 向 Struts ActionServlet 传递一个初始化参数决定的。下面这个片断，参数的名称是 application:

```
<init-param>
  <param-name>application</param-name>
  <param-value>application</param-value>
</init-param>
```

这个参数没有缺省值。在应用中使用 Struts 应用资源束之前必须首先进行配置。应用资源文件位于应用的 CLASSPATH 之中，这样 Struts 可以找到它。最好是放在你的应用的 classes 文件夹中。这可能是在 WEB-INF/classes 文件夹中，或者，如果你以二进制部署你的应用时，则在 WEB-INF/lib 下的一个 JAR 文件中。

param-value 应该是你的文件按包命名格式的全限定名称。这意味着如果如果你将资源文件直接放在 classes 下，你可以直接使用文件名，如前面的代码片断所示。

如果你将文件放在一个子目录中，那么该子目录就相当于一个 Java 包。如果应用资源束在一个名为 resources 的子目录下，你就应该这样来标识：

```
<init-param>
  <param-name>application</param-name>
  <param-value>resources.application</param-value>
</init-param>
```

物理文件的系统路径应该是：

```
WEB-INF/classes/resources/application.properties
```

如果你将类移到了 JAR 中，不需要进行什么改变。Struts 可以在 JAR 文件中找到资源束，就如找到其他类一样。为了本地化你的应用，为每个支持的场所添加资源文件，并修改基本名称：

```
WEB-INF/classes/resources/
application.properties
application_es.properties
application_fr_CA.properties
```

Application 名称只是一个习惯。对框架来说没有缺省设定。你可以将名称改为你认为适合的任何名字。另一个常用的缺省是使用 appplicationResources 作为名称，因为它在一些早期的 Struts 例子中经常使用。

关于更多属性文件和资源束的信息，参见 Sun Java 教程，以及本书地 13 章的国际化[Sun, i18n]部分。

本章稍后表述的 Ant 构建文件通过自动将资源束从源代码树中拷贝到二进制 class 文件夹

中来在构建应用时帮助你管理应用资源束。这可以将原始文件和其他源代码放在一起。

4.6. Ant构建文件

虽然不是使用和配置 Struts 需要的严格要求的部分，许多开发人员还是使用 Ant 及其构建文件来装配，部署，甚至测试他们的应用。我们的 logon 应用 (第 2 章)中的 build.xml 文件就是基于空白 Struts 应用中提供的一个实际框架。

构建文件设置为使用源代码存储在 WEB-INF 子目录下的项目树。这使得整个应用，包括源代码和编译文件，都集中在一个目录系统之中。这就能使你得应用的工作目录可以位于你的开发服务器中。如果容器可以很好的重装类文件，你就可以重新构建应用来测试最新的改变，而不用重启容器。

这个 build.xml 希望的源代码书结构请参见第 4.3 节。

清单 4.3 展示的是我们的一个简单的完整 build.xml 文件。

```
<!—①-->
<project name="logon basedir="." deafult="dist">
  <property name="project.title" value="Logon"/>
  <property name="project.version" value="1.2"/>
  <property name="dist.name" value="logon"/>
  <!—②-->
  <path id="project.class.path">
    <pathelement path="lib/struts.jar"/>
    <pathelement path="./classes/"/>
    <pathelement path="${classpath}"/>
  </path>

  <!—③ -->
  <target name="prepare">
    <tstamp/>
  </target>

  <!—④-->
  <target name="resources">
    <copy todir="classes" includeEmptyDirs="no">
      <fileset dir="src/resources">
        <patternset>
          <include name="**/*.properties"/>
        </patternset>
      </fileset>
    </copy>
  </target>
```

```
<!—⑤-->
<target name="compile" depends="prepare,resources">
<!-- property name="build.compiler" value="jikes"/ -->
  <javac srcdir="src" destdir="classes">
    <classpath refid="project.class.path"/>
  </javac>
</target>

<!—⑥-->
<target name="clean" description="Prepare for clean build">
  <delete dir="classes"/>
  <mkdir dir="classes"/>
</target>

<!—⑦-->
<target name="javadoc" description="Generate JavaDoc API docs">
  <delete dir="./doc/api"/>
  <mkdir dir="./doc/api"/>
  <javadoc sourcepath="./src/java"
           destdir="./doc/api"
           classpath="lib/struts.jar:"
           packagenames="app.*"
           author="true"
           private="true"
           version="true"
           windowtitle=" API Documentation"
           doctitle="&lt;h1&gt;${project.title}Documentation
                       (Version${project.version})&lt;/h1&gt;"
           bottom="Copyright &#169; 2002"/>
</target>

<!—⑧-->
<target name="dist" description="createe binary distribution">
  <delete dir="./dist"/>
  <mkdir dir="./dist"/>
  <war warfile="./dist/${dist.name}.war"
         webxml="../WEB-INF/web.xml"
         manifest="../META-INF/MANIFEST.MF"
         basedir="../"
```

```
          excludes="WEB-INF/dist,WEB-INF/web.xml,
                    META-INF/MANIFEST.MF"/>
  </target>

  <!—⑨-->
   <target name="project"
      depends="clean,prepare,compile,javadoc,dist"/>
</project>
```

清单 4.3 Ant 的 Build.XML

①*project* 给出一个构建文件的总体名称，并且标识一个基础目录和缺省目标。当 Ant 装入文件时，目标会首先锁定它的调用。要使用不同的目标，改变这个缺省设置并存储文件，或者在命令行中覆盖它。缺省基准目录设置为 build.xml 的当前目录。脚本的其他部分这是 WEB-INF 文件夹，并且要在这个基础目录的子目录下查找源代码。这个块中还有一些属性要设置，以备后用。要让这个文件用于另一个应用，你可以只修改这些属性，而让剩下的其他属性保持原样。

②*path* 块建立了 Ant 构建应用是要使用的 classpath。它每次都会执行而不管是选择哪一个目标。通常，这是一个 WEB-INF/lib 文件夹中的 JAR 的清单。

③ *prepare* 帮助 Ant 通过比较类文件和源文件的时间戳来最小化编译工作。

④ *resources* 目标从源代码树中拷贝一些属性文件 (java.util.Properties) 到 classes 树。这样你可以保持原始的属性文件和文件源代码中的保持一致。

⑤*compile* 目标首先调用 *prepare* 和 *resources* 目标，然后开始构建源文件。Jikes [Jikes]或者标准的 javac 编译器都可以使用。

⑥*clean* 目标通过删除和恢复类文件夹来确保所有的东西都重新构建。

⑦ *javadoc* 目标为应用构建 JavaDoc。通常，你需要象标明项目的 classpath 一样为 JavaDoc classpath 标明相同的 JAR 路径。注意，这是一个冒号分隔的列表。JavaDoc 编译器会发出警告，但会继续为它能找到的类产生文档。

⑧*dist* 目标为应用创建一个 Web 归档(WAR) 文件。这个文件可以用来在你的生产服务器上部署你的应用。

⑨ *project* 目标将全部构建所有东西，并准备一个二进制的分发包。

关于 Ant 得更多信息，我们强烈推荐你阅读 *Java Development with Ant*[Hatcher] 。

4.7. 配置 Struts 核心

当此为止，我们已经全部涉及了要使你的 Struts 应用运行需要构建和定制的四个文件。

- 部署描述符 (web.xml)

- Struts 配置文件 (struts-config.xml)
- 应用资源束 (application.properties)
- Ant 构建文件 (构建.xml)

现在，我们把它们结合起来，作为一个 Struts 配置的检查表。

4.7.1. 安装Java和Java servlet 容器

第一步是安装一个 servlet 容器，比如 Tomcat 。我们在第 1 章讲述过这个内容，但现在是一个让你能从头开始的快速检查表：

从 JavaSoft 站点[Sun,Java]下载并安装 Java 1.4 (或更高版本)。

从 Jakarta [ASF, Tomcat] 下载并安装 Tomcat 4 LE (或更高版本)。

当然，Struts 也可以工作于其它 web 容器和其他 Java 虚拟机。这仅仅是我们的一个建议。Struts 分发包中包括配置各种容器的技术规范和注意事项。

4.7.2. 安装开发环境

我们在第 3 章也涉及了安装开发环境。Struts 在大多数 Java 环境下都能工作的很好。如果你还没有一个好的环境，作为入门比较好的选择是 jEdit 和 Ant：

下载并安装 Ant 1.4 (或更高版本) [ASF, Ant].

下载并安装 jEdit [jEdit].

安装 Ant add-in for jEdit

但是再次提醒，这仅仅是建议。如果你已经有一个开发环境了，它也会工作的很好。

4.7.3. 安装Struts 核心文件

运行 Struts 需要的常备文件都在 Struts 库文件分发包(jakarta-struts-1.1-lib.zip)中提供了。这些包括几个 JAR， 标签库描述符，DTDs，和标准的 XML 配置文件。这些常备文件和你要提供的 4 个配置文件一起，创建了一个核心的 Struts 配置。

下面是一个检查表：

- 下载并解压 Struts library distribution [ASF, Struts].
- 拷贝所有的*.jar 文件到应用的 /WEB-INF/lib 文件夹.
- 拷贝所有的*.tld 文件到 /WEB-INF 文件夹.
- 拷贝所有的*.xml 和 *.dtd 文件到 /WEB-INF 文件夹.
- 创建部署描述符 (4.2).
- 创建 Struts 配置文件 (4.4).
- 创建资源束 (4.5).
- 创建 Ant 构建文件 (4.6).

4.7.4. 配置Tiles框架

Tiles 是一个 Struts 框架的可选组件，是一个强大的页面组装和布局工具，在其意义上是一个真正的框架。我们在第 11 章讨论 Tiles 的使用。使用 Struts 框架的其它部分时并不需要

使用和配置 Tiles 。但是如果你喜欢 Tiles，这就是一个练习。

注 这个 Tiles 设置的例子是基于 Struts1.1 beta Release 2。这个过程在 beta1 和 beta2 之间就做了修改，并且可能在 1.1 Final Release 中也发生改变。参见本书的网站[Husted]获取最新信息。

你使用 Tiles 的所需的所有文件都在 Struts library distribution 中提供了(4.7) 。如果你让你的应用基于 Struts 空白应用(4.10)或者你已经安装了所有必需的 Struts 库文件夹到应用的 /WEB-INF 或者 r /WEB-INF/lib 文件夹中，那么基本的所需文件已经有了。

下面是检查表：

- 从 Struts lib 文件夹拷贝 struts-tiles.tld 和 tiles-config.dtd 文件 (如果没有)到 /WEB-INF 文件夹.
- 插入下面的语句块(如果没有)到 /WEB-INF/web.xml 文件中，并且紧跟其它 <taglib> 元素：

```
<taglib>
  <taglib-uri>/tags/tiles</taglib-uri>
  <taglib-location>/WEB-INF/tiles.tld</taglib-location>
</taglib>
```

- 创建一个空白的 tiles-defs.xml (如果没有)在 /WEB-INF 文件夹中，像这样：

```
<!DOCTYPE tiles-definitions PUBLIC
"-//Apache Software Foundation//DTD Tiles Configuration//EN"
"http://jakarta.apache.org/struts/dtds/tiles-config.dtd">
<tiles-definitions>
  <!-- skeleton definition
  <definition
       name="${name}"
       path="${path}">
    <put
         name="${name}"
         value="${value}"/>
  </definition>
  end blank definition -->
</tiles-definitions>
```

- 插入这个<plug-in> 元素到 struts-config.xml 中，位置在关闭的</struts-config> 元素之前：

```
<plug-in className="org.apache.struts.tiles.TilesPlugin" >
    <set-property
        property="definitions-config"
        value="/WEB-INF/tiles-defs.xml" />
</plug-in>
```

至于能运行的具体 Tiles 应用的例子，请参见 Struts 分发包中的 Tiles 示例应用，以及第 16 章我们要讨论的 Artimus 1.1 示例。Struts 空白应用 (4.10) 其实也配置了 Tiles 。

至于 Tiles 应用的 Struts 1.0 版本，参见第 15 章的 Artimus 1.0 示例。

4.8. 配置Struts Validator

像 Tiles 一样，Struts Validator 也是框架的一个可选组件。

我们会在第 12 章涉及 Struts Validator 。使用 Struts 框架的其他部分时并不需要使用和配置 Validator 。不过如果你喜欢，这也是一个练习机会。

注 这个 Validator 设置的例子是基于 Struts1.1 beta Release 2。这个过程在 beta1 和 beta2 之间就做了修改，并且可能在 1.1 Final Release 中也发生改变。可才能见本书的网站[Husted]获取信息。

你使用 Validator 所需的所有文件都在 Struts library distribution 中提供了(4.7) 。如果你让你的应用基于 Struts 空白应用(4.10) 或者你已经安装了所有必需的 Struts 库文件夹到应用的/WEB-INF 或者 r /WEB-INF/lib 文件夹中，那么基本的所需文件已经有了。

下面是检查表：

- 从 Struts lib 文件夹拷贝 struts-validator.tld 和 validator-rules.xml 文件 (如果没有)到/WEB-INF 文件夹.
- 插入下面的语句快到(如果没有)到 /WEB-INF/web.xml 文件中，并且紧跟其他 <taglib> 元素：

```
<taglib>
  <taglib-uri>/tags/validator</taglib-uri>
  <taglib-location>/WEB-INF/struts-validator.tld</taglib-location>
</taglib>
```

- 创建一个空白的 validations.xml (如果没有)在 /WEB-INF 文件夹中，像这样：

```
<form-validation>
  <formset>
  <!-- skeleton form
    <form name="${}">
      <field
          property="${}"
```

```
        depends="${}">
      <arg0 key="${}"/>
    </field>
  </form>
  end skeleton form -->
 </formset>
</form-validation>
```

- 插入这个<plug-in> 元素到struts-config.xml ，位置在关闭的</struts-config> 元素之前：

```
<plug-in

className="org.apache.struts.validator.ValidatorPlugIn">
 <set-property
    property="pathnames"
    value="/WEB-INF/validator-rules.xml,
           /WEB-INF/validation.xml"/>

</plug-in>
```

至于能运行的具体 Validator 应用的例子，请参见 Struts 分发包中的 Validator 示例应用，以及第 16 章我们要讨论的 Artimus 1.1 示例。Struts 空白应用（4.10）其实也激活了 Validator。至于 Validator 应用的 Struts 1.0 版本，参见第 15 章的 Artimus 1.0 示例。

4.9. 从空白Struts应用开始

Struts 分发包中的 *Struts Blank* 应用提供了开发新应用时可以采用和修改的骨架配置。它包含了来自于分发包中的所有的基础文件以及开发人员应提供的四个配置文件的初始版本(4.7)，以及 Tiles 和 Validator 的骨架配置。

Struts Blank 是以一个 WAR 文件的形式分发的。开始时你需要做的是：

- 从二进制分发包中的 webapps 文件夹拷贝 struts-blank.war 文件到容器的 WAR 部署目录(通常是 webapps).
- 将 struts-blank.war 更名为你想要的应用名称，比方说 myApp.war.
- 启动或重启 servlet 容器 Struts Blank 应用就会以你的新应用名称进行部署。如果你访问缺省页面：

```
http://localhost/myApp
```

你的新应用就会显示一个简单的欢迎页面。

Blank 应用并没有包括一些 Java 源代码文件—这是你的工作—但它的确包含了一些有用的文件和页面，我们总结在表 4.3 中。

Struts Blank 应用设计目的是让你可以很容易地开始一个 Struts 项目。如果将它部署在一个 servlet 容器的本地拷贝，比如 Tomcat 或者 Resin， 你就可以马上在上面工作了。构建文件的目的是为了使你在过程中可以就地工作，只是需要时重启容器或者重新装入应用。

如果你正在一个团队内工作，你可能需要使用一个不同的方案，因为你需要随时将文件 Check in 或者 Check out 。但是简单的事情总归简单，Struts Blank 应用使得开始构建一个新的 Struts 应用真正变得非常简单。

表格 4.3 Struts Blank 起始文件

文件名	目的
/index.jsp	一个常备的欢迎页面，它转发到一个 Struts Welcome action.
/pages/Welcome.jsp	一个欢迎 JSP， 通过一个 Welcome action 访问
/classes/resources/application.properties	部署的应用资源文件。这是一个运行拷贝，原始文件和其他源代码在一起
/WEB-INF/struts-config.xml	一个起始 Struts 配置文件，常用元素有示范性说明
/WEB-INF/tiles-def.xml	一个起始 Tiles 配置文件,常用元素有示范性说明
/WEB-INF/validator-rules.xml	一个起始标准 Validator 配置文件，设置基本的 validator。

/WEB-INF/validations.xml	一个起始 Validations 配置文件可以在此描述你的 form 。
/WEB-INF/*.tld	Struts 标签的标签库描述符文件。
/WEB-INF/lib/*.jar	框架类依赖的 JAR 文件
/WEB-INF/src/build.xml	Ant 构建文件.
/WEB-INF/src/java/	Java 源代码文件的起始目录
/WEB-INF/src/resources/application.proper	原始的 application.properties 文件。编辑后要重新构建
ties	

需要注意的一件事情是 build.xml 文件引用的本地系统路径可能存在或者不存在。

```
<property name=" jdbc20ext.jar"
  value="/javasoft/lib/jdbc2_0-stdext.jar"/>
<property name="servlet.jar"
  value="/javasoft/lib/servlet.jar"/>
<property name="distpath.project"
  value="/projects/lib"/>
```

jdbc2_0-stdext.jar 和 servlet.jar 需要用来产生 JavaDoc 。Blank build.xml 会在你的默认驱动器的/javasoft/lib 文件夹下查找这些文件。如果必要，你可以修改这些设置以指向你保存这些文件的地方。

你的容器应该包含其所支持的 servlet API (如, 2.2 或 2.3)的 servlet.jar 。

Tomcat 4 在其$TOMCAT/common/lib 文件夹中保存 servlet.jar 文件。

jdbc2_0-stdext.jar 包含在 Struts 库文件分发包中。它没有和 Struts web 应用绑定在一起。许多容器，如 Tomcat ，会共享这个文件的一个拷贝。在 WEB-INF/lib 文件夹中放置另一个拷贝可能会造成冲突。

其他目录是保存你的一个 WAR 文件的地方。这可以用来在开发完成之后在生产系统上部署你的应用，或者在准备测试的时候也行。再一次提醒，你可以修改路径或者创建目录。Blank build.xml 工厂设定是默认驱动器上的/projects/lib 文件夹。

4.10. 配置模块化应用

Struts 架构的一个关键优点是所有的请求都通过一个单一的控制点。开发人员可以集中精力于那些应用于每一个请求的功能，并且避免整个应用中的代码重复。因为 Java 是一个多

线程平台，Struts 使用一个单独的控制器 servlet 也提供了最好的性能可能性。开发人员编写更少的代码，机器在更少的资源下更快的运行。一切都很完美。

在 Struts 1.0 中，应用以单独模式运行。Struts 配置文件对用户来说是一个。如果多个开发人员工作于一个应用，他们需要有一个方法来管理 Struts 配置的更新。

在实践中，开发人员倾向于将他们的工作分割为逻辑单元，这非常类似于他们将 Java 代码分离到各个包中。团队成员在应用中有他们自己的名字空间的情况并不是不常见。

在一个在线拍卖应用中，一个团队可能工作于 registration 模块；另一个团对可能工作于 bidding 模块。第 1 个团队可能具有一个/reg 文件夹来存放他们的 JSP 页面以及一个 app.reg 包来存放他们的 Java 代码。同时，团队 2 可能有一个/bid 文件夹存放 JSP 页面以及一个 app.bid 包来存放 Java 代码。在资源文件中，每个团对可能都有他们以 reg. 或 bid. 为前缀的关键字。

当然，这种方式并不是只可在大型团队中组织大型项目。许多单独的开发者也做同样的事情。限制一个文件夹中的文件数量和包中的类数量，被认为是很好的项目管理。

许多开发人员也按逻辑模块来组织项目，而不管是否需要共享文件。

4.10.1. 分而治之

在 Struts 1.1 中，将应用分为模块的理念成为了一种习惯—现在已经集成到框架中了。让我们回头看看，Struts 是如何将应用组织为模块的。

web 应用容器是我们可以为每个应用创建上下文来共享服务。这个上下文对应于服务器的 webapp 目录的一个子目录。按同样的道理，Struts 1.1 通过为每个应用创建一个前缀来共享应用的使用。多个模块可以运行于相同的应用空间，每一个都在起自己的前缀之下，与多个应用可以运行于同一个服务器空间非常相似—每一个都有其自己的上下文。

我们在编写 web 应用时，我们经常引用到上下文-相关的 URI。这是一个不包含应用名字或上下文的路径。同时，我们在编写 Struts 应用时， 我们也经常应用一个模块-相关的 URI。不用惊讶，这也是一个路径，不包含模块名称，或者模块前缀。表 4.4 所示的是关于绝对的，上下文相关的，和模块相关的同一个 URI 的比较。

就像你可以编写一个 web 应用，并将它配置在某个上下文中一样，你可以编写一个 Struts 应用模块并配置它在某个前缀下运行。

编写一个模块和编写一个独立运行的应用并没有多大的不同。

所有世纪的配置都在部署描述符中。如果你将某个模块从某个前缀移动到根，或者从根移动到某个前缀下，或者从一个前缀移动到另一个前缀，模块中没有 JSP，Java 代码，或者 XML 代码需要改变。

要设置一个应用来使用分离的 reg 和 bid 模块，我们可以这样配置 servlet 描述符：

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>
    org.apache.struts.action.ActionServlet
  </servlet-class>
  <!-- The default (or "root") module -->
```

```
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
<!-- The register module -->
  <init-param>
    <param-name>config/reg</param-name> <-- Includes prefix! -->
    <param-value>/WEB-INF/struts-config-reg.xml</param-value>
  </init-param>
  <!-- The bidding module -->
  <init-param>
    <param-name>config/bid</param-name> <-- Includes prefix! -->
    <param-value>/WEB-INF/struts-config-bid.xml</param-value>
  </init-param>
  <!-- ... other servlet elements ... -- >
</servlet>
```

表格 4-4 表 4.4 Absolute, 上下文-relative, and 模块-relative portions of the same URI

URI 类型	URI
绝对 URL	http://localhost/myApp/mymodule/myAction.do
域-相关	/myApp/mymodule/myAction.do
上下文-相关	/mymodule/myAction.do
模块-相关	/myAction.do

在这个例子中，reg 团队可以工作在 struts-config-reg.xml 而 bid 团队可以工作在 struts-config-bid.xml 配置。每个团队都工作于他们自己的模块，就像她们工作于一个单模块的应用一样。框架做了有关模块前缀的所有调整，就像容器为应用上下文所作的调整和配置一样。

对，几乎是所有的调整...

4.10.2. 给页面加前缀

当框架针对模块前缀调整了 URI， 它会调整所有 URI， 不管它是指向一个 ActionMapping 或者是模块中的其他资源，如 JSP。在 struts-config-reg.xml 文件中，我们可能有一个模块相关的 URI，比如/input.do 。则该 URI 的上下文相关版本就成为/bid/input.do 。同时，我们可能有一个到/pages/Index.jsp 的引用，那么该引用的上下文相关就成为

/bid/pages/Index.jsp 。

迁移 ActionMapping 是非常容易的，它们实际上是虚拟的。但是许多 URI 引用到 JSP，它们实际上和物理文件相关。其它 URI 也可能引用到 HTML 页面，图像文件等等。

这意味着虽然 bid 团队可以在 Struts 配置中忽略模块前缀，但是他们仍然需要知道，他们是在使用哪一个前缀，这样才知道应该在哪里存放他们的文件。如果前缀改变，需要重新命名那个子目录。不需要修改 JSP, Java, 或者 XML 编码，尽管如此，他们必须使他们的页面目录名称和模块名称保持同步。

这其实和我们要在应用层次要干的事并没什么不同。如果我们上载文件到应用中，我们需要知道其应该存放的上下文目录。如果我们上载文件到模块，我们就需要知道他在哪一个模块目录下面。

4.10.3. 修改配置

如果你已经你已经有了一个设置为类似一个模块应用的 Struts 1.0 应用，诀窍就是使模块前缀从模块的配置文件中分离出来。

如果页面已经在以模块名称命名的子目录中，它们继续留在那里就可以了。如果不是，你也能够将它们移动到模块目录下面，而不用作任何改动 (当然，除非一些硬编码的连接，比如 `<html:link>` 标签，必须修改)。

4.10.4. 共享Struts JAR

如果你的容器支持这个功能，Struts JAR 可以被 web 容器中的所有应用所共享。 (现在来说多数都行或者即将行)。请参考你的容器的文档，看如何配置共享的 JAR。大多数情况下，你可以简单地将 Struts JAR 放在共享的 lib 文件夹下面，然后在/WEB-INF/lib 文件夹下删除它们既可。

对 Tomcat 4 来说，共享库文件夹位于

```
$TOMCAT/common/lib
```

对缺省的 Windows 安装，这可能是对应于\Program filess\Apache Tomcat 4\common\lib 。当然，这意味着所有共享这些 JAR 的应用必须是同一版本的 Struts 应用。

4.11. 小结

没有一个应用应该是孤岛。要将应用的所有组件集合在一起工作，你必须创建并维护一些配置文件。要使一个 Struts 应用建立并运行，需要开发者提供一个 Struts 配置文件，一个资源文件， web 应用部署描述符文件，以及一个 可选的 Ant 构建文件。本章一步步讲述了创建这些文件的细节。

Struts 配置的一个关键优点是它集中了变更管理。因为实现细节被集中了，Struts 应用就可以从 Struts 配置进行全面重新配置，而不用去进行一些修改 Action 类或者表现页面的工作。

所有的组件都由 Struts 配置进行初始化—`ActionForm`，`ActionForward`，`ActionMapping`— 它们形成了 Struts 控制器的核心。组合在一起，这些对象表达了应用的外部和内部 API —剩下的就是具体的实现细节了。这些核心组件制定了应用能接受的输入，要转发的地方，以及要做的事情。

那么，接下来是什么呢？

本书的第 2 部分， 我们会深入到每一个核心组件来帮助你全面了解它们。在第 3 部分， 我们集中于你的应用的可见部分：页面。第 4 部分，会主要用几个实例研究来使用 1 到 3 章所表述的技术。

但是如果你已经非常渴望想要开始了，你如今应该已经有足够的知识可以开始构建你的 Struts 应用了。而且，如果你喜欢，就象本书剩余部分所展示的，你可以使用我们介绍的新技术重构，改进，或者扩展你的现有应用。

5.用 ActionForm 传递数据

Ted Husted 和 George Franciscus合著

本章内容

- *理解 ActionForm 生命周期*
- *检查 ActionForm 的职责*
- *讨论 ActionForm 设计特征*
- *使用 ActionForm 的最佳实践*
- *组装和报告 ActionForm*
- *介绍 Scaffold BaseForm*

The sweat of hard work is not to be displayed. It is much more graceful to appear favored by the gods.

—Maxine Hong Kingston,

The Woman Warrior: Memoirs of a Girlhood among Ghosts

5.1. 吃的是草，挤出的是奶

使用Web应用的人通常会花费大量的时间和精力通过HTML form提交数据。有时是一些新的数据，通过填写空白表单来提交。而有些时候，则可能是将修改过的数据重新提交。

HTML form 向Web开发人员提出了两个挑战：一是在数据被提交时获取数据，以及用用户可以修改的数据预装一个表单。如果用户选择一个必须被修改的地址记录，我们就需要将数据从记录拷贝到HTML form。这就意味着，我们必须能够通过传递动态值来针对每次请求而改变页面。

HTML 并没有提供一个内建的手段来使用动态值预组装一个控件。需要定制的页面通过混合静态组件和动态的组件，以一种特殊的方式写成，作为运行时的响应。

有许多方式可以为Java Web应用编写动态页面，最通常的办法是JSP页面。Struts 分发包中包括了一套JSP标签，你可以用它们来编写动态 HTML 控件。和其它许多标签库一样，Struts标签设计来就是同JavaBean一起工作。如我们在第一章所见，JavaBean 是一个遵循一定设计规则的，简单但是非常强大的对象。

和HTML 元素不同，Struts 标签提供了标准的方式来组装控件。每个 HTML 标签对应一个标准的HTML标记元素。每个JSP 标签都有一个属性项提供bean中的属性名称。JavaBean属性的返回值用于控件的value 属性。

所以，如果有一个HTML元素象这样：

```
<input type=”text” name="address"/>
```

它可以由这样的Struts JSP 标签来代替：

```
<html:input property="address"/>
```

标签会从Javabean中获取 address 属性，并将它作为HTML 元素的 value来插入。当浏览器获得这个标签时，它可能看起来像这样：

```
<input name="address" value="6 Lost Feather Drive"/>
```

这里，它实际上是调用 ActionForm的getAddress()方法，而该方法则返回字符串"6 Lost Feather Drive"。

注 在某些编程上下文中,词语“属性（property）”等同于 attribute, field,或者variable。在这些场合，属性（property）表达了一个存储地址。JavaBean 属性(property)通常使用字段/域(field)来存储值，但是 JavaBean “properties”确实说来是可以用来检索值的方法。当我们说一个公共属性public properties,我们实际上是在说 JavaBean 对象的公共域。我们说的方法则是用来检索和设置值的。有时候，这些值存储于域（field）中。其它时候，它可能是从几个域

中计算出来，或者从其它对象中检索出来。JavaBeans的强大之处在于对象可以控制值的存储方式，以及使他们可以通过mutator 和accessor方法进行公共访问。

为完成这个处理，当表单被提交到Struts 控制器时，它将HTTP 参数传递给 JavaBean。大部分来自于HTML 表单的输入都将在传递给业务层之前进行校验。如果一个字段假定是包含数字，我们就得确保它就是数字。如果校验检查失败，我们可以将 JavaBean 回传给页面。然后JSP 标签根据JavaBean 属性重新组装HTML元素，用户就可以纠正输入，并重新尝试。

任何JavaBean 都可以用于 Struts JSP 标签以组装控件。但为了要提供输入的自动校验，Struts使用它自己的JavaBean 子类，称作 ActionForm。

一旦从HTML表单的输入传递给`ActionForm` bean，并且属性经过了校验，所有的属性就要作为一个漂亮整齐的JavaBean传递给`Action`。Struts `Action` 对象使用form bean来完成其业务操作，处理错误，并选择相应响应页面。第8章将详细讨论Action 对象。

5.1.1. ActionForm 的要求

创建一个ActionForm 并不困难，但是你的类必须符合一些要求：

- ActionForm 必须扩展自`org.apache.struts.action.ActionForm`。基类`ActionForm` 是不能实例化的。
- ActionForm 必须为每个应该从请求中收集的HTML输入控件定义一个公共属性。*(Struts 1.0 **要求每个属性都要** mutator **和**accessor。Struts 1.1 **则没有如此严格**)*

ActionForm还可能要符合一些可选的要求:

- 如果你要求ActionForm 在传递属性到Action之前校验它们，你就必须实现validate方法；
- 如果想在组装前初始化属性，必须实现 reset，它在ActionForm 组装前被调用；

下面是一个简单的ActionForm 类：

```
import org.apache.struts.action.*;

public class MyForm extends ActionForm {
  protected String name;
  protected String address;

  public String getName()
    {return this.name;};
  public String getAddress()
    {return this.address;};
  public void setName(String name)
    {this.name = name;};
  public void setAddress(String address)
    {this.address = address;};
```

```
};
```

在Struts 1.1, 你也可以使用 DynaActionForm 类在Struts 配置文件中声明你的属性。下面是一个简单的使用DynaActionForm类的ActionForm：

```
<form-bean
  name="myForm"
  type="org.apache.struts.action.DynaActionForm">
  <form-property
    name="name"
    type="java.lang.String"/>
  <form-property
    name="address"
    type="java.lang.String"/>
</form-bean>
```

关于Struts 配置文件的详细情况，请参见第4章。

虽然ActionForm的要求非常简单，但它在许多应用的开发中都扮演了令人惊奇的强大角色。ActionForm也许是Struts 框架最令人费解的部分。这个“勤劳”的JavaBean扮演了一个字段收集器、防火墙、类型转换器、数据缓冲器以及传输对象的多种角色。

有时它们似乎显得是多余的，有时它们的价值又是无法估量的，但是每次，它们都是框架的焦点部分——也是Struts之所以是 Struts的关键部分。

5.2. 千面女郎 ActionForm

ActionForm是一个多能的对象。就如我们先前解释的，它可以扮演字段收集器，防火墙，数据缓冲, 数据校验器，类型转换器，以及传输对象——在一个单一请求的范围内的所有东西。让我们一一来看看它可以在应用充当的各种角色。

5.2.1. ActionForm 作为字段收集器

绝大部分应用都需要用户输入数据。许多应用甚至需要大量的数据。在web 环境中, 有效收集数据成了其一大挑战。

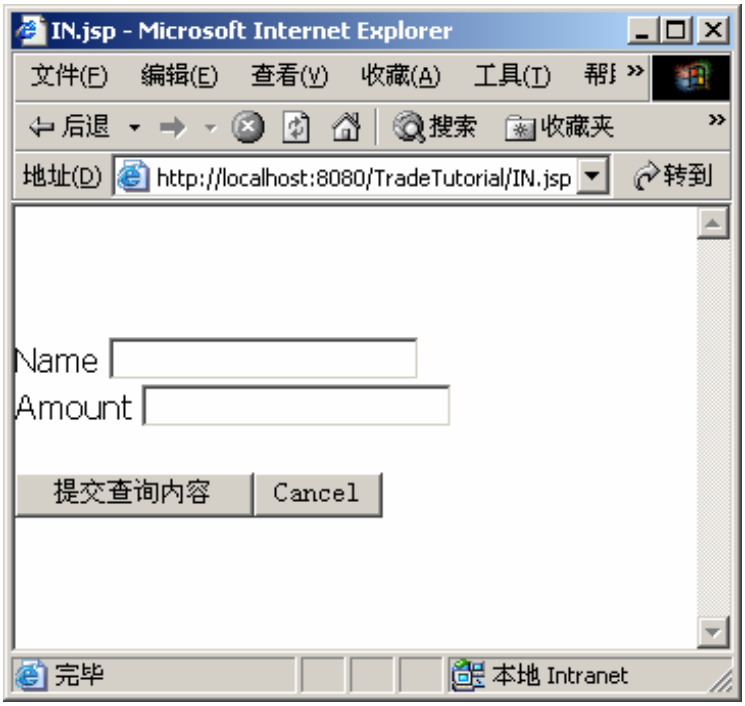
HTML 定义了一组几乎不能使用的数据输入控件。HTTP 也定义了一个几乎不能使用的数据传输协议。Struts 使用 ActionForm来帮助补偿HTML和HTTP的不足。HTTP 是个如此的简单协议，它使用可能是最基本的方式来传输数据。但作为协议本身，它非常易于实现，并且在它自己的方式上似乎运行得非常有效率。但作为应用程序使用的角度，HTTP将许多实现细节作为挑战和考验留给了开发人员。

通过 HTTP提交字段

当一个HTML 表单通过HTTP被提交, 所有的内容都被视为文本。web 服务器接收到

的表单元素是以名-值对的方式来到的。它们是文本字符串，不是二进制数据。

下图是一个简单的表单，两个字段，Name 和Amount。



图表 5-1 有两个字段的简单表单

而下图则是浏览器要提交这个表单的URI。

```
http://localhost/app/submit.do?name=ted&amount=1.00
```

图表 5-2 来自表单的 Get 请求

而如下图，POST 方法则可以用来隐藏提交的内容，但结果是一样的：表单中输入的内容被作为名-值对编码进URI之中。

```
POST /app/submit.do HTTP/1.0
Content-Type: application/x-www-form-URL-encoded
Content-Length: 22

name=ted
&amount=1.00
```

图表 5-3 来自表单的 POST 请求

定义 HTTP 规范允许表单数据作为URI的一部分进行提交,但不是每个字符都可以用在URI之中。如果其它字符需要被表单数据使用，他们必须先进行编码。 URL 使用% 符号，紧跟两位十六进制ISO-Latin代码数据 (非大小写敏感)。详细内容见 RFC 1738 和 2396 [W3C URL; W3C URI]以及 HTML

4.01规范 17.13.4 [W3C HTML4].

通过HTTP提交文件

一些表单允许将文件作为一个附件进行上载。这时，附件要进行特殊编码，以便仅使用文本内容编码就可以提交二进制文件。这样就允许我们能够通过HTTP传递二进制文件，文件被转换成文本字符流来传输，然后再接收后又被转换回来。

通过HTTP提交空字段

如果一个文本字段是空白的，大部分浏览器都会提交一个空参数。但是如果一个选择框是空白的，浏览器则根本不提交任何东西。应用必须根据浏览器没有提交选择框时判断出，这种情况意味着它是false 或者 null。实践中，这仅适用于选择框，但规范允许浏览器忽略任何空白字段。

通过HTTP接收

通过解析HTTP 文本流来查找名值对可不是一件愉快的事。

虽然Servlet API 提供一些有用的方法帮助查找提交的内容，并找出你需要的内容。但许多烦人的代码仍然需要调用。

ActionForm方案

Struts 对HTTP 参数处理的方案是将输入参数传递到 JavaBean 属性来进行处理。当ActionForm的属性与某个请求参数匹配，框架自动以参数的值设置属性。Struts 开发人员可以工作在一个可以信赖的JavaBean上，并将处理HTTP的麻烦留给框架处理。

要收集请求参数，Struts 开发人员需要做的所有事情就是提供一个具有JavaBean属性的ActionForm，这些属性名称与HTTP请求参数匹配。其余的工作将会自动进行。

5.2.2. ActionForm 作为数据缓冲

在一个常规(非web)用户接口中，捕获用户输入的控件通常都具有一个内部缓冲，以便可以对数据进行校验。如果校验失败，用户不能离开控件。如果校验成功，数据被传送到适当类型的其它字段。开发人员通常看不到内部缓冲，但它确实在。

HTML 控件并没有一个内建的缓冲器，或者其它有效的方法，来使数据在提交之前进行校验。当然，可以用JavaScript来做这些，但JavaScript 是可以被浏览器禁止的。

ActionForm 扮演了我们想要的HTML控件的输入缓冲。它保持输入数据，直到数据被校验然后送到相应类型的字段为止。

如果用户在一个`Integer` 字段中输入了一个错误字母，经过校验后原始输入应该被返回，包括无效的字符。用户可以看看到底是什么地方出错了，以便纠正数据，然后再试。这也意味着ActionForm 属性应该是`String` 类型，以便各种类型输入都可以捕获，不管有效还是无效。

ActionForm 字段不是输入的目的地，但它要缓冲数据，以便在提交之前进行校验。

5.2.3. ActionForm 作为数据校验器

虽然现有的项目也许已经有一些JavaBean来执行校验，但它们很少有能在输入无效时提供缓

冲，以便能进行纠正。ActionForm的 validate 方法是一个扩展点，你可以在此插入对业务方法（它们知道如何校验数据）的调用。**当校验失败， ActionForm 可以将整个内容都回传给web 页面，这样用户就可以重新尝试。**

但是ActionForm的校验职责并不比它的纠正职责多一些。许多字段在传递给业务逻辑处理之前必须具有正确的类型。在一个Integer 字段中的数据不能包含字母。如果这样，你应该在让用户继续之前使用户能纠正此数据。

通常，这只是一个表面形式上的校验。即一个字段是一个Integer 并不能告诉我们它是正确的Integer。许多应用将校验分为两段执行。首先，他们使用ActionForm的validate 方法来决定输入是否是正确的类型以及它们是否可以被业务过程使用。一旦这个阶段完成，Action 可以执行额外的校验，来决定输入是否满足业务层的其它要求。如果业务层校验失败，你可以将控制返回到输入页面，就象这是在ActionForm 的validate 方法校验失败一样。

Struts 框架给了你处理数据校验的灵活性，用 ActionForm，用Action，或这两者，按你所需而定。

在第12章，我们将讨论Struts Validator，它扩展了ActionForm validate 方法的使用。

5.2.4. ActionForm 作为类型转换器

ActionForm的一个强制之处是应该使用 String 和 boolean属性。实际上，这意味着属性必须得从一种转换到另一种类型。大部分应用也需要一些属性，比如电话号码或者数量，并以一种格式化的方式出现。核心Java 包提供一些工具来做这种事情，但是要干净利落地将它们集成到应用中仍然是一个挑战。

Struts 开发人员经常在ActionForm中包含一些helper方法，来进行类型转换。helper 方法可以有很多种实现方式，这我们在5.6种叙述。

5.2.5. ActionForm 作为传输对象（TO）

ActionForm 可以被其它bean或者过程作为数据载体。就是说ActionForm 是一个***传输对象***（*Transfer Object*）。象其它传输对象一样，它承载的数据通常对应着持久层中的不止一个实体 (比如不止一个数据库表)。但不像常规的传输对象，ActionForm 的各个属性都必须是可变的。HTTP 将每个属性表达为名-值对。如果每个属性都是可以独立设置的，这就足够简单了。

另一方面，其它传输对象通过仅在实例化时被设置，而后不可改变。

定义

传输对象（也称为值对象（value object） [Go3])用来通过发送粗粒度的数据视图来交换细粒度的数据。通常用在远程应用环境之中， 传输对象可以将多个相关的属性组织成组，以便它可以被序列化，并且可以在一个单一操作中被传送到远程服务器。

Murable意味着“可以被改变或者变更”， 计算机程序可以包含可变（mutable）和不可变（ immutable ）元素。某些对象设计来就是不可改变的，并且作为不可变对象进行引用。大部分对象则是设计来是可变的，作为可变对象应用。

STRUTS TIP

使用粗粒度的 ActionForm来减小类维护。实践中，应用中的表单一般都会共享属性。这通常更易于创建一个基本的 ActionForm，具有表单需要的所有属性。如果必要，你可以子类化这个粗粒度的基础bean，并提供特殊的 validate 和 reset 方法。

5.2.6. ActionForm 作为防火墙

当请求提交时，ActionServlet 使用一个自动组装机制来从请求参数中设置请求的ActionForm属性。这使得你可以通过控制哪个ActionForm 属性被暴露的方式来控制哪个请求参数是可接受的。这也意味着如果你的ActionForm 的粗心设计，你可能失去控制哪个参数可以接受的能力。ActionForm 一定不能包含看起来像是一个JavaBean 属性，但却不能从HTTP 请求设置的属性方法。

设计ActionForm时你应该当心的是它的自动化组装机制。自动化机制自己是乐于根据请求来设置ActionForm的公共属性的，而不管它们是否是来自于HTML表单。所以，如果你要将一个bean重用为ActionForm，bean上的一些公共属性—以及它的超类—就可从HTTP 请求中直接设置。因为引用是可以被嵌套和链接的，某些作为成员属性的bean也被暴露，连同其超类和一些成员属性。如果这些bean中的某些可以在系统状态中立即改变，那么一个欺骗就可以影响到状态改变—即便那不是一个有意的用法。

如果你从头创建一个ActionForm，并按其原本的意图，你就完全没有必要担心自动组装。但某些开发人员喜欢将业务对象置于ActionForm中以便它们可以从请求中通过值来传递。如果这样，某些看起来像JavaBean 属性的方法，如果它们接受一个 String 值，就可能被从HTTP 请求来调用。

一个例子是ActionServlet的上载缓冲区大小。在 Struts 1.0中，ActionServlet 被暴露为一个ActionForm成员属性。这意味着在 Struts 1.0，你可以从HTTP请求中来调用ActionServlet.setBufferSize。幸运的是，这并没什么影响，因为ActionServlet 仅在启动初始化时使用这个值。然而，如果 servlet 是在运行时引用它，一个欺骗可以将它设置为0，来创建一个拒绝服务攻击。

使用ActionForm这样的可以影响系统状态的bean，或者一个ActionForm的一部分，就像直接传递输入字段到一个外壳脚本。 这就如同不能区分在误导的人群中哪些是狂欢纵乐的聪明人一样。

ActionForm 就像一个防火墙系统中的DMZ：它使你在数据被允许传递到应用的其它部分之前必须检查数据。

5.3. ActionForm设计推论

ActionForm的设计产生一些推论。一个ActionForm 可以:

- 与业务逻辑bean共享属性名称
- 最小化用户代码
- 封装助手方法
- 包含其它JavaBean

5.3.1. ActionForm可以共享名称

因为它可以和业务逻辑 bean进行交互, ActionForm通常可以使用业务逻辑bean中的一套相同的属性名称。通常，这些属性间并没有直接相关性，所以这不失为一个很好的实践方法。虽然form bean和逻辑bean最终都是表达相同的数据，但是它们在各自的生命周期内表达数据的不同视图。业务逻辑bean表达的是模型的状态。form bean表达的是状态的改变。ActionForm bean收集和校验输入。业务逻辑 bean处理捕获的数据并将新数据合并到模型中。

所以，虽然bean可以共享属性名称并创建一个公共协议，它们实际上并没有共享数据。已经被接受到模型中的数据是一回事，可以被接受到模型中的数据又是另一回事。

定义

消息协议（message protocol）(或称消息接口（message interface)）是一个基于反射的技术， 它允许在一个共享的公共层次场合下发现公共的命名约定来使对象共享信息。如果两个JavaBean具有能返回可比值的相同的属性名称，它们可以共享相同的消息协议。不管JavaBean是同一个类或者超类都无所谓。消息协议仅通过反射进行操作，也仅仅关心的是属性（“消息”）的名称。不同的JavaBean可以使用相同的Struts JSP，只要这些bean遵循相同的协议。如果bean 具有相同名称的属性，Struts 标签就可以找到正确的方来来调用。许多Struts 应用都会直接从业务层返回一个Bean，并将它传递给HTML form。当该页面被提交时，一个共享相同协议的ActionForm bean被用于捕获和校验参数。如果校验失败，控制会返回到同一个页面，标签将使用来自ActionForm bean的属性。每个都是不同的类，但是因为JSP 标签是基于反射，标签才不在乎是来自于哪个,只要bean是使用相同的消息协议 (或者说相同的方法名称集) [Johnson]

5.3.2. ActionForm可以最小化用户代码

在一个典型的部署中, 在ActionForm 中使用的代码直接相关于表现层特别是一个HTML表现层。定制的助手方法，以及特定的业务输出，通常在业务逻辑bean之中, 这样它们才可以被其它环境所重用。ActionForm和Action 类都是设计来作为一个适配器，鼓励将业务代码保持在业务层，表现代码保持在表现层。

5.3.3. ActionForm可以封装Helper

实际应用中, 开发人员发现根据其应用的业务规则来让ActionForm 格式化和校验数据是非常有用的。因为ActionForm 是一个JavaBean, 它可以传递输入到业务助手方法，来执行准确的格式化和校验，然后将结果返回表现层。

静态方法是有用的，因为操作通常是简单的过滤处理：

```
public String getTelephoneText() {
    return ContactBean.
              formatTelephone(this.telephoneText,getLocale());
}
```

这儿，ActionForm从业务层引入了一个 ContactBean 。大部分开发人员都同意控制层可以从业务层引入方法 (但其它方式则不行!)。

5.3.4. ActionForm可以嵌套其它bean

因为 Struts 标签扩展和自动组装机制都支持点号语法来从ActionForm访问其它bean。这是一个方便的方式，可以通过ActionForm来组装存在的bean。在 JSP 页面，你可以象这样引用一个嵌套的bean：

```
<html:text
        propety="values.telephoneText"
        size="14"
        maxlength="14"/>
```

这相当于是调用：

```
aForm.getValues().getTelephoneText()
```

浏览器将通过HTTP提交参数：

```
values.telephoneText=555-1234
```

自动组装机制然后会调用相应的方法：

```
aForm.getValues().setTelephoneText((String)request.
   getParameter("values.telephoneText"));
```

避免嵌套冲突

在进行嵌套bean时，请记住 ActionForm 必须准备好能处理各种HTTP 请求，并不是仅仅是来自于你控制的表单的提交。HTTP 请求是非常易于欺骗的。如果被嵌套的bean具有一个接受String 参数的方法，并且看起来像是一个JavaBean 属性，该方法就可能被从HTTP请求传递入一个值。

在Struts 1.0, ActionForm基类暴露了一个 servlet 属性链接到应用的ActionServlet。对 servlet 的存取需要处理一个MIME 请求。一个后果是，这个查询字符串可能被在运行时被传递到Struts 1.0 Action 来改变 TempDir属性：

```
?servlet.setTempDir=/whatever
```

这相当于是调用：

```
ActionForm.getServlet().setTempDir("/whatever")
```

幸好，实际上这并没有什么影响，因为这个属性仅在实例化时被使用。在Struts 1.0.1 以及后来版本中，将使用一个包装类（ wrapper class）来保护 ActionServlet对象:

```
public class ActionServletWrapper {
  protected transient ActionServlet servlet = null;
  public void log (String message, int level) {
    servlet.log(message,level);}
    public void log(String message) {
```

```
    servlet.log(message);
  }

  public String getMultipartClass() {
    Return servlet.multipartClass;
  }

  public void setServletFor(MultipartRequestHandler Object) {
    Object.setServlet(this.servlet);
  }
  public ActionServletWrapper (ActionServlet servlet) {
    super();
    this.servlet = servlet;
  }
}
```

包装类仅仅暴露了框架需要的属性。其它属性则保护起来，避免篡改。嵌套冲突强调了一个基本设计原理： **对象应该仅仅暴露其希望被改变的东西。**这是一个很好的设计原理，不仅仅是针对ActionForm，也包括其它JavaBean。

嵌套bean是一个很强大的技术但必须小心使用。

5.4. ActionForm 的风情

在Struts 1.1中，引入了两个基本的ActionForm的替代类: Map支持的mapbacked）ActionForm 和动态DynaActionForm。

5.4.1. Map支持（Map-backed）的ActionForm

许多大型 web 应用需要使用大量的属性。这些应用的开发人员开玩笑说，创建并维护ActionForm bean仅仅比声明一个字段， getter， 和setter多一点点工作而已。这些应用通常联合使用一些原本在其它平台一起使用的成熟的业务组件，现在被改编到web 应用中。因为bean 协议已经是良好定义的，所有这些应用真正需要的是，以最小的错乱保存好这些提交的字段。

当然，同时，并不是每个Form的属性都是简单类型的属性——也许十之八九是。

在Struts 1.1中, ActionForm可以支持map属性。和其它东西一起，这使得在一个ActionForm中混合使用map和常规的JavaBean属性成为可能。Map可以用来在没有事先定义相关的属性的情况下捕获提交的参数。每个字段只是Map中的一个项目。

从ActionForm的定义中，你可以定义这些方法来访问Map中的项目：

```
public void setValue(String key, Object value)
```

```
public Object getValue(String key)
```

然后你就可以在JSP中这样使用标签:

```
<html:text property="value(key)"/>
```

以及

```
<bean:write name ="formBean" property="value(key)"/>
```

这使你可以在没有其它方式的帮助下将Map用作一个简单属性。Map也有accessor，所以如果一个属性需要特殊的处理，你可以检查它，并插入一个助手方法。在下面的代码中：

```
public object getValue(String key) throws Exception {
// telephone number needs to be reformatted.
  if ("telephone".equals(key)) {
    return (object) getTelephone();
  }
  return getMap().get(key);
}
```

你需要为 telephone 属性定义accessor，但其它所有东西都可以直接传送，并存储在Map之中。很显然，调用 telephone 属性要多做一点工作。但大量的工作节省了，因为并不需要定义在ActionForm 使用的其它属性。因为ActionForm可能具有很多简单属性，这种节省可能是很可观的。

通常的做法是，页面引用Map的值和引用标准的属性是不同的。如果页面已经使用了标准的JavaBean 属性标签，JSP 代码需要更改，以使用Map值标签来代替。

如果HTML 标签引用的是：

```
<html:text property="telephone"/>
```

应该改为

```
<html:text property="value(telephone)"/>
```

存储在Map中的字段也没有象通常JavaBean属性一样来暴露。某些Java 开发工具可以使你直接使用JavaBean。这些工具并不将这些字段看成独特的属性，所以你不能完全使用这些工具的能力。

如果觉得将每个字段看成是JavaBean 属性是很重要的，而且你也需要最小化维护开销，还有一个可以考虑的就是使用DynaActionForm 类。

5.4.2. DynaActionForm

对一个大型应用来说，以常规方式声明简单属性可能会带来大量工作。要创建一个常规JavaBean 属性，你需要对field, getter, setter进行编码—大量的基础工作仅仅是*get这个或者*

set那个。 如果你仅仅是声明某个属性，而不用对每个属性单独进行设置可能会快些。

`DynaActionForm (org.apache.struts.action.DynaActionForm)` 就是设计来让你可以通过Struts配置文件来说明简单属性的对象， 就象我们在程序清单5.2种所见一样。DynaActionForm 是基于Jakarta Commons [ASF, Commons]的DynaBean组件。这是一个聪明的对象，它可以将字段存储在一个内部map之中，但把它们作为标准的JavaBean 属性来暴露。你可以在使用ActionForm 的任何地方使用DynaActionForm。你也可以用DynaActionForm 替代常规的 ActionForm 而不用改变任何现有的Java 或者JSP 代码。

还有什么

这里有个告诫：缺省情况下, DynaActionForm的`reset` 方法将设置所有的字段为其初始值。对一个常规ActionForm来说，开发人员可以决定哪些字段将受到reset方法的影响。许多开发人员将它们全部初始化，所以在大多数情况下，将ActionForm 换成DynaActionForm其行为没有什么改变。

5.5. 关于ActionForm的疑问

在开源社区，很少有设计是不受争议的。下面是一些开发人员关于Struts ActionForm 设计的常见问题。

5.5.1. 为什么ActionForm不仅仅是一个Map?

这里有一些用于设计JavaBean 和Map的准则，如下表：

理由	解释
封装性	绝大多数情况下，属性会返回简单值，但有时又有些值的确需要被生成
扩展性	Map不能被被扩展，以提供适合的方法，比如validate和reset方法
自省	一个清晰定义属性的bean可以被GUI接口和其它工具所使用

表 5-1 JavaBean 和 Map 的设计理由

Struts 1.1 对两者都提供了最好的设计。你可以定义自己的Map来存储字段或者直接使用DynaActionForm 来自动声明属性。(5.4.2)

5.5.2. 为什么ActionForm不是一个普通 JavaBean?

虽然一些JavaBean 也可以用来和Struts JSP 标签一起组装HTML表单，但也需要一个具有一些已知方法的类来对提交后的输入进行管理。

当通过HTTP来接受输入时，校验是必须的。某些从web层进来的数据在使用前必须经过检验，如不适合，坚决拒绝。

ActionForm 具有一个 validate 方法，可以被 ActionServlet 调用，以确保数据输入是正确的。其它业务校验可以在此基础上随后进行，但是具有ActionForm 的初始校验将使得设计成为一个健壮的系统。

另一个问题是选择框（checkboxe）。如果一个选择框没有被选择，浏览器将不会送出什么东西。有就是真，无就是假。我们所要做的是记住在组装它之前，将其设置为false。这样，

如果有选择则返回属性 true。否则，保持false不变。为解决类似问题， ActionForm 有一个内建的reset 方法。因为这些方法，ActionForm可以正确处理输入的数据，而旧式的普通 JavaBean 则不行。Form bean 不得不提供这两个方法给框架使用。

5.5.3. 为什么ActionForm不是一个接口?

这问题问得好。

总的来说，使用一个基类而不是接口可以使我们能扩展这个类而不是仅仅使用一个预定义的类。ActionForm 最常用的是暴露来自于业务对象的属性。如果它是一个接口,我们也许可以直接使用业务对象并且避免通过一个分离的ActionForm 对象来传递数据。

这看起来似乎很好，但实际上，这样做仍然有一些障碍。下表列出了这些问题的总结。

原因	解释
扩展性	添加一个方法到ActionForm的接口（在未来的Struts版本中）可能会破坏应用的Form Bean，如果它严重依赖于前一版本的接口的话。作为一个基类，我们可以添加一个具有默认行为的新方法。现有的子类可以简单地继承这个默认行为
正确使用	如果一个ActionForm有一个接口，开发人员可能试图使用现存的数据bean作为ActionForm使用。ActionForm应该被视为是控制层的一部分，提供的API应该鼓励这样使用它。ActionForm相关的核心API契约必须是真实反映用户提交的输入内容，而不管用户提交的东西是否是确实有效的。这就允许用户能纠正输入的数据并重新提交。大多数对象都不是设计来缓冲不正确的数据的，并且许多输入都是不可更改的。
性能	一些数据Bean，特别是EJB，其设计可能会开始一个事务，并且锁定一些资源，比如基本的数据库资源。应用应该具有很好伸缩性，因为直至输入起码在形式上看起来有效之前，数据库事务会处于延迟状态。
安全性	Struts的自动组装首先设置属性，然后对它进行校验。在一个新的输入值传递到一个对象的属性时，它可以设计来可以改变系统的状态。这时，一个敌意的用户就可能从浏览器的地址栏直接输入一个值给ActionFrom的属性，从而进行非授权的状态改变，即便你从来不曾希望从浏览器改变属性。ActionServlet并不知道那一个属性是要在HTML Form中使用，它只知道它在请求中发现了什么。

这还有一个问题: “为什么业务 bean 不是一个接口?” ActionForm 可以实现业务接口,而且，一旦通过校验，将传递到业务层。

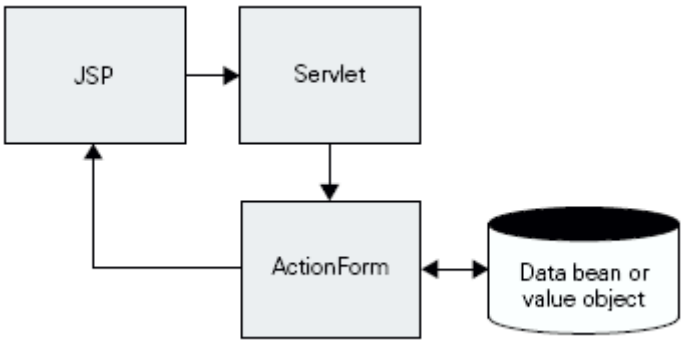
传输数据从ActionForm到业务层是一个非常重要的步骤，我们在下一节说明。

5.6. 小结ActionForm

Struts 标签扩展有助于根据ActionForm组装 HTML 控件。ActionServlet根据HTTP 请求组装ActionForm 。

但是，如图5.4, 我们仍然需要越过最后一英里。为了数据能完成其生命周期，我们不得不在

ActionForm和业务对象间传递值。



图表 5-4 数据的生命周期循环：Data bean 组装 ActionForms；ActionForms 组装 JSP；ActionServlet 组装 ActionForm；而 ActionForm 组装 data bean。

Struts 开发人员使用一些策略来在层间传递值。在这一节中，我们将讨论最通用的策略。业务层的需求通常决定项目应该采用的最好方法。许多web 开发人员被要求为业务方法使用一些预先存在的层，这可能会限制它们的选择。有时，整个应用都需要重建，这样开发人员就有一些富裕的选择，可以选择它们最喜欢的方案。在我们的讨论中，我们使用词语传输对象 （*transfer object*）来描述用来从一个地方到另一个地方传递数据的JavaBean 。这种方法通常用在一个使用 Enterprise JavaBean 的环境中，以减少应用需要的对远程数据库的调用次数。这种办法是，将多个属性绑定在一个单一的JavaBean值中。Bean中的属性可以和多个表现关，但它们是在远程数据库中处理。

你的应用或许根本没有使用传输对象，但可能会使用某种 “数据 bean”来访问本地数据系统。 这种方法其实是一样的工作原理。我们讨论的策略总结在下面的表中：

策略	说明
实现业务层接口	ActionForm 实现了业务层接口，所以可以直接使用
嵌套可改变的值对象	具有独立属性的值对象是ActionForm的一个成员属性
设置不可改变的值对象	由Action调用构造器或设置器
设置可改变的值对象	Action调用一个或者多个设置器
使用工厂方法	一个helper方法封装，实例化和组装一个业务bean.
传递一个Map	一个数据Map由Action 传递给业务层实体
通过反射数据传递值	通过匹配属性名称从一个bean 传递到其它Bean

使用适配器类	一个特殊的类从ActionFrom复制属性到几个业务层实体

为帮助你选择流程，我们在每一个策略中包含了“结论”和“转换数据类型”两个内容。

5.6.1. 实现业务层接口

业务层通常为其中的bean定义接口。当一个业务层接口可用时，你可以使你的 ActionForm 实现该接口，以便它可以直接传递到业务方法:

```
public class ArticleForm extends ActionForm
  implements ArticleBean {
  // ...
}
```

然后，在你的 Action中，你就可以直接传递经过校验的ActionForm到那些需要业务层类型的方法:

```
articleModel.save((ArticleBean) form);
```

转换数据类型

在某些时候，Web客户端提交的许多基于String的输入必须转换为业务层需要的原生类型。不幸的是, JavaBean 规范不允许属性的 getter 和 setter方法被重载。如果你试图在ActionForm中这样做， ActionServlet自动组装机制就会抛出一个异常。

一个有用的技术是使用一个具有可预知名称的助手方法来按需要转换属性类型:

```
private String keyDisplay = null;

public String getKeyDisplay {
  return keyDisplay;
}

public Integer getKey() {
  return new Integer( getKeyDisplay() );
}
```

标签扩展可以这样引用keyDisplay:

```
<html:text property="keyDisplay"/>
```

但是 Action 或者业务层方法都可以简单的引用原方法：

```
ArticleBean articleBean = (ArticleBean) form;
articleModel.update(articleBean.getKey(),articleBean);
```

结论

- ActionForm 成为业务 API 无缝的一部分；
- 必须注意输入在业务方法中使用前必须经过校验;
- 非字符串属性必须事先经过转换。

5.6.2. 嵌套可变值对象

如果你足够幸运，使用的事仅仅具有String 和 boolean类型属性的值对象，那么你就可以使你的值对象成为ActionForm的一个属性。而值对象中的属性则可以通过点号语法来进行引用，像这样：

```
values.telephone
```

这相当于是调用

```
getValues().getTelephone();
```

和

```
getValues().setTelephone(values.telephone);
```

转换数据类型

不适用。这个策略要求不需要进行转换的String 和 boolean 值。

结论

- 这个策略快捷、易于实现；
- 这个策略不适用于非文本类型，比如Integer 或者 Date；
- 如果值对象拒绝没有良好格式化的数据时，这个策略可能不能很好工作；
- 这个策略将表现层绑定到特定的值对象。如果值在对象中发生改变，表现代码也需要反映这种改变；

注 前两种策略，实现业务层接口和嵌套可变值对象，将自动传输数据。其它策略则需要Action协助触发传输。接下来的策略的代码片断将可以在Action 类中看到。

5.6.3. 设置不可变值对象

如果你被提供以一个使用批量构造器的不可变值对象，你可以这样来组装它：

```
ArticleForm aForm = (ArticleForm) form;
ArticleBean aBean = new ArticleBean{
  aForm.getArticleKey(),
  aForm.getContributor(),
```

```
  aForm.getconstructor(),
  aForm.getTitle(),
  aForm.getContent()
};
```

批量setter方法也看起来是一样，除非bean 已经被创建，并且随后调用setter方法。如果涉及到大量的参数被调用，这有助于将它们分组，然后为每个组定义它们的setter方法 (如果这是个选择的话)。这就是经典的分而治之策略：

```
ArticleForm aForm = (ArticleForm) form;
ArticleBean aBean = new ArticleBean();
  aBean.setRecordHeader(
    aForm.getArticleKey(),
    aForm.getEditor());
  aBean.setPeople(
    aForm.getContributor(),
    aForm.getconstructor());
  aBean.setText(
    aForm.getTitle(),
    aForm.getContent());
```

当然, 这样做是值得的,每个块都带了一些参数,但你获得了你想要的方法。有时,多个setter可能和部分修改相关，即是说，文本字段可以被修改而人员字段却没有触及。如果数据是通过一个远程链接来传送的，注意仅传送你想要更改的数据是非常重要的。

转换数据类型

如果属性类型需要转换，你可以在数据被传输的时候或者放入你的ActionForm的助手方法时进行。 因为数据可能是已经经过校验并且可以进行转换的， ActionForm 助手就比较乐观，认为转换会成功。例如：

```
Integer getArticleKeyInteger() {
  return new Integer(this.getArticle())
}

ArticleBean aBean = new ArticleBean(
  aForm.getArticleKeyInteger(),
  aForm.getContributor(),
  aForm.getconstructor(),
  aForm.getTitle(),
  aForm.getContent()
```

```
)
```

当然，你也可以在这个语句中使用 try … catch，以便在不希望的事情发生时，抛出异常：

```
try {
  ArticleBean aBean = new ArticleBean(
  aForm.getArticleKeyInteger(),
  aForm.getContributor(),
  aForm.getconstructor(),
  aForm.getTitle(),
  aForm.getContent())
}
catch (Throwable t) {
  errors.add(ActionErrors.GLOBAL_ERROR,
  new ActionError("error.conversion"));
}
```

结论

- 这个策略提供类型检测；
- 这个策略可以使用任何值对象；
- 如果属性易变这个策略将导致较高的维护性；
- 这个策略强烈地和一个具有特殊 ActionForm 的 Action 类耦合；
- 有很多属性的值对象通过这种方式非常难以管理。

5.6.4. 设置可变值对象

如果值对象没有批量setter，并且因为某些原因不能添加新的方法，这就会显得混乱：

```
ArticleBean aBean = new ArticleBean();
aBean.setArticleKey(aForm.getArticleKeyInteger());
aBean.setContributor(aForm.getContributor());
aBean.setconstructor(aForm.getconstructor());
aBean.setTitle(aForm.getTitle());
aBean.setContent(aForm.getContent());
```

结论和数据类型转换同上一节。

5.6.5. 使用工厂方法

如果 ActionForm 值需要传输到另一个值对象，你可以将传输封装在ActionForm 中而不

是将处理流程暴露给Action类。

如果必须创建值对象，一个好办法是使用工厂方法，它会实例化，组装并返回一个值对象。下面是一个示例方法，可能会在ActionForm中找到:

```
public ArticleBean getArticleBean() {
  ArticleBean aBean = new ArticleBean(
    this.getArticleKey(),
    this.getContributor(),
    this.getconstructor(),
    this.getTitle(),
    this.getContent()
  );
  return aBean;
}
```

另外，你可以直接传递一个已有的值对象:

```
public void setArticleBean(ArticleBean aBean) {
  aBean.set (
    this.getArticleKey(),
    this.getContributor(),
    this.getconstructor(),
    this.getTitle(),
    this.getContent()
  );
}
```

转换数据类型

对这个策略关于数据转换要考虑的因素与前两节相同。这个策略只是简单地把代码从Action移到ActionForm中。

结论

- 这个策略将 ActionForm 绑定到业务层类型；
- 这个策略简化了 ActionForm；
- 这个策略允许多个 Action 传输数据而不需要代码复制。

5.6.6. 传递Map

值对象通过Map(java.util.Map)来传递属性并不是不常见。如果 ActionForm 属性与值对象属性匹配，传输数据就会非常容易。

如果你有一个接收 Map的值对象，有两种办法可以使ActionForm使用Map,这取决于你使用的Struts版本，是Struts 1.0 还是Struts 1.1 及以后版本。

Struts 1.0

BeanUtils类包含一个describe 方法，它可以返回一个JavaBean的公共属性的一个Map。它也提供一个相应的populate 方法，可以从一个Map组装JavaBean 的公共属性。在下面的例子中，我们假定我们假设的值对象具有表达其值的 setMap 和 getMap 属性。

我们先从Form bean到值对象传递：

```
Map map = BeanUtils.describe(form);
bean.setMap(map);
```

而从值对象到Form bean则可以这样进行：

```
Map map = bean.getMap();
BeanUtils.populate(form,map);
```

(为了清楚起见，我们将Map视为一个中间变量。在你的代码中你可以结合使用这两个语句)

如果因某些原因属性名称不匹配，ActionForm 还可以给定别名（alias） 属性：

```
public String getKey() {
  return getArticle();
}
public String getAuthor() {
  return getconstructor();
}
```

或是 ActionForm可以给定一个定制方法来返回相应的 Map:

```

public class ArticleForm extends ActionForm {
  // ...
  public Map describe() {
    map = new HashMap();
    map.add("key",this.getArticle());
    map.add("author",this.getconstructor());
    // ...
    return map;
  }
}
```

在下面的情况下，你可以调用 ActionForm的 describe 方法而不是BeanUtil的:

```
bean.setMap(form.describe());
```

Struts 1.1

如果你的模型接受Map (java.util.Map)的数据输入，极有可能它也以Map形式返回数据。从 Struts 1.1开始，你可以结合使用Commons BeanUtils [ASF, Commons] 类的强大功能和Struts 的点号语法。这使得可以很容易地使用Map来存放ActionForm 属性。

这个技术十分简单。首先，添加一个属性到 ActionForm 中来访问Map:

```
private Map map = null;

public void setMap(Map map) {
  this.map = map;
}

public Map getMap() {
  return this.map;
}
```

然后，添加一个属性来访问Map中的元素:

```
public void setValue(String key, Object value)
              throws Exception {
   getMap().put(key,value);
}
public Object getValue(String key) throws Exception {
   return getMap().get(key);
}
```

在 Struts 1.1 JSP 标签 (第10章)中，你可以象这样访问Map中的元素：

```
<html:text property="value(key)"/>
```

以及

```
<bean:write name ="formBean" property="value (key)"/>
```

这里 key 是属性的名称。

如果业务层值对象已经使用了Map，你可以通过使用各自的getMap 和 setMap 方法 (或者等同的方法)来传递数据:

```
form.setMap(bean.getMap());
bean.setMap(form.getMap());
```

转换数据类型

如果业务层期望一个原生类型的Map而不是String类型, ActionForm 的助手方法被要求将String转换为原生类型，并返回一个修改过的Map:

```
public Map getTypedMap() {
  Map map = this.getMap();
  String keyString = (String) map.get("key");
  Integer keyInteger = new Integer(keyString);
  map.put("key",keyInteger);
  return map;
}
```

结论

- 这个策略在Map已经用来传输数据的情况下，将导致和业务层很好的集成；
- 因为如果没有相应项目，Map将返回null, 所以需要一些额外的校验代码来监视哪些缺失的参数。

5.6.7. 通过反射传递值

如果你使用的是一个比较新的JVM, 将 ActionForm 数据传输到其它JavaBean 的一个很好的方法是使用反射。起初，反射机制会造成性能损失，但每个JVM都减少了这个损失。在Sun的 Java 1.3以及以后的版本，性能差异已经可以忽略不计。

你可以添加一些简单的方法到ActionForm 基类中，使其易于使用反射机制来与其它bean之间传出和传入数据，就象在程序清单5.3使用的一样。

这些方法其实是对框架其它地方使用的BeanUtils类方法的包装。ActionServlet 使用BeanUtils 来从HTTP 请求中组装ActionForm。

```
public Map describe() throws Exception {
  try {
    return BeanUtils.describe(this);
  } catch (Throwable t) {
    throw new PopulateException(t);
  }

public void set(object o) throws Exception {
  try {
```

```
    BeanUtils.populate(this,BeanUtils.describe(o));
  } catch (Throwable t) {
    throw new PopulateException(t);
  }
}

public void populate(object o) throws Exception {
  try {
    BeanUtils.populate(o,this.describe());
  } catch (Throwable t) {
    throw new PopulateException(t);
  }
}
```

清单 5-1 数据传输对象方法

1.0 vs 1.1

Struts 1.1所使用的Commons BeanUtil 包比原来Struts 1.0提供了更好的类型转换能力。如果你正使用反射传递你的数据，我们推荐你导入Commons BeanUtil 包，而不管你的Struts是使用哪一个版本。最全面的数据传递方法是 `BeanUtils.copyproperty`。下面演示如何使用copyproperty来从其它Bean中组装你的bean：

```
BeanUtils.copyproperty(this,source);
```

copyproperty方法将自动在两个方向上对原生类型转换应用字符串 String 类型。新的 copyproperty使在你的ActionForm和 业务 beans之间的“往返旅行”更加容易:

```
BeanUtils.copyproperty(mybusinessBean,myActionForm);
myBusinessOperation(myBusinessbean);
BeanUtils.copyproperty(myActionForm,myBusinessBean);
```

这个代码片断传递myActionForm中的String 属性到myBusinessBean中的原生类型，允许业务操作来更新值，然后将原生类型的值统统传递回myActionForm中的字符串。真漂亮！

你也可以用BeanUtils注册你自己的转换器来处理特殊的类型转换，请参考Commons 的站点获取更多详细内容 [JSF, Commons].

转换数据类型

BeanUtils 方法可以在字符串类型和原生类型间转换数据，所以当你实现其它接口时，并不需要写很多“桥接”方法 (虽然，非原生类型将是很大的问题)。

当使用 BeanUtils 方法来传递数据时，所有的东西都必须通过原生类型来进行传递。这对 ActionForm 来说并不是问题，因为它仅仅使用 Strings 或者 boolean类型。因为试图在

```
    String setCurrent();
```

和

```
    Date getCurrent() ;
```

间传递数据，对业务bean那一方来说则是个问题。另外还有其它Struts 1.0 BeanUtils 类不支持的非原生数据。

一个解决方案是在业务Bean那一方提供转换方法，将Date 转换为String 然后传回来。下面是一些方法，可以将 String 转换为 Timestamp 或者转换回来。这些方法都应该是业务bean的成员—而不是ActionForm的成员。通过将它们放入业务层，他们对其它也需要处理字符串的组件有效：

```
public String getTicklerDisplay() {
  Timestamp tickler = getTickler();
  if (ConvertUtils.isNull(tickler)) return null;
    return tickler.toString();
}

public void setTicklerDisplay(String ticklerDisplay) {
  if (ticklerDisplay==null)
    setTickler(null);
  else try {
    setTickler(Timestamp.valueOf(ticklerDisplay));
  } catch (Throwable t) {
    setTickler(null);
  }
}
```

而在业务bean的其它地方则是实际的Timestamp的getter和setter，实际的Timestamp存储在数据库中：

```
private Timestamp tickler = null;
public Timestamp getTickler() {
  return (this.tickler);
}
public void setTickler(Timestamp tickler) {
  this.tickler = tickler;
}
```

在 ActionForm 中，我们可以简单地使用String 属性，但是名称前要加上Display前缀：

```
private string ticklerDisplay = null;
public String getTicklerDisplay() {
  return this.ticklerDisplay;
};

public void setTicklerDisplay(String ticklerDisplay)
  this.ticklerDisplay = ticklerDisplay;
};
```

当使用反射进行设置和组装时，这个机制等于是调用：

```
bean.setTicklerDisplay(form.getTicklerDisplay());
```

或者

```
form.setTicklerDisplay(bean.getTicklerDisplay());
```

这取决于我们是在做哪个方向的转换。业务方法处理了String 和 Timestamp间的转换，所以我们可以在ActionForm 像通常一样使用String 属性。

STRUTS TIP 可以在你的业务Bean中使用显示助手来自动转换复杂类型。通过使业务 bean 处理与字符串之间的转换，你可以确保业务层的需求停留在业务层之上，并且可以在其它平台上重用它们。

在实践中，你的应用可能需要用到这些方法，因为我们很少将数据显示为其原生类型。即使没有转换的问题，某些诸如下面的 getDateDisplay 类似的方法也通常需要被用来对值进行本地化和用户友好方面的渲染处理。

```
public String getDateDisplay() {
  if (this.dateDisplay==null) return null;
  DateFormat dateFormatter = DateFormat.getDateInstance(
    DateFormat.default,
    this.getLocale()
  );
  return dateFormatter.parse(this.dateDisplay);
}
```

对一些类似于timestamp的值，你也许需要将它们分解为单个的值到几个字段之中，以便每个部分可以在HTML form的下拉列表中进行选择。

下面的 setTicklerDisplay 方法从Timestamp字符串中提取了几个部分，以便它可以被不同的控件使用：

```
public void setTicklerDisplay(String ticklerDisplay) {
```

```
  this.ticklerDisplay = ticklerDisplay;
  boolean longEnough = ((this.ticklerDisplay!=null) &&
          (this.ticklerDisplay.length()>11+2));
  if (longEnough) {
    // Snag Components: YYYY-MM-DD HH:MM
    setTicklerYear(this.ticklerDisplay.substring(0,0+4));
    setTicklerMonth(this.ticklerDisplay.substring(5,5+2));
    setTicklerDay(this.ticklerDisplay.substring(8,8+2));
    setTicklerHour(this.ticklerDisplay.substring(11,11+2));
    // Parse AM/PM/EV
    Integer hour = null;
    try {
      hour = Integer.valueOf(getTicklerHour());
    }
    catch (Throwable t) {
      hour = null;
    }
    int tod = 0;
    if (hour!=null) tod = hour.intValue();
    setTicklerTod(AM);
    if (tod>12) setTicklerTod(PM); // after 1pm
    if (tod>16) setTicklerTod(EV); // after 5pm
  }
}
```

在很多情况下，Display 方法将承担双倍的义务。它们格式化Strings值然后传递给ActionForm。ActionForm 方法然后才可以按它需要的格式显示这些值。

结论

- 代码基的规模减小了；
- 总体维护成本减小了；
- Action 类和其它组件间的耦合减小了；
- 此方法可能要求创建一些桥接方法或类；
- 某些依赖于日期信息的开发人员可能不愿使用反射。

5.6.8. 使用适配器类

很多时候ActionForm bean和模型（业务）bean是非常类似的。而其它时候，它们仅有一点儿相似。一个ActionForm 内的字段可能需要传递给几个不同的模型bean。或者ActionForm的属性名称可能需要有别于业务对象使用的属性名称。在这种情况下，一个适配器类可能有助于将业务对象的方法映射到ActionForm 的属性。

适配器和ActionForm 将共享一个或者几个具有相同方法体的方法。适配器类设计来就象一个或者几个业务对象的包装类。适配器中的getter和setter方法调用业务对象中的相应方法。然后，适配器便可以用来代替业务bean (作为代理/proxy 或者代表/delegate):

```
// Get data into ActionForm
DataBean dBean = new DataBean();
dBean.execute(something);
AdaptorBean aBean = new AdaptorBean(dBean);
aBean.populate(form);
// Fetch data from ActionForm
AdaptorBean adaptor = new AdaptorBean(new DataBean());
BaseForm actionForm = (BaseForm) form;
adaptor.set(actionForm);
DataBean model = (DataBean) adaptor.getBean();
Data.execute();
```

我们已经讨论过的数据传递技术都可以用在适配器类中。适配器的关键之处在于它封装了ActionForm 和业务类之间的差异。这在业务模型非常复杂的时候非常有用，因为适配器可以明确地去掉Action对业务实现的耦合。

转换数据类型

适配器可以封装一些现有的数据转换方法或者根据需要实现新的转换方法。

工作区

如果你需要值对象是在其它任何地方都是不变的，你可以使同一个对象同时实现可变和不变两种接口。

业务和数据层将通过只读接口使用这个对象，Action则可以使用可变接口，但它们都可以共享下面的数据字段。

结论

- 类是针对应用特定的，仅有一丁点儿重用性；
- 这个策略增加了应用中对象的数量；
- 这个策略对业务和表现层的改变是敏感的；
- 这个策略可以避免来自于其它层的改变（有利于层间的去耦合）；

5.7. BaseForm

Scaffold 包提供了一个基本的ActionForm ，实现了本书讨论的几种技术(org.apache.struts.scaffold.BaseForm)。这个类包括处理本地化，分派，以及管理自动组装的方法。下表是BaseForm 的方法。

分类	方法

SessionLocale	`public void setSessionLocale(Locale locale);`
	`public Locale getSessionLocale();`
	`protected void resetSessionLocale (HttpServletRequest request);`

Dispatch	`public void setDispatch(String dispatch);` `public String getDispatch();`
Mutable	`public void setMutable(boolean mutable);` `public boolean isMutable();`

Autopopulation	`public Map describe() throws Exception;`
	`public void set(Object o) throws Exception;`
	`public void populate(Object o) throws Exception;`
	`public Map merge(Object profile) throws Exception;`

STRUTS TIP 如果你的应用使用几个 ActionForm类，可以定义一个基类来包括一些在应用中可能通用的属性或者方法。

5.7.1. SessionLocale

缺省情况下，ActionServlet会为在会话上下文中的用户自动创建一个Locale 对象。这些方法将用来帮助你管理Locale 对象:

```
public void setSessionLocale(Locale locale);
public Locale getSessionLocale();
protected void resetSessionLocale(HttpServletRequest Request);
```

resetLocale 方法将被BaseForm的 reset 方法调用。它从会话中重新取得Locale 对象，以便ActionForm可用。如果你有一个 HTML 控件要改变locale，你可以在validate 方法中进行改变。 因为这里Locale 对象引用到 session 对象，改变将通过用户会话持续保持。

关于 Struts 本地化的更多内容请参考第13章。

5.7.2. 分派（Dispatch）

Struts 开发人员通常使用同一个Action处理相关的操作。一个选择不同操作的通常做法是在HTML表单中使用隐藏属性。BaseForm 的dispatch 属性也可用作这个目的：

```
public void setDispatch(String dispatch);
public String getDispatch();
```

5.7.3. 自动组装

ActionServlet 会处理ActionForm的缺省自动组装。这套方法帮助Action 对象履行其它数据传递任务。describe 方法 返回一个ActionForm bean的Map。

方法从另一个JavaBean中组装ActionForm。populate方法设置另一个JavaBean的属性来匹配这个bean:

```
public Map describe() throws Exception;
public void set(Object o) throws Exception {
public void populate(Object o) throws Exception {
protected Map merge(Object profile) throws Exception {
```

merge 方法稍微有点复杂。其潜在意思是，一般情况下，你的应用中可能在会话范围内有一个profile bean，它用特定的设定来覆盖缺省设置。merge 方法是你可以结合用户特定的设置和ActionForm中的标准设置到一个统一的Map中。

所以，如果 ActionForm具有一个sessionLocale 属性而且 profile bean 也具有一个 sessionLocale 属性，profile bean的设定将在Map中返回。如果profile bean 没有 sessionProfile 属性，或者这个属性为null，那么Map中返回的则是ActionForm的属性。

5.7.4. BaseMapForm

在 Struts 1.1 或者后面版本中，我们可以使用Map作为 ActionForm 属性。Map中值的引用和常规的JavaBean 属性的引用是不同的:

```
// form.getValue(key);
<html:text property="value(key)"/>
```

但这通常是一个公平的对灵活性的折衷，即Map可以引入到你的ActionForm中。

Scaffold 中的BaseMapForm (org.apache.struts.scaffold.BaseMapForm)提供了标准方法来从 Map 属性中存储和检索字段。

这个方法扩展了BaseForm 因此也提供了该类的所用功能。下表是BaseMapForm 方法：

分类	方法

Value Map	public void setValue(String key,
	public Object getValue(String
	public void setValues(Map values);
	public Map getValues();

Value Map

为了使用预装的Map来初始化一个ActionForm，可以使用 setValues方法。为了返回描述当前BaseMapForm 值的Map,可以调用 getValues 方法。为了添加或者改变一个特定的值，可以使用 setValue 和getValue 方法。

5.8. 小结

通常，ActionForm是一个惹恼Struts开发人员的源头。这可望在Struts 1.1中的改善，因为Map和 DynaBean可以减小ActionForm 的维护。

在这一章，我们讨论了ActionForm的原理，有助于明确它在框架中所处的角色。我们也探讨了几种在ActionForm和业务对象间传递数据的方式。

Scaffold BaseForm 类支持本章所述的几种数据传递策略,可以作为你的ActionForm基础的更好的选择。

ActionForm描述了什么东西可以进入你的应用。在下一章，我们来看看ActionForward，则描述了“人们”可以从什么地方进入你的系统。

6.连线 ActionForward

本章内容

- 理解ActionForward 最佳实践
- 使用运行时参数
- 使用动态转发

Inanimate objects classified scientifically into three major Categories—those that don't work, those that break down and those that get lost.

—Russell Baker

6.1. ActionForward做什么

在整个应用中，许多组件，比如 Action 对象，都会问这样的问题："OK, 操作成功了，然后呢？" Struts配置将使用一个连接到这个 "success" 的ActionForward作为该操作的响应。然后ActionForward 将会回传到ActionServlet。ActionServlet 使用其中的路径将控制传递给另一个地方。没有一个其它组件知道路径的细节；唯一知道的就是 ActionForward 说，这就是我们接下来要去的地方。

注 从技术上来说， path 仅仅是ActionForward保存的URI的一部分。URI 也可以包括一个带有你的应用可以使用的参数的查询组件。但是，这里我们所说的path，指的是完全的URI。

如果每个链接都通过一个forward，则ActionForward 记录了一个应用的工作流。理想情况下，ActionForward应该被用在应用的每一个入口点—即那些一个页面连接到其它页面的的地方。

1.0 VS 1.1 在Struts 1.1中, ActionForward 子类化了 ForwardConfig 类(org.apache.struts.config.ForwardConfig)并添加了向后兼容所需的 API 方法。ActionForward已经不赞成了，不过在以后的发布版中其层次体系如何还没有决定。现在，我们将引用ActionForward 类，但你应该注意到，在Struts 1.1中，所有的forward 属性都实际上是在ForwardConfig 超类中定义的。ActionForward 可以工作在两个版本之中。

ActionForward 代表一个应用的URI [W3C, URI]。虽然这个属性被称为 " 路径 " *path*，ActionForward可以包含完整的URI，并且也可以包含一个查询组件。这意味着你也可以使用这样的字符串

```
path="/do/item?dispatch=edit"
```

在你的ActionForward 路径中。如果 /do/item 是一个Action类，其 ActionForm具有一个属性名称叫 `dispatch`，URI将令Struts 将`dispatch` 属性设置为 `edit`。当然，更复杂的查询字符串也可以工作得很好。下面的URI：

```
path="/do/item?dispatch=edit&amp;key=17"
```

就会让Struts 设置`dispatch` 属性为 edit，设置 `key` 属性 为 17。

注 因为 path是一个 URI，我们将 ampersand编码为 &。Ampersands是URL中查询组件的一个限制字符。

6.2. ActionForward如何工作

ActionForward 实际上是一个简单但富有效率的包装类。其基类仅有四个属性：name, path, redirect, 和 classname，如表6.1所示。

属性	描述
name	指定 ActionForward 的逻辑名字。其它组件可以通过此名称来引用该 ActionForward。以便其他属性可以很容易地被修改。
path	指定 ActionForward 的 URI。URI 是 web 应用通信的重要方式。
redirect	如果设置为 true，控制被重定向。缺省设置为 false ，而且几乎是更好的选择。
className	可选。当实例化该 Forward 时，用来指定一个 org.apache.struts.action.ActionForward 类的子类名称。 [Struts 1.1].

6.2.1. 转发和重定向

HTTP 协议本身就具有一个内建的 “重定向（redirect）”命令，服务器可以用它来将控制从一个页面发送到另一个页面。Java web 开发人员在他们的方法代码中则有另一个选择，调用转发（forward）。这二者都在各自的方式上很有用处。

- *Forward* 将保持存储在HTTP 请求和请求上下文中的所有东西。仅可用在同一个应用中。
- *Redirect* 指示web 客户端进行一次新的HTTP 请求。请求的资源可以在同一个应用中，也可以不在。

转发请求

Java Servlet 容器有一个内部机制，可以使一个请求被传递或者称被转发（*forwarded*）到另一个组件。这将允许一个请求在产生其对应的响应之前被多个组件进行处理。通过这个方法，组件可以在请求上下文中添加和修改对象，甚至修改请求参数。

当一个Action 对象返回一个ActionForward时，servlet 就为这个ActionForward路径创建一个RequestDispatcher。如果路径包含查询字符串，查询字符串参数将成为被转发请求的一部分。然后你就可以使用请求对象的方法来重新取得参数：

```
Object parameter = Request.getParameter("ParameterName");
```

如果这是一个新的参数，它也被传递到随后的转发之中。如果参数已经存在，在后面的转发期间，新的值将覆盖原有的值。然后，旧的参数再次浮出水面。关于servlet如何处理请求，参见 Servlet 规范 [Sun, JST] 中的“Dispatching Requests”。

不要把参数和请求属性（Attribute）搞混淆。参数是 HTTP 请求的一部分。属性则是保存于请求上下文中的，它是 Java 提供的东西。HTTP请求的元素通常可以通过 Java 请求上下文访问，所以有时候它们看起来很相似。 HTTP 请求的参数用来组装 ActionForm (第5 章)。典型情况下，ActionForm 将包含应用期望的所有输入。Struts 开发人员不需要直接与请求参数打交道。他们使用请求属性来完成其大部分工作。请求属性在整个请求的转发期内都有效，但如果被重定向后，它便会消失。

重定向请求

当 ActionForward的 `redirect` 属性设置为 true时， ActionServlet 将向浏览器发送一个 HTTP 响应，告诉浏览器重新提交一个新的请求到一个新的路径。原来的请求参数不再保留，原来的请求上下文也消失了。新的HTTP 请求仅包含在 ActionForward的 `path` 属性里包含的参数，如果有的话。URI在其被送到客户端前被重新编码，并且如果路径在同一个应用中，用户的会话上下文也会被维护。但是重定向则总是会清除请求上下文。

6.3. 全局和局部转发

Struts 配置对ActionForward提供两个级别嵌套:

- *Global（全局）* ActionForward 对整个应用中的所有 Action 对象均有效；
- *Local（局部）* ActionForward 则在 ActionMapping 元素中定义。仅对那些在这个 ActionMapping 中调用的 Action 对象有效。

从Action 对象内部来看，forward 通常这样来选择：

```
ActionForward forward = mapping.findForward("continue");
```

当ActionServlet 调用 Action时，Action的 mapping 对象要传递给它。mapping 对象包括一个局部转发的列表，以及一个到全局转发的链接。 Mapping的`findForward` 方法会首先检查局部转发列表。局部转发查找失败才会进行全局转发的查找。

如果在两个地方都没有找到相应的 forward ，`findForward` 方法将返回null。如果不幸发0生了这种事情，因为Action 类和Struts配置不一致，Action 对象就会返回一个空（null）ActionForward，这样浏览器就会报告错误。

理想情况下，你所使用的所有全局变量都应该定义为`String`常数，以避免某些误解和拼写错误。这也使得它们可以用在JSP中。(但不幸的是, 不能用在XML配置文件中)。Jakarta Commons Scaffold [ASF, Scaffold] 包在其Token类中定义了多个常用的全局变量：

```
public static final String CANCEL = "cancel";
public static final String CONTINUE = "continue";
public static final String DONE = "done";
public static final String EMPTY = "empty";
public static final String ERROR = "error";
public static final String EXIT = "exit";
public static final String FORWARD = "forward";
public static final String LOGON = "logon";
```

```
public static final String LOGOFF = "logoff";
public static final String MENU = "menu";
public static final String NEXT = "next";
public static final String PREVIOUS = "previous";
public static final String TOKEN = "token";
public static final String WELCOME = "welcome";
```

6.4. 运行时参数

当然，一个web 应用后面的驱动力是假定它是动态的，并允许在运行时改变。对ActionForward来说，有两个地方可以在运行时进行调整，以添加和修改查询组件：

即在页面中和在Action类中。

6.4.1. 在页面中添加参数

如果你通过`<html:link>` 标签 `(org.apache.struts.taglib.html)`使用ActionForward，你可以这样来在运行时添加参数到查询组件中：

```
<html:link
     forward="article"
     paramId="key"
     paramproperty="key"
     name="articleForm">
     <bean:write name="articleForm" property="name">
</html:link>
```

假定 `articleForm` bean 具有方法 `getKey`，标签就可以获取`key`属性并将其值添加到URI中去。 如果 ActionForward 是这样定义的:

```
<forward
  name="article"
  path="/do/article?dispatch=view"/>
```

而 `<html:link>` 标签是这样的:

```
<html:link forward="article"
           paramproperty="articleForm"
           paramproperty="articleKey"
           paramId="key">
  News from the edge
```

```
</html:link>
```

则将产生这样的HTML 标记：

```
<a href="http://localhost/artimus/do/
                article?dispatch=view&key=17">
  News from the edge
</a>
```

这里已经有了一个查询字符串了，请注意标记是足够地聪明，添加了另外的参数（key=17）。

关于Struts标签扩展的详细信息，请参考第10章。

6.4.2. 在Action类中添加参数

你也可以在一个Action类中添加参数到ActionForward中，可以这样使用：

```
ActionForward forward = mapping.findForward("article");
StringBuffer path = new StringBuffer(forward.getPath());
boolean isQuery = (path.indexOf("?")>=0);
if (isQuery)
  path.append("&amp;dispatch=view");
else
  path.append("?dispatch=view");
return new ActionForward(path.toString());
```

这样就添加了一个查询参数（?dispatch=view）到URI中，并且在添加之前还进行了检查，是否已经有了一个参数了。如果有，则需要加入“ & ”字符。

Scaffold ConvertUtils 类（`org.apache.scaffold.text.ConvertUtils`）提供了一个方法来自动完成这个工作：即`addParam` 方法，如下所示：

```
// 基本路径和第一个参数
aForward.setPath(
ConvertUtils.addParam("/do/article","dispatch","view"));
// 获得当前路径，并且添加另一个参数
aForward.setPath(
ConvertUtils.addParam(aForward.getPath(),"key","17"));
```

这将产生如下的请求参数：

```
/do/article?dispatch=view&key=17
```

除非 ActionForward 被设置为重定向，新参数将被合并到当前请求的参数中。如果新参数使用的名称和原有参数名称相同，在新的转发期内将使用新的参数值。

6.5. 动态转发

虽然在一个Struts 配置文件中定义ActionForward是个好的选择，如果必要，你也可以从头创建一个ActionForward，并设定其path 和其他参数:

```
ActionForward forward = new
        ActionForward("/do/itemEdit?action=edit");
```

一旦你创建了 ActionForward，你就可以使用上一节的运行时参数设置技术来帮助构建和修改你的ActionForward路径。

如果对设置运行时参数和动态转发有更多兴趣，请参考第8章 (8.4节)。

注 如果你发现你不断地创建动态转发来链接Action，这可能是个警告信号，意味着太多的业务逻辑由一个Action 类来表达。一般地，你应该在在Action类之间重用用业务对象，以便你不必用这种方式来传递“接力棒”。关于如何使业务逻辑独立于Action 类之外，参见第8章。

6.6. 为什么地址栏不变？

使一些新开发人员迷惑的是，为什么一个Action URI, 比如 /do/article/View, 在浏览器的地址栏内保持不变，即使一个表现页面, 比如 /pages/article/View.jsp, 被显示在窗口中的时候也是如此。地址栏显示的是浏览器最后被给定的URL。当URL 被提交后，在某个组件返回一个响应给浏览器之前，你的应用可能转发请求多次。所有这些都发生在服务器端，浏览器并不清楚发生了什么事情。当一个HTTP 响应被返回时，它并没有包含地址栏的值。所以浏览器仅仅显示其用来作初始请求的地址。

注 当然, 通常这就是你想要的。表现页面的名称和地址并非实质性的东西。经常，这也是没有益处的。动态页面的表现典型依赖于Action 对象提供的数据。如果页面被直接访问，数据可能被丢失。

改变浏览器地址栏显示的唯一办法是使用redirect而不是forward (见 6.2.1节)。这样向浏览器发送一个标准的响应，指示浏览器提交一个新的请求。因此，浏览器将修改地址栏的显示来反映新的URL。不幸的是，这也意味着数据不能通过请求上下文传递到页面。必须通过URI来传递。

6.7. 玩转ActionForward

开发人员也可以提供自己的ActionForward 子类，添加额外的属性或者方法。在 Struts 1.0 中， 你可以在部署描述符 (web.xml)对 ActionServlet的配置部分来进行配置，如下所示：

```
<init-param>
  <param-name>forward</param-name>
```

```
  <param-value>app.MyActionForward</param-value>
</init-param>
```

在Struts 1.1中，你可以在Struts 配置文件中，作为 `<global-forwards>` 元素的属性来进行配置：

```
<global-forwards type="app.MyActionMapping">
```

单独的forward也可以通过设置`class`名称属性来使用另一个类型的转发:

```
<forward classname="app.MyActionMapping">
```

关于Struts配置，参见第4章。

框架提供两种基本的ActionForward类，如表6.2所示。

对象名称	说明
org.apache.struts.action.ForwardingActionForward	缺省设置 redirect 属性为false
org.apache.struts.action.RedirectingActionForward	缺省设置 redirect 属性为true

它们都可以选为缺省设定，或者用作你的子类的基类。框架的缺省是ForwardingActionForward (`redirect=false`)。对其子类化可以提供新的属性，并可在Struts配置文件中使用标准属性设置机制来进行设置:

```
<set-property
    property="myproperty"
    value="myValue" />
```

这个技巧避免了仅仅是为了在配置文件中识别新属性而子类化ActionServlet。

6.8. 小结

尽管是非常简单的对象，ActionForward在Struts应用的设计中扮演了一个重要的角色。正确使用它，它可有助于你容易既见森林数据又见树木。一个web 应用其实就是一个URI的乱麻，ActionForward 可以将它梳理成一个优雅的逻辑框图——就象你墙上挂的流程图一样。

在下一章，我们来看看应用的骨架ActionMapping。ActionForm描述了什么东西可以进入应用。ActionForward描述了他们可以去向哪里。ActionMapping则告诉我们这个应用到底想要干什么。

7. 设计 ActionMapping

本章内容

- *理解ActionMapping*
- *定义ActionMapping 属性*
- *使用局部和全局ActionForward*

Decide what you want, decide what you are willing to exchange for it.

Establish your priorities and go to work.

—H. L. Hunt

7.1. 进入ActionMapping

Model 2 架构 (第1章)鼓励在同一个应用中使用servlet和JSP页面。在 Model 2下，我们总是从调用一个servlet开始。

Servlet 处理业务逻辑并将控制转到相应的页面来完成响应。我们可以在web应用部署描述符(web.xml)中将一个URL模板映射到一个servlet。这个模版可以是一个常用的模板格式，比如*.do, *或者什么特别的路径*, 比如saveRecord.do。某些应用通过给每个业务操作映射一个Servlet来实现 Model 2。这种方法可以工作，但许多应用都可能涉及到大量的业务操作。因为servlet是多线程的，实例化这么多servlet 显然不是使用服务器资源的最好方式。Servlet设计来是可以处理大量的并行请求的。简单的创建很多 servlet并没有什么性能优势可言。

Servlet的主要工作是与容器以及HTTP进行交互。对于具体的处理业务操作，则是servle委托其它组件干的事情。Struts 则通过ActionServlet 委托其它对象来进行业务操作。它用servlet接受请求并将其路由到一个处理器（称为***前端控制器*** *模式* [Go3]）。

当然, 简单地将业务操作委托给其它组件并没有解决将URI [W3C, URI]映射到业务操作的问题。我们和Web浏览器进行通信所能使用的唯一途径就是HTTP 请求和URI。如何组织URI来触发业务操作是开发web 应用的关键部分。

同时，在实际应用中，许多业务操作都以相似的方式来进行处理。因为 Java 是多线程的，如果我们使用同一个Action 对象来处理相似的操作，就能更好地利用服务器资源。但是要让它能工作，我们可能需要向对象传递一些配置参数，以用于每一个操作。

那么，底线是什么呢？为了以一种灵活和有效的方式实现Model 2，我们需要

- 将对业务操作的请求路由到一个 servlet
- 决定哪个业务操作和请求相关
- 装入多线程的 helper 对象来处理业务操作
- 将每个请求特定的配置细节传递给 helper 对象。

这样，ActionMapping出场了！

7.1.1. ActionMapping bean

ActionMapping (`org.apache.struts.action.ActionMapping`) 描述了框架是如何处理每一个离散的业务操作 (或 *action*)的。 在Struts中，每个ActionMapping 通过其`path` 属性和一个特定的URI 相关。

当一个请求到来， ActionServlet 使用 `path` 属性来选择对应的ActionMapping。一组 ActionMapping 对象被放在一个*ActionMappings* 集合之中 (`org.apache.struts.action.ActionMappings`)。原本, ActionMapping 对象用来扩展Action *对象*而不是Action *类的*。当用于Action时，Mapping 给了一个特定的Action 对象一些额外的职责和新的功能。所以从本质上来说，Action 是一个装饰器（*decorator* [Go4]）。通过这种方式， ActionMapping 以自己的方式发展为一个对象，可以和Action一起使用，也

可以不。

定义 装饰器模式（decorator pattern）的意图是为一个对象动态地附加上额外的职责和功能。Decorator 在扩展功能时提供了一个除子类化之外的选择。[Go4].

ActionMapping通常通过Struts 配置文件创建。更多信息，参见第4章。

7.1.2. ActionMapping 目录

ActionMapping将对Struts 应用中有效的业务逻辑进行分类。当一个请求到达时，ActionServlet在ActionMapping目录中查找对应条目，并调用相应的Action Bean。

ActionServlet 使用ActionMapping bean 来决定接下来该做什么。它也许需要将控制转发到其它资源。或者也许它需要组装并且校验一个 ActionForm bean。某些时候，它也许会将控制传递给一个Action 对象，并且当Action 返回时，它可能会查找和这个mapping相关的ActionForward。

ActionMapping工作起来就像是ActionServlet的一个路由联络官。取决于mapping 如何被填写设置，请求可能被传递到任何地方。ActionMapping表达了Struts 应用的核心设计。如果你想知道一个 Struts 应用是如何工作的，可以从ActionMapping开始着手。如果你想知道如何编写一个新的Struts 应用，也请从ActionMapping开始。Mapping处于每个Struts 应用的绝对核心。

在本章，我们将详细讨论ActionMapping 属性，并展示它们将如何有助于设计Struts 应用的流程。

1.0 VS 1.1 在 Struts 1.1中, ActionMapping 子类化了ActionConfig (org.apache.struts.config.ActionConfig) 类，并添加了向后兼容的API 方法。ActionMapping 已经不推荐，但后继版本中类的层次还没有决定。

当前，我们引用ActionMapping 类，但你应该注意到在 Struts 1.1 中，所有的 action 属性实际上都是在ActionConfig超类中定义的。ActionMapping类在两个版本下都可以工作。

7.2. ActionMapping 属性

表 7.1 描述了基本的属性。当与其它配置组件一起使用时，开发人员可以扩展ActionMapping以提供额外的属性。

表格 7.1 ActionMapping 基本属性

属性	说明
path	来自于请求的 URI 路径，用来选择该 mapping。(API 命令)
forward	通过一个 forward 服务这个请求的上下文相关的资源路径。

	实际上是 forward，include，type 属性中的一个，必须指定。
或	
include	通过一个 include 服务这个请求的上下文相关的资源路径，。 实际上是 forward，include，type 属性的一个，必须指定。
或	
type	可选，表明一个 `org.apache.struts.action.ActionMapping` 的子类名称，在实例化这个 mapping 时使用。
classname	该 Mapping 使用的 Action 类的全限定名称
name	与该 Mapping 相关的 form bean 的名称，如果有的话。这不是类名称。而是在 form bean 配置中使用的逻辑名称。
roles	可以访问该 mapping 的安全角色列表。
scope	范围(请求或会话)识别符，如果有的话。与该 mapping 相关的 forma Bean 在其中创建。
validate	如果与该 mapping 相关的 form bean 的 validate 方法 (如果有)需要被调用，则设置为 true。
input	输入表单的上下文相关的路径，如果校验错误，控制应该被返回到该表单。可以是任何: HTML，JSP，VM，或者另一个 ActionMapping。
parameter	通用配置参数，用来向 ActionMapping 选定的 Action 传递额外的参数信息。
attribute	果它不是 bean 的特定名称的话，则表示 form bean 在要在其中被访问的请求-范围或者会话-范围的属性名称。
prefix	如果有的话，用来将请求参数名称匹配到 form bean 属性名称的前缀。
suffix	如果有的话，在组装 Actionform bean 属性时，用来匹配请求参数名称的后缀。
unknown	如果该 mapping 要被配置为应用的缺省 mapping(处理那些没有被其它 mapping 处理的请求)，可设置为 true。在一个应用中仅有一个 mapping 可定义为缺省的 unknown mapping。
forwards(s)	该 mapping 使用的 ActionForward，如果有的话。
exception(s)	该 mapping 使用的 ExceptionHandler 如果有的话。

在接下来的一节，我们将详细讨论每个属性。

7.2.1. path属性

ActionMapping URI，或者path，对用户来说就象web server上的一个文件。但实际上它并不代表一个文件。它是一个对ActionMapping的虚拟引用。

因为它暴露给其它系统， path 并不真正是个逻辑名称，就象那些我们用于ActionForward中的一样。 path 可以包括反斜杠和扩展名--如果它引用一个文件系统——但它们都只是一个单一名称的一部分。

ActionMapping自身是一个“平面的” 名称空间，完全没有内部层次关系。它仅仅是正好使用了一个和层次性文件系统中一样的名称而已。

当然, 这对处理ActionMapping仍然是有帮助的，就象它们是一个层次关系或者在同一个"文件夹"下的相关命令组的一部分。唯一的限制是名称必须匹配于在应用部署描述文件中为ActionServlet指定的样式。这通常是/do/*或者 *.do，但是其它相似的样式也可以使用。如果你在一个团队环境中工作，不同的团队可能有不同的 ActionMapping 名称空间使用。某些人可能工作于/customer ActionMapping, 而另一些可能工作于/vendor ActionMapping。这也可能和各个团队使用的Java 包的层次相关。因为 ActionMapping URI是逻辑的构造，所以你可以用它来以各种方式组织来适合你的项目。

对Struts 1.1来说, 这种类型的名称空间可以被提升为应用模块（module）。每个团队可以独立工作于它们各自的模块, 连同他们自己的配置文件和表现页面。关于在应用如何配置多个模块的问题，请参考第4章。

定义

Web运行在URI之上，大多数 URI都映射到物理文件。如果你想改变资源，你就得改变相应的文件。某些URI, 象Struts action,则是虚拟的引用。它们并没有对应的文件，但是通过编程组件来处理。要改变资源，我们要改变组件的编程方式。 但是因为path是一个 URI 并同我们控制之外的其他系统交互，path 也不是一个真正的逻辑引用—例如，是一个 ActionForward 的名称。我们可以改变ActionForward的名称而不用顾及其它系统。它是一个内部的逻辑引用。 如果我们改变了一个到ActionMapping的路径，我们可能需要修改通过公共URI引用该ActionMapping 的其它系统。

7.2.2. forward 属性

当指定了forward 属性时，servlet将不会把请求传递给Action类，而是调用RequestDispatcher.forward。因为操作没有使用Action 类, 它也可以被用来进行Struts 和其它资源之间的集成，或者用来构建系统原型。forward, include, 以及 type 属性是互斥的。（参见第6章的详细信息。）

7.2.3. include 属性

当指定了include 属性时, servlet将不会把请求传递给Action 类而是调用RequedtDispatcher.include方法。因为操作没有使用Action 类, 它也可以被用来机械进行Struts 和其它组件之间的集成。forward, include, 以及 type 属性是互斥的。

7.2.4. type 属性

大部分 mapping都会指定一个 Action 类类型，而不是一个forward 或者 include。Action 类可以被多个mapping使用。 Mapping可以指定form bean，参数，forward，或者exceptions属性。forward, include, 以及 type 属性是互斥的。

7.2.5. classname属性

如果被指定，classname 是应该用于此对象的ActionMapping子类的全限定Java类名称。这就允许你使用你自己的ActionMapping 子类来引入特定的方法和属性。参见7.4节。

7.2.6. name属性

如果在Struts 配置文件中配置了相应的formbean段的话，这个属性指定了form bean的逻辑名称。缺省情况下，这也是将form bean 放入请求或者会话上下文时使用的名称。使用该类的attribute 属性来指定不同的属性（Attribute）关键字。

7.2.7. roles属性

这个属性是一个逗号分隔的，允许访问该ActionMapping 对象的安全角色称的列表。缺省情况下，与基于容器的安全机制所使用的同一个系统将用于这里给定的角色列表。这意味着你可以通过在部署描述符中指定特定的URL模板而使用基于action的安全，或者两者都使用。

安全检查由**请求处理器**(org.apache.struts.action.RequestProcessor)的processRoles方法处理。通过子类化请求处理器，你也可以在基于应用的安全下使用 roles 属性。

7.2.8. scope属性

ActionForm bean 可以存储在当前的请求或会话范围中（这样它可以服务于另外的请求）。虽然大部分开发人员使用请求范围来存储ActionForm, 框架的缺省设置却是会话范围。如何设置请求为缺省值, 见7.4节。

7.2.9. validate属性

ActionForm 生命周期内的一个重要步骤是在其数据提供给业务层之前需要进行校验。当 mapping 的validate 属性设置为true时，ActionServlet将调用ActionForm的 validate 方法。如果 validate 返回false，请求将被转发到input 属性指定的资源。

经常，开发人员将为每个数据输入表单创建两个mapping。一个mapping 将使 validate 设置为 false, 所以你可以创建一个空的表单。另一个则设置validate 为true，用来提交完整的表单。

> **注** ActionForm的 validate 方法是否被调用与ActionServlet的 validating 属性无关。后者只是控制Struts 配置文件如何处理。

7.2.10. input属性

当validate被设置为true时，重要的是需要提供一个有效的输入路径。这也是当ActionForm validate 方法返回 false时，控制应该被传递到的地方。通常这是一个表现页面。有时也可以是另外一个用来产生页面所需要的数据对象的Action path (其 validate 必须设置为 false)。

> **注** input 路径通常会将用户引导到提交请求的页面。虽然对框架来说返回请求的原发地点似乎是很自然的，但是在web 应用中这却不是一个简单的事情。请求通常在响应被返回给浏览器之前被从一个组件发送到另一个组件。浏览器只知道它用来获取input 页面的路径，而该路径可能是，也可能不是用作input 属性的正确路径。虽然也可以根据HTTP referrer 属性产生一个缺省 的input 页面， 但是Struts 的设计者认为该方法并不可靠。

输入转发

在Struts 1.0中，ActionMapping的 input 属性通常是一个字面的URI。在Struts 1.1中，它也可以是一个ActionForward 的名称。ActionForward 被获取之后，其path 属性被用作input 属性。这也可以是一个全局或者局部转发。

这儿为了使用 ActionForward而不是字面的path，可以设置这个模块的<controller> 元素的inputForward 属性为 true:

```
<controller inputForward="true">
```

7.2.11. parameter属性

这个普通的paramenter属性允许Action能够在运行时被配置。许多标准的Struts Action都使用这个属性，标准的Scaffold Action也经常使用它。Parameter属性可以包含URI，方法名称，类名称，或者其它一些Action在运行时可能需要的信息。这种灵活性允许一些Action履行双倍或者三倍的职责，削减应用所需的不同的类的数量。

在一个Action 类中，parameter属性是从传递到perform的mapping中取得的：

```
parameter = mapping.getparameter();
```

多重参数

虽然多重参数不被标准的ActionMapping类支持，也有一些容易的方式实现它，包括使用HttpUtils, StringTokenizer，或者一个属性文件 (java.util.property)的方式。

HttpUtils

虽然不被Servlet API 2.3 规范赞成，HttpUtils 包(javax.servlet.http.HttpUtils)提供了一个静态方法，可以以对一个查询字符串一样的方式来解析字符串，然后返回一个Hash表(java.util.Hashtable)：

```
HashTable= parseQueryString(parameter);
```

Mapping的 Parameter属性就好像成为另一个查询字符串，你可以在Struts 配置中使用它。

stringTokenizer

另一个简单的方法，使用你选择的记号来分隔参数，比如逗号，冒号，或者分号。并且使用StringTokenizer 方法将它们读回来：

```
StringTokenizer incoming =new
```

```
        StringTokenizer(mapping.getParameter(),";");
int i = 0;
String[] parameters = new String[incoming.countTokens()];
while (incoming.hasMoreTokens()) {
  parameters[i++] = incoming.nextToken().trim();
}
```

属性文件

虽然稍微比其他两个方法要复杂些，但另一个为ActionMapping提供多重参数的通常做法是使用标准的属性文件（`java.util.property`）。取决于你的需要，属性文件可以存储在文件系统的一个绝对位置，或者应用的`CLASSPATH`之中。

Commons Scaffold 包 [ASF, Commons] 提供了一个 `ResourceUtils`包（`org.apache.commons.scaffold.util.ResourceUtils`），它具有从一个绝对路径位置或者应用的`CLASSPATH`装入属性文件的方法。

7.2.12. attribute属性

不时地，你可能需要同时在同一个上下文中保存同一个ActionForm 的在两份拷贝。这在ActionForm被存于会话上下文中作为工作流的一部分时，经常发生这种情况。为了避免它们之间的名称冲突，你可以使用 `attribute` 属性来给一个 ActionForm bean 一个不同的名称。

另一个方法是在配置中定义另一个ActionForm bean，使用相同的类型但使用不同的名称。

7.2.13. prefix和suffix属性

与 `attribute`属性一样，`prefix` 和`suffix` 属性可用来避免应用系统中的名称冲突。如果指定了这个属性，这就表明允许在属性名称前面加上前缀或者后缀，形成它们从请求中被组装时的一个别名。

如果指定了前缀 `this`，那么在ActionForm组装时：

```
thisName=McClanahan
```

相当于

```
name=McClanahan
```

而它们都可以通过调用

```
getName("McClanahan");
```

取得。

这并不影响属性如何被标签扩展输出。它只影响到自动组装机制如何从请求中感知它们。

7.2.14. unknown ActionMapping

在Web上冲浪时，大都遇到过讨厌的*404— pages not found*信息。大部分web server 都提供一

些特殊的特征来处理对未知页面的请求，所以 webmaster 可以将用户引导到正确的方向上。

Struts 提供了一个类似的服务来处理404错误—即*unknown ActionMapping*。

在Struts 配置文件中，你可以指定一个ActionMapping 来接收所有与其它ActionMapping不相匹配的请求：

```
<action
    name="/debug"
    forward="/pages/debug.jsp"/>
```

如果没有设置它，没有匹配对的ActionMapping请求就会抛出：

```
400 Invalid path /notHere was Requested
```

请注意*通过对ActionMapping请求*, 我们的意思是URI首先必须匹配为servlet指定的前缀或者后缀 (通常是 /do/* 或者 *.do)。对于其它样式的请求, 无论好坏，都将被容器中的其它servlet处理：

```
/do/notHere (goes to the unknown ActionMapping)
/notHere.txt(goes to the Container)
```

7.3. 嵌套的组件

当ActionMapping 试图要让一个Action运行业务操作时，其属性是非常有帮助的。但它们只告诉了故事的一小部分。在Action 返回结果之前仍然还有许多事情要做。

一个Action可以具有不止一个结果。我们也许需要注册多个ActionForward以便在返回时Action 可以获取它们。

7.3.1. 局部转发

在正常情况下, ActionMapping 被用来选择一个 Action 对象以处理请求。而 Action 则返回一个 ActionForward，指示使用使用哪一个页面应该来继续完成响应。

我们使用 ActionForward 的原因在于，在实践中，表现页面要么经常被重用，要么经常改变，或者两种情况均存在。在这两种情况下，比较好的做法是将页面的地址封装到一个逻辑名称后面，比如“success” 或者 “failure”什么的。ActionForward 对象使得我们可以分派一个逻辑名称给一个给定的 URI。

当然, 逻辑概念，如success 或者 failure 是相对的。对某个Action 代表成功或许对另一个Action意味着失败。每个ActionMapping 都可以有它自己的一套局部ActionForward。当一个Action 需要查找一个转发 (通过名称)时，局部设置会被首先检测，然后才是全局转发。

参见第 6 章，关于 ActionForward 的更多知识。

7.3.2. 局部异常

通常，一个应用的异常处理器 (`org.apache.struts.action.ExceptionHandler`)都声明为全局的。然而，如果一个给定的 ActionMapping 需要以不同方式处理一个异常，它可以

设置一个自己的局部异常处理器来在全局异常处理器之前进行检测。

局部异常处理通常在 Struts 配置文件中设置。参见第 4 章。

7.4. 玩转ActionMapping

虽然 ActionMapping 提供了一些给人深刻印象的属性，开发人员也可能需要提供一些他们自己的子类来引入额外的属性和方法。在 Struts 1.0 中，这可以在部署描述符 (web.xml)中对 ActionServlet 的配置部分进行配置：

```
<init-param>
  <param-name>mapping</param-name>
  <param-value>app.MyActionMapping</param-value>
</init-param>
```

在 Struts 1.1 中，这也在 Struts 配置文件中配置的，但作为 `<action-mappings>`元素的属性来配置：

```
<action-mappings type="app.MyActionMapping">
```

另外也可以通 `classname` 属性来设置单独的 mapping 来使用另一个类型：

```
<action classname="app.MyActionMapping">
```

框架提供了两种基本的ActionMapping类， 见表 7.2。

表格 7.2 缺省 ActionMapping 类

ActionMapping	说明
org.apache.struts.action.SessionActionMapping	缺省时，scope 属性为session
org.apache.struts.action.RequestActionMapping	缺省时，scope 属性为Request

它们可以选作为缺省设置，或者用作你的子类的基类。

框架的缺省设置为 SessionActionMapping，所以 `scope` 属性的缺省设置为 session。

提供新的属性的子类可以在Struts 配置中通过标准机制设置属性：

```
<set-property property="myProperty" value="myValue" />
```

使用标准机制可以帮助开发人员避免仅仅因为在解析配置文件时，识别新属性而子类化 ActionServlet。这确实是一个Struts简单继承的Digester 的一个特征。

7.5. 小结

Sun的 Model 2 架构告诉我们，应该在同一个应用中使用servlet 和 JSP页面。Servlet可以处

理流程控制和数据获取， JSP页面则可以处理HTML。

Struts 对此一步领先，并且将很多流程控制和数据获取的职责委托给Action 对象。然后应用就仅需要一个单一的servlet来扮演交通警察的角色。所有要做的工作就是包装除Action和Struts 配置对象之外的其它对象。

就象servlet一样，Actions也是很有效率的，多线程的单态对象（Singleton）。一个单一的Action对象可以同时处理很多请求，这样就优化了服务器资源的使用。

为了最大限度地利用 Action，ActionMapping 对象被用来作为Action 对象的装饰器。它对Action 指定一个 URI，或者几个URI，以及一种根据哪一个URI被调用来传递不同设置给Action的方式。

在本章，我们详细探讨了ActionMapping 的属性，并解释了每个属性在整个配置中的相应角色。我们也看到了如何使用用定制属性来扩展标准的ActionMapping 对象——在你的解决方案需要更多东西的时候。

在第8章，真正的好戏开场了。迄今为止我们讨论过的配置对象主要是作为支撑系统。它们帮助控制器将输入的请求匹配到服务器端的操作。既然，我们已经有了支撑，那我们就来会会Struts中的女主角: Action 对象。

8.和女主角 Action 对象共舞

本章内容

- *理解 Action 最佳实践*
- *使用 BaseAction*
- *串接 Action*
- *依赖标准的 Action*
- *使用 helper Action*

I will work harder!

—Boxer *(Animal Farm*, by George Orwell)

8.1. 准备好了，设定，行动!

如果Struts配置是一个应用系统的大脑，Action 类则是其强健的四肢。它们是Struts应用中真正干活的组件，开发人员一般都在这里耗费大量的时间。

Action也是Struts框架中最灵活的类。这个家伙可以用来在你需要的时候创建你所想要的所有功能。

Struts Action的核心职责是：

- 访问业务层
- 为表现层准备数据对象
- 处理在其中出现的错误

但这个列表并没有包含Action可以做的所有事情。当ActionServlet将请求影射给Action时，它也"代表"了这个对象，以便它可以做通常一个servlet能做的事情。例如，一个Action 可以:

- 对请求创建它自己的响应
- 在 servlet 范围内访问或者创建其它对象
- 通过 RequestDispatcher 包含或者转发另一个 Servlet

虽然许多Action都是定制来完成特定任务，其它一些则是良好构建的，可以通过它们的mapping来进行配置，并在整个应用中都可以重用。

在这一章，我们详细探讨Struts提供的标准Action，Scaffold包中的标准Action 类，以及可重用的Action编码技术。

8.2. 搞定Action对象

Struts应用中的其它大部分组件都只是简单地提供一个基础架构，以便Action对象可以专心做应用系统想要做的事情。如果应用需要保存记录到数据库，其流程可能是:

- ActionForward 提供一个到输入页面的链接
- ActionForm 捕获输入
- ActionMapping 配置 Action.
- Action 将输入送到数据库

注 在实际应用中，大多数开发人员都会将数据送到业务代表 (见14章)而不是实际的数据库。不过这取决于你而不是Struts。框架将业务操作委派给Action代表，并使 Action 按其自己的方式工作。

在这一节，我们描述这个重要的类的基础：Action是什么，它何时被调用，它做些什么，以

及它看起来象什么。

8.2.1. 什么是Action?

和常规的 web 应用相比，Struts Action 类工作起来就象一个小型的 servlet。在大多数 Java 应用中，诸如访问业务层的任务、错误处理等任务均是由 servlet 承担的。在一个 Struts 应用中，servlet 扮演着一个分派器的角色。而 Action 对象则干实际的工作。象 servlets 一样，Action 对象是多线程的。每个应用只需要一个 Action 类的实例。

虽然工作起来和 servlet 一样，Action 对象自身并不是 servlet。Action 是简单，轻量的 Java 类，ActionServlet 将处理请求和响应的事情委托给它。Action 被连接到 ActionServlet 并可以调用其所有公共属性，但没有实例化另一个 servlet 的负担。

Servlet 引用 ActionMapping 列表 (第 7 章)来选择一个 Action 来处理请求。Servlet 然后调用一个 Action 的进入方法，并传入一些有用的对象。当 Action 的进入方法完成时，它会返回一个 ActionForward。ActionServlet 使用这个 ActionForward 来决定接下来控制应该传递到哪里来完成这个请求。

典型情况下，ActionForward 将控制转发到一个表现组件，比如 JSP 或者 Velocity 模板什么的。ActionForward 也可以引用其它的 Action，一个 HTML 页面， 或者其它任何具有 URI 的资源 [W3C, URI]。如果为了表示它已经产生了响应并完成了请求，Action 会返回一个 null 值。

从外部看， Action 象一个返回一个 ActionForward 的小型 servlet 。

本章余下部分，我们将研究 Action 从内部看起来是什么样子的。

线程安全

Actions 是多线程的；每个应用中对一个给定 Action 子类仅有一个实例。这意味着 Action 的编写必须是线程安全的。在你写一个 Action 子类的时候，最重要的事情是记住类属性不能在成员方法间共享值。如果使用了成员方法，那么所有的方法都必须通过方法签名来传递。这样就是通过 stack 来传递值，是线程安全的。

定义

线程安全意味着某个特定的库函数必须以可以被多个并发线程执行的方式来实现。关于线程安全和多线程，可以阅读书籍 Java 语言环境第7章 [Gosling, JLE]。

成员方法是一个很重要的设计元素，可以在编写良好的 Action 子类中发现(包括在本章中后面要涉及的 Action)。请确保通过方法签名来传递共享值，就象这些方法是在不同的对象上一样。

8.2.2. Action 何时被调用?

当需要 Action 时，ActionServlet 是通过 Action 的 `perform` 或 `execute` 方法来调用它的。

STRUTS 1.1

Execute 方法是Struts 1.1加入的，改善了异常处理能力。同时也可以使用在原来使用perform 方法的地方。perform 方法在Struts 1.1已经不推荐使用，但还提供向后兼容能力。

Perform 或者 execute 方法是Action的唯一进入点。方法接受四个参数，如表8.1:

表格 8.1 传递给 Action perform / execute 方法的参数

参数	说明
mapping	用来调用这个Action的ActionMapping
form	mapping 指定的ActionForm，如果有的话
Request	请求上下文
response	创建的响应

mapping 和form 参数的传递是很方便的。在这里，它们已经被放入请求之中，并可从那里获取：

```
ActionMapping mapping =(ActionMapping)
       Request.getAttribute(MAPPING_KEY);
ActionForm form =(ActionForm)
       Request.getAttribute(mapping.getName());
```

但几乎每一个Action 类都需要使用这些对象，对servlet来说更简单的办法是通过方法签名来传递它们。相反，servlet并不传递session对象，因为Action 类大多不需要使用 session上下文。如果需要session时，可以从请求中获取：

```
HttpSession session = Request.getSession();
```

然而，重要的是，你要记住传递到 Action 类的 ActionForm，可能根据不同的请求而不同。由 ActionMapping 来决定哪一个 ActionForm 子类传递给 Action。ActionServlet 可以使用任何请求中的 ActionForm 类型。这使得你可以创建一个灵活的 Action 类，可以被多个不同的 ActionForm 使用。

然而，在实际应用中，许多 Action 类都是设计来使用特定的 ActionForm 类型，并让它访问需要的属性:

```
MyActionForm myForm = (MyActionForm) form;
```

如果传递的 form 不是一个 MyActionForm 类型或者其子类，将抛出一个运行时异常。但只要正确的类型在 ActionMapping 中指定了，这种方式将工作得很好。

8.2.3. Action做些什么?

一个典型的 Action 的职责通常是：

- 校验前提条件或者声明
- 调用需要的业务逻辑方法
- 检测其它处理错误
- 将控制路由到相关视图

这个列表并不意味着要限制 Action 能做的事情。 一个 Action 可以做一个完整的 servlet 能做的所有事情。如果想要直接输出一个响应给客户，Action 就可以。如果想转发到其它 servlet， Action 也可以。但，在实际应用中，大多数开发人员都是编写 Action 来做这四种事情，而让控制器来转发请求，让其它组件来渲染响应视图。

Action校验声明

Action 的主要角色是作为 web 和业务层之间的适配器。Action 从 web 层获取我们需要的东西并将它们传递给相应的业务操作。但是，通常情况下，业务操作过分讲究，而 web 操作则过分的粗鲁。

因为和 web 的松散耦合,大部分 Action 需要在将请求传递给业务操作之前校验表单的数据并且检查其它前提条件。很多时候，数据校验可以委派给 ActionForm 进行。所以，Action 需要做的就是确认 ActionForm 是否是需要的类型。通常，这是通过将 ActionForm 临时转换为需要的类型来完成的。

```
MyActionForm myform = (MyActionForm) form;
```

其他前提条件包括是否用户已经登陆，或者是否已经进行了某些适当的清理。在 Struts 1.1 中，这可以通过 ActionMapping 的 `roles` 属性来进行自动处理。也可能有其它的前提，是否用户已经执行了工作流的其他步骤。

如果这些前提条件的某些检查失败， Action 将产生一个本地化了的信息列表，并将控制路由到一个错误页面。我们再回头来简单看看错误处理。

Action 的首要任务是检查前提条件，然后是调用业务逻辑。

Action调用业务逻辑

业务逻辑，从广义上说， 是一系列管理特定业务功能的指南。

使用面向对象的方法可以使开发人员能将业务功能分解为称之为业务对象的组件和元素....

特定的业务规则帮助我们标识业务对象的行为和结构，以及当对象在系统中将其行为暴露给其它对象是必须符合的前置和后置条件，这就是逻辑。.

— Taken from the J2EE Blueprints at 5.1 [Sun, Blueprints]

—

Action 类是 HTTP 与应用系统中其它部分之间的适配器。最重要的是要避免将业务逻辑放入 Action 之中。Action 类应该只是简单地收集业务方法需要的数据并传递它到具体的业务对象。如果你同时在编写业务类和 Action 类，可能会受到要将它们编写在一起的诱惑。一定要抵挡这种诱惑，并且将业务方法放入 Action 可调用的单独的类之中。Java 虚拟机(JVM)针对这种方法调用作了优化；性能损失可以忽略不计。

同时你也得到了一些设计上的优势，如表 8.2。

表格 8.2 为什么要将业务逻辑和 Action 分开

原因	解释
更鲁棒的设计	如果业务方法位于单独的其他类中，在写Action类时你大可不必担心破坏它们。

更简单的设计	如果业务方法封装在对另一些代码段的调用中时，更容易看到你的业务方法是如何工作的。
更广泛的设计	目前可能只需要一个Struts web 应用。或许明天又会需要一些新的东西，从而重构整个系统。如果业务逻辑没有放在Action类中，你可以很方便的重用它们。
更灵活的设计	最后，你可能会发现你需要在其它Action类中使用同一个业务逻辑。某些开发人员就试图在Action间转发请求。如果业方法在单独的类中，你就可以从不同的Action中调用它们。

Action检测错误

Struts具有一个设计良好的错误处理系统，允许你可以：

- 同时截获几个错误
- 在请求中传递错误数据包
- 显示本地化信息

这个处理流程涉及到两个对象 (ActionErrors 和 ActionError) 和一个注册错误的工具方法(saveErrors) 。其它两个对象 (MessageResources 和 一个定制标签)则用来显示错误信息。我们将讨论这些对象和 servlet 方法。 关于如何利用 JSP 标签显示错误信息，参见第 10 章。

注册错误

总体流程归结为：

- 创建一个空的ActionErrors 实例
- 在错误发生时，为错误信息添加关键字；
- 检查是否添加了某些信息
- 保存ActionErrors 集合对象到请求中
- 转发控制到错误页面以显示信息
- 否则，正常继续

或者，例如：

```
ActionErrors errors = new ActionErrors();
  try {
  // * 调用业务对象 *
  }
  catch (ModelException e) {
    errors.add(ActionErrors.GLOBAL_ERROR,
               new ActionError("error.detail",e.getMessage()));
```

```
  }
  if (!errors.empty()) {
    saveErrors(Request, errors);
    return (mapping.findForward("error"));
  }
  // * 正常继续 *
```

当然，error.detail 不是用户所看到的内容。这是一个引用消息资源的关键字。和这个关键字相关的信息可以与占位参数进行以及显示给用户的结果进行合并。不同的本地场所可以有不同的消息资源。消息内容可以是不同的，但它们都有相同的关键字设定。

这个例子将消息从异常发生之地送到错误页面。在资源文件中文件，关键字error.detail可能是一个大的占位符参数:

```
error.detail={0}
```

典型的，你可能想要使用用户友好的信息，比如这样

```
error.database=An error occurred while accessing the database
```

你也可以发送友好的错误信息和实际的错误信息，象这样做：

```
errors.add(ActionErrors.GLOBAL_ERROR,
               new ActionError("error.database"));
errors.add(ActionErrors.GLOBAL_ERROR,
               new ActionError("error.detail",e.getMessage()));
```

Struts 消息可以接受四个占位参数。 你可以用它们来定制具有记录编号或者其它特殊内容的消息。关于消息资源和国际化的更多信息，参见第13章。

在JSP中，你可以使用值得信赖的Struts 1.0 标签来输出错误信息:

```
<html:errors/>
```

或者更加灵活的Struts 1.1消息标记:

```
<logic:messagesPresent>
  <UL>
    <html:messages id="error">
      <LI><bean:write name="error"/></LI>
    </html:messages>
</UL>
</logic:messagesPresent>
```

在Struts 1.0中排队消息

在Struts 1.0, 你可以使用同一个队列来确认消息。不管消息是一个错误，或者是一个典型的上下文内容。对用户来说，区分下面这两个消息的不同并不困难：

```
"The Amount field is required"
```

和

```
"Record #1412 deleted"
```

在Struts 1.1中排队消息

在Struts 1.1中，你可以将消息和错误分别注册到两个不同的队列。在表现页面中，你就可以分开输出每个队列，也许还可以使用不同的样式。注册消息的流程和注册错误的流程相同：

- 创建一个空的ActionMessage实例
- 根据需要为消息添加关键字
- 调用 saveMessages(HttpServletRequest, ActionMessages)方法。

在 JSP中, 你可以检查消息，就象处理错误一样，方法是在消息标记中标识message=true:

```
<logic:messagesPresent message="true">
  <UL>
    <html:messages id="message" message="true">
      <LI><bean:write name="message"/></LI>
    </html:messages>
  </UL>
</logic:messagesPresent>
```

要检测错误，而不是消息，可以忽略message属性(缺省设置为 false)。

Struts 1.0的异常处理

在 Struts 1.0 应用中，Action 对象处于调用树的最顶层。某些在此不能被捕获的异常就不能被捕获了。这会导致一个“白屏幕” 或者显示你创建的 JSP 错误页面。

perform 方法将抛出 IOExceptions 和 ServletExceptions 异常。如果这些异常发生，让它们转到一个缺省的错误页面也许是最好的选择，因为令人讨厌的事情发生了。

而你必须检查的是业务逻辑抛出的异常。我们推荐的方法是，业务对象应该抛出其自己的异常类。这有助于正确的检查。也给你的业务层一个可以在业务上下文中处理其自己的异常的机会。

Struts 1.1中的异常处理

在 Struts 1.1 中, Action 的首选的进入点是 execute 方法。这个方法设计来和 Struts ExceptionHandler 对象(org.apache.struts.action.ExceptionHandler)一起使用。你可以在 Struts 配置中注册全局和局部异常处理器。如果你喜欢，你也可以委派所有的异常处理职责给 Struts handler。或者你仍然可以让 Action 处理可恢复的异常而允许其它异常传递给处理器。

关于配置 Struts 异常处理器的细节，参考第 4 章。

关于编写异常处理器的细节，参考第 9 章。

Action路由控制

Struts ActionForward 可以定义“要去的地方”，但是具体的运行路线则是由 Action 对象来选择的。

ActionForward 定义去哪里，而 Action 对象则决定何时去。

mapping.findForward.

ActionServlet 通过 `perform` 或者 `execute` 方法调用 Action 对象，并以返回一个 `ActionForward` 或者 `null` 来退出调用。Action 类选择 forward 的最通常的方式是通过 ActionMapping 的 `findForward` 方法:

```
return mapping.findForward("continue");
```

`mapping` 在查找全局转发之前会首先寻找局部转发。如果命名的 forward 没有找到，`findForward` 返回 null。通常，Action 返回 null 意味着响应已经送出。如果你接收到浏览器的一个消息，指示说没有响应，绝大多数情况是因为 `findForward` 返回了 null。这意味着两种转发都没找到，或者 Action 类命名错误。

STRUTS TIP

避免这种类似错误的一个做法是为ActionForwards定义字符串记号。

不幸的是，你不能在 XML 配置文件中使用它们，并且在JSP中也很难以使用。但是它们的确至少可以记录forwards 如何被调用，并且可以在Action类中防止输入错误。

动态选择.

虽然传递一个 `String` 常数是选择 ActionForward 最通常的做法，但也不是唯一的办法。`mapping.findForward` 方法是运行时解析的，所以转发名称可以在运行时决定。例如，一个基础 Action 类可能提供一个 `getForward` 方法，而这个 Action 类可以被子类化，以提供特定的行为。而主 `execute` 法块调用你的这个方法来返回结果:

```
return mapping.findForward( this.getForward() );
```

你的子类可以重载`getForward` 方法来返回不同的结果，而不用改变主`execute` 方法。Action类就可以根据操作的结果选择一个响应的forward。例如，如果查找结果返回空可以使用一个特定的页面来作为响应：

```
Collection result = businessClass.getResult();
if (0==result.size()) {
  return mapping.findForward(Tokens.EMPTY);
}
Request.setAttribute(Tokens.RESULT, result);
```

```
return mapping.findForward(Tokens.SUCCESS);
```

虽然不是通用的方法，Action 也可以明确的选择一个转发，或者根用户的场所，浏览器，安全角色，或者其它规则来选择另一个转发。

动态构造

ActionForward 自身也可以在运行时构造。这是一种在ActionForward被传递到另一个Action之前向其中添加参数的便捷方式。关于构造如何构造ActionForward，参见第6章。

8.2.4. Action象什么?

下面是 `execute` 方法的一个骨架， 它综合了前面所表述的内容 (对 Struts 1.0，你可以将同样的代码用在 `perform` 方法中):

```
public ActionForward execute(
                ActionMapping mapping,
                ActionForm form,
                HttpServletRequest Request,
                HttpServletResponse Response)
        throws Exception {
  ActionErrors errors = new ActionErrors();
  // 如果所有的异常都注册了处理句柄
  //  try .. catch 语句可以被忽略
  try {
    // * 此处调用业务逻辑*
  }
  // * 此处捕获你的业务异常*
  catch (ChainedException e) {
    // Log and print to error console
    servlet.log("Exception: ", e );
    e.printStackTrace();
    // General error message
    errors.add(ActionErrors.GLOBAL_ERROR,
         new ActionError("error.general"));
    // Generate error message from exceptions
    errors.add(ActionErrors.GLOBAL_ERROR,
         new ActionError("error.detail",e.getMessage()));
    if  (e.isCause()) {
      errors.add(ActionErrors.GLOBAL_ERROR,
```

```
            new ActionError("error.detail",e.getCacuseMessage()));
      }
    }
    // Report any errors
    if  (!errors.empty()) {
      saveErrors(Request, errors);
      if (mapping.getInput()!=null)
        return (new ActionForward(mapping.getInput()));
      // if no input page, use error forwarding
      else
        return (mapping.findForward(Tokens.ERROR));
    }
    // * if your biz logic created any *
    // * helper objects, save them before returning *
    // * request.setAttribute("yourKey",your biz object); *
    // * return the ActionForward you use for success *
    return findForward("continue");
}
```

当然, 还有其它聪明的事情你的 Action 可以去做，但这只是一个每个 Action 都应该遵循的基本骨架。

Scaffold 包提供了一个这个骨架的重构版本，可以在你的应用中用作基本 Action(`org.apache.struts.scaffold.BaseAction`)。 它用一个公共或保护的方法代替了上面的代码块，这些方法你可以在子类中根据需要覆写。

8.3. 标准Action

许多 Struts Action 类都是编写来在特定的应用中作特定的事情。而其它一些 Action 则编写来为各种应用所使用。一些 Action 设计来可以做多种工作。而另一些 Action 可能仅提供流程控制，而本身并不做任何事情。而有些 Action 则向其它 Action 提供公共支撑基础。

Struts 发布包提供了几个分别作不同事情的“标准的” Action。它们中有桥（ *bridge* ）Action，以帮助 Struts 与你的应用中的其它 servlet 一起工作，以及几个用来扩展新功能的基础（ *base* ）Action。表 8.3 表述了这些 Action

表格 8.3 表述了这些 Action

Action	目的	示例
Bridge Action	集成Struts和其它servlet	ForwardAction

		IncludeAction
Base Action	扩展功能	DispatchAction LookupDispatchAction [Struts 1.1] SwitchAction [Struts 1.1] BaseAction [ASF, Scaffold]

8.3.1. 标准桥式Action类

Struts 希望和其它组件友好相处，所以提供了标准类来帮助集成 Struts 和应用中的其它 servlet 。这是通过调用一个标准的 servlet `RequestDispatcher` 类 (`javax.servlet.RequestDispatcher`)，并且使用其 forward 或者 include 方法来达到的，比如表 8.4。

表格 8.4 标准 bridge Action

Action	**目的**
ForwardAction	发出一个RequestDispatcher 转发
IncludeAction	发出一个RequestDispatcher 包含

ForwardAction

如其名，ForwardAction 只是简单地将控制转发到其它资源。这可能是另一个 Action，一个 JSP，另一个 servlet，或者具有 URI 的其它应用资源。[W3C, URI]。

ForwardAction 创建一个请求分派器，并根据 ActionMapping 提供的上下文相关的 URI 来转发控制。上下文相关的路径在 ActionMapping 的 parameter 属性中给出:

```
<action
  path="/saveSubscription"
  type="org.apache.struts.actions.ForwardAction"
  name="subscriptionForm"
  scope="request"
  input="/subscription.jsp"
  parameter="/path/to/application/resource"/>
```

ActionServlet 将沿着其正常路径实例化一些 form bean 并校验它，如果适合就转发请求。ForwardAction 的绝大多数使用是作为 Action 的占位符。许多 Struts 开发人员避免从一个页面直接连接到其他地方而是通过 Action 或者 ActionForward 来传递控制。这保证了工作流在 Struts 配置的控制之下，在这里可以进行集中管理。

然而，许多页面并不需要特殊的预处理(至少还不需要)。如果为这些页面创建 ActionMapping ，你可以使用 ForwardAction ，来仅进行路由控制。如果以后，需求改变，又需要进行预处理，你可以改变 mapping 来为那个页面引用到一个 Action。因为链接

是引用到 mapping，而不是 Action 类，所以你可以修改 Action 类而不用改变链接。

1.0 vs 1.1

支持模块化应用是Struts 1.1引入的。使用这个特征要求控制要在使用 Struts 标记库处理一个JSP页面之前通过ActionServlet进行传递。即便没有使用一个模块化的设计，你也可以使用这个机制，以便使你的应用易于重构，方法是只将连接引用到Action而不是JSP。如果你的页面根本不需要Action，则使用一个ForwardAction来代之。

ForwardAction 也可以用来集成 Struts 与应用中那些希望处理为 URI 的其它组件。许多 servlet，比如 Cocoon [ASF, Cocoon]的设计都是通过 URI 来访问的。这些 Servlet 的 URI 可以用 ForwardAction 进行封装。 这样就可以使我们能够容易具有操纵 Struts 控制流和 form bean 的能力，而不需要放弃其它 servlet 提供的特殊服务。

如果需要时，其它 servlet 也可以在请求中访问 form bean。Mapping 可以通过关键字 `Action.MAPPING_KEY` 访问到，连同原来的 HTTP 请求参数。这些对象都可以用来向其它 servlet 提供信息 (如果它能够导入 Struts 类的话):

```
ActionMapping mapping = (ActionMapping)
                  Request.getAttribute(Action.MAPPING_KEY);
EditForm editForm = (EditForm)
                  Request.getAttribute(mapping.getName());
```

如果你需要将对象放入其它组件所期望的请求之中，那么可以简单地创建你自己的 Action 子类，并且返回一个引用组件的 URI 的 ActionForward。通常的做法是这个 forward 可以从 Struts 配置中读取，当然也可以动态创建。

IncludeAction

类似于 ForwardAction，IncludeAction 也可帮助你集成其它应用组件到 Struts 框架中。但它不是转发到 parameter 属性指定的路径，而是出一个 `include` 指令。

通过包含转发，你可以开始对客户的响应，同时发出一个 include 指令。当其它 servlet 完成时，控制要返回。相反，一旦一个响应开始时候，就不能再发出一个转发了，而且控制也不返回到发出的 servlet，如表 8.5 所示。

表格 8.5 Forward 和 include

Action	响应	控制
Forward	一旦响应开始就不能再发出	控制不返回
Include	在响应时也可以发出	控制要返回

包含多用于表现系统的一部分，并且是某些JSP 模板系统的基础，比如在第11章介绍的Titles系统。关于servlet如何分派请求，参见servlet 规范 [Sun, JTS]。

IncludeAction可以使用的地方是当一个Action对象开始一个响应并且希望其它一些 servlet 来完成它的时候。嗯…有点深奥 … 但就是这样。实际上，IncludeAction的源代码有助于指导你开发你自己的需要包含来自于其它Servlet输出的Action类。

8.3.2. 标准 base Action

标准的base Action 包括：

- BaseAction (Scaffold)
- DispatchAction
- LookupDispatchAction
- SwitchAction

BaseAction (Scaffold)

本章前面，我们归纳了Action 通常要做的事情：

- 检测错误和排队消息
- 调用业务方法
- 捕获异常
- 记录消息
- 路由控制

如果你需要将所有的任务都放入一个良好构建的 Action中，你可以这样干：

```
public ActionForward execute(
                    ActionMapping mapping,
                    ActionForm form,
                    HttpServletRequest Request,
                    HttpServletResponse Response)
       throws Exception {
// 检查前提条件错误；如果发现，则失败
  preProcess(mapping,form,Request,Response);
  if (isErrors(Requset)) {
    return findFailure(mapping,form,Request,Response);
  }
  //尝试逻辑，如果需要，调用 catchException()
  try {
    executeLogic(mapping,form, Request,Response);
  }
  catch (Exception e) {
  // 保存异常；调用扩展点
    setException(Request,e);
    catchException(mapping,form, Request,Response);
  }
```

```
  finally {
    postProcess(mapping,form, Request,Response);
  }
  // 如果队列中有消息，则失败
  if (isErrors(Request)) {
    return findFailure(mapping,form, Request,Response);
  }
  // 否则，检查消息和 succeed (仅对 1_0)
  if ((isStruts_1_0()) && (isMessages(Request))) {
    saveErrors(Request,getMessages(Request,false));
  }
  return findSuccess(mapping,form, Request,Response);
}
```

这就是来自于附加Scaffold包的 Struts BaseAction (`org.apache.scaffold.http.BaseAction`)的 `execute` 方法。这个 Action 在Action工作流内为主要事件提供了一个热点方法。如果需要改变，其子类可以覆写这个热点方法。

STRUTS TIP

总是在你的应用中为定制Action创建一个基础Action类。Action仅会实例化一次，所以创建一个深一点的类层次几乎没有什么性能损失。如果你仔细观察哪些公共需要，可以抽象到一个工具方法，你就会发现Action之间其实有很多代码可以共享。

实践中，你通常可能需要覆写`executeLogic` 方法，有时也覆写`postProcess`方法，但其他具有缺省行为的方法在绝大多数环境下都工作得很好。如果不行，你尽可以覆写这些热点方法为特殊的环境提供特殊的行为。如表 8.6。

表格 8.6 Base Action 热点

Action	说明
preProcess	可选的扩展点，为Action处理前提条件
findFailure	为错误条件返回相应的ActionForward。缺省方法返回一个到输入路径的转发，当有这个输入路径时。如果没有，返回错误转发
executeLogic	执行Action的业务逻辑。可覆写它提供新的功能
catchException	处理这个Action的异常处理

postProcess	可选扩展点，处理Action的某些后置条件
findSuccess	返回正常的无错误状态下的ActionForward。缺省返回mapping.findForward("success").

你可以将 BaseAction 作为你的 Action 类的祖先类，采用或者改变此技术以用到那些你已经使用的基础 Action 中。

1.0 vs 1.1

BaseAction Scaffold 1.0发布包，设计来对Struts 1.1向前兼容。它包括一个存根/stub process 方法， 该方法调用我们首选的execute 方法。这使你的代码可以基于execute 方法，而不管现在使用的是Struts 1.0还是 Struts 1.1。在Struts1.1中，execute 方法被直接调用而旧的perform被忽略。在 Struts 1.0，调用perform 方法是在调用新的execute 方法之后。其中也包括其它钩子来适应这些改变，比如新的消息队列。

关于从Struts 1.0迁移到Struts 1.1，参见第16章。

DispatchAction

Struts 开发人员的一个常用策略是使用同一个 Action 类来处理几个相关的任务。一个很好的例子是对一个数据记录执行基本的 CRUD (创建读写修改删除)数据操作。因为这些操作非常相似，它们就可以最简单地放入同一个类中进行维护。

如果没有 DispatchAction，通常的办法是使用一个 ActionForm 中的隐藏字段来选择相应的动作。 而通过 DispatchAction(`org.apache.struts.actions.DispatchAction`)，开发人员就可以将多个方法在一个单一的 Action 内进行分组。DispatchAction 可以通过隐藏字段的关键字自动选择正确的方法 ;它使用的是反射机制而不是大部分开发人员所使用的逻辑方法。

每个分派方法都必须使用象通用的 Action `perform` 或者 `execute` 方法一样的方法体。(对 Struts 1.0 来说是 `perform` 而对 Struts 1.1 来说是 `execute`。) 隐藏字段的名称在通用的 ActionMapping 的 parameter 属性中传递给 Action。DispatchAction 然后会从请求中获取字段的值，并使用反射来调用相应的方法。

STRUTS TIP

使用 DispatchAction来将相关的操作到组织到一个统一的Action中。将相关操作放在一起简化了维护和流程控制。

例如，你的 DispatchAction 子类就可以包含这样的 “分派” 方法：

```
public ActionForward Create(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest Request,
        HttpServletResponse Response)
throws IOException,ServletException;
```

```
public ActionForward read(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest Request,
        HttpServletResponse Response)
throws IOException, ServletException;

public ActionForward update(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest Request,
        HttpServletResponse Response)
throws IOException, ServletException;

public ActionForward delete(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest Request,
        HttpServletResponse Response)
throws IOException, ServletException;
```

并且在Struts 配置中可以创建这样的配置条目：

```
<action
  path="/dataRecord"
  type="app.recordDispatchAction"
  name="dataForm"
  scope="request"
  input="/data.jsp"
  parameter="method"/>
```

要选择 `delete` 方法, 你可以调用

```
http://localhost/app/dataRecord?method=delete
```

以及包括选择记录所需要的任何其它属性。

实践中，method 字段的值通常是按钮或者表单中隐藏属性的名称 (可以通过JavaScript来设置)。

分派机制本身是透明和富有效率的。开发人员只需要子类化DispatchAction 并提供相应的方法。但是perform 或者 execute 方法不能被覆写， 因为它们要用来选择另一个方法。任何通常可以放入perform 或 execute 方法的功能都可以放入相应的分派方法中。

LookupDispatchAction

选择分派方法的一个方便的方式是将它们连接到按钮。这对于本地化应用来说是个问题，因为按钮的标签可能根据用户的场所来改变。对一个用户来说，按钮可能读着*Delete*； 而对另一个用户来说，可能读着*Borre*。

STRUTS TIPS

当本地化控件和避免依赖JavaScript来选择dispatch 操作都很重要时，可以使用 LookupDispatchAction 而不是DispatchAction。

LookupDispatchAction (org.apache.struts.actions.LookupDispatchAction)通过将标签映射回原本的消息关键字来解决这个问题。那么关键字就可以映射到相应的分派方法。 因为消息关键字不能是相应Java 方法的名称，开发人员就提供一个 hash 表来映射消息关键字和分派方法名称:

```
protected Map getKeyMethodap(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest Request) {
  Map map = new HashMap();
  map.put("button.add", "cerate");
  map.put("button.view", "read");
  map.put("button.update", "update");
  map.put("button.delete", "delete");
  return map;
}
```

在 JSP中, 按钮就应该这样来创建:

```
<html:form action="/dataRecord">
  <html:submit property="method">
    <bean:message key="button.add">
  </html:submit>
  <html:submit property="method">
    <bean:message key="button.view">
  </html:submit>
  <html:submit property="method">
    <bean:message key="button.update">
```

```
  </html:submit>
  <html:submit property="method">
    <bean:message key="button.delete">
  </html:submit>
</html:form>
```

按钮的标签可以根据用户的场所不同而不同，但最终都会选择相同的方法。

概括一下，DispatchAction 和 DispatchLookupAction 间的不同之处在于后者做了一个反向查找来将按钮提交的本地化值匹配到相应的消息关键字。然后消息关键字再映射到相应的分派方法的名称。

规整 dispatch 方法

通常，分派方法在做它们各自的操作时，往往都需要做一些类似的事情：监视异常，检查错误，记录消息等等----即所有那些Action 常做的事情。

虽然Action类需要是线程安全的，你仍然可以提供一些工具方法来供各个分派方法所共享。重要的只是必须记住通过工具方法的方法签名来传递它们需要的变量。这样就将变量放入stack中，它是线程安全的。

例如，如果每个分派方法都需要检查用户的场所，它们便可以通过调用同一个工具方法来进行查找，只要每个实例都传递它们自己的请求拷贝：

```
Locale locale = getLocale(Request);
```

同样的原则也适用于其它数据结构。一个多线程的类，比如Action，是不能在方法间共享数据的，但是方法可以在其方法签名内向其它方法传递实例变量。本节前面涉及的BaseAction类也依赖于这种技术。如果你想要将相关的操作路由到同一个Action，但是有时又需要它们之中的某一个在其自己的Action内进行处理，你便可以让分派方法返回一个forward 以便其它 Action 可以完成工作。

8.3.3. SwitchAction

所有的 Struts 应用至少具有一个模块。某些应用可能被配置为使用多模块。每个模块具有其自己的一套配置文件和表现页面，并且可以象应用中仅有一个模块的情况一样独立进行开发。

当然，在某些时候，如果模块将要以一个单独的应用工作时，它们之间就需要进行交互。SwitchAction (org.apache.struts.actions.SwitchAction) 是一个标准Action，可以切换到另一个模块，然后转发控制到此模块中的路径。

SwitchAction 需要传递两个请求参数：

- Page：一个指向控制在切换这后应该转发去的地方的模块相对的URI (以 / 开始)
- Prefix：控制应该切换被到的应用模块的模块前缀 (以 /开始)。缺省使用0长度的字符串。

相应的应用配置对象将保存为一个请求属性，所以所有后续的逻辑都将假定是在新的应用模块之内。

8.4. 串链Action

当应用增长时，其Action会倾向于封装到一个内部API。开发人员们发现自己总是想把Action类串联成链，创建一个工作流或者宏流程。通常，这是一个信号，意味着业务逻辑严重地和Action相耦合，或者Action 的层次结构太浅。如果这是个需要在Action间共享的功能，它应该被重构进单独的业务类或者在Action 超类中作为Helper类提供。

然而，仍然有一些技术可以让你在需要时将Action链接在一起。Action的 `perform` 方法返回的ActionForward 可以是任何URI, 包括另一个Action。开发人员经常使用这个技术来创建工作流，这时，每个Action只是在一个请求的流程中扮演一部分角色。.

如果以这种方式将控制从一个Action 转发到另外一个，ActionServlet 处理该被转发的请求就像其是直接来自于客户的请求。ActionForm bean 将被重设，重新组装，并且重新校验。并且，如果一切运行良好，则转发到第二个Action。这也经常困扰开发人员, 因为它们想在form bean中设置属性，然后传递给链条中的下一个Action。

最好的解决方式是添加一个开关到你的bean 中，来使其属性保持不变：

```
private boolean mutable = true;
public void setMutable(boolean mutable) {
  this.mutable = mutable;
}

// ...
public setProperty(String property) {
  if (isMutable()) this.property = property;
}
```

在第一个Action中修改了属性之后，可调用`setMutable(false)`方法。然后切换到第二个`Action`。

如果你的 `reset` 方法调用了公开的 setter 方法，那么 `reset` 也会被有效地禁止。如果 `reset` 是直接设置属性，那么它就需要被修改来首先检测 immutable 的状态。如果校验有问题，那么在 immutable 状态为 false 的时候，使 `validate` 返回 true。然而，你想在 Action 间传递的数据可能仍然有效，这在正常条件下使不会失败的。

这个技术的一个变体是，为第二个 Action 创建一个新的 form bean，组装它，并将它添加到请求中去。如果两个 Action 使用同一个 bean 类型，并且你两个都需要，你可以添加一个 `attribute` 属性到其中一个 Action 的 mapping 之中，以便它可以被保存在另一个不同的属性（Attribute）名称下：

```
<action
  path="/item/Edit"
  type="org.apache.gavel.http.ModelHelper"
  name="itemForm"
```

```
  attribute="itemFormAdd"
  scope="Request"
  validate="false"
  parameter="org.apache.gavel.item.Select">
  <forward
    name="continue"
    path="/pages/item/Form.jsp"/>
</action>
```

8.4.1. 来点新鲜的

换句话说，如果你想清除请求上下文以便链接的 Action 可以从新开始，那么就用重定向好了：

```
ActionForward startOver = mapping.findForward("start");
startOver.setRedirect(true);
return startOver;
```

注

从软件架构的角度而言，以某些方式串链Action并不是我想要做的事情。理想情况下，你应该能够在所有需要的地方从Action中调用所有的业务对象。想要将控制转发到其它Action往往意味着业务对象可能过于紧密耦合。或者这意味着Action应该从一个包含子类可以覆写的热点方法的公共类中继承下来。当你在两个Aciion间迷茫的时候，请仔细研究 Scaffold BaseAction 的设计。

当然，在凑合使用串链Action的时候也会有一个机会，例如，如果其它 Action 是用于在一个表现页面的场合中渲染页面的时候。但有效的使用情况非常少见，最好的办法是就事论事。

8.5. Scaffold Action

Scaffold 包包括几个标准的 Action 类，可以在多个应用中多次使用。它们分为两类，见表 8.7 所示。

表格 8.7 Scaffold Action

仅作转发的Action	助手（Helper） Action
SuccessAction	BaseHelperAction
RelayAction	ProcessAction
ParameterAction	AttributeExistsAction

FindForwardAction	RemoveAttributeAction

所有的 Scaffold 标准 Action 都子类化了 BaseAction。针对 Struts 1.0 版本的 BaseAction 提供一个向前兼容的 `execute` 方法体。在本节中，当我们引用到 `execute` 时，我们是针对两个版本的同一方法。

8.5.1. 仅作转发的Action

这是一个十分简单但强大的技术，这里，Action 类被用作一个简单的分派器。它查找一个给定的 ActionForward 并对控制进行中继。Scaffold 包提供这个方案的四个变体，如表 8.8.

表格 8.8 **Scaffold的 forward-only Action**

Action	目的
SuccessAction	转发到成功状况
RelayAction	基于一个已知的运行时参数查找一个转发
ParamterAction	中继之前添加一个运行时参数
FindForwardAction	基于一个已知的运行时参数动态查找一个转发

SuccessAction

类似于标准的 ForwardAction，`SuccessAction` (`org.apache.struts.scaffold.SuccessAction`)可以将控制路由到其它资源，通常是一个 JSP。 但它不是从 parameter 属性获得 URI，而是查找全局转发得到的。

`SuccessAction` 类的 `execute` 方法仅有一行实现:

```
Return mapping.findForward(Request.getparameter(Tokens.SUCCESS));
```

这里，一个 `ActionMapping` 元素就可能是这样配置的:

```
<action
  path="/myPackage/myForm"
  type="org.apache.scaffold.http.SuccessAction"
  name="myFormBean"
  scope="Request"
  validate="false">
  <forward
    name="success"
    path="/pages/myPackage/myForm.jsp"/>
</action>
```

`SuccessAction` 具有同 `ForwardAction` 一样的功效，但是它让你使用 forward 来指定资源，而不是使用 mapping 的 parameter 属性。当然，因为 SuccessAction 非常简单，

如果你想使用你自己的格式而不仅仅是 success，你尽可以创建你自己的版本，那是小菜一碟。

中继Action（RelayAction）

RelayAction (org.apache.struts.scaffold.RelayAction) 查找一个给定的参数，并根据它的值来查找 forward。这在一个页面中有不止一个提交按钮，并且每个提交按钮都有不同的 ActionMapping 调用的时候尤其有效。

注 这是DispatchAction 和 LookupDispatchAction的一个可选替代方案，它希望相关操作都被提交到同一个ActionMapping。其他设计可能需要通过不同的ActionMappings由不同的操作。

一个简单的JavaScript可用来在表单中设置隐藏属性。象SuccessAction一样，Action 的 execute 方法的实现只有一行：

```
return mapping.findForward(Request.getParameter(Tokens.DISPATCH));
```

但是，这里，我们从请求中查找一个参数，并将它的值传递给 findForward()方法。相反，SuccessAction 只是传递一个固定的值给 findForward ("success")方法。

如果传递一个这样的URL

```
http://whatever.com/artimus/do/prospect/Submit?dispatch=update
```

RelayAction 就会在 mapping 中查找一个名称为 update 的 ActionForward，并返回它。这里是一个 RelayAction mapping 的例子:

```
<action
  path="/prospect/Submit"
  type="org.apache.scaffold.http.RelayAction"
  name="prospectForm"
  scope="Request"
  validate="false">
  <forward
    name="update"
    path="/do/prospect/Store"/>
  <forward
    name="cancel"
    path="/do/prospect/Current"/>
  <forward
    name="create"
    path="/do/prospect/createeDonor"/>
  <forward
```

```
    name="donor"
    path="/do/donor/Detail"/>
</action>
```

依赖于 dispatch 参数的值，mapping 可以将控制中继到其它四个 ActionMapping：Store，Current，CreateDonor，以及 Detail。

在 JSP 中， forward 参数可以用一个很小的 JavaScript 来设置，象这样:

```
<html:form>
  // ...
  <html:hidden property="dispatch" value="error"/>
  <html:submit onclick="set('update');">
    UPDATE PROSPECT
  </html:submit>
  <html:cancel onclick="set('cancel');">CANCEL</html:cancel>
  <html:submit onclick="set('create');">CREATE DONOR</html:submit>
  <html:submit onclick="set('donor');">UPDATE DONOR</html:submit>
</html:form>

<script>
  function set(target) {document.forms[0].dispatch.value=target;};
</script>
```

注 隐藏的 dispatch 属性缺省为error。如果 JavaScript 被禁止，那么用户就会被转发到一个错误 error 页面，以便解释运行需要。这里假定dispatch 是ActionForm的一个属性 (推荐)。如果dispatch不是一个ActionForm 属性，那么就应该是使用 <input type="hidden" name="dispatch" value="error"> 。

另一个不需要使用 JavaScript 的技术是本章所讨论的的 FindForwardAction 方法。

STRUTS TIP 请使用 RelayAction 来从Struts 配置中选择和分派Action。这样可以将控制以一种开放的形式进行编码，而不是隐藏在Action之中.

许多菜单页面提供了一系列业务操作。通常，这些菜单项为了运行正确都需要一个参数。例如，一个 search 操作需要一个 String 参数来限制查寻范围。典型地，我们都是开始于一个这样的 URI

```
/do/find/Content
```

并且想将其定制为这样的格式

```
/do/find/Content?content=glockenspiel
```

这里， glockenspiel 是在运行时由用户提供的。

ParameterAction (org.apache.struts.scaffold.ParameterAction) 一看起来就有点象 RelayAction。它查找一个名称为 dispatch 的请求参数并使用 dispatch 的值来查找一个 forward。但是，它接下来还要查找第二个请求参数。第二个参数的值被添加到 URI 的尾部。

Action 要查找的请求参数由 ActionMapping 的 parameter 属性指定。下面是一个示例的 ActionMapping:

```
<action
  path="/menu/Find"
  type="org.apache.struts.scaffold.ParameterAction"
  name="menuForm"
  validate="false"
  parameter="keyValue">
  <forward
    name="title"
    path="/do/find/Title?title="/>
  <forward
    name="author"
    path="/do/find/Author?Creator="/>
  <forward
    name="content"
    path="/do/find/Content?content="/>
</action>
```

以及使用这个 ActionMapping 的 JSP 代码片断：

```
<html:form action="/menu/Find">
  <TR>
    <TD>Find articles by: </TD>
    <TD>
      <html:select property="dispatch">
        <html:option value="title">Title</html:option>
        <html:option value="author">Author</html:option>
        <html:option value="content">Content</html:option>
```

```
      </html:select>
      <html:text property="keyValue"/>
    </TD>
    <TD><html:submit>GO</html:submit></TD>
  </TR>
</html:form>
```

如果用户选择 Content 并且在文本字段中输入 glockenspiel，浏览器将提交

```
dispatch=content
keyValue=glockenspiel
```

然后 parameterAction 会查找 content 的 forward ，并添加关键字值到 URI 的末尾，最后产生/do/find/Content?content=glockenspiel 这样的请求

关于创建菜单页面的详细信息，参见 8.8 节。

FindForwardAction

在一个页面内有多个提交按钮是比较复杂 :特别是它们中有些还进行了本地化或者干脆使用图像按钮的情况下。在使用标准的 HTML 按钮时，浏览器将提交按钮的名称和其标注（label）。但是当使用图像按钮时，浏览器提交的却是按钮的 x/y 坐标。(添加 .x 和 .y 到按钮名称上）。原有的名称根本没有被传递。在这两种情况下，提交的值其实根本没有值。标签可能被本地化了或者干脆因设计人员一时兴起给修改了。即便 x y 坐标是不变的。

这里有一个可靠的信息是按钮的名称，它可以用作关键字来决定请求的是哪一个任务。Action 对象可以遍历所有可能的按钮关键字，但是改变按钮的设置可能会影响到要改变 Action 的源代码。一个好一点的方法是从 Struts 配置中将按钮映射到它们相应的 ActionMapping。 FindForwardAction (org.apache.struts.actions.FindForwardAction) 就会使之容易实行。因为这个技术不是那么直接，让我们从一个具体的 JSP 例子开始谈起。这个例子显示了两个按钮。一个是标准的提交(submit)按钮，名称为 *add* 并有一个本地化标签。另一个是图像按钮，名称是 *delete*:

```
<html:form action="/addDelete">
  <!-- BUTTON --!>
  <html:submit property="button_add">
    <bean:message key="button.add"/>
  </html:submit>
  <!-- IMAGE --!>
  <html:image pages="/images/delete.gif" property="image_delete" />
</html:form>
```

在提交时，请求将包含一个名称为 button_add 的参数，或者名称为 image_delete.x 和

image_delete.y 的参数集。

下面是处理这个表单的示例 ActionMapping。请注意，它为每个按钮都指定了一个局部转发。对于 image_delete 按钮，我们使用 image_delete.x 名称，因为它是一个图像按钮。(大当然 image_delete.y 也可以使用):

```
<action path="/addDelete"
  type="org.apache.scaffold.http.FindForwardAction"
  name="addDeleteForm"
  scope="request"
  input="/pagess/addDelete.jsp">
  <forward name="button_add" path="/do/Add"/>
  <forward name="image_delete.x" path="/do/Delete"/>
</action>
```

或者，如果需要包含参数的话：

```
<action path="/addDelete"
  type="org.apache.scaffold.http.FindForwardAction"
  name="addDeleteForm"
  scope="request"
  input="/pagess/addDelete.jsp">
  <forward name="button_add" path="/do/crud?dispatch=add"/>
  <forward name="image_delete.x" path="/do/crud?dispatch=delete"/>
</action>
```

虽然其它只做转发的 Action 类也通过给定的名称来查找 forward，但是 FindForwardAction 却要更加动态一些。它获得一个有效的 ActionForward 名称的列表，然后检查是否有一些匹配请求参数中的名称。如果有，该 ActionForward 就被选定。所以，如果方法找到 button_add，局部转发 button_add 就会被触发。如果方法找到 image_delete.x，局部转发 image_delete.x 则被触发。

FindForwardAction 的 perform 方法总共只有 5 行：

```
String forwards[] = mapping.findForwards();
for (int i=0; i<forwards.length; i++) {
  if (request.getparameter(forwards[i])!=null) {
    return mapping.findForward(forwards[i]);
  }
}
return null;
```

这就将按钮名称与它们调用的 Action 类之间去除了耦合。如果页面修改了它们使用的按钮，

或者不管怎样使用它们，所要做的修改都只是在 Struts 配置中进行。

8.5.2. Helper Action

Scaffold 提供几个Action类，它们提供了我们通常希望Action做的那些服务之外的服务。 除了作为分派器之外，它们还管理那些用来完成手头任务的特殊的helper对象。如表 8.9。

表格 8.9 Helper Action

Action	用途
BaseHelperAction	创建专门的helper对象
ProcessAction	实例化和执行helper bean，并处理结果
ExistsAttributeAction	检查给定范围内是否存在某个属性
RemoveAttributeAction	从session 上下文中删除一个对象

BaseHelperAction

几乎所有Struts 框架使用的对象都可以在Struts 配置中标识—ActionForward，ActionForm，ActionMapping，Action 对象—即除了业务对象外的所有对象。 业务对象通常封装 (或者“埋藏”) 在Action类中。

STRUTS TIP

将 helpe 对象指定在 ActionMapping 会将一切都处于Struts 配置的控制之中，使应用的架构易于设计、评审和修改。

如果你使用一些轻量的业务对象，BaseHelperAction 可以允许你使得业务对象保持开放，并且一个 mapping 相关的业务对象类型可以指定为它的 parameter 属性:

```
<action
  path="/search/Content"
  type="org.apache.struts.scaffold.BaseActionHelper"
  name="articleForm"
  validate="false"
  parameter="org.apache.artimus.article.SearchContent">
  <forward
    name="continue"
    path="/pages/article/Result.jsp"/>
</action>
```

BaseHelperAction (org.apache.struts.scaffold.BaseHelperAction) 继承了 BaseAction 类。BaseAction 提供了缺省的 Action 控制流程，以便其子类，比如 BaseHelperAction，可以覆写热点方法并且继承剩余的处理流程。因为 helper 多是业务对象，BaseHelperAction 可以覆写 executeLogic 方法来实例化 helper 类。然后就可以调

用其新的热点方法，以及一组 helper：

```
executeLogic(mapping,form,request,response,helpers);
```

子类应该覆写这个新的 executeLogic 方法签名，以使用由 BaseHelperAction 类传入的 helper 类。 缺省版本提供了一些测试代码，指示输入一些字符串，表示每个 helper 作了响应:

```
protected void executeLogic(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response,
        Object[] helpers)
    throws Exception {
  // override to provide new functionality
  response.setContentType(Tokens.TEXT_PLAIN);
  for (int i = 0; i < helpers.length; i++)
    response.getWriter().print(helpers[i].toString());
}
```

ProcessAction

Scaffold ProcessAction (org.apache.struts.scaffold.ProcessAction) 类是 BaseHelperAction 的一个子类，使用 helper 业务对象并且构建在 BaseAction 提供的控制流上。ProcessAction 希望下面的 helper 也是另一个 Scaffold 类 ProcessBean (org.apache.commons.scaffold.util.ProcessBean)的子类。这是一个封装了业务操作的轻量类。象 Struts Action 一样，一个 ProcessBean 可以通过 execute 方法来访问。

当与 ProcessAction 一起使用时，ProcessBean 的 execute 方法被要求返回一个 ProcessResult (org.apache.commons.scaffold.ProcessResult)类。这是一个传输对象，用来封装可能由任意业务操作返回的一些数据、消息等结果的任意组合。

ProcessAction 对 ProcessResult 进行分析。如果结果返回消息，Action 将调用 saveMessage 来将信息暴露给 Struts。如果结果返回数据，它会将数据存储到 servlet 上下文中，在此处表现页面可以找到它们。

ProcessAction 是作为一个 “灰盒” 组件来设计的, 就象 Struts ActionServlet 一样。你可以按其原样使用它，也可以覆写其扩展点，或者扩展某些缺省行为。实际的编程往往发生在 ProcessBean 中。

关于实现 ProcessBean 以及与 ProcessAction 一起使用的例子，参见第 15 章。

ExistsAttributeAction

某些工作流可能希望某些属性已经被设置好了。这可能是一个值对象，用来收集一个向导精灵的一部分字段。也可能是一个用来为 HTML 提供选项的字段。也可能是一个描述记录概

要（profile），用作某个安全方案的一部分，或者其它一些什么东西。

web 应用的混乱使我们要确保这些属性的有效非常困难。有很多方式，用户可以跳过某个应用，或者不经过第一步而直接到达第二步。也有一些对象，我们只有在需要时才进行创建，但一旦我们创建了它们，就有不再需要它们了。

ExistsAttributeAction 就是这类问题的解决方法。它在一个给定的范围内检查一个给定的属性是否存在。如果是，它转到 success forward。如果不是，则转到 failure forward:

```
<action
  path="/Menu"
  name="menuForm"
  type="org.apache.struts.scaffold.ExistsAttributeAction"
  parameter="*;HOURS">
  <forward
    name="success"
    path="/pages/article/Menu.jsp "/>
  <forward
    name="failure"
    path="/do/MenuCreate"/>
</action>
```

范围和属性在 parameter 属性中指定。第一个记号是范围(应用，会话，请求)。要指示所有范围，可以用*号统配符。

第二个记号是属性。请记住 Java 是大小写敏感的，所以 HOURS 和 hours 并不是同一个属性。

RemoveAttributeAction

这个 Action 类可以和需要从会话上下文移除某个属性的工作流一起使用。 它简单地移除 ActionMapping 的 parameter 指定的属性，并且将控制转发到 success:

```
<action
  path="/account/Logoff"
  type="org.apache.gavel.http.RemoveAttributeAction"
  validate="false"
  parameter="userProfile">
  <forward
    name="success"
    path="/pages/account/Logon.jsp"/>
</action>
```

这样可用作为对用户进行登出，或者一个向导工作流的末尾部分。

8.6. Base View Action

Struts 框架提供了 Model/View/控制器架构的控制器组件 (第 1 章)。

但这并不意味着 Action 就不能直接创建响应和渲染视图。框架支持 Action 直接创建响应的理念。Action 必须做工作的是返回 null 并且框架要考虑请求/响应是否满足。如果你的应用具有特殊的视图要求，Action 可以做你需要它做的一切。你也许想要创建你自己的视图 Action 来渲染动态图像，创建 PDF，或者合并 XML 和 XSLT。

一个办法是简单地串链你的视图 Action 并且将常规的控制器 Action 转发到渲染视图的另一个 Action。在这种方式下，你的控制器 Action 可以继续准备数据，并将它转发到其它显示视图的地方。

除了 JSP 或者 Velocity 模板，“其它地方”还可以是一个 Action。

另一个方法是创建一个基本的视图 Action，并且扩展它来创建那些你可能需要的任何控制器 Action。基础 Action 可以实现这样的方法：

```
public ActionForward render(
          ActionMapping mapping,
          ActionForm form,
          HttpServletRequest Request,
          HttpServletResponse Response)
   throws IOException, ServletException;
```

采用设个方法有几个结果：

- 因为这个方法在其方法签名中传递形式变量，它是线程安全的
- 代码容易维护。一个团队可以工作于祖先类，而同时另一个团队则可以专注怎样扩展它
- 因为 `render`方法使用 `perform` 方法体, 方法的接口就如同Action的其它部分一样稳定
- 因为请求不需要被转发，所有事情都可以当场完成
- ActionServlet将不会再次处理请求，或者覆盖原先的ActionForm 和 ActionMapping
- 如果你有不止一个特殊的渲染技术，则可能有不止一个基类。 如果接口兼容，你可以仅仅修改它扩展的那个。
- 如果基类扩展了`DispatchAction` (`org.apache.struts.actions.DispatchAction`), 那么你可以在子类之前直接调用render 方法。
- 在适当的因素下，请求-渲染代码是轻便的。Action 可以传递请求, 响应, 以及其他需要的数据到另一个类中的方法。第二个类可以编写实际的响应，就象是它调用业务层的方法访问实际数据库一样。

8.7. Helper Action 技术

在前一节，我们探讨了可重用的标准Action类。这些Action类实现了几种技术，你可以在你的类中包含它们，以使你的类更加灵活。为了节省你查看源代码的时间，并从中获取精华的部分，我们在此展示了一些可重用的技术： (表 8.10 是一个总结)。

表格 8.10 Helper Action 技术

Action	用途
可选转发	逻辑判断是否某个特定的forward 存在
向前调用	调用有意进行覆写的基方法
捕获链式异常	为链中的每个异常显示一个信息
智能错误转发	可选的转发到输入页面或者标准的错误页面
确认成功	将消息排队以报告状态
替换视图	基于业务逻辑事件选择视图helper
反射方法	通过名称调用 Action 的方法
反射类	通过名称调用Action的helper 对象

8.7.1. 可选转发

这个技术多次使用在helper中，用来提供可选的行为，取决于Action 对象如何被配置。这允许同一个Action 类以不同的方式使用。在Struts 框架中, Action 对象由ActionMapping配置。ActionMapping 定义了一些属性并且也提供一个局部转发列表。

STRUTS TIP 使用可选转发将使你的基本 Action更加灵活和易于使用.

可选转发技术首先检查看是否有特定的ActionForward被定义。如果没有，选择行为被跳过。如果找到，在环境允许的情况下选择行为就会发生。

一个很好的例子是检测 *cancel* 状态。 取消 HTML 表单的提交被内建进 Struts 框架中了。当然，并不是每个表单都是可以取消的，如果可以，控制流会根据 mapping 不同而不同。

下面是一个 ActionMapping，包括一个路由到 cancel 事件的转发:

```
<action
  Path="/item/Store"
  type="org.apache.gavel.http.ModelHelper"
  name="itemForm"
  scope="Request"
```

```
  validate="true"
  input="/pages/item/Form.jsp"
  parameter="org.apache.gavel.item.Store">
  <forward
    name="cancel"
    path="/do/item/Detail"/>
  <forward
    name="token"
    path="/pages/item/tokenError.jsp"/>
  <forward
    name="continue"
    path="/do/item/Detail"/>
</action>
```

下面的代码表示 BaseAction 是如何使用选择转发技术来检测 cancel 命令的。这是 BaseAction 的 execute 方法的第一条语句:

```
ActionForward forward = mapping.findForward(Tokens.CANCEL);
if ((forward!=null) && (isCancelled(Request))) {
  return (forward);
}
```

首先,它检查看是否有 cancel ActionForward 被定义。如果有一个合适的 ActionForward 被定义 (!=null) 并且请求就是取消,则 cancel ActionForward 就会被返回,方法结束。

另一个内建的特征是同步器令牌(*synchronizer tokens*) (见第 10 章)。这样可确保一个表单不被重复提交。同样, 如果定义了一个针对 token 的 ActionForward ,并且 token 确实无效,那么该 token ActionForward 就会被返回。方法剩下的部分将不被处理。

```
forward = mapping.findForward(Tokens.TOKEN);
if ((forward!=null) && (!isTokenValid(request))) {
  return (forward);
}
```

重要的是要记住选择转发的名称(象 cancel 或 token) 应该只在局部上下文中使用。如果在全局转发中也使用了相同的名称,就会发生循环和其它问题。findForward 方法总是首先搜索局部转发,但是如果没找到,它也会在返回 null 之前搜索全局转发。

8.7.2. 向前调用

作为一个通用模式,使一个祖先方法调用一个希望被子类化的抽象方法或者基方法是有帮助的。许多 Struts 开发人员都是在 Action 的 execute 方法进行,以便确保某些标准的操

作会首先发生。(在 Struts 1.0 中，则调用 `perform` 方法)。

一个通用的使用场合是在基于应用的安全下鉴别用户身份。

基类祖先 Action 对象首先检测用户是否登入 (通常在 HTTP session 中检查某个对象)。如果是，则祖先类的 `execute` 方法调用已经被子类化且提供了实际功能的另一个方法。如果没有登入，用户便被路由到登录页面。

通常其它同方法也会被给予一个相似但不同的名称(比如，`doExecute`)。开发人员就可以扩展应用的基础 Action 类，并且覆写 `doExecute` 方法而不是 `execute` 方法。

另一个方法是从子类了的 `execute` 方法调用

```
super.execute(mapping, form, request, response);
```

并且返回非 `null` 的结果。

BaseHelperAction 使用第三种变体形式。它调用另一个名称为 `perform` 的方法，该方法使用了一个不同的方法签名。在此方法实例化业务对象之后，它以

```
return execute(mapping,form,request,response,helpers);
```

语句结束。

这样调用 “其它” `execute` 方法并且传递给它一系列 helper 对象。当子类方法返回，那就是传递到堆上的 ActionServlet 的结果。.

8.7.3. 捕获串链异常

许多 Java 专家都推荐业务对象要抛出自己的异常。在内部，某个组件可以捕获一个 SQL 或者 IO 异常，但我们真正需要的是告诉用户数据存取发生了错误。当然，同时，我们也不想告诉太多的异常的细节。从异常中保留细节对一个分层的、多层的、多平台的应用是特别重要的。业务组件不能直接访问日志，并且异常机制是它自己告诉我们到底那里不对了的唯一方式。

简言之，我们需要有一种方式来保持异常中原有的细节信息，但同时还要有一种更加用户友好的业务异常产生。前者可能是在抱怨说我不能处理一个 SQL 查询了。而后者则可能简单地报告数据存取错误并且建议报告或者联系数据库管理员。我们应该确保日志对这两种异常都忠实记录了，以确保维护所有的细节。

因为异常类是原生设计的，如何做这些事情是个问题。Java 异常机制允许你只能抛出一次异常而不是两次或者三次。当然也可以容易的包裹异常类，用一个异常来初始化下一个异常，但结果是经过多层包裹，失去了细节。我们真正需要的是堆叠（stack）或者串链（chain）异常，以便每一层都可以根据变化加上自己的视点。

然后，在末尾，我们可以将它们全部显示出来，连同在列表底部的起源异常。

这个方法在多层架构下工作的出奇的好。最顶层最接近于用户，所以抛出最对用户用好的异常。最底层则抛出 “低级” 错误，它对我们解决错误有用。

当我们通过链接将异常串链起来，用户友好的消息首先到来，然后是比较详细的消息。用户被告知他们应该首先知道的信息，其余的则可以留给系统管理员。

STRUTS 使用串链的异常在应用的各个层次捕获异常和错误，不会丢失细节。

TIP

令人兴奋的是串链异常在 Struts 中是非常容易实现的!

Java 1.4 提供了一个新的功能来串链异常，但是编写你自己的 ChainedException 类，并用于另一个 JVM 并不困难。Commons Scaffold toolkit [ASF, Scaffold] 包括一个 ChainedException 类可以工作于较老的 JVM。

下面是一个来自于基础 Action 的 try/catch 语句块，它使用了 Scaffold ChainedException 类。它调用了一个成员方法来执行业务操作然后分析得出的结果:

```
// ①
ActionErrors errors = new ActionErrors();
try {
    executeLogic(mapping,form,request,response);
}
catch (Exception exception) {
  // ②
  servlet.log("*** ACTION EXCEPTION: ", exception);
  exception.printStackTrace();
  //③
  errors.add(ActionErrors.GLOBAL_ERROR,
         new ActionError("error.general"));
  //④
  StringBuffer sb = new StringBuffer();
  if (exception instanceof ChainedException) {
    ChainedException e = (ChainedException) exception;
    e.getMessage(sb);
  }
  else {
    sb.append(exception.getMessage());
  }
}
// ⑤
errors.add(ActionErrors.GLOBAL_ERROR,
         new ActionError("error.detail",sb.toString()));
// ⑥
```

①设置一个空的 ActionErrors 集合以备后用，调用业务操作，捕获可能抛出的所有异常。

②如果一个异常被抛出，我们首先记录这个消息，并且加上一个标记以便随后可以容易找到它。我们也输出 stack 踪迹，确保该信息被记录以便后用。

③我们开始添加自己的错误信息到错误集合中。只是说些诸如“处理未能完成，细节随后给出。”之类的套话。

④如果业务回传一个 ChainedException 的子类,我们遍历这个链并且收集所有消息。

⑤为使消息更容易显示在一个表现页面中，我们将它们包裹进一个通用消息模板。消息模板是一个单独的替换码：

```
error.detail={0}
```

所以它只是逐字逐句原样输出消息。

这样错误消息中的结果就是：

1 The process did not complete. Details should follow.

2 A required resource is not available.

3 Cannot connect to MySQL server on localhost:3307. Is there a MySQLserver running on the machine port you are trying to connect to?(`java.net.ConnectException`)

它们依次是:

1 通用错误消息

2 业务对象消息

3 JDBC driver 的原生信息

⑥如果错误集合不为空的话，我们切换到一个错误页面。

这个方法的主要优势是它提供了一个高层次的消息给用户 (“A required resource is not available”) 和较低层次的消息给技术支持(“Cannot connect to MySQL server...”)。所以，所有人都高兴！用户获得一个合理的消息。支持人员获得详细的消息来解决问题。

8.7.4. 智能错误转发

Struts ActionErrors 类的设计目的是使你可以加入任意数量的消息，并且随后可以通过请求送出，并通过一个页面组件来显示出来。这个类也用作校验校验。如果一个校验错误发生，ActionServlet 会自动转发到 mapping 的 `input` 属性。所以，当 `input` 属性存在时，它可能是其它错误信息的合理的目标。

下面这一段代码检测某些错误信息是否都被收集了。如果是，它再检查看是否有一个 `input` 属性针对于当前的 mapping，或者如果没有 `input` 属性，则查找是否有一个错误 mapping 存在：

```
if (!errors.empty()) {
  saveErrors(request, errors);
  if (mapping.getInput()!=null)
    return (new ActionForward(mapping.getInput()));
```

```
  return (mapping.findForward(Tokens.ERROR));
}
```

8.7.5. 确认成功

Scaffold `ProcessResult` 类 (`org.apache.commons.scaffold.util.ProcessResult`) 提供了它自己的消息集。它们主要是用来作为成功操作的确认，或者提醒，而不是报告一个错误条件。这个消息集和 `ActionErrors` 没有关系。BaseAction 提供了一个工具方法，所以它们可以进行协作。此工具方法 `mergeAlerts`，象一个处于消息列表和 Struts ActionErrors 类之间的适配器。下面是这个工具方法的方法签名:

```
protected void mergeAlerts(
        HttpServletRequest request,
        ActionErrors alerts,
        List list) {
```

`mergeAlerts` 方法假定消息是 Struts 风格消息的替换参数。列表中的第一个元素是作为模板。其它一些项目则视为是参数。`mergeAlerts` 调用相应的 `errors.add` 方法直到达到最大四个参数。

BaseAction 提供了其他的支持方法来根据需要创建消息队列，并经它们放入请求上下文中。如果队列已经在请求中，`mergeAlerts` 就添加到其中。应用则通常只需调用 `saveMessages(HttpServletRequest,List)` 方法。

Struts 1.1 也具有一个 `saveMessages(HttpServletRequest,ActionMessages)` 方法。BaseAction 版本向后兼容设计的，并且也能和 Struts 1.1 消息标签一起工作。

在 Struts 1.0 应用中, BaseAction 自动将消息队列存储为错误队列，以便 Struts 1.0 标签能找到它们。

8.7.6. 替换视图

绝大多数 Action 要执行的最终任务是将控制转发到下一层。Action 也可以返回 null，如果由自己直接响应请求的话——但这很少见。 Action 的典型路经一般会分支到一个错误页面，或者在末尾查找一个 `success` 或 `continue` 转发。有时，可能会出现不只一个成功结果。例如，搜索操作经常返回一个空的集。一个页面，比如我们前面展示的，也可能包含能优雅处理这个问题的表现逻辑。另外，控制也许会被送至一个针对空集结果的页面。

注 是否一个MVC 页面应该包含表现逻辑还在广泛争论。一个观点是，一个视图不应该应付业务逻辑事件，比如空的结果集或者用户是否登入。另一个观点是，没有表现逻辑，某些应用中的视图的数量会呈指数增长。如此多的视图会成为一个支持负担，并和MVC 架构的目的背道而驰。

我们的建议是，你根据你的应用需要作决定。

下面是一个展示分支到替换转发的例子。与可选转发技术相反，这个语句块首先查找触发器 (空结果集) ，然后检测一个有效的 ActionForward。这个块可以放在 `Action.execute` 方法的末尾：

```
ActionForward forward = null;
if (resultList.getSize()==0)
  forward = mapping.findForward(Tokens.EMPTY);
if (forward!=null) return forward;
  return mapping.findForward(Tokens.SUCCESS);
```

在这里， 如果 ActionMapping 包含一个 EMPTY 转发，在结果集为空的情况下，我们就会分支到那个页面。否则，我们分支到通常的成功页面，假定这个成功页面的版本能处理空的结果集。如果另一个 mapping 提供了一个 empty forward，Action 将使用那个针对空结果集的转发，并且在结果集有项目返回的时候转到成功页面。

8.7.7. 反射方法

大多数 Struts 开发人员喜欢将相关的任务聚集到一个单一的 Action 类之中。这可以简化应用设计和减小维护成本。具体选择执行哪一个任务的方式是通过反射调用它。Struts DispatchAction (`org.apache.struts.actions.DispatchAction`) 就使用了这个技术。

其基本策略是首先标识方法的名称。DispatchAction 使用 mapping 的 parameter 属性来做这个事情。一旦方法名称被决定，你就可以通过方法签名来查找方法。下面是对 DispatchAction 查找方法的一个改进版本:

```
protected Class clazz == this.getClass();
protected Class types[] = {
        ActionMapping.class,
        ActionForm.class,
        HttpServletRequest.class,
        HttpServletResponse.class }
protected method getMethod(String name)
   throws NoSuchMethodException {
{
  return clazz.getMethod(name, types);
}
```

DispatchAction 还要进行缓存方法以提高性能，但是为了简化去除了那部分代码。你可以在 Struts 源代码中找到完整的代码[ASF, Struts]。

8.7.8. 反射类

一个类似的策略是向 Action 类传递需要调用的一个或多个方法的名称。Scaffold BaseHelper 类就是做这件事——再次使用 mapping 的 paramenter 属性来获取类名称。因为有 Java 的反射技术，从其名称中创建对象是很简单的:

```
Object helper = null;
```

```
try {
  helper = Class.forName(className).newInstance();
} catch (Throwable t) {
  // 错误处理
}
```

如果 helper 使用了一个未知的祖先类或者接口， 它可以被转型至调用的操作方法。下面是一个使用 ActionForm 作为输入调用 ProcessBean 的例子：

```
ProcessBean processBean = (ProcessBean) helper;
ProcessResult processResult = processBean.execute(
    beanUtils.describe(form));
```

8.8. 使用智能转发

使 WWW 如此普及的一个原因是容易导航。页面上的文本可以转换为超链接。用户只需要指向和点击，就可以到达目标页面。在文本后面是指向该页面的路径，路径可能是很长和很繁琐的，但用户并不需要知道这些。他们只需要点击说明文字，然后余下的交给系统来做。

除了随处可见的超链接，HTML 还向我们提供了各种用户接口组件，比如单选按钮，复选框，以及选择列表等等。与超链接一样，它们允许我们向用显示一个普通的语言描述，但将技术描述符返回给服务器。通常这些控件用来有助于容易地填充一些表单，但也可以用来创建菜单系统。用户在选择列标准列表中选择一个地址，然后系统将他们带到相关页面。

构建菜单系统最简单的方式可能是在页面中将地址，或者 URL 嵌入到控件中。并且许许多多的 web 应用都采用这个办法，特别是在使用象 Perl 这样的 CGI 系统中。但我们在用 Struts 构建应用时不采用这种嵌入路径的办法。我们希望用逻辑标识符设计页面，并让 Struts 来匹配标识符和系统路径。

Struts 匹配标识符和系统路径的基本方式是通过 Struts 配置 (struts-config.xml)。Struts 应用持续使用配置来将诸如 `success` 和 `failure` 这样的 ID 匹配到各种应用的地址。那么，我们如何做同样的事情来支持菜单系统呢？

本章早些时候，我们介绍了一个标准的 RelayAction 可以用来在多个提交按钮中做出选择。现在，我们来看看能够如何使用 RelayAction 和 一些其它标准 Action 来在一个选择列表的选项间做出选择。

这是一个多重技术，使用了标准 Action，ActionForm bean，JSP 标签，以及 Struts 配置。为了将它们搞在一起，我们将介绍每个这些组件的编码。

最简单的例子是在应用的各个地址间做出选择。

下面是一个有两个选项的地址:

```
<html:select property="dispatch" >
  <html:option value="reload">Reload Config</html:option>
  <html:option value="create">create Resources</html:option>
```

```
</html:select>
```

这个控件的使用就象是多个提交按钮技术。表单的 dispatch 属性设置为哪个选项被选择。表单被提交到一个具有针对每个选项的局部转发的 RelayAction。然后 RelayAction 在将请求转发到该局部转发指定的路径:

```
<action
  path="/menu/Manager"
  type="org.apache.scaffold.struts.RelayAction"
  name="menuForm"
  validate="false">
  <forward
    name="reload"
    path="/do/admin/Reload"/>
  <forward
    name="createresources"
    path="/do/admin/createResources"/>
</action>
```

这对于简单请求来说非常好，但是如果我们需要包括一个参数又该如何?如果控件被用来为同一个 ActionMapping，或者使用相同的参数名称的 mappings 选择参数，你可以只给选项以参数名称。下面是一个单选控件，显示象 *Day* 和 *Week* 这样的标识符，但是传递相应的小时数给 mapping:

```
<html:form action="/find/Hours">
  <P>List articles posted in the last:</P>
  <P>
    <INPUT type="radio" name="hours" value="24">Day
    <INPUT type="radio" name="hours" value="168">Week
    <INPUT type="radio" name="hours" value="720">Month
  </P>
  <P>
    <html:submit property="submit" value="GO"/>
  </p>
</html:form>
```

当用户提交这个表单，浏览器产生这样的 URI

```
/find/Hours?hours=24
```

或者

```
/find/Hours?hours=168
```

或者

```
/find/Hours?hours=720
```

取决于哪个单选按钮被选择。

实践中，我们可能希望从一个集合中编写这个控件，可使用这样的代码：

```
<html:form action="/find/Hours">
  <P>List articles posted in the last:</P>
  <P>
    <html:options collection="FIND"
                  property="value"
                  labelproperty="label"/>
  </P>
  <P>
    <html:submit property="submit" value="GO"/>
  </p>
</html:form>
```

这是一个教育性的例子。所以我们以硬编码的形式给出了选项，即便我们在实践中并不这么做。硬编码参数，或者通过一个集合来传递，在我们所有的选项到同一个 Action 或者 goto action 使用相同的参数名称时工作得很好。但究竟参数名称是否能够不同？我们可能有大量的 action 针对这个字段或者那个字段的记录查找，以及也许需要将字段提供为参数名称:

```
/do/find/Title?title=Struts
/do/find/Author?cerator=husted
/do/find/Content?content=menus
/do/article/View?article=12
```

我们经常想要在一个单独的组合框内提供的地址，可以让我们选择搜索类型(Title, Author, Content,ID)，然后提供一个用户提供的参数，就像上面的一样。

在每种情况下，我们需要做的就是将参数粘贴到 URI 的末尾。表单也仍然需要在相同的名称下提供参数，但是如果我们这样做:

```
/do/menu/Find?dispatch=title&value=Struts
```

以及转换为这样:

```
/do/find/Title?title=Struts
```

这正是我们想要的。

令人高兴的是,这正是标准 ParameterAction 干的事情。这里是设置它的 Actionmapping 元素:

```
<action
  path="/menu/Find"
  type="org.apache.scaffold.struts.parameterAction"
  name="menuForm"
  validate="false"
  name="keyValue">
  <forward
    name="title"
    path="/do/find/Title?title="/>
  <forward
    name="author"
    path="/do/find/Author?cretor="/>
  <forward
    name="content"
    path="/do/find/Content?content="/>
  <forward
    name="article"
    path="/do/article/View?article="/>
</action>
```

你可能注意到， action 使用了一个叫 menuForm 的 Form Bean。这是一个具有许多菜单选项的公共属性的简单 bean。下面是其源代码:

```
private String keyname = null;

public String getKeyname() {
  return this.keyname;
}

public void setKeyname(String keyname) {
  this.keyname = keyname;
}

private String keyValue = null;
```

```
public String getKeyValue() {
  return this.keyValue;
}

public void setKeyValue(String keyValue) {
  this.keyValue = keyValue;
}

private String dispatch = null;

public String getDispatch() {
  return this.dispatch;
}

public void setDispatch(String dispatch) {
  this.dispatch = dispatch;
}
```

Scaffold 包中的版本(org.apache.struts.scaffold.MenuForm)还包含其它一些方便的属性，但这三个是最重要的。当然，你的 form 可以使用需要的所有属性。它们都会进入请求上下文并呆在那儿。当某一个标准 action 转发了请求，所有的原始参数也跟着转发。你可以从同一个请求中组装多个 ActionForm，只要你喜欢。当请求到达目标 Action 的 mapping 时，它使用的所有 form bean 都会自动组装。

到此为止，我们看到了使用 RelayAction 来在不同的提交按钮和菜单选项间进行选择，以及 ParameterAction 添加一个值到查询字符串之中。在我们的节目中还有另一个这种类似的 Action： FindForwardAction (org.apache.struts.scaffold.FindForwardAction)。RelayAction 依赖于已经有一个已知名称的参数在请求之中——例如，[dispatch=save]。它查找名称为 dispatch 的参数，然后再查找名称为 save 的转发。FindForwardAction 也比较动态。它遍历所有参数名称，逐个检查看是否有和转发名称一样的参数名称。如果有，就返回匹配的 ActionForward。

这可能是匹配多重提交按钮的一个比较好的方式，而不用使用 JavaScript 来设置按钮的 dispatch 属性。如果你有一个名称为 Save、Create 和 Delete 的按钮，forward 也命名为 save, create, 和 delete，FindFowardAction 会自动将一个和另一个进行匹配。

这是一个 JSP 代码的例子，它创建了多个提交按钮:

```
<html:submit name="save">SAVE</html:submit>
<html:submit name="Create">SAVE AS NEW</html:submitl>
<html:submit name="delete">DELETE</html:submit>
```

然后，在 Struts 配置文件中，我们为每个按钮输入对应的 forward：

```
<action
 name="articleForm"
 path="/do/article/Submit"
 type="org.apache.scaffold.FindForwardAction">
 <forward
   name="Create"
   path="/do/article/Create"/>
 <forward
   name="save"
   path="/do/article/Store"/>
 <forward
   name="delete"
   path="/do/article/Recycle"/>
</action>
```

唯一的告诫是，你必须得小心地管理你的转发和控件的名称。FindForwardAction 会检查所有的参数和所有有效的转发，第一个找到的就是它的结果。所以如果有几个控件名称匹配几个转发名称，结果就可能不是你想要的了。

这在你不能添加 `dispatch` 属性到表单，或者不能使用 JavaScript 来设置 `dispatch` 属性的时候尤其有用。但是 RelayAction 在可能的情况下作为首选，因为它更具确定性。

联合运用这些技术可以满足令人惊讶的菜单需求，并将控制流集中在 Struts 配置之中。

8.9. 小结

本章研究了 Struts 应用中主角 Action 类的输入和输出。重载和其它管理 Action 可以在运行时修改 Struts 配置，这是一个令人高兴的开发特征。其它标准 Action，比如 ForwardAction 和 IncludeAction，可以帮助把 Struts 与应用中的其它 servlet 进行集成。DispatchAction 和 新的 LookupDispatchAction 可用来在运行时在 Action 对象中选择方法，并有助于减少单独的 Action 类。 Scaffold 包中可重用的 Action 可以用在你的整个应用中，并且可以减少你需要编写和维护的的定制 Action 类的数量。

9.扩展 ActionServlet

本章内容

- 理解ActionServlet 在应用中的角色
- 使用ActionServlet 扩展组件
- 使用 ActionServlet 扩展点

If the only tool you have is a hammer,

you tend to see every problem as a nail.

—Abraham Maslow

9.1. 来点实在的

首先，对大部分应用来说，Struts ActionServlet 组件可以就让其保持原样就好了。它不需要被子类化（尽管可以子类化），缺省的类就可以在很少的困扰下完成一般的工作任务。从架构的角度来看， Struts ActionServlet 是一个黑盒组件 [Johnson]。

在这本书中，我们经常引用 Struts 控制器 servlet，并描述了它是如何与组成 Struts 框架的其它组件和谐共处。这在实践中也工作得非常好，因为要在日常开发基础上使用 Struts，大部分开发人员仅需要知道 ActionServlet 如何同他们所使用的其它组件进行交互即可。只有在很少的时候，Struts 开发人员，才直接使用 ActionServlet。

1.0 vs 1.1

在Struts 1.0 和 Struts 1.1中，ActionServlet的实现已经完全不一样了——出现了很多变化，在这里我们仅仅涉及Struts 1.1 的ActionServlet。大多数开发人员都并不子类化Struts 1.0的ActionServlet, 或者只在某种很小的方式下进行。此外，ActionServlet设计来就是作为一个单态/singleton的。多数情况下， 应用仅需要实际子类化ActionServlet一次。因为Struts 开发人员在日常开发基础上并不直接使用ActionServlet，所以我们就不比较两个发布版的差异，而做些其它的事情。

对于为什么 Struts 开发人员要远离 ActionServlet，有两个主要原因：

首先，它确实是一个单态（ *singleton*）对象。在很多 Struts 应用中都只有一个 ActionServlet。我们一般不会象创建新的 Action 或者 ActionForm 一样来创建新的 ActionServlet。在 Struts 中，简单得实际上没给 servlet 没留下多少工作需要做了。

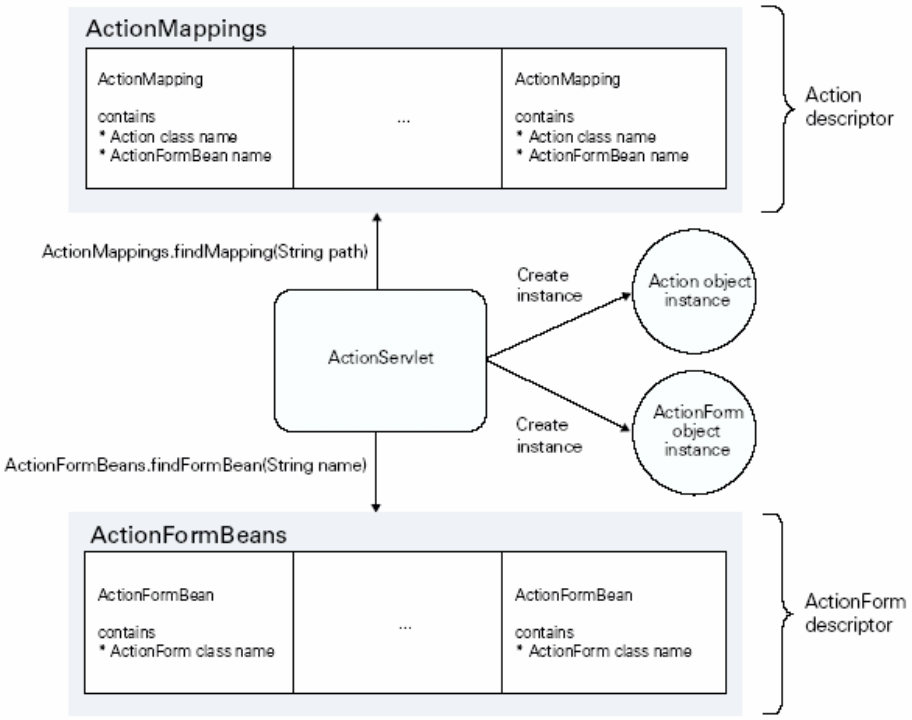
第二，ActionServlet 会在调用其它对象之时花费大量的时间。

虽然我们不需要编写 ActionServlet，但我们需要编写将要被它调用的对象。许多框架都使用这种方式。事实上，这被视为是一个正式的设计模式，控制倒置模式[Johnson]。由 ActionServlet 来协调应用的活动，用户编写来适应框架而定义的方法仅仅被 ActionServlet 所调用。而它们并不在 servlet 中声明的。

定义

控制倒置 (来源于著名的“好莱坞原则”，即“Don’t call us, we’ll call you”)是一个设计模式。在这种模式下，对象在框架中注册为某个事件的处理句柄。事件发生时，框架将调用注册对象的钩子方法。然后对象执行事件需要的特定处理过程。这种模式允许由框架管理事件的生命周期循环，以及允许开发人员将定制的事件处理句柄插入到框架之中。 [Earles]

Struts 框架中的其它对象设计来都是可以子类化的，可以根据应用的需要具体调整设计。这些都是框架的白盒组件 [Johnson]。与此相反，ActionServlet 是个黑盒组件。图 9.1 描述了 ActionServlet 是应用的中心，由它来协调其它对象进行活动。



图表 9-1 ActionServlet 是应用的中心，调用其他对象进行活动

尽管非常象一个黑盒，Struts 1.1 ActionServlet 的设计也是非常具有扩展性的。其实有很多扩展点可以利用，可以非常容易而简洁地创造一个子类，以实现特定的行为。

大多数扩展点都通过能插入框架控制器的对象来提供。使用可插入组件允许你可以在完全不用创建一个新的 ActionServlet 子类的情况下改变其关键的行为。使用这些新组件的一个原因是，这样可以允许不同的应用模块使用不同的行为。

(Struts 模块应用配置参见第 4 章)

9.1.1. Servlet三人帮

所有的可插入组件中，*RequestProcessor* (`org.apache.struts.action.RequestProcessor`) 是最强大的。RequestProcessor 是 ActionServlet 的实际核心；它处理顶层的请求，而 Action 则处理某个特定的 URI 请求。

RequestProcessor 要处理的一个顶层的问题是异常处理。异常类是可以注册到某个处理句柄中的。如果某个已注册的异常被抛出，RequestProcessor 就将该异常传递到相应的 *ExceptionHandler*(`org.apache.struts.action.ExceptionHandler`)。你也可以使用缺省的异常类，或者为每个异常类型提供相应的子类。

许多应用都需要访问它们特定的资源。为了使 Struts 更容易初始化特定的资源，你可以注册一个插件 Action 到控制器中。然后由控制器在启动时调用 Action 的 `init` 方法，在退出时调用 `destroy` 方法。Action 能够访问调用的 servlet，并能够用来做一切常规 servlet 能做的所有事情。

这三个扩展点——RequestProcessor, ExceptionHandler, 和 PlugIn Action——并没有给 ActionServlet 留下太多的工作要做。事实上，ActionServlet 在运行时要做的所有事情仅仅是为相应的应用模块选择相应的 RequestProcessor 而已。

剩余的类简化了 Struts 配置生命经周期的管理。它使用 Digester 来根据配置文件创建需要调用的对象，然后在应用退出时销毁它们。

本章剩余部分将讨论关于使用“3 人帮”来扩展 ActionServlet 的问题，而不是子类化 ActionServlet 本身。这些类，包括它们的子类，都可以通过 Struts 配置文件插入到框架控制器之中，具体配置参见第 4 章。

9.2. RequestProcessor

当一个请求到达时，ActionServlet 选择应用模块并将请求转给相应的请求处理器。每个模块都可以载入其自己的 RequestProcessor 子类，或者使用框架提供的默认请求处理器。

RequestProcessor 是一个灰盒组件。你可以原样使用它，就像一个黑盒组件一样，你也可以扩展它以提供更加特定的行为，就像一个白盒组件。戴上它的白帽子，请求处理器就提供了一些扩展方法，你可以对它们进行覆写来方便地改变它的某些行为。而戴上黑帽子后，请求处理器提供了大多数应用都适合的缺省行为。这些扩展方法总结与表格 9.1 之中。

扩展方法	备注
processMultipart	用一个包装类包装多部分(MultiPart)请求
processPath	表示一个 path 组件，将用来选择一个mapping
processLocale	如果请求发生，为当前用户选择一个场所
processContent	设置请求中的内容类型
processNoCache	如果在该模块中被激活，则设置请求中的no-caching 头
processPreprocess	作为一个通用的预处理钩子
processMapping	表示请求的mapping
processRoles	检查执行该action需要的角色
processActionForm	为该mapping创建或者取得ActionForm bean
processPopulate	从请求中组装 ActionForm
processValidate	处理该mapping标明的包含
processForward	处理该请求相关联的ActionForm bean
processInclude	处理该mapping标明的 forward
processActionCreate	创建或者取得处理该请求的Action 实例
processActionPerform	调用Action 实例处理该请求，返回一个ActionForward

processActionForward	处理返回的ActionForward 实例

请求处理器的子类可以通过 Struts 配置文件向控制器中注册，其配置元素可以是这样配置：

```
<controller
  processorClass="myApp.MyRequestProcessor"/>
```

关于通过 Struts 配置文件注册对象的更多细节，参见第 4 章。

9.2.1. process 方法

所有的扩展方法都在 process 方法中被依次调用。process 方法负责处理 HttpServletRequest 并创建相应的 HttpServletResponse。缺省的请求处理器通过调用表 9.1 中所列方法来完成这个任务。

为了创建响应，processActionForward 方法通常将请求发送到一个 JSP 页面或者其它资源。但是在 process 方法返回时，API 要检查状态，看响应否已被完成。

重新覆写所有的扩展方法是一个最直接的方式，特别是在你对所有的资源都能支配的情况下。通常情况下，你可以执行你的定制行为，然后调用超类的方法来执行默认的行为。

9.2.2. processRoles

请求处理器的扩展最有可能被完全覆写的方法可能是 processRoles。该方法负责检查是否某个用户具有访问某个 action 的权限许可。缺省的行为是使用标准的 Java 安全 API [Sun, JAAS]，当然很多用户也有它们自己的安全方案。通过覆写 processRoles，你可以完全调用你自己的 API。

缺省的行为是检查 ActionMapping 指定的安全角色列表是否匹配请求参数暴露所的标准 isUserInRole 方法。如果用户在特定的列表中，访问将被授权允许。

标准的基于容器的安全系统一般都需要你设置一些 url-pattern，然后配置能够访问这些相应模式所匹配的资源的角色。Struts 基于 action 的安全系统能够让你指定那些用户可以访问特定的 ActionMapping。

当使用标准 API 时，这些系统不会发生冲突，能够一起使用。程序清单 9.1 展示了 processRoles 方法的一个实现。

```
protected boolean processRoles(
        HttpServletRequest request,
        HttpServletResponse response,
        ActionMapping mapping)
    throws IOException, ServletException {
  // ①该 action 是否被角色安全所保护?
  String roles[] = mapping.getRoleNames();
  if ((roles == null) || (roles.length < 1)) {
    return (true);
  }
```

```
  // ②检查当前用户是否在要求的角色列表之中
  for (int i = 0; i < roles.length; i++) {
    if (request.isUserInRole(roles[i])) {
      if (log.isDebugEnabled()) {
        log.debug(" User '" + request.getRemoteUser() +
             "' has role '" + roles[i] + "', granting access");
      }
      return (true);
    }
  }
  // ③当前用户没有获得该对 action 的授权
  if (log.isDebugEnabled()) {
      log.debug(" User '" + Request.getRemoteUser() +
         "' does not have any required role, denying access");
  }
  response.sendError(HttpServletResponse.SC_BAD_REQUEST,
         getInternal().getMessage("notAuthorized",
         mapping.getPath()));
  return (false);
}
```

通过覆写 processRoles，你可以很容易地将这个特征转移到专用的基于应用的安全方案之上。不需要调用 Request.isUserInRole 方法，你的子类化方法可以在你自己的 API 中调用相应的方法。当然，角色并不是必须包含角色本身，而是包含你的系统可以用来量化访问权限的记号。

取决于你的安全概要，你可能需要要求每一个 ActionMapping 都有一个 role 属性，即便是他只是一个记号，比如是表示匿名访问的*通配符。 你甚至也可能根本不需要使用 `roles` 属性。不管如何方法都会被调用，并被用来实现一些不同的访问方案。请求和 `mapping` 对象被传入方法体签名之中，这意味着你可以访问你可能需要用来检测用户凭证的所有东西。

实现 `processRoles` 是典型的请求处理器的扩展方法。所有这些方法都可以被以相同的通用方式覆写以提供你所需要的特定行为。

9.3. ExceptionHandler

Action 对象大多数情况下是 ActionServlet 的代表，并可以包括异常处理。另外，你也可以让 Action 将所有或者部分检测到的异常传递给 ActionServlet，因为它可以捕捉所有异常。

如果 servlet 没有捕捉到异常，它就会进行检查，看是否为某个异常或者其超类注册了异常

处理。它首先检查 ActionMapping 的局部异常处理器，然后再是全局异常处理。

如果 Servlet 发现一个异常，它就会使用异常的详细信息调用处理器。如果没有，Servlet 就会重新抛出一个异常，并且可能会中止在一个声名狼藉的“白屏”之处，因为它没有什么信息。

ExceptionHandler 可以通过 Struts 配置文件中的一个元素来进行配置，就像这样：

```
<exception
  type="org.apache.struts.webapp.example.ExpiredPasswordException"
  key="expired.password"
  path="/changePassword.jsp"/>
```

如果 ExpiredPasswordException 被抛出，缺省的处理器就会根据指定的关键字创建一个 ActionError，并使用其中指定的路径创建一个 ActionForward 。控制跟着被转发到模块中的/changePassword.jsp 页面。

一般来说，该 JSP 会从模块的默认资源束中查找和显示关键字为 `expired.password` 的信息，并给用户一个重新修改密码的机会。如果忽略了 `path` 属性，缺省的处理器将使用 ActionMapping 的 `input` 属性代之。

`<exception>` 元素还可以接受一些其它属性，它们都列在附录 B 的 struts-config API 参考之中。这些属性包括 `handler` 和 `className`，可以用来表示一个定制的 ExceptionHandler，以及(如果需要)该处理器的定制的配置 bean。

定制的 ExceptionHandler 必须子类化默认的 handler (`org.apache.struts.action.ExceptionHandler`)。进入方法是 `execute`，它提供了一个参数集：

```
public ActionForward execute(
        Exception ex,
        ExceptionConfig ae,
        ActionMapping mapping,
        ActionForm formInstance,
        HttpServletRequest request,
        HttpServletReponse response)
    throws ServletException
```

就像编写 Action 的 `execute` 方法一样，你在完成时返回一个 ActionForward 之前可以做很多东西。如果基于某些原因你的处理器自己完成了响应，你也可以返回一个空的 ActionForward。

ExceptionConfig bean (`org.apache.struts.config.ExceptionConfig`)代表了原始的 Struts 配置文件中的`<exception>` 元素。如果你需要某些特定的属性集，你可以创建你自己的 ExceptionConfig 子类，并使用`<set-property>` 元素来从模块的配置文件中初始化它们。

9.4. PlugIn

许多应用都需要访问特定的资源。它们可能是数据访问组件，属性文件，或者其它大量的业务层组件。许多开发人员扩展了 Struts，比如菜单系统，可选的校验器，数据转换器，XSL 转换器等等。这些组件都需要解析自己的配置文件并创建它们自己的控制器对象。

在一个常规的 Java web 应用中，这些对象通常在 Servlet 的 `init` 方法中初始化并在 `destroy` 方法中销毁。我们可以子类化 ActionServlet，并做同样的事情，但在开发人员使用第 3 方扩展组件的时候可能会造成冲突。

因为在大多数情况下，Action 已经是 Servlet 的代表了，框架提供了一个 PlugIn 接口(`org.apache.stuts.action.PlugIn`)以允许 Action 实现 `init` 和 `destroy` 方法，就像在 Servlet 中一样。因为 Action 可以作为成员变量访问 Servlet ，那么一些可以在 Servlet 中初始化和释放的资源也可以在 PlugIn Action 中进行处理。

PlugIn 是通过下面的配置元素在配置文件中向控制器注册的：

```
<plug-in className="myApp.MyAction>
  <set-property
    property="key"
    value="MY_APP_KEY"/>
</plug-in>
```

className 属性指定的 Action 类必须实现 PlugIn 接口并且提供 init 和 destroy 方法。如果 Action 有一些需要初始化的成员变量，标准的<set-property> 元素可以用来传递一个值给任何属性。

启动时，控制器先初始化 PlugIn Action，设置一些属性，然后调用 `init` 方法。而在退出时，控制器在释放每个 PlugIn Action 前将调用它们的 `destroy` 方法。

9.5. 小结

为扩展控制器 servlet，大多数应用其实根本不需要子类化 Struts 1.1 ActionServlet。通常，你可以通过子类化控制器的扩展组件或者插入一个预编写的组件来创建你希望的特定行为。通过插入标准的组件，或者你的子类，你可以优雅地修改控制器以产生需要的行为。

在本书第 3 部分，我们离开控制层，仔细关注表现层，那么你就可以编写应用的余下部分。

Part 3
构建表现页面

本书第 3 部分将关注应用的表现部分——页面。下面讨论的页面将反映 Struts/MVC 架构的关键原则：在页面显示之前，很多真正的东西已经完成。

10. 显示动态内容

本章内容

- *理解为什么使用 JSP 标记*
- *安装标记扩展*
- *使用 Struts JSP 标记*
- *JSP 标记一瞥*

Nothing is particularly hard if you divide it into small jobs.

—Henry Ford

10.1. 标签—就是你

本书的第一部分，我们就涉及到了业务数据和规则的处理。在这一章，我们会集中来看看在数据已经获取之后发生的东西，顶层的逻辑已经应用完毕，随即控制器将请求送出到表现层——将所有需要的数据装配起来完成工作。

Struts 发行包包括一个强大的预编写的 JSP 标签库以帮助将框架和 JSP 页面集成在一起。一个新的标准标签库，JSP 标准标签库[Sun, JSTL]，也已诞生。

在这一章，我们将介绍将 Struts 标签扩展用在你的应用中。

你可以使用 JSP 标签来预装文本字段或者选择列表，以及管理复选框和单选按钮数组。Struts 也可以和 JavaScript 一起工作得很好,所以你可以创建其它一些让页面设计者羡慕的 HTML 几乎不能支持的聪明的部件。

10.1.1. JSP标签—你到底好在哪里?

我们在整本书中都在使用 JSP，但还没有谈到它到底它是什么，或者为什么使用它。我们回头看看并快速复习一下为什么我们要用 JSP 标签来构建 web 页面。如果你已经是一个定制标签的热衷者并且没什么耐心再听我罗索，可尽管跳到第 10.3 节。

大多数 web 应用都依赖于标准的浏览器来显示信息。而标准的浏览器又依赖于标准的 HTML 来格式化信息。经常，我们的应用都要求显示的信息不是静态的预编写的页面，而是针对特定用户的动态内容。首先，这看起来并不是十分困难。毕竟 Java servlet 可以容易地动态编写 HTML——我们可以象写 HTML 一样，送出平面文本来显示和打印 HTML：`out.println("whatever")`。图 10.1 显示了如何使用 Java servlet 来编写动态页面。

```
app.ChaseBean chase = new ChaseBean("dog","cat");

// ...
out.println("<HTML>");
out.println("<HEAD><TITLE>The Chase<TITLE><HEAD>");
out.println("<BODY>");
out.println("<H1>Welcome to the Chase</H1>");
out.println("<H2>Our story so far:</H2>");
out.println("<FONT SIZE=\"+1\" FACE=\"Times\" COLOR=\"#FFFFFF">);
out.print("The big ");
out.print(chase.getChaser());
out.print(" &amp; ");
out.print("the little )"
out.print(chase.getChasee());
out.println(" chased each other. ");
```

```
out.println("</FONT>");
out.println("</BODY>");
out.println("</HTML>");
```

图表 10-1 以古老的方式来渲染 HTML 页面

在 HTML 代码被创建的时候，两个动态变量，dog 和 cat，也会合并到页面之中。web 浏览器将接收和显示这种方式送到的页面，就象显示预编写的静态的页面一样。如果 ChaseBean 被设置为象 fox 或 hound 这样的东西，用户就会获得一个不同的结果。图 10.2 显示了浏览器接收到标签，以及显示结果。

What the browser sees:

```
<HTML>
<HEAD><TITLE>The
Chase<TITLE><HEAD>
<BODY>
<H1>Welcome to the Chase</H1>
<H2>Our story so far:</H2>
<FONT SIZE=\"+1\" FACE=\"Times\"
COLOR=\"#FFFFFF">The big dog &amp;
the little cat chased each other.
</FONT>
</BODY>
</HTML>
```

What the browser displays:

Welcome to the Chase

Our story so far:

The big dog & the little cat chased each other.

图表 10-2 浏览器看到的东西和显示出来的样子

但是以这种方式进行复杂的页面输出简直是一件令人发疯的事情，并且非常难以维护。不使用 println 的另一个选择就是使用特殊的 Java 类和方法来产生 HTML。图 10.3 展示了渲染 HTML 的另一个方法，使用嵌套类来产生标签。

```
app.ChaseBean chase = new ChaseBean("dog","cat");
  // ...
  Html html = new Html()
    .addElement(new Head()
    .addElement(new Title("The Chase")))
    .addElement(new Body()
    .addElement(new H1("Welcome to the Chase"))
    .addElement(new H2("Our story so far:"))
    .addElement(new Font().setSize("+1")
    .setColor(HtmlColor.WHITE)
    .setFace("Times")
    .addElement(chase.getChaser());
    .addElement(" &amp; ");
    .addElement("the little )"
```

```
    .addElement(chase.getChasee());
.addElement(" chased each other. ");
output(out);
```

图表 10-3 设置 println 的包

至少图 10.3 中的方法摆脱了生硬的 HTML 语法。但是即便最好的 HTML 生成器也是很难从这些树来看到森林。只有极少数开发人员能够想象出页面会看起来是什么样子。为了看到最后版本的样子，开发人员不得不重新编译这些类，然后重新部署它们。

服务器页面

解决编写动态页面问题的一个流行方法是使用服务器页面（*server page*）。有多种风格的服务器页面: ASP, ColdFusion 页面, PHP 页面, 服务器段包含（SSI），Velocity 模板, 当然还有 JSP。 它们都使用相同的基本方法：

- 使用服务器页面标签来创建一个HTML风格的页面以表现动态特征；
- 当接收到对该页面的请求时，服务器页面用来构造一个动态响应；
- 响应作为标准的HTML格式返回。

下面是关键的优点:

- 服务器页面不是一个需要编译进核心应用的程序文件；
- 服务器页面更加类似于标准的web 页面。

这些优点使服务器页面非常容易创建和维护，特别是，对非编程人员来说。服务器页面可以使用可视化编辑器来进行维护，比如Macromedia's Dreamweaver [Macromedia]。

JSP

JSP 对服务器页面标记提供了两种截然不同的方法。开发人员可以使用脚本小程序 *scriptlet* 直接在页面中放入 Java 代码。Scriptlet 快速，相对容易，并且十分强大。图 10.4 展示了 JSP scriptlet 方法。

```
<jsp:UseBean id="chase" scope="page" class="app.ChaseBean"/>
<HTML>
  <HEAD><TITLE>The Chase</TITLE></HEAD>
  <BODY>
    <H1>Welcome to the Chase</H1>
    <H2>Our story so far:</H2>
    <FONT SIZE="+1" FACE="Times" COLOR="#FFFFFF">
      The big "<%= chase.getChaser() %>" &amp;
      the little "<%= chase.getChasee()%>">
      chased each other.
    </FONT>
  </BODY>
```

```
  </HTML>
```

图表 10-4 Scriptlets 允许在 HTML 混合 Java 代码和表达式

第二个方法是使用标签扩展。 JSP 标签类似于 HTML 标签，并且使用类似的格式和语法。编写它们要困难一些，但仅需要编写一次， JSP 标签比较容易使用并在长期运行时容易维护。 图 10.5 显示了 JSP 标签方法。

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %
<HTML>
  <HEAD><TITLE>The Chase</TITLE></HEAD>
  <BODY>
    <H1>Welcome to the Chase</H1>
    <H2>Our story so far:</H2>
    <FONT SIZE="+1" FACE="Times" COLOR="#FFFFFF">
      The big <bean:write name="chase" property="chaser"/>
      &amp; the little
      <bean:write name="chase" property="chasee"/>
     chased each other.
    </FONT>
  </BODY>
</HTML>
```

图表 10-5 JSP 页面看起来很像 HTML.

10.1.2. Struts和JSTL

虽然 Struts 标签强大和易用，但它却不是这城里唯一可玩的游戏。Sun 的新的 Java 服务器页面标准标签库(JSTL) 实现了一套非常有用的标签，有好些与 Struts 标签库重叠。JSTL 要求容器支持 Servlets 2.3 和 JSP 1.2，比如 Tomcat 4 或者 Resin 2。简言之，我们可以希望容器针对 JSTL 作了优化，使它作为是大部分应用的基本标签库选择。

当然，JSTL 不能消除对定制标签的需要。开发人员会发现很多实例证明，可以很方便的编写一个标签，以一种确实的方式完成准确的内容。但是，象常规的 Struts 标签库一样，JSTL 消除了对那些执行大部分应用需要的公共职责编写定制标签的需要。

你应该尽可能使用 JSTL 标签而不是 Struts 标签吗？是的，如果你的容器支持 Servlets 2.3 和 JSP 1.2，并且这就是你想做的事情。

大部分 Struts 标签都是填补空缺，并没有和框架耦合的很紧。如果 JSTL 已经存在，大部分 Struts 标签将不再需要编写。

当然，这个时候，Struts 标签是许多应用的一个整体的部分，并且随着 Struts 发行更新。但是你可以确定的是，Struts 小组并不是和 JSTL 竞争。你可能期望一些新的 Struts 标签基于 JSTL 或者作为 JSTL 的附加件。

> **注** 本书出版时，Struts 标签库的 JSTL 版本已经加入到开发构建之中。这个库可能是在 Struts 1.1 的最终发布版的一部分。参见本书的站点[Husted]获取 Struts 1.1 勘误表或者附件。

JSTL表达式语言

JSTL 包括一个表达式语言(EL)，提供了一个针对 scriptlet 的更清晰的替代选择。表达式语言对那些定制标签用户来说，在他们需要传递多个动态参数到 JSP 标签时特别有用。

现在，定制标签的语法仍然有一个盲点。属性可以在标签体中被传递，该标签又可能包括其它定制标签的输出。所有其它属性都必须在标签自身中被传递，并且对其它标签的嵌套是不允许的。

所以，这个写法：

```
<some:tag attribute="<bean:message key='...'/>">
```

是不能被编译的，因为JSP语法希望你这样做：

```
<some:tag><bean:message key="..." /></some:tag>
```

这个一般能够工作，但是有时不只一个属性是动态的。Struts 标签通过让你指定Bean的名称而不是数据来进行。标签然后从Bean中获取属性。但对其它来说，你可能需要使用一个运行时表达式，比如

```
<some:tag attribute='<%=myDynamicString + "txt" %>'/>
```

当在 JSP 标签属性使用运行时表达式时， 重要的是对属性值使用完整的表达式，就象我们这里一样，否则表达式不能被编译。

JSTL 表达式语言提供了另一个选择:

```
<c:set var="msg">
  <bean:message key="..." />
</c:set>
<tag attribute="$msg"/>
```

这里，我们首先将<bean:message>标签的输出捕获为一个页面范围的属性（Attribute）变量，然后我们使用 JSTL 表达式语言来将其写入到非 JSTL 标签中的属性。当然，对 JSTL 来说，其内容远不止这个简单的例子这么多。

通常，你应该能使用 JSTL 来代替一般的 Struts bean 和 logic 标签 (如果你想的话)。但同时你也许需要继续使用 Struts html 标签，因为这些标签要特别些。

JSTL 和 Struts 框架能在一起工作得很好，因为它们某些目标是相同的。它们都试图避免使用 scriptlet，都鼓励使用 JSP 页面来作为 MVC 架构的视图部分。

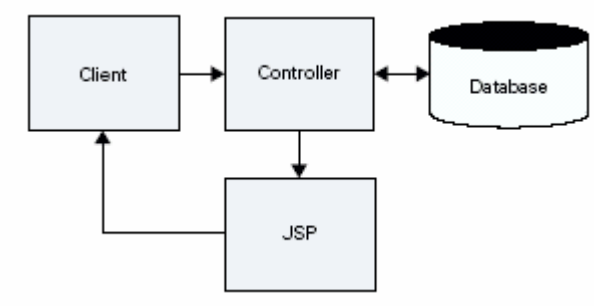
10.1.3. Struts标签和MVC

Struts JSP 标签提供了大部分应用需要用来创建 MVC 架构表现页面的所有功能。这很重要，

因为许多开发人员都想从过去那种以页面为中心的 Model 1 web 应用中解脱出来。

在一个彻底的 Model 2–MVC 应用中，请求并不直接到达表现页面。请求首先经过控制器。如图 10.6 所示，仅仅当业务数据被获取，并且业务规则被应用后，控制器才处理表现页面。因为其它所有工作都完成了，页面的唯一职责就是输出结果——这就是 JSP 标签引入的地方。JSP 标签可以访问保存在 servlet 上下文中的值对象，并且使用从这些值对象获得的数据来创建动态内容。

如果你现在还没猜到，第 10.1 节的标题是双关的。在一个 MVC 环境中，控制器发送请求到页面，说，“标签—就是你了！”另一层意思是，和以前的替代技术相比，标签就足够了。



图表 10-6 MVC 应用中的程序流

往后，在第 10.2 节，我们将简短地讲述标签扩展的工作基本原理，包括如何编写和安装，以及你可以用他们来做什么。在 10.3 节，我们介绍 Struts 标签库，着眼于标签库的总体设计。然后在 10.4 节，涉及到具体内容，我们将使 Struts 标签库运行起来。这里我们将讲述标签的特别之处，包括基本原理，重要技术以及通用应用。本章的目的是缩短你的学习曲线，以便你可以快速地将 Struts 标签集成到你的项目之中。Struts 标签库开发人员指南和技术文档[ASF, Struts] 非常好，推荐阅读。但是要帮助你开始使用它们，本章提供了一个关于标签扩展和 Struts 标签的优雅的介绍，并强调它们如何工作，以及你如何将它们用在你的系统之中。

10.2. 标签扩展

在 10.1 节，我们沿记忆的小道愉快的漫步了一下，看了看 Java web 应用从最初的命令行形式，到使用模板的方式，再到可以使用可视化的外部编辑器来进行管理的服务器页面。在这一节，我们将看看标签扩展是如何编写的，以及标签扩展和其它一些技术的区别，然后介绍 Struts 标签。在此基础上，本章剩余部分将探索如何使用 Struts 标签。

10.2.1. 标签扩展是如何编写的?

在Java中编写JSP标签是使用标签扩展API。这个类设计来解析 XML 格式的标签，并使用标签的属性作为传递类方法的参数。实际用起来，标签扩展就是允许你可以用类似于HTML的XML语法调用Java函数。例如，标准的HTML base 标签看起来像这样:

```
<base href="http://mydomain.com/myapp/index.jsp">
```

而 Struts `base` 标签看起来却是:

```
<html:base/>
```

在被 JSP 页面渲染时，Struts `<html:base>` 标签将会被转换成标准的 HTML base 标签，而且适当的路径会自动包含进去。在此包裹之下，其实有一个 Base 标签类与 base JSP 标签对应。Base 标签扩展了一个 API 类，标签支持并重写了热点方法，即 doStartTag 方法。这是一个构造 HTML 的非选拔的 Java 方法，非常类似于你用来产生响应的普通 Servlet。

```
public class BaseTag extends TagSupport {
  // ...
  public int doStartTag() throws JspException {
  HttpServletRequest request =
            (HttpServletRequest)PageContext.getRequest();
  StringBuffer buf = new StringBuffer("<base href=\"");
  buf.append(request.getScheme());
  buf.append("://");
  buf.append(request.getServerName());
  buf.append(request.getRequestURI());
  buf.append("\"");
  if (target != null) {
    buf.append(" target=\"");
    buf.append(target);
    buf.append("\"");
  }
  buf.append(">");
  JspWriter out = PageContext.getOut();
  try {
    out.write(buf.toString());
  }
  catch (IOException e) {
    PageContext.setAttribute(Action.EXCEPTION_KEY, e,
    PageContext.REQUEST_SCOPE);
  throw new JspException(messages.getMessage(
          "common.io", e.toString()));
  }
  return EVAL_BODY_INCLUDE;
  }
}
```

清单 10.1 BaseTag 示例

这段代码可能让你想起在图 10.1 和 10.3 中我们展示的东西。主要不同之处在于一个特殊的

类，它设计来可以重用，是它封装了 HTML 标记。应用可以共享定制标签库，Tag 类型的标记和普通的 JSP/HTML 标记集成得很好。

在实践中，大多数开发人员都不需要编写他们自己的标签。可以使用某些通用的标签库，包括新的 JSTL。但是一旦需要，这也是一个选择。典型地，开发人员会简单的导入一个现有的标签库到页面中 (是否可以说重用?) 并且以标记方式开始工作，看起来就象扩展了 HTML 一样。

在讨论标签扩展时，开发人员使用了大量相关的词句。一个词汇表可以给你直接的印象。表 10.1 定义了一些在讨论 JSP 标签扩展时使用的词汇。

10.1 Struts 标签库术语

术语	定义
标签扩展	通过标签库可以在 JSP 中使用标记的机制或者 API.
标签库	一个封装了功能的动作(标签)集，可以用在 JSP 之中
taglib	标签库的缩写。
定制标签或 JSP 标签	组成标签库的各个动作(标签)

10.2.2. 如何安装标签扩展?

安装标签扩展并在你的页面中使用它们，有 3 个步骤

1 安装 JAR 和 TLD 文件.

2 更新应用的 web.xml 文件

3 导入新的 Taglib 到你的页面中

安装JAR和TLD文件

标签通常是以二进制的 Java Archive (JAR) 文件发布的。你可以将 taglib JAR 文件放在 WEB-INF/lib 文件夹下，就像其它 jar 组件一样。

在 taglib 发行包里面还有另一样东西: 标签库描述符(TLD) 文件。TLD 通常被直接放在 WEB-INF 目录下面。JSP 服务在编译 JSP 时使用 TLD 来检查标签语法：

```
\WEB-INF\lib\struts.jar
\WEB-INF\struts-bean.dtd
\WEB-INF\struts-html.dtd
\WEB-INF\struts-logic.dtd
```

修改应用的 web.xml文件

为提供最大的灵活性，你的 JSP 将使用一个逻辑引用到相应的标签库。这使得你可以迁移甚至修改标签库而不会影响到页面(只要标签库是二进制代码兼容的)。逻辑引用被映射到位于应用部署描述符文件(web.xml)中指明的实际的标签库。

下面是本章我们涉及的 Struts 标签的标签库说明块:

```
<web-app>
  <!-- ... other web-app elements ... -->
  <taglib>
    <taglib-uri>/tags/struts-bean</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/tags/struts-html</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/tags/struts-logic</taglib-uri>
    <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
  </taglib>
</web-app>
```

某些开发人员在 uri 和 location 属性是用同样的字符串，但这种做法有悖于该特征的初衷并且如果 TLD 的位置发生改变容易造成混淆。

注 因为标签库通常以 JAR 形式进行分发，某些开发人员将 TLD 放置在 \WEB-INF\lib 下，和 JAR 文件在一起。这是将所有的东西都放在一起。

另一个通常做法是为 TLD 创建一个单独的文件夹。因为 TLD 的位置要在 web.xml 文件中标明，这种做法会工作得好一些。

导入新的标签库到页面中

哪一个标签在哪一个页面有效，是由开发人员来决定的。规范向开发人员提供了很好的灵活性。你可以导入一个标签库并且在当前页面中为它指定一个前缀。请注意，在导入标签库时，是从 web.xml 引用 `uri` —而不是 `location`:

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ tag 记 lib uri="/tags/struts-logic" prefix="logic" %>
```

然后每个标签都可以通过该页面指令定义的前缀来进行引用:

```
<bean:write name="userForm" property="firstname"/>
```

这个方法漂亮地解决了标签库名称冲突问题，并且使得在兼容的标签库间切换变得非常容易。

很多时候，开发人员都在整个应用中都使用相同的前缀，但这不是必需的。如果存在冲突，

你可以在不同的页面中使用不同的前缀。你也可以改变导入到页面中的标签库，而使用旧的前缀 (只要DTD是兼容的的)。

注 如果你发现页面中的 JSP 标签的动作很奇怪，或者也许就不工作，请检查看其标签库是否导入。因为 JSP 编译器和 web 浏览器都会忽略它们不懂的标签。所以如果你在页面中使用了标签，却没有导入其标签库，则什么都不会发生。

10.2.3. 标签不是什么

标签扩展是非常有用和有发展前途的——但，在写作本书时，这也还是一个发展中的技术。下面是一些应该记住的要点:

- 标签扩展并不被所有的 HTML 可视化编辑器支持
- 标签扩展不是脚本小程序（Scriplet）的简单替换
- 标签扩展不是 JavaServer Faces

标签扩展不被所有HTML可视化编辑器支持

在 Macromedia's UltraDev 4 [ASF,CTLX]中有一些有效的插件支持，但目前还不完全。UltraDev 4 仅支持 JSP 1.0，它最早引入定制标签。另一个 Macromedia 产品，HomeBase，也支持定制标签，但该产品设计来是作为手工编码的。当然，对 JSP 标签的 GUI 支持很可能增长起来，因为定制标签越来越广泛的使用。

标签不是小脚本程序的替代

几乎所有你想在 MVC 风格的 JSP 应用中所做的事情都可以用 Struts 标签库或者 JSTL 来完成。但也仅仅是*几乎*。开发人员仍然发现需要借助 scriptlet 来做一些事情。很多时候，持续使用 scriptlets 会克服标签中的设计缺陷，或者避免编写或者扩展一个新的标签。一个真正纯粹的对 scriptlet 的需要是很少见的。

JSTL 的脚本特征设计来就是为了克服对 scriptlet 的需要。但在写作本书时，JSTL 发布了最新版本，并且需要一个支持 Servlet 2.3 和 JSP 1.2 API 的容器支持。JSTL 表达式语言特征将能够比较容易地代替 scriptlet。

标签扩展不是JSF(JavaServer Face)

一种主动方式是定义一个标准的 JSP 标签和 Java 类集，可以用来简化构建 JavaServer 应用的 GUI，称为 JavaServer Faces (JSF)[Sun, JSF]。因为 JSF 明显地构建于标签扩展 API 之上，所以它有其自己的组件框架。JSF 组件将和一些 Struts 标签交迭，但是 Struts 小组保证会在适当的时候提供一个清楚的迁移路径。

注 在本书即将出版时，Struts 小组声明 struts-faces 标签库的开发的发布将符合 JavaServer Faces 1.0。 Struts-faces 标签库设计来使开发人员能从 Struts 标签迁移到 JavaServer Faces 页面。关于最新的细节，请参考本书站点 [Husted]。

关于其它标签扩展，我们推荐 JSP 标签库 [Shachor]，这是另一本 Manning 出版的书籍。

10.3. Struts标签库

首先，你的应用并没有被约束来只能使用 Struts 标签。标签扩展是一个标准的 Java 技术。其它标准库也可以使用，包括 Sun 的 JSP 标准标签库 [Sun, JSTL]，Jakarta 标签库 [ASF, taglibs]，以及 Improve 的 Struts-Layout 标签 [Improve]，当然还有其它。所有这些标签，其及其它好多标签，都可以在应用中和 Struts 标签库一起使用。如果你有些需求现有的标签不能满足，或者你想研究一下标签的处理流程，你也可以自己编写你自己的标签。 你自己的标签可以和其它资源混合或者匹配使用。

Struts 的分发包包括了几个关键的标签库: *bean*, *html*, *logic*, 和 *nesting*，*列于*表 10.2。 位于 html 库 (`org.apache.struts.taglibs.html`)的大部分标签要依赖于框架。而其它库中的大部分标签并不依赖于框架，也可以在其它应用中使用。

10.2 四类关键的 Struts 标签库

库	说明
bean 标签	在访问 JavaBeans 及其属性，以及定义一个新的 bean 时使用
html 标签	用来创建能够和 Struts 框架和其他相应的 HTML 标签交互的 HTML 输入表单
logic 标签	管理条件产生的输出和根据对象集产生的循环
nested 标签	提供对其他的 Struts 标签的增强嵌套能力

在本节中，我们将讨论 Struts 标签库所享有的公共特性。这里的内容并不是想要代替 Struts 开发人员指南和技术文档。关于每个标签的详细细节，我们希望你参考技术文档。本节只是用来为 10.4 节打个基础，在那里，我们将使得 Struts 标签运行起来。

10.3.1. Struts标签公共特征

Struts 标签库中的 JSP 标签提供了大量的公共特性来帮助标签的方便使用，包括自动区分范围，公共属性名称，扩展语法，运行时表达式以及公共错误属性。

自动区分范围

Java servlets 可以在大量称之为***上下文*** 的共享区域中存储对象。上下文类似于一个公共的公告板。应用中的 servlet 可以提交它们想提交的东西，而其它 servlet 则可以查看提交了什么东西。在搜索一个对象时，Struts 标签可以自动查找所有标准的上下文——页面，请求，会话，和应用——并且使用它找到的第一个实例。你也可以明确标明上下文 (或者范围)确保标签锁定正确的对象。

公共属性名称

了解 Struts 标签的一个重要事情是知道它们是设计来暴露传递给页面的 JavaBean 的。标签的公共属性——`id`, name, peroperty, 和 scope——可以与一些 helper bean 一起使用，并由控制器来传递。表 10.3 列出了这四个公共属性。

10.3 使用固定属性名称的 Struts 标签

属性	说明
id	命名定制标签创建的脚本变量
name	指出关键字值，在该关键字下可以找到一个存在的 bean 。如果给出了范围属性，则仅仅在范围上下文中查找。否则，根据标准的顺序在各种上下文中查找：(页面，请求,会话，应用).
property	指出 bean 中的某个属性，可以在其中检索值。如果没有标明。则使用对象本身的值。
sscope	表示出上下文范围(页面，请求,会话，应用)，在其中查找 Bean。如果没有标出则在标准的顺序下查找各种上下文范围。脚本变量(见 id)将在相同的上下文中被创建。

下面是使用了全部四个公共属性的 Struts 标签:

```
<logic:iterate
    scope="request"
    name="result"
    property="collection"
    id="row">
  <%-- markup --%>
</logic:iterate>
```

这个特殊的例子说明，在标准请求上下文中查找名称为 result 的 Bean。然后检索一个叫 collection 的属性(例如，getCollection)。通过在该集合中进行迭代，每一次将每个元素暴露为脚本变量 row，然后处理其中嵌套的标签，关闭 </logic:iterate> 标签。

其它 Struts 标签将以相似的方式使用这些属性，有助于平坦学习曲线。

扩展语法

Struts 标签支持 JSP 动作中经常使用的简单引用和嵌套引用。使用嵌套引用，可以在 Struts 标签中将一个属性表达为：

```
Property="foo.bar.baz"
```

这相当于告诉 Struts 进行下面的调用

```
getFoo().getBar().getBaz();
```

或者作为 setter

```
getFoo().getBar().setBaz(value);
```

从 Struts 1.1 开始，也可以包含索引引用:

```
property="foo[2]"
```

这相当于是调用

```
setFoo(2, value);
```

请注意，引用索引是零基的，是典型的 Java 表达式。

运行时表达式

虽然 Struts 标签的设计原意是为了避免使用 scriptlet，但所有的 Struts 标签属性都可以通过运行时表达式(scriptlet)提供，这种情况下还没有替代。当使用 scriptlet 来产生属性时，请确保使用完整的表达式:

不正确

```
<html:link href='<%= "/" + name %>/index.jsp>'>
```

正确

```
<html:link href='<%= "/" + name +"/index.jsp" %>'>
```

如上面的代码，表达式必须逐个地提供整个属性。关于混合使用 scriptlet 和 JSP 标签的例子，参见 Struts Bean 标签库开发人员指南 [ASF, Struts]。要注意的是，Struts 小组，以及其他许多开发人员，现在都将使用 scriptlet 作为最后的手段。

公共错误处理

JSP 规范 [Sun, JSP] 允许定义缺省的错误页面。如果在处理页面时异常发生，容器将定向控制到错误页面而不是抛出一个标准的 “空白页面”。如果 Struts 标签抛出一个异常，它将被传递到错误页面，并在请求属性下使用关键字 `org.apache.struts.action.EXCEPTION`。

这使得你的 JSP 错误页面有机会可以处理造成问题的实际的异常。

10.3.2. Bean标签

在定制标签和 JavaBean 之间具有很强的联系。定指标签的原本设计意图就是要在 JSP 和 JavaBean 之间提供一个接口。最后，Struts 提供了一个小巧但非常有用的标签库来操纵 JavaBean 和相关对象。Bean 标签总结在表 10.4 中。

表格 10.4 Struts bean 标签

标签名称	说明
cookie	根据特定的请求 cookie 定义一个脚本变量

define	根据特定的 Bean 的属性值定义一个脚本变量
header	根据特定的请求头定义一个脚本变量
include	从一个动态应用请求中装入一个响应，并将其作为一个 bean
message	输出一个国际化消息字符串
page	从页面上下文中某个项目暴露为一个 bean
parameter	基于一个特定的请求参数定义一个脚本变量
resource	装入 web 应用的资源，并将其作为一个 bean
size	定义一个包含 Collection 或者 Map 元素个数的 Bean
struts	将某个 Struts 内部配置对象暴露为 bean
write	输入特定的 bean 属性的值

这些标签可以用来做这些事情：

- 从 HTTP header，请求参数，cookie，或者一定范围的现有对象中创建一个脚本变量；
- 根据对其它请求(include)的响应，应用资源，或者 Struts 配置对象创建一个新的 bean ；
- 决定在 collection 或者 map 中元素的个数
- 自动从应用资源束中为当前用户输出本地化信息
- 在某些可用的 bean 写入给定的属性值

当然，仅有两个， `message` 和 `struts` 标签，以某种方式绑定到框架。其它 9 个都可以在其它应用中很好地工作。

如果一起使用，这些标签提供了大量功能，这些功能以前通常 JSP 设计人员仅通过 scriptlet 来完成。包括通常的 CGI 技巧，比如写 HTTP headers:

```
<bean:header id="browser" name="user-Agent"/>
<P>You are viewing this page with: <bean:write name="browser"/></P>
```

以及其他更多有用的事情，比如根据 cookie 创建一个新的 Bean:

```
<bean:cookie
        id="username"
        name="UserName"
        scope="session"
        value="New User" />
```

```
<P>Welcome <bean:write name="username" property="value"/!</P>
```

如果用户名称已经存储在 cookie 中，则它将不会被显示为一个新的用户。

Bean标签最佳实践

在开发人员从 JSP Model 1 迁移至 Model 2 应用时，Struts bean 标签库提供了许多高级的有用的特性。许多标签，象 `<bean:cookie>` 和 `<bean:header>`，提供了也可以被 Struts Action 处理的服务。许多 Struts 开发人员喜欢简单地使用 JSP 页面来进行邮件合并的任务。带有业务逻辑意味的东西，象管理 cookie，最好在 Struts Action 中处理，然后再传递给页面。

在实际应用中中，最常用的标签是<bean:write> 和<bean:message>。

Bean write 和 message 标签

`<bean:write>` 标签与下一节讨论的 Struts html 标签相比来说，它是个只读组件。它是一个灵活和有效的标签，使用反射来输出给定属性的值:

```
<bean:write
        name="shoppingCart"
        property="itemSum"/>
```

如果属性不是一个 Java 基本类型，对象的标准 `toString()` 方法将被调用。

`<bean:message>` 标签可以帮助本地化你的应用。Java 定义了一个标准的对象来存储用户的区域和语言，称之为场所 *locale*(`java.util.locale`)。 缺省情况下，Struts 控制器为每个用户创建一个 locale 对象，并将其存储在会话上下文中的已知关键字下面。

开发人员需要做的就是提供消息的关键字。相关的 Struts 组件，如 `<bean:message>` 标签，会查找该关键字对应的消息。 框架自动从用户场所的资源中提供相应的消息。

如果一个标签是这样:

```
<bean:message
        key="inquiry"/>
```

则 bean message 标签可能会针对法语 French/加拿大 Canada 的场所用户渲染成这样

```
"Comment allez-vous?"
```

而针对英语/美国的用户则可能是

```
"How are you?"
```

10.3.3. Html标签

HTML 格式 [W3C, HTML] 提供了一个浏览器必须支持的小巧但有用的控件集。这些包括按钮，复选框，单选按钮，菜单，文本字段，以及隐藏控件。所有这些都是设计来可以针对动态数据进行预组装的。如何将数据装入到控件取决于具体应用。

用脚本语言写成的动态应用，包括 JSP scriptlet，通常通过混合使用 HTML 和脚本来组装

HTML 标签。例如，为使用标准 avaBean 和 JSP scriptlet 组装一个 HTML 文本表单标签，大多数开发人员可能会这样使用:

```
<input  type="text"
        name="firstName"
        value="<%= formBean.getFirstName() %>"/>
```

注 顺便说一下，没有任何东西不允许你用脚本程序来写 HTML 标签。Struts 标签并没有特殊的优先。当表单提交给 Struts 控制器时，控制器看到的所有东西都是标准的 HTTP 请求。而不管该请求是如何被初始化的，控制器将忠实地履行它的职责：装配所有相关的 ActionForm bean，调用 ActionForm bean 的 validate 方法，并且将 ActionForm 传递给 Action 的 perform 方法或者将请求转发给输入页面。而关于请求后面的 HTML form 是如何被渲染的这并不重要。重要的是(至少对 Struts 开发的团队) JSP 开发人员有一个完整的一套标签，可以用来编写 HTML 控件，而勿须使用 scriptlets.

而下面是我们使用 Struts html 标签来组装同一个控件:

```
<html:text property="firstName"/>
```

注意这两种方式的不同:

- 虽然没有表明，scriptlet 版本需要在使用前在页面中将 formBean 声明为一个脚本变量。Struts 标签则不需要声明就能找到该 bean。
- 缺省时，Struts 标签将对剩下的表单使用同一个 bean，所以 bean 不需要对每一个控件都进行指定。

对应的HTML元素

Struts html 标签库提供了 20 多个标签来帮助组装 HTML 控件和相关元素。如表 10.5 所示，它们大部分对应于相关的标准 HTML 元素。

表格 10.5 Struts HTML 标签与 HTML 标记元素的对应

Struts Html 标签	对应的HTML 元素
base	<base>
link	<a>
button	<input type="button">
messages	无— 显示一组积累的错误消息[Struts 1.1]
option, options	<option>

checkbox, multibox	<input type="checkbox">
password	<input type="password">
errors	无—显示一组积累的错误消息
radio	<input type="radio">
file	<input type="file">
reset	<input type="reset">
form	<form>
rewrite	无— 输出一个编码的URL路径
hidden	<input type="hidden"> select <select>
html	<html>
submit	<input type="submit">
image	<input type="image">
text	<input type="text">
img	<img> textarea
textarea	<input type="textarea">

这种一对一的对应关系可以直接将现有的 HTML 或者 JSP 页面转换为 Struts 页面。例如，如果有这样的一个 HTML 元素

```
<input type="text" name="UserName">
```

被找到，就可以被下面的对应的 Struts html 标签代替:

```
<html:text property="UserName"/>
```

有一个自动转换器，可以读入现存的HTML 页面并自动转换为对应的Struts 版本 [Objectwave], [Ramaiah]。像这种工具可以为你的项目提供一个很好的开始。

公共属性

如所有的 Struts 标签，Struts html 标签共享一些公共的属性，如表 10.6 所示。

表格 10.6 Struts html 标签库使用的属性名称

属性	目的

name	ActionForm 的名称, 或者其他 JavaBean的名称,用来为该控件提供数据。如果没有指明，将使用Form标签包围的相应ActionForm bean 。
on*	每个html 标签都包括对应的JavaScript 事件处理器[Netscape]，包括onblur、onchange、onclick、ondblclick、onfocus、onkeydown、onkeypress、onkeyup、onmousedown、onmousemove、onmouseout, onmouseover、Onmouseup、onreset、以及onsubmit。都是小写格式，以兼容于XML。
Accesskey	可访问的键盘字符(某些流览器会忽略)。按下某个元素的访问键将聚焦到该元素之上。
Tabindex	指明当前元素在当前文档中的Tab顺序的位置。Tab顺序定义了在使用键盘导航时每个元素接受焦点的顺序。
Style	应用于该HTML元素的CSS 样式[W3C, CSS]
StyleClass	将应用于该HTML元素之上的CSS 样式表类[W3C, CSS]

10.3.4. Logic标签

如表 10.7 所示，Struts 提供 3 个风格的逻辑标签: 取值标签，控制流标签，以及反复标签。

表格 10.7 Struts 逻辑标签的 3 种类型

标签	目的
取值标签	测试值是否相等，小于，大于，空（空白或者null）,或者是否存在
控制流标签	转发或者重定向请求
重复标签	通过某些集合类型进行迭代

取值标签

Struts 逻辑标签通常用来提供同一个表现页面的替换版本。取决于一个对象的存在或者一个属性的值，一个给定的标签块被选择并展现给用户：

```
<logic:notPresent name="logonForm">
  <html:link forward="logon">Sign in here</html:link>
</logic:notPresent>
<logic:present name="logonForm">
  <html:link forward="logoff">Sign out</html:link>
</logic:present>
```

依赖于特定的环境，许多页面也许稍微有别，比如用户是否已经登入。已经登入的用户可能会看到一个按钮以使其可以登出系统。而登出的用户则可以看到一个对话框，可以让他们可以登入系统。当用户登入时，他们也可以访问管理型的控制。另外，用户可能可以访问不同的控制，这取决于他们的安全角色。通常，页面的大部分是相同的，不同的可能仅是一两行而已。

因为 MVC 架构允许我们在模型和控制器层处理业务逻辑，在试图中能应用某些逻辑构造仍然是有用的。实践中，为每种可能的环境创建不同的页面将导致维护负担。如果相似的页面被创建并且需要更新，每个不同的页面则都需要单独更新。这并不符合 MVC 的主要原则，创建健壮的系统，容易维护的系统。所以，从实践的角度出发，许多开发人员放置一点表现逻辑到页面中，只是为了减少要维护的页面数量。

注 第 11 章讨论的 Tiles 库，提供了这一问题的另一个解决方案。可以为每一个环境创建各种不同的页面定义，而不用创建大量冗余的标记文件。使用 Tiles 可以减少和消除应用的表现逻辑。

因为期望主要的逻辑都在 Action 里面来处理， Struts 逻辑标签基于一些基础阶段的处理。你可能注意到在前面的代码片断中缺乏一个类似于 `if...then...else` 结构。

一个标签，`present`，被用来测试看是否 `logonForm` 存在，而另一个标签 `notPresent`，则用来测试一个 ActionForm 是否不存在。这种相同的格式遵循所有的 Struts 取值标签。在逻辑标签库中没有象 `if...then...else` 结构的标签。这种结构在定制标签中非常难以公式化，并给 `Struts` 页面中处理的逻辑数量带来限制，开发小组认为带来的问题要多一些。

取值标签—公共属性

如上所诉， Struts 提供一个完整的取值标签：`empty` [Struts 1.1]，`notEmpty` [Struts 1.1]，`equal`，`notEqual`，`greaterEqual`，`lessEqual`，`greaterThan`，`lessThan`，`match`，`notMatch`，`present`，和 `notPresent`。所有这些标签都需要一个 `value` 属性，并且基于它来求值。当然，如果需要，`value` 属性可以由运行时表达式决定:

```
<bean:define id="value2" name="bean2" property="value"/>
<logic:equal
        value="<%=(String) value2 %>"
        name="bean1" property="value">
  <%-- markup for when bean1.value equals bean2.value --%>
  HIT!
</logic:equal>
```

如表 10.8 所示，取值标签可以和一个 cookie，HTTP header，请求参数，bean，Bean 的属性比较值。Match 和 `notMatch` 标签也使用一个可选的 `location` 属性。这可以用来指示字符串的 `start` 或者 `end` 被匹配。否则，匹配将针对整个字符串。

表格 10.8 比较标签的比较属性

属性	目的
cookie	值将要和一个cookie的属性比较

header	值将要和HTTP header比较的
parameter	值将和某个请求参数比较的值
name	值将要和该名称指明的某个对象进行比较
property	值将和命名对象的该属性进行比较

关于详细的每个Struts 标签接受的属性的细节，参见Struts 技术文档。

控制流标签

虽然，最好在控制器中提供控制流，但 Struts `redirect` 和 `forward` 标签也可以在 JSP 中处理这些任务。典型地，流程的转发和重定向会基于取值标签的输出。

`redirect` 或者 `forward` 标签的一个很好的应用是将控制从一个缺省的 welcome 页面发送到一个 Action。虚拟的 URI，象 Struts Action，不能被应用作为缺省的 welcome 页面。(容器将查找物理的页面)。另外，你可以提供一个 index.jsp 页面，由它将控制转发到 Struts 配置所控制的元素:

```
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<logic:forward name="welcome"/>
```

反复标签

Struts `<logic:iterate>` 标签十分灵活。它提供了很多 MVC 表现页面需要的循环功能。集合（collection）被传递为一个 bean，或者是一个 bena 的属性。标签则根据每一个元素进行反复，将它们暴露为一个脚本变量。`bean` 和 `html` 标签随后就可以用它们来输出每个元素：

```
<UL>
  <logic:iterate id="item" name="list">
    <LI><bean:write name="item"/></LI>
  </logic:iterate>
</UL>
```

我们在下一节还要讨论 iterate ，在那里我们将使用这些标签，并使他们一起工作。

Struts 标签库的完整 API 包含在 Struts 分发包中 [ASF, Struts]。在你开始使用标签时，我们推荐你先看看 Struts 标签库开发人员指南，其中包含了一些额外的实用信息和示例。Struts 分发包中的标签库练习应用也包含了标签的一些有趣的各种用法。所有这些都可以作为你的工作的起始点。

10.4. 使用Struts JSP标签

在本书的第一部分，我们强调了使用分层架构的重要性。

通过这种方法，所有的数据处理都通过一个 Action 类。表现层则专注于显示数据而不是如何获取它们。

在本节中，我们将展示如何使用 Struts 定制标签来显示动态数据。

我们开始之前需要注意最后一个问题。需要记住的是，在最后，所有的东西都是要转换成 *HTML 的。* 在页面到达浏览器之前，它已经不再是 JSP 格式或者 Struts 格式了。浏览器看到的仅仅是常规的 HTML，以常规的方式进行渲染。必然是这种方式，因为 JSP 和 Struts 设计来是和现有的各种浏览器一起工作的，不需要改编。有些技术，如 Java applet，可以插入浏览器并提供特殊的能力。但是 JSP 和 Struts 设计来与现有 web 浏览器工作。他们仅能做 HTML 可以做的事情。相反，你可以用 Struts 来做哪些可以用 JSP 和 HTML 来做的所有事情。它自己开启了一个可能的世界。

10.4.1. Struts 标签

实践中，Struts 各种标签库中的标签都经常一起使用。某些表单的属性如果只读，则使用 `<bean:write>` 而不是 `<html:text>` 标签。逻辑标签可以用来在运行时根据求得的值提供或者排除一个表单。`<html:link>` 标签通常和 `<logic:iterate>` 一起来渲染一个超链接列表。等等。

在本节，我们专注于如何一起使用 Struts 标签来提供需要的效果。从动态数据创建 web 页面是一个有趣的事情，并且可以用一整本书来描述。更多有趣的效果取决于客户端的 JavaScript 而不是服务器端的动态数据。因此，我们在此着眼于如何获取动态的，服务器端的数据到页面中，然后，其他资源，如 JavaScript，就可以使用它们了。

首先，在 “基础”一节，我们看到了组装控件和设定缺省值的基础信息。然后，在“技术” 一节，我们将检验 Struts 标签的一些主要特征，以提供通用问题的较广范围的解决方案。这些包括使用数组来捕获重复元素和使用 `rewrite` 来加工指向其他资产的路径。

10.4.2. 基础

本节的内容帮助你开始你自己的Struts应用。首先，我们涉及到基本的信息，比如声明一个表单，然后组装控件。然后我们将接触到新的Struts 开发人员想要了解和征服内容，包括如何完成以下工作：

- 选择单选按钮
- 过滤 (或者不过滤) HTML
- 清除密码
- 使用会话令牌
- 与`options` 标签一起使用collection
- 使用 `multibox` 来处理checkbox 数组
- 本地化按钮和标签

第13章将深入讨论本地化。在本节，我们仅仅涉及使用Struts 标签的本地化问题。

声明一个Form

Struts html 标签设计来从一个 JavaBean 中组装其对应的 HTML 表单元素。JavaBean 表现了全部的表单。JavaBean 的属性标识表单的元素。因为某些 JavaBean 将和标签一起工作，架构鼓励开发人员使用 ActionForm (见第 5 章)。

ActionForm 是特殊设计来从 HTML 表单中传递数据到应用的其他部分。

STRUTS TIP 学习使用 Struts JSP 标签的最好方法就是开始用对应的 Struts HTML 标签代替你的 JSP 页面中的 HTML 元素。

就象和 HTML 中对应的标签，Struts 以 `<html:form>` 标签开始一个表单，后面跟着需要的控件元素，最后以`</html:form>` 标签结束，如清单 10.2。

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<html:html>
  <HEAD><TITLE>Sign in, Please!</title></head>
  <BODY>
    <html:errors/>
    <html:form
          action="/logonSubmit"
          name="logonForm"
          type="app.LogonForm" scope="request">
      <table border="0" width="100%">
        <TR><TH>username </TH>
          <TD><html:text property="username"/></TD>
        </TR>
        <TR>
          <TH>Password: </TH>
          <TD><html:password property="password"/></TD>
        </TR>
        <TR>
          <TD><html:submit/></TD>
          <TD><html:reset/></TD>
        </TR>
      </table>
    </html:form>
  </BODY>
</html:html>
```

在清单 10.2 中，我们为表单使用的缺省的 JavaBean 标明了 name，type，和 scope 属性。实践中，这些属性通常被忽略，而`<html:form>` 标签则从 ActionMapping (第 7 章)获取他们。如果 JavaBean 不存在， `<html:form>` 标签将创建它， 并设置 bean 的缺省值，以为页面中的其它元素使用。

组装一个html控制元素

缺省时，清单 10.2 中的 text 和 password 标签将从 logonForm bean 中自动组装自己。如果 bean 在转发到页面前被放置到请求中，某些预先存在的值将被使用。这使得你可以事先在 Struts Action 类中设置 bean 的属性，然后转发到页面，在那里这些属性可以被显示出来。

所以

```
<html:text property="username"/>
```

就象等同于这样的scriptlet

```
<input type="text" value="<%= logonForm.getUserName() %>"">
```

这里， logonForm 已经被暴露为一个脚本变量。

表单中的每一个元素并不是一定要从同一个 bean 中组装。<html:form>标签使用的 bean 仅是一个简单的缺省情况。这段代码

```
<html:text property="username" name="accountBean"/>
```

将从一个名为 accountBean 的 from bean 中进行组装。因为没有指定范围，通常的顺序是页面，请求，会话，应用。将使用根据名称找到的第一个 bean。

NOTE 当一个表单被提交时，浏览器将使用 UserName=....的格式发送该元素。Struts 控制器将仅装配和该表单的 action 对应的 ActionForm。如果如果使用了另外的 bean，开发人员就需要自己捕获额外的参数。如果可能的话，最好让表单所使用的所有属性都由同一个 ActionForm bean 来表达。

.

选择单选按钮

每个单选按钮元素都需要一个能够与其它单选按钮区分开来的指定值。当创建一个静态的单选按钮数组时， 你需要指出它们哪一个，如果有的话，是选中的。这在单选按钮从动态数据中组装时并不需要这么做。控件会将其自身和form bean的属性进行比较，然后选择适当的项目。

如果一个 <html:radio> 控件像这样：

```
<html:radio property="expectedVia" value="UPS"/>UPS
<html:radio proeprty="expectedVia" value="FEDX"/>Federal Express
<html:radio property="expectedVia" value="AIRB"/>Airborne
```

并且 form bean 的 expectedVia 属性已经设置为 UPS，则 HTML 单选按钮元素将被渲染这样：

```
<input type="radio"
       name="expectedVia"
```

```
       value="UPS"
       checked="checked">UPS
<input type="radio"
        name="expectedVia"
        value="FEDX">Federal Express
<input type="radio"
        name="expectedVia"
        value="AIRB" >Airborne
```

过滤HTML

HTML 的特殊处理有一些特别的字符，包括标签的括号，&符号，以及其他东西。如果这些字符作为了文本字段的一部分，可能会造成 HTML 的崩溃，或者产生其他不好的影响。缺省情况下,这些字符会被 Struts `<bean:write>` 标签过滤,并被替换为它们的 HTML 等价物。这使你可以在数据文件中存储实际的字符，而且可以安全和自动在 web 页面种渲染它们。

有时，你也需要使用一个字段来捕获一个实际的 HTML (由信任或者由相关知识的人输入)并显示为页面的一部分。这时，你可以将过滤功能关闭:

```
<bean:write name="scriptForm" property="article" filter="false"/>
```

但这应该仅在你确实需要的时候才进行，而且要真正知道你正在做什么。渲染未经验证的输入到页面具有安全风险，并应尽可能的避免。

清除密码

缺省时， `<html:password>` 域将会自动组装自身，就像其他 html 标签一样。如果页面有个校验错误,password 属性将会被从 form bean 中读回,并放置在 password 标签中。实际的密码将会被浏览器隐藏，避免不经意的看见，但仍然可以在 HTML 源代码中看到。如果这种行为被视为一个安全风险，password 标签可以设置为不进行自动组装:

```
<html:password property="password" redisplay="false"/>
```

这意味着 password 域将总是以空白开始。如果表单在提交时校验失败，密码将不得不重新输入，即使原来的输入是正确的。在实际的生产应用系统中设置登录表单的 redisplay 为 false 是个好的做法。

使用会话令牌

重复提交对 web 应用通常是个问题。Struts 支持一个同步令牌策略来避免重复提交。

使用同步令牌对 Struts `<html:form>` 标签来说是自动的。如果它看到使用了同步令牌，它们就会自动加入一个相应的隐藏字段。在 Action 一端，你可以在 Action 的 perform 或者 execute 方法的某处通过调用下面的语句来激活会话令牌:

```
saveToken(request);
```

当调用返回，为使 Action 检查是否令牌仍然完好，可调用

```
boolean valid = isTokenValid(request);
```

如果这个方法返回false，则会被分支到一个错误页面。如果返回true，则调用

```
resetToken(request);
```

虽然参与事务通常和表单一起发生，也可用于超链接和其它资产。因此，Struts <html:link>和 <html:rewrite> 标签提供了一个可选的事务支持。为了在连接中包含适当的令牌，可设置 transaction 属性为 true。一个这样的标签:

```
<html:link
        forward="addItem"
        paramname="row"
        paramname="itemId"
        paramId="id"
        transaction="true"/>
```

将产生下面这样的链接:

```
<a href="/gavel/do/addItem?
id=3017&amp;org.apache.struts.taglib.html.TOKEN=72da1d3fdede66c">
```

使用选项集合

options 标签使用来从一个或者多个集合中输出菜单选项。你可以针对labels 选项和 values 选项使用一个collection ，也可以分别使用单独的collection 。作为选择，你也可以使用一个 collection，其集合是对label 和 value属性的accessor。借助于 <bean:define>的一点帮助，你也可以将 collection作为 form bean的一部分传递。

options 标签的语法比较复杂。所以我们最好提供一些常见情况的使用示例：

- 一个集合，存储为 ActionForm 的一个属性
- 一个集合，存储为一个单独的 bean
- 一个集合，具有label名称 和 label属性 访问器（accessor），存储为一个单独的bean
- 一个集合，具有 Code名称 和 Label属性 accessors

一个集合，存储为ActionForm的一个属性.

同一个的字符串将同时用作 value 和 property。collection 将通过一个具有方法签名为 Collection getImageOptions()的方法返回：

```
<TR>
```

```
  <TD>Item Image Source:</TD>
  <TD>
    <html:select property="hasImage">
      <html:options property="imageOptions" />
    </html:select>
  </TD>
</TR>
```

一个集合，存储为一个单独的bean

这里，集合被直接放入请求，会话，或者应用范围的上下文中，名称为imageOptions:

```
<TR>
  <TD>Item Image Source:</TD>
  <TD>
    <html:select property="hasImage">
      <html:options collection="imageOptions"/>
    </html:select>
  </TD>
</TR>
```

一个集合，具有label名称和label属性访问器（accessor），存储为一个单独的bean.

这样将针对每一个集合中的元素调用 getValue() 和 getLabel() 属性：

```
<TR>
  <TD>Item Image Source:</TD>
  <TD>
    <html:select property="hasImage">
       <html:options
               collection="imageOptions"
               property="value"
               labelproperty="label"/>
    </html:select>
  </TD>
</TR>
```

一个集合，具有 Code名称 和 Label属性 accessor

这将针对集合中的每个元素调用getValue() 和 getLabel() 方法。这里我们需要使用<bean:define>来暴露集合以使它可以像前面的例子中一样被使用：

```
<TR>
  <TD>Item Image Source:</TD>
  <TD>
    <html:select property="hasImage">
      <bean:define
              id="imageOptions"
              name="itemForm"
              proerty="imageOptions"
              type="java.util.Collection"/>
      <html:options
              collection="imageOptions"
              property="value"
              labelproperty="label"/>
    </html:select>
  </TD>
</TR>
```

从 Struts 1.1 开始，提供了一个 LabelValueBean 类 (`org.apache.struts.util.LabelValueBean`)，但也可以使用一些兼容的类。关于本地化 LabelValue 类，参见第 13 章。

使用复选框数组

许多应用需要使用大量的复选框来跟踪选项和选定的项目。为了帮助完成这种事情， Struts 提供了一个 `multibox` 控件。它十分便捷，但在第一次接触时可能难以理解。`multibox` 使HTML 可以方便处理复选框。如果复选框没有被选择，浏览器将不提交控件的任何值。

如果复选框被选定，控件的名称和其值将被提交。这种行为也是ActionForm为什么要有个 `reset` 方法的原因。因为浏览器将不会通知一个复选框没有被选择，唯一的方法是重设所有的复选框，然后重新选择当前表现在请求中的那个选项。

STRUTS TIP 使用<html:multibox> 标签来将一系列复选框管理为一个数组

`multibox` 控件设计来是使用字符串数组。数组中的每个元素表示一个复选框。为选择一个复选框，在数组中添加一个标识复选框名字的字符串。要不选择一个复选框，将其从复选框中移除。 (看起来类似?)

在传递值时， `multibox` 控件会扫描数组中的元素看是否有匹配的项目。如果有，该复选框选中。如果没有，该复选框就保持未选择状态。

如果用户选择了复选框并且提交了表单，则该复选框的值就会包含在请求中。控制器就会将其加入到已选择（`checked`）的数组之中。如果复选框没有被选择，什么都不会提交，那么也不会有什么加入到数组中。如果ActionForm bean 被保持在会话上下文中，在请求之间，`reset` 方法需要将数组长度为减小为0（但非null）。

在这个例子中：

```
<logic:iterate id="item" property="items">
  <html:multibox propety="selectedItems">
    <bean:write name="item"/>
  </html:multibox>
  <bean:write name="item"/>
</logic:iterate>
```

每个复选框的标注（label）位于items 属性之中。选定项目的列表位于一个名为selectedItems的数组之中。没有选定的项目不会出现在selectedItems 数组中。multibox 检查selectedItems数组获得当前的项目。如果出现，它就写一个选择的复选框。如果没出现，就写一个没选定的复选框。

如果一个ActionForm 设置成这样

```
private String[] selectedItems = {};
private String[] items = {"UPS","FedEx","Airborne"};
public String[] getSelectedItems() {
  return this.selectedItems;
}
public void setSelectedItems(String[] selectedItems) {
  this.selectedItems = selectedItems;
}
```

例子中的标签将产生 3 个复选框，分别标签为 UPS，FedEx，和 Airborne:

```
<input type="checkbox" name="selectedItems" value="UPS">UPS
<input type="checkbox" name="selectedItems" value="FedEx">FedEx
<input type="checkbox" name="selectedItems"
value="Airborne">Airborne
```

开始， selectedItems 数组是空的。如果 UPS 被选择并提交，则等同于：

```
private String[] selectedItems = {"UPS"};
```

如果 UPS 和 Airborne 都被选择，等同于

```
private String[] selectedItems = {"UPS","Airborne"};
```

并且，当复选框在渲染时，适当的元素将自动被 multibox 标签检测:

```
<input
        type="checkbox"
```

```
         name="selectedItems"
        value="UPS"
        checked="checked">UPS
<input type="checkbox"
        name="selectedItems"
        value="FedEx">FedEx
<input type="checkbox"
        name="selectedItems"
        value="Airborne"
        checked="checked">Airborne
```

为提供不同的标注和值，标准的 LabelValueBean 类 (`org.apache.struts.util.LabelValueBean`) (从 Struts 1.1)可以用于 `multibox` 控件:

```
<logic:iterate id="item" property="items">
  <html:multibox property="selectedItems">
    <bean:write name="item" property="value"/>
  </html:multibox>
  <bean:write name="item" property="label"/>
</logic:iterate>
```

本地化标签

你可以使用 `<bean:message>` 来为Html标签创建本地化标签，就象

```
<TH><bean:message key="username"/></TH>
<TD><html:text property="username"/></TD>
```

这里username是应用消息资源的关键字。

消息资源可以针对每个场所提供。针对当前场所用户的消息会自动渲染。关于本地化的更多信息，参见第13章。

在按钮标签的情况下， `<bean:message>` 可以作为按钮的内容来给出:

```
<html:submit><bean:message key="submit"/></html:submit>
```

因为按钮的标注也有其值，本地化消息就使浏览器将提交的内容。

本地化选项

The `<html:options>` 标签允许你对选项的标注提供一个集合，而对选项的值提供另一个集合。这使得你可以针对一个给定的用户场所传递一个标注（label）集合。这种情况下标签

并没有改变，因为标签简化了输出给定的东西。实际的集合最好在Action 类中产生。参见第13章，可获取关于产生本地化内容的信息。

如果你在页面中硬编码了选项，你也可以使用<bean:message>来输出本地化消息:

```
<html:option value="status">
    <bean:message key="status"/>
</html:option>
```

本地化集合

关于更多准备本地化集合，包括基于LabelValueBean类的集合，可参见第13章。当一个本地化集合被传递到一个页面，对标记并不需要改变，因为页面简化了对集合中所提供的东西的渲染。

10.4.3. 技术

既然我们已经通过了基础教程，现在我们来看看一些高级的技术，Struts 开发人员会发现它们非常有用，包括:

- 使用 ImageButtonBean 来表示ImageButton
- 使用 bean 标签创建定制控件
- 使用数组来捕获多重参数
- 使用 <bean:size> 测试集合大小
- 通过集合进行迭代
- 如果需要暴露迭代索引
- 使用嵌套的 <present> 和 <notEmpty> 标签来测试bean 属性
- 针对稳定的选项列表使用应用范围（ scope）对象
- 使用rewrite来渲染样式表，JavaScript或者其他资产的URL
- 使用 Struts JSP 标签来渲染JavaScript
- 使用 <bean:write> 来从bean中渲染JavaScript
- 重命名提交按钮来避免JavaScript冲突
- 使用无form按钮
- 使用Action 作为输入属性来重新创建一个依赖对象
- 使用动态 form action
- 使用可选的消息标签

使用ImageButtonBean来表示图像按钮

无尽的烦恼之源是HTML Input image 元素。HTML规范说明浏览器应该像处理图像Map一样处理这种控件。和其它按钮不同，它并不提交一个标识按钮标注的字符串；它提交的是按钮的 x 和 y 坐标。如果你看到 HTTP 提交了一个图像按钮，你会看到这样的东西：

```
myImageButton.x=200
myImageButton.y=300
```

对其它大多数控件而言，Struts 开发人员都可以创建一个简单的 String 属性来表示这个元素。这对图像按钮来说没有效果，因为它提交了两个“点” 属性，而不是一个通常的“名值对”。

可喜的是，Struts 允许一个ActionForm 包含，或者嵌套，其它的JavaBean，并且使用图像元素的相同语法自动组装bean。(好一个粉饰太平!)

为了在你的ActionForm中表示一个图像输入元素，展示你的意图，可使用一个 ImageButtonBean 来捕获 x 和 y 参数，就象清单10.3中所述：

```
public final class ImageButtonBean extends Object {
  private String x = null;
  private String y = null;
  public String getX() {
    return (this.x);
  }

  public void setX(String x) {
    this.x = x;
  }

  public String getY() {
    return (this.y);
  }

  public void setY(String y) {
    this.y = y;
  }

  public boolean isSelected() {
    return ((x!=null) || (y!=null));
  }
} // End ImageButtonBean
```

注意我们在这个bean中包含了一个helper 方法，isSelected。这个方法在 x 或者 y 属性不为空的情况下返回 true。 如果它们都为空，那么 isSelected 返回 false。

下面是如何在一个 ActionForm 中声明两个 ImageButtonBeans :

```
// ..
private ImageButtonBean logonButton = new ImageButtonBean();

public void setLogonButton(ImageButtonBean button) {
  this.logonButton = button;
}
public ImageButtonBean getLogonButton() {
  return this.logonButton;
}
private ImageButtonBean cancelButton = new ImageButtonBean();
public void setCancelButton(ImageButtonBean button) {
  this.cancelButton = button;
}
public ImageButtonBean getCancelButton() {
  return this.cancelButton;
}
// ...
```

清单 10.2 一个 ImageButtonBean 类

下一个问题可能是“好，那么到底点击了哪一个按钮呢?” 所以，让我们在 ActionForm 中定义另一个 helper 方法来告诉我们:

```
public String getSelected() {
  if (getLogonButton().isSelected()) {
    return Constants.LOGON;
  }
  if (getCancelButton().isSelected()) {
    return Constants.CANCEL;
  }
  return null; // nobody home
}
```

在 Action中，决定哪个按钮被点击就成了一个简单的事情，只需要问表单哪个被选择了:

```
String selected = ((MyForm) form).getSelected();
If (Constants.CANCEL.equals(selected)) ...
```

当然，因为`getSelected` 可能在 Action内被调用，方法并不需要返回一个字符串。它可以

是int，一个表示你API函数的定制类型，甚至是DispatchAction (org.apache.struts.actions.DispatchAction)使用的另一个方法的名称。

使用bean 标签创建定制控件

尽管 html 标签足够的好，它们也不可能涵盖所有的应用环境。当你需要以某种方式写一个现存标签不支持的表单元素时，你可以用 <bean:write>来进行修补。下面是某些常见的使用情况:

添加 wrap="soft" 到 textarea 控件:

```
<textarea name="description" rows="5" cols="60" wrap="soft">
<bean:write name="scriptForm" property="description"/></textarea>
```

重命名一个属性:

```
<input type='hidden' name='prospect'
value='<bean:write name="donorForm" property="donor"/>'>
```

从请求中设置一个控件到一个参数:

```
<bean:parameter id="item" name="item"/>
<input type='text' name='item' value='<bean:write name="item"/>'>
```

设置一个控件到cookie的值:

```
<bean:cookie id="username" name="username"/>
<input type='text' name="username" value='<bean:write
name="username"
property="value"/>'>
```

当然，这些情况都是最普遍的例子。还有其他使用情况遵循相同的格式。

使用数组来捕获多重参数

HTML 和 HTTP 允许具有相同名称的参数被添加到同一个请求中。你可以通过使你的ActionForm属性为字符串数组来捕获多重参数。:

```
private String items = {""};
public String[] getItems() {
return this.item;
}
public void setItem(String item[]) {
this.item = item;
}
```

请注意到你必须编码你的数组，以使它们不为空null。否则，在遍历所有项目进行迭代时会

抛出一个异常。

在表单中，你可以使用iterate 标签输出数组:

```
<logic:iterate name="logonForm" property="items" id="item">
<TR>
<TD>Item:</TD>
<TD>
<input type='text' name="item"
value='<bean:write name="item"/>'>
</TD>
</TR>
</logic:iterate>
```

STRUTS TIP

ActionForm 中的属性可以包括集合和数组和简单属性。如果你以相同的名称提交参数，它们会被捕获在一个数组中。

使用 <bean:size>测试集合大小

一些标准的Java 集合，比如ArrayList，在其属性中并不使用JavaBean 约定。为了解决这个问题， <bean:size> 标签返回给定集合的大小。使用这个标签的一个好的用法是如国际和为空则输出一个特殊信息，而不是进行0次的遍历:

```
<bean:size id="listSize" name="list"/>
<logic:equal name="listSize" value="0">
<P>No records were selected.</P>
</logic:equal>
<logic:notEqual name="listSize" value="0">
<logic:iterate id="row" name="list" >
<%-- markup for each row --%>
</logic:iterate>
</logic:notEqual>
```

然而，如果一个应用经常返回一个搜索的结果集，那么定义一个方便的类来包装集合和相关属性是值得的。

参见Scaffold 包中的ResultList 类(org.apache.commons.scaffold.util.ResultList)。

迭代部分集合

<logic:iterate> 标签可以可以针对集合中应该暴露的部分设定偏移和计数次数。例如，

使用这个方式从第5个元素开始，先是后面的5元素：

```
<logic:iterate id="element" name=”list" offset="5" length="5">
```

如果需要暴露迭代索引

通过访问每个迭代器的索引可以产生独一无二的表单名称或其它元素：

```
<OL>
<logic:iterate id="element" name="list" indexId="index">
<LI><EM><bean:write name="element"/></EM>
[<bean:write name="index"/>]</LI>
</logic:iterate>
</OL>
```

index 其实是一个脚本变量，所以上面的代码也可以写成是：

```
<OL>
<logic:iterate id="element" name="list" indexId="index">
<LI><EM><bean:write name="element"/></EM>
[<% index %>"/>]</LI>
</logic:iterate>
</OL>
```

使用嵌套的 <present> 和 <notEmpty>标签来测试 bean 属性

如果你需要测试一个Bean的属性是否存在或者为空，并且bean 自身可能并不存在，你可以进行嵌套求值：

```
<logic:present name="bean">
   <logic:notEmpty name="bean" property="value">
      <bean:write name="bean" property="value"/>
   </logic:notEmpty>
</logic:present>
```

NOTE

<logic:empty> 标签是 Struts 1.1 中加入的。

Struts 1.0 中，<notPresent>可以用来测试 null 值，但在 Struts 1.1，<empty> 可以用来测试 null 值和空字符串。.

对稳定的选项列表使用应用范围对象

许多应用需要使用静态的，半静态的选项列表，用在不同的页面中。通常这些列表都是来自于数据库表或其他资源。这使得他们是很好集合选项。如果这些集合对每个用户都是一样的，它们也是应用范围对象的很好的候选者。这里是一个例子:

```
<TR>
<TD>Item Image Source:</TD>
<TD><html:select property="hasImage">
<html:options collection="imageOptions" scope="application"
property="value" labelProperty="label"/>
</html:select>
</TD>
</TR>
```

如果列表需要过滤，或者需要针对用户进行本地化，可以在会话中放入一个代理对象，该对象使用应用对象作为数据源。

使用 rewrite 来渲染样式表，JavaScript，和其他资产的 URL

Struts `<html:rewrite>` 标签可以转换上下文相关的URI为用来访问样式表、JS、图像或者其它HTML资产的基本的URI。例如：

```
<LINK rel="stylesheet" type="text/css"
href="<html:rewrite page='/assets/styles/base.css'/>">
```

如果你想隐藏样式表的存放位置，也可以使用ActionForward:

```
<LINK rel="stylesheet" type="text/css"
href="<html:rewrite forward='baseStyleSheet'/>">
```

同样，对JavaScript的引用，以及由JavaScript处理的URI，也可以以这种方式进行渲染：

```
<SCRIPT language='javascript'>
  src='<html:rewrite page="/assets/scripts/remote.js"/>'>
</SCRIPT>
<SCRIPT>
  <!--
  function doPreview (aRecord) {
    aBase = '<html:rewrite forward="preview"/>';
    doOpenRemote(aBase + '?record=' + aRecord);
  }
  // --
```

```
</SCRIPT>
```

在下面是例子中，我们首先从其他页面中包含一些JavaScript(包括 doOpenRemote 脚本)。在调用脚本之前，我们从ActionForward中查找所调用的JavaScript 函数的基本URI。最后，在我们调用JavaScript时，我们添加了基本数量到函数中。这种类型的函数通常从超链接中调用，比如：

```
<a href='javascript:doScript(10011)'>10011</a>
```

是使用这样的JSP代码产生的：

```
<a href='javascript:doPreview(<bean:write name="row"
property="script"/>)'>
<bean:write name="row" property="script"/>
</a>
```

STRUTS TIP 经常使用 <html:rewrite>来引用 HTML 资产.

因为我们是调用JavaScript 函数，所以不必费心用<html:link>标签来提供URL编码。超链接会在客户端进行处理，所以维护会话状态不是个问题。由rewrite 标签产生的URI，随后被JavaScript使用，也可以进行 URL编码，以便在cookies无效的时候对会话进行维护。

使用 Struts JSP 标签渲染JavaScript

Struts 框架可以保证我们在不使用 JavaScript的情况下进行数据校验。但这并不意味着在我们的Struts应用中不能使用JavaScript。很多web 开发人员依赖于JavaScript来提供核心的表现层特征，对一些Struts 开发人员也不例外。

大多数JavaScript可以像在其它页面中一样用在Struts JSP 中。因为最终，所有的东西都是要转换成HTML。因为要转换成HTML，你可以混合使用JSP 标签和JavaScript引用。JSP 代码被首先渲染，到浏览器看见它们时，许多动态引用已经被解决，看到的仅是静态的引用了。

我们来看看一个打开一个远程窗口，并查看一个数据库中的记录的脚本。

首先，这是一个基本远程窗口脚本，就象你正在使用一个静态页面：

```
// Open window
function openWin(newURL, newName, newFeatures, orgName) {
  var newWin = open(newURL, newName, newFeatures);
  if (newWin.opener == null)
    newWin.opener = window;
  newWin.opener.name = orgName;
  return newWin;
}
```

```
// Open centered remote
function doOpenRemote(aURL, newName, aHEIGHT, aWIDTH, aFeatures,
orgName){
  if (aHEIGHT == "*"){ aHEIGHT = (screen.availHeight - 80) };
  if (aWIDTH == "*"){ aWIDTH = (screen.availWidth - 30) };
  var newFeatures = "height=" + aHEIGHT + ",innerHeight=" + aHEIGHT;
  newFeatures += ",width=" + aWIDTH + ",innerWidth=" + aWIDTH;
  if (window.screen){
    var ah = (screen.availHeight - 30);
    var aw = (sreen.availWidth - 10);
    var xc = (( aw - aWIDTH ) / 2);
    var yc = (( ah - aHEIGHT ) / 2);
    newFeatures += ",left=" + xc + ",screenX=" + xc;
    newFeatures += ",top=" + yc + ",screenY=" + yc;
    newFeatures += "," + aFeatures;
  }
  var newWin = openWin(aURL, newName, newFeatures, orgName);
  newWin.focus();
  return newWin;
}
```

我们将这个脚本看作是假设的程序，并不去逐步看它是如何工作的。示例脚本也不是练习的目的。我们来看看它是如何从JSP里面进行调用的。

我们想用这个脚本来在不同的时间打开不同的数据库记录。

为了达到这个目的，我们需要给它输入不同的URI 告诉它这时要打开那一个数据库记录。在许多应用中，URI 可能看起来像这样:

```
/do/item/View?item=117
```

`/do/item/View` 是相对静态的。它只需要被重写以维护会话。我们可以将它存储为一个ActionForward 并使用 `<html:rewrite>` 标签来在运行时渲染它，如果必要，保证它是经过URL编码的格式。例如，在Struts 配置 文件中，我们放入

```
<forward
  name="item"
  path="/do/item/View"/>
```

然后在JSP中使用这个标签:

```
<html:rewrite forward="item"/>
```

不确定的部分是?item=117。这是URI中真正动态的部分，而这一部分也是我们的确需要传递给脚本的部分。正因为这样，我们将它作为一个参数传递给我们的JavaScript函数。下面是结果:

```
<script>
<!--
function doItem(aItem) {
aBase = '<html:rewrite forward="item"/>';
doOpenRemote(aBase + '?item=' +
aItem,'preview','*','600','scrollbars','form');
}
// -->
</script>
```

请注意我们仅需要传递项目代码 (比如, 117)到该函数。然后函数找出URI的基本部分，并将它和查询参数以及我们的传递参数连接起来。

现在剩下的就是传递参数给脚本函数。 项目编号可以在JavaBean中传递到页面，所以我们可以使用<bean:write> 来做这个事情:

```
<a href='javascript:doItem(<bean:write name="itemForm"
property="item"/>)'>Item Number</a>
```

在运行时，它被渲染成

```
<a href='javascript:doItem(117)'>Item Number</a>
```

因为JavaScript 重写URI，我们对URI的各个部分都不必担心，而且可以使用常规的超链接标签。

使用 <bean:write>从Bean中渲染JavaScript

另一个方法是将整个脚本写入到页面中：

```
<SCRIPT>
<!--
  <bean:write name="fancyForm" property="javaScript" filter="off"/>
// -->
</SCRIPT>
```

这样将输出fancyForm.getJavaScript()方法返回的字符串。这使得你可以用各种必要的方法创建JavaScript。Struts Validator 使用这种方法来创建一个复杂的一系列脚本，可以从一个单一的JSP标签渲染到一个页面中。

当然，同样的技术也可以用在其他HTML文本资产中，包括层叠性样式表(CSS)。

重命名提交按钮避免JavaScript冲突

某些JavaScript可能试图调用一个表单的提交操作。缺省情况下，<html:submit>按钮也命名为(令人惊讶) 提交（Submit）。为了避免这种冲突，可以给Submit按钮另外的名称：

```
<html:submit property="submitButton"/>
```

使用无表单按钮

我们经常使用一个按钮来代表超链接或者JavaScript的动作。可是问题是，<html:form> 标签期望每个表单都有其自己对应的ActionForm bean。解决方法是给标签它想要的东西，并定义一个没有属性的简单的表单：

```
public class BlankForm extends ValidatorForm {
// blank form
};
```

在需要‘ 无形 ’表单的时候，随后这个表单可以在Struts 配置中引用。例如，提供一个JavaScript回退按钮：

Struts 配置文件:

```
<action
  path="/Back"
  type="org.apache.struts.ForwardAction"
  name="blankForm"
  scope="request"
  validate="false"
  parameter="/do/Menu"/>
```

JSP 页面:

```
<html:form action="/Back">
  <html:button property="page"
          onclick="history.go(-1)">DONE</html:button>
</html:form>
```

使用Action作为输入属性来重新创建一个依赖对象

如果页面从放在请求中的对象中显示选项或者其他控件，如果在校验失败时这些对象必须被

重建。所以不能像这样来设置input 属性

```
<action
  path="/item/RecvStore"
  // ...
  validate="true"
  input="/pages/item/RecvForm.jsp"/>
```

应该这样使用

```
<action path="/item/RecvStore"
  // ...
  validate="true"
  input="/do/item/RecvForm"/>
```

这里，RecvForm action 将重建页面希望在请求中找到的对象。

如果依赖对象是ActionForm 的属性，你也可以在From中将他们保存为隐藏属性。那么依赖对象就可以从请求中和用户提供的值一起被重新构造。

使用动态Form Action

表单收集属性并提交到Action 对象。实践中，不同的Action希望相同的属性集但对它们进行不同的操作。经典的例子是插入和更新纪录。对数据库而言，这是不同的操作。对一个应用而言，他们看起来是同一个表单。

问题是， `<html:form>` 标签假定一个特定的表单使用`action` 属性的相同的值。框架并不提供一个自动化的方式来传递`action` 属性给一个表单。

当想要在不同的ActionMapping中使用同一个表单时，开发人员的典型做法是将`action` 属性硬编码到标签中:

```
<html:form action="/saveRecord'>
<%-- ... -->%
</html:form>
```

然后使用一些其他技巧来改变提交的目的地址。

第6章展示了用ActionForward处理这些的方式。第8章，使用标准的Dispatch Action和Scaffold FindForward Action提供其他解决方案。第11章涉及的 Tiles 框架也通过将表单内容放入一个单独的文件中提供一些帮助。另一个工作方式是使用运行时表达式。

象所有的Struts标签属性一样， `action` 属性的值可以在运行时用表达式提供。你可以使你的Action 对象通过请求传递表单的`action` 属性的值。当页面被渲染时，页面可以将它放入到`<html:form>` 标签之中：

```
 <bean:define name="dispatch" id="dispatch" type="java.lang.String">
<html:form action ="<%=dispatch%>">
```

```
<%-- ... -->%
</html:form>
```

如果dispatch String 已经通过请求被传递，就象这里展示的一样，你必须确保请求首先通过页面的Action传递。否则，dispatch String 就不会在请求之中，并且 <html:form> 标签 会抛出一个异常。

NOTE 相同的原理适用于一些依赖属性。参见本节前面的“ 使用 Action 作为输入属性以重新创建一个依赖对象”。

避免路由问题的方法是在ActionForm bean设置一个dispatch 属性:

```
<bean:define name="recordForm" property="dispatch"
id="dispatch" type="java.lang.String">
<html:form action ="<%=dispatch %>">
<html:hidden property="dispatch"/>
```

通过包括一个dispatch 属性为一个隐藏字段，你可以确保校验在步骤进行到其他其他 ActionForm 属性时，它也可以被自动组装。如果校验失败， dispatch 属性被回写到 <html:form>标签中，就象其他ActionForm 值。

如果你使用这个技术，一个好办法是将dispatch 属性放入一个基础ActionForm 对象，这样其它ActionForm就可以扩展它。Scaffold 包中的BaseForm 类 (参见第5章)(org.apache.scaffold.http.BaseForm) 提供了一个 dispatch 属性，还有其他属性。

使用替换消息标签

许多Struts 应用都在页面定部放入一个简单的<html:error/> 标签来获取错误。如果一些错误消息出现，就由这个标签来将它们全部输出。为了帮助漂亮的输出错误消息，<html:error>标签检测应用资源的 errors.header 和 errors.footer 消息。如果找到，标签在消息块的前面或者后面输出它们。一个通用的设置是:

```
errors.header=<UL>
errors.footer=</UL>
```

Struts 1.0 开发人员可以包括 <LI> 和 </LI> 标签以及文本到使用这种方式的每个消息。在Struts 1.1中,这种情况改善了,因为加入了 errors.prefix 和errors.suffix 消息。仅仅需要header 和 footer在消息块之前和之后输出，而 prefix 和 suffix 在每个单独的消息前输出。所以，输出一个简单的消息列表，可以这样包括

```
errors.header=<UL>
errors.footer</UL>
errors.prefix=<LI>
```

```
errors.suffix=</LI>
```

到你的应用资源中，然后 `<html:error>` 标签会做余下的事情。

然而，坚持技术纯正的人可能会抱怨， HTML不能放入消息资源文件之中。他们也许是对的。即使使用Struts 1.1 的prefix 和suffix 特征，你也仍然可能需要在不同的页面使用不同的标签。

对 Struts 1.0 应用，Struts 校验扩展(见第12章)提供了一个有用的标准`<html:error/>` 标签的替代方案。 你可以使用这些标签，不管是是否需要使用该包中的其他校验特征。不是提供一个综合性标签， Validator 方法是使用一个迭代起来暴露每一个错误消息，然后将它们发送到页面并提供其他必要的格式化。例如:

```
<validator:errorsExist>
<UL>
<validator:errors id="error">
<LI><bean:write name="error"/></LI>
</validator:errors>
</UL>
</validator:errorsExist>
```

在 Struts 1.1中，这些标签被收到核心标签库中。这是个使用Struts 1.1相同的例子：

```
<logic:messagesPresent>
<UL>
<html:messages id="error">
<LI><bean:write name="error"/></LI>
</html:messages>
</UL>
</logic:messagesPresent>
```

如果你想批量输出你的错误消息，这敢情好。但是许多错误是和数据输入校验相关并涉及到特定的字段。许多页面设计希望消息和字段相关，并且在字段的旁边进行输出。

没问题！

当错误消息排在队列中时，你可以给它们都标明一个 “属性”。如果你没有为它们指定一个属性 (使用我们描述的标签)，则所有的消息都会输出。如果你指定了一个属性，则仅有针对该属性排队的消息被输出。排队一个错误消息的缺省代码为：

```
errors.add(ActionErrors.GLOBAL_ERROR,
new ActionError("error.username.required"));
```

如果表明这个消息是针对色username属性的，我们可以使用这个代码:

```
errors.add("username", new ActionError("error.username.required"));
```

如果表明了一个属性，我们可以像这样使用<html:errors/> 标签 (或其他替代标签):

```
<P>Username: <html:text property="username"/></P>
<P>Password: <html:password property="password"/></P>
```

Username会错误紧跟着Username字段输出，而password 错误紧跟password 字段输出。

但是，究竟你是想输出特定错误还是通用错误？

再说一下，还是没问题！

你也可以标明一个通用属性，就象在 Java 代码中一样。首先，在 JSP的顶部，导入Action 和 ActionErrors包，你就可以引用适当的常数：

```
<%@ page import="org.apache.struts.action.Action" %>
<%@ page import="org.apache.struts.action.ActionErrors" %>
```

然后，在标签中，使用运行时表达式来标识常数:

```
<logic:present name="<%=Action.ERROR_KEY%>">
<P><html:errors property="<%=ActionErrors.GLOBAL_ERROR%>"/>/P>
</logic:present>
```

万岁！

特定的消息可以在特定的地方输出，而通用的错误仍然可以在它们应该的地方输出。

当然，你不必一定要设置全部标准标签。如果这些变化还不能满足你的特定要求，就请你看看源代码，自己修改它们了。框架提供了队列，但如何输出，则在于你。

10.4.4. 成功的控件

我们编写的许多动态页面涉及到提交HTML form。当工作于HTML表单时，重要的是记住空的，或者“不成功的”控件是如何处理的。如果一个控件不成功，浏览器则可能不包括在请求之中。在这种情况下，你不能得到一个空字符串或者表示null的任何东西。控件的有关参数将将不会出现。

下面是一些注意事项，来自于HTML 4.01 规范 [W3C, HTML] 中关于成功控件的内容：

- 被禁止的控件不能成功
- 隐藏控件和因为样式表设置没有被渲染的控件仍然可以成功
- 如果在提交时，控件还没有当前值，用户代理 (例如，web 浏览器) 就不要求将它处理为成功控件
- 如果表单含有不止一个提交按钮，仅有激活的那个会成功
- 所有 on 的复选框都会成功
- 对共享相同的名称属性值的单选按钮来说，只有on 状态的单选按钮会成功

- 对菜单来说，控件名称由SELECT 元素提供，而值由OPTION 元素提供。只有选定的选项会成功。

当没有项目选中，控件不会成功，而且名称和值都不会提交给服务器。因为 HTML 处理不成功控件的方式， Struts ActionForm 具有一个 `reset` 方法。`reset` 方法可用来在控件不成功而没有被浏览器提交时，设置控件地缺省值。问题是当使用复选框地表单被重复提交，而 ActionForm 是在用户的会话中维护的情况下。如果用户去选择了一个复选框，控件变为不成功，而根本不会被提交。作为结果，会话对象中的 ActionForm 值，不会变成未选择。解决方法是使用 `reset` 方法关闭所有的复选框，然后让请求打开成功的那些。

10.5. 其他可代替的视图

Struts 并不是仅仅针对JSP的。现在有一些新的有效扩展，可以将框架连接到其他表现系统上。比如XSLT 和 Velocity 也是Struts 应用的首选。

10.5.1. Struts和JSP

在第2章，我们讨论了MVC架构模式，以及框架是如何作为控制器的角色。当然，没有一个控制器应该是一个孤岛。要使其有用，一个应用还需要模型和视图组件。JSP是在Java中创建动态视图的标准手段。为帮助 JSP 开发人员集成控制器元素和应用的其它部分，Struts 包括了一个我们在本章前面涉及过的综合性标签库。

当然，Struts 自己不会渲染JSP。这是容器的工作。在内部，.jsp 扩展名被映射到运行在容器中的一个服务。当某些组件，比如Struts，请求一个JSP时，容器服务会接受请求，并给出一个响应。 JSP可以包含我们的Struts所提供的标签库中的标签，但是调用标签后面的代码是JSP自己的事情。这种安排的优点是在Struts 和JSP间强迫进行分层。因为Struts实际上并不渲染JSP，并且在容器中它并没有特殊的权限， Struts 和JSP的通信是和servlet 相同的方式—通过servlet 上下文[Sun, JST]进行的。

10.5.2. Servlet上下文

Sun的 Java web架构鼓励开发人员将应用创建为相关servlet的集合。每个servlet 负责处理特定类型的请求，然后返回响应或者将控制转发到另一个servlet。

在实践中，servlet 通常需要包含附加的其它信息到HTTP 请求中去。在进行转发时，信息可以被加入到原始的HTTP 请求中，但是 HTTP 请求是基于字符串的，并被编码成一个古怪的形式，最多将它视为一个很笨的办法。

为了提供给servlet以HTTP 请求和关于请求的运行时信息，容器将HTTP 请求包装进一个 Java 对象，称之为 *HttpServleRequst对象*。HttpServletRequest提供一个类似于剪贴板的风格的组件，servlet可以使用它来交换信息，它称之为*上下文*。

DEFINITION

Java servlet 可以保存对象到各种称为上下文的共享环境中。上下文中的每个对象都有一个名称和一个对应的对象。同一个应用中的 servlet 可以通过名称来在上下文中检索对象。应用中的 servlet 常常需要共享一些比单个 HTTP 请求存活更长的信息。为了有助于管理共享变量的生命周期，提供了 3 个标准的上下文：应用，会话，请求。对 JSP 来说也可以使用页面范围的上下文。

页面范围的对象不能与其它 JSP 和 servlet 共享。

象Struts这样的应用框架依赖于标准的servlet上下文来允许其组件相互通信和协作。servlet框架提供了各种范围，以便不同类型的对象都具有其自己的生命周期。如果Struts 控制器递交了一个Locale 对象在名称`org.apache.struts.action.LOCALE`下，应用中的其它servlet就可以根据同一个名称对其进行引用。表 10.9 展示了标准的上下文。

表格 10.9 标准上下文

上下文	用途
Application	其中的对象对应用中的所有servlet 都可用
session	其中的对象对于访问用户的HttpSession 对象的所有servlet 都可用。这通常通过调用HttpRequest上的一个方法来发现。Session 上下文可以被servlet或者容器终止。
request	其中的对象对每个处理HttpRequest的servlet 有效。请求可以被从一个servlet 转发到另一个Servlet。当一个Servlet包含另一个Servlet时，请求中的对象请求也可以共享。 (参见第8章 8)
page	对象只对当前JSP在请求的生命期内有效。页面范围只对JSP，而不针对标准的servlet。

Struts 控制器广泛的使用标准上下文。其所有资源都暴露在一个或另一个上下文中。上下文使得 JSP，以及应用中的其它对象，能够被集成到Struts 控制器中。

这允许通过访问标准上下文，而 Struts 被一些视图和模型技术所使用。上下文对应用中的每个 servlet 都有效。

所以，如果你可以在servlet中做的事情，你也可以在Struts做。

NOTE

虽然我们只是从 Web 的角度来介绍了 servlet 请求和上下文的概念，但是这些类的扩展接口的 HTTP 版本对常规应用来说也可用。所以，这不仅仅是 web 的事情。

10.5.3. JSP之外

虽然绝大部分Struts 应用只是用JSP写成，许多开发人员也在它们的应用中使用其他表现方式。将Struts 和XSLT 以及Velocity 一起使用已经是行得通的。紧跟着还有更多。Struts 开发人员现在可以混合和匹配使用它们的表现系统技术，合适的时候使用XSLT，其他合适的地方使用JSP 或 Velocity模板—或者只用一个，或者还有其他系统，只要最符合应用的要求即可。

在第17章，我们将看看如何将JSP 标签迁移到Velocity 模板。

10.6. 小结

JSP 标签是开发结合体的一部分，它从编写粗略的HTML给HTTP 响应开始，发展到使用GUI编辑器创建服务器页面。JSP 标签的最终设计目标是简化来暴露软件工程师可定义的，而页面设计者可使用的JavaBean的属性。近来的进展，如JSTL标签库和JSF使我们越来越接近于目标了，但也还有许多工作要做。Struts JSP 标签提供给Java 开发人员他们今天想要的功能。Html库中的标签紧密地和标准的HTML 元素对应，并通常可以一一交换。其他 Struts 标签使开发人员尽量避免使用scriptlet而且帮助他们重构Model 1 应用到 Model 2。总而言之，结合JSP 标签和Model 2–MVC 架构 提供了一个更加强壮的设计，并打开了向未来扩展的大门。

当然，旅程刚刚走完第一步。本章中的基本使用方法使让开发人员在使用Struts标签构建自己的页面时迈开正确的一步。为了打下更加复杂页面的基础，我们也展示了一些高级的技术，使Struts 开发人员可以使用来对实际的应用构建实际的页面。

非常好，但是现在我该如何...

这时，回答“我应该如何做？”问题的答案应该是“噢，就按你通常做的方式就行。” 定制标签并不是一个新的环境，但简化了使动态数据到经过检验的HTML 环境中的方式。 通过这里描述的基础知识和技术，你应该可以现有的某些HTML 或者JSP表单，将它么迁移到Struts中去。

在下一章，我们将探索使用Titles框架使JSP应用提升到下一个层面。Tiles 能帮助你根据特定的组件构建你的页面，以便页面更容易设计和维护。

11. 用 Tiles 开发应用

Cedric Dumoulin 和 Ted Husted合著

本章包括

- *用动态包含设计动态应用*
- *使用Struts 和 Tiles 框架*
- *理解Tiles 定义和属性*
- *将应用迁移到Tiles*

A foolish consistency is the hobgoblin of little minds, adored by little statesment and philosophers and divines.

—Ralph Waldo Emerson

11.1. 页面布局

可用性是现今应用设计的主要考虑——而一致性是可用性的主要因素之一。用户想要专注于手边的任务，却容易被应用接口和屏幕布局的一些小毛病所打扰。

一致性对动态应用来说还不是个小的挑战；但对手工编码的每个页面来说却是个平常事情。如今已经有许多针对静态页面的布局设计工具可以提供给设计人员。但却很少有针对动态的服务器页面的布局工具。

更糟糕的是，一个应用的外观和感觉通常是最后需要处理的细节，然后又在各个版本中经常变动——或者干脆来一个更大的整个站点重新改版。这对最后的编码和测试来说仍然是个噩梦，即使是仅仅改变了背景色或者添加了一个连接到菜单中也是如此。

当然，一致性不是一个妖怪；它是一个好的设计特性。比如一些其它的组件，web 页面包含许多通用的元素，页头，页脚，菜单等等。经常，这就是我们在页面间进行拷贝和粘贴的工作。但是像一些组件一样，最着bug被发现，特征被增强，又导致另一轮的剪切-粘贴动作以便“重用”。在 web 应用中，页面标签是一个编程组件，就象其它组件一样，并且应该纳入相同的重用标准。

11.1.1. 动态模板的分层

在本书的第1部分，我们强调了应用分层的重要性，通过这样来隔离变化的影响。通过划分应用层次，我们可以修改某一个部分而不影响另外的部分。同样的概念可以用在表现层，用来分离外观感觉和具体的内容。

将页面布局和内容分离的一个方法是使用动态 JSP 包含。JSP 规范提供了动态和静态的包含。标准 JSP 动态包含的动作是<jsp:include>。

我们将服务器页面分割为几个碎片来使用动态包含，每个碎片都有其各自应该完成的职责。一个背景模板可以用来设置缺省的格式和布局，页面碎片可以在运行时包含进来，以提供实际的动态内容。动态包含会降被包含页面得输出组合到原始的页面之中。它就象一个开关，将处理流程调用所有被包含页面，然后再返回到调用者。就象图 11.1 所示，被包含的模板被正常处理，就象它被自己调用一样。

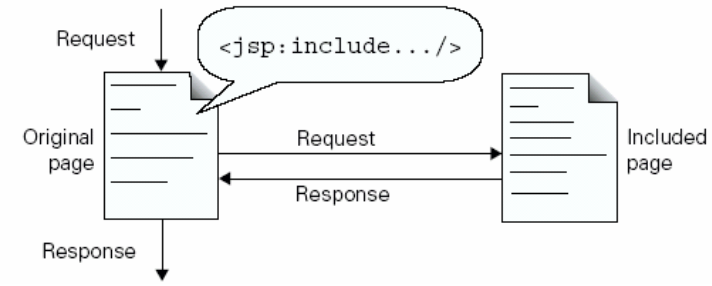


图 11-1 <jsp:include>动作在请求处理中的效果

本章涉及的 Tiles 框架使用一种更加高级的 JSP 包含形式。在一个 Tiles 应用中，背景，

或者布局，模板通常定义页头，菜单体，内容和页脚的位置。其它页面然后被包含进来填充这些位置。如果 header 改变了，仅有那个模板文件需要改变。这个改变将自动影响到包含该页面的所有页面。改变所造成的影响是最小的——妖怪被消灭了。

标准的 HTML 组件，象层叠式样式表 (CSS)，也可以很好的和动态模板一起工作。样式表可以保持模板内部的一致性，并进一步最小化改变的影响。

11.1.2. 模板推论

每一种技术都伴随着一定的折衷。这里是模板技术的一些推论：

- 用，不用，或者折衷——都会伴随在应用中使用动态模板而来
- JSP 包含技术是构建良好的和可靠的，可以扩展至大型应用。包含动态模板的根本技术是 Java Servlet API 核心的一部分
- 大多数容器都针对 JSP 以及象 servlet 包含之类的特征作了优化
- 被包含的页面一般输出 HTML 片断，而且并不完整。这可以防止在标准的 HTML 编辑其中维护模板，这些编辑器要求标记是完整的，而且是独立的页面。
- 在代码改变后，大部分网站都重新编译 JSP 页面。模板创建的更多页面都要监控这种改变。
- 模板实际上重用了那些可能在页面间进行复制的代码。这可以导致可观的节省操作步骤和节省服务器资源

11.1.3. 使用模板

在这一章我们要详细讨论怎样使用模板，特别是使用 Titles 框架实现它。这是一个成熟的产品，和 Struts 框架集成的非常好。Tiles 模板甚至可以从 Struts ActionForward 来部署，大大消除很多其他模版系统需要的“形式”文件。

Struts 和 Tiles 是一个强大的组合。使用动态模板来产生表现页面和其它一些编写 web 应用的手段是一致的。在这一章，我们将展示如何最好地集成 Tiles 和 Struts 以及其它资产，比如 CSS。 介绍完 Tiles 之后，我们也提供了一个重构指南，帮助你将现有的应用迁移到 Tiles 上来。

如果你发现 Tiles 很匹配你的应用，请一定要仔细研究第 15 章的示例应用。表 11.1 本章使用的一些特殊的词汇表：

表格 11.1 动态模版使用词汇

词汇	定义
动态元素	JSP 的一部分，会被JSP翻译器识别，包括action，directive，expression, JSP 标签, 或者scriptlet.
模版数据	JSP文件的一部分，通常不被JSP 翻译器转换，而被逐字传递到响应之中。通常是标记或者可视文本
模版页面	一个包含其他页面或者被其他页面包含的JSP
模版文件	一个包含模版页面的静态文件或者JSP文件

Tile	模版页面的同义词
布局	描述模版文件、tiles应该如何在页面中定位
Tiles	一个强大易用的模版和布局框架
定义	一个Tiles 特征，允许一个布局被标识为一个模版页面，或者一个JavaBean。 定义也可以用一个XML 文档来描述

11.1.4. 组合模板，Tiles和Struts

当 HTML 表格第一次引入时，页面设计者一刻将采用为一种布局机制。无边框的表格可以用来包含其它表格以及内容，通过这种方式来创建一种其它方式所不能的布局。同样的理念经常用在动态模板中。如图 11.2 所示，一个主模板用来提供页面的布局和定位组件，页面片段被包含进来填充这些组件。页面片段可以被包含进各种类型的布局 :使用无边框表格的；使用 标记的，甚至最简单的将一个组件堆在另一个组件之上的布局。

Tiles 是一个使用简单但有效的标记来实现模板和布局的框架。它可以用来替换 Struts1.0 中的模板标记库但功能更加强大。Tiles 包可以被其它 JSP 应用所使用。

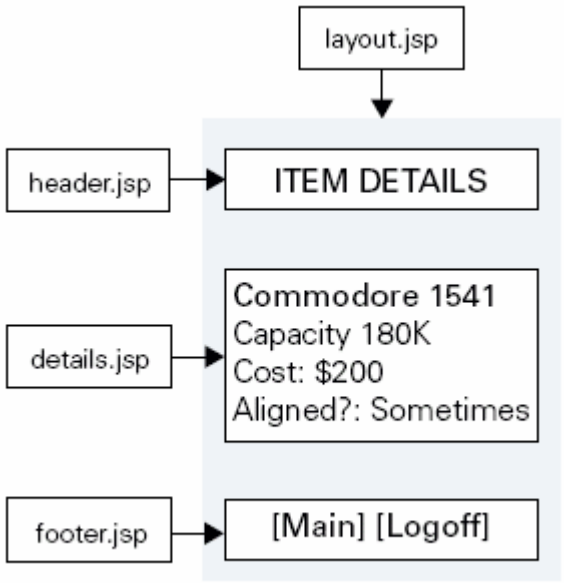


图 11-2 主模版提供了页面的布局

1.0 vs 1.1 Tiles 在Struts 1.1中提供。配置Tiles 请参见第4章。Tiles也可以用在Struts 1.0中，参见Struts Resources 页面中的扩展分类 [ASF, Struts].

通常，JSP 模板系统都是使用一个模板作为布局，而其它则作为填充组件。Tiles 称之为布局文件定义（Definition）。大多数模版系统都要求有一个额外的文件用来标识布局。为避免这种麻烦， Tiles 允许布局定义存储为一个 JavaBean。而且，在 Struts 中还有个扩展，允许定义是一个 ActionForward 的目标。这是一个激动人心的特征，我们将在 11.3.3 节中进行讨论。在 11.5 节，在探讨完使用 Definition 和 Struts 将一个现有的应用重构为基于 Titles

的应用之后，再返回来讨论 Definitaion。

但，现在，让我们从头开始，创建一个新的布局模板。

11.2. 构建一个布局模板

构建布局的第一步是标识组件。对一个经典的 web 页面来说，这些部件可能是页头，菜单，页面体，以及页脚。通常一个简图可以有助于怎样关注页面布局。

为提供一个快速的例子，我们将构建一个经典的 web 页面布局，具有页头，菜单，内容体，以及页脚。我们的草图见图 11.3。

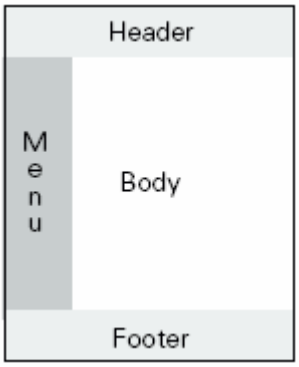


图 11-3 经典的，包括页头，页脚，菜单和内容体的主模版布局

创建一个象图 11.3 一样的模板页面是非常容易的，就象其看起来一样。

- 开发一个新的 JSP 页面
- 导入 Tiles 标记库
- 创建一个 HTML 表格，其单元各匹配这个草图

对每个布局的部件使用一个 Tiles JSP 标签 (<tiles:insert>) 来命名。你可以在清单 11.1 中见到，在我们创建来准备放置一个布局组件的地方，我们都放置了一个 JSP 标签，样子象 <tiles:insert attribute="aName"/>。这个标签是说在这里插入由属性值指定的片断 (或者模板文件) 。当要在应用中使用这个模板的时候，我们会对每一个这些 Tile 传递一个路径。这将允许我们在多个地方使用同一个模板，只需要简单的传递一个或多个不同路径的组件或片断即可。

```
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
  <table border="0" width="100%" cellspacing="5">
    <TR>
      <TD colspan="2">
        <tiles:insert attribute="header"/>
      </TD>
    </TR>
```

```
    <TR>
      <TD width="140" valign="top">
        <tiles:insert attribute="menu"/>
      </TD>
      <TD valign="top" align="left">
        <tiles:insert attribute="body"/>
      </TD>
    </TR>
    <TR>
      <TD colspan="2">
        <tiles:insert attribute="footer" />
      </TD>
    </TR>
</table>
```

在大多数情况下，body 部件可能每个页面都不同，但是在同一区域的所有页面将共享相同的页头和菜单部件。同时，可能同一站点的所有页面都要共享同一个页脚部件。当新的一年过去，就需要在每个页面中更新版权声明，这样便仅有页脚部件需要被更新。

只需要加入剩下的 HTML 标记，我们的经典布局就可以是一个完整的独立的模板页面。

在清单 11.2 中，你将会注意到加入了一个新的 Tiles 标签，<tiles:getAsString name="title"/>。这个标记意思是，将属性值返回为字面字符串，而不是路径名称或者其它命令。为此， Tiles 调用标准的对象的 toString() 方法。结果在运行时直接插入到页面之中。

```
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<HTML>
  <HEAD>
    <TITLE>
      <tiles:getAsString name="title"/>
    </TITLE>
  </HEAD>
  <BODY>
    <table border="0" width="100%" cellspacing="5">
      <TR>
        <TD colspan="2">
          <tiles:insert attribute="header" />
        </TD>
      </TR>
```

```
      <TR>
        <TD width="140" valign="top">
          <tiles:insert attribute='menu'/>
        </TD>
        <TD valign="top" align="left">
          <tiles:insert attribute='body' />
        </TD>
      </TR>
      <TR>
        <TD colspan="2">
          <tiles:insert attribute="footer" />
        </TD>
      </TR>
    </table>
  </BODY>
</HTML>
```

11.2.1. 但什么是小部件（tiles）？

Tiles 框架提供的模板特征要远远超过标准 Servlet 和 JSP 包含功能所提供的。框架称其模板为小部件 tiles。这有助于说明小部件比简单的 JSP 模板功能更加强大。Tiles 是你的表现层的构建块。

技术上，tile 是一个 JSP 页面中的一个矩形区域，又是称之为区域 region 。tile 也可以由其他 tile 装配而成。Tile 可以递归构建，并表现为一个树，如图 11.4。书中的每个节点是一个区域。

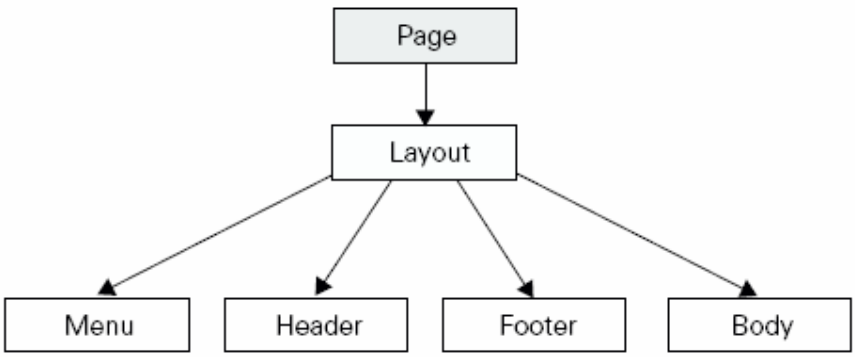


图 11-4 Tiles 可以被表达为树 页面是根，布局是分支，包含其他叶子（页眉、页脚等）

根节点通常是页面。最终的节点，叶子，包含了页面的内容。中间的节点通常就是布局了。布局节点是工具 tile，它可以定位页面中的 tile，或者为内容 tile 提供背景标记。tile 对象支持一些重要的特征，包括参数和定义。

参数

Tile 可以在运行时接受不同的信息，格式为参数(Parameter)或者属性（Attribute)。这意味着 tile 是可参数化的。它们可以接受不同的信息并根据这些信息出现不同的效果。tiles 参数通常称为属性（**attribute**）以避免和请求参数相冲突。

Tile 属性在插入 tile 时定义，并仅在 tile 内可见。它对子 tile 和包含该 tile 的页面也是不可见的。这样在同一个 tile 被在一个页面中使用数次的情况下也可以避免名称冲突。开发人员可以专注于如何最好的使用好 Tile 而不用担心名称冲突。

Tile 属性可以是字符串或者其它类型。11.4 节有关于 tile 属性的详细信息。

定义（Difinition）

如果综合在一起，传递给 Tile 的各种属性实际上创建了一个屏幕的描述。实际上，这些描述中许多是联系在一起的，而且可能一个会构建在另一个之上。

定义存储了一组属性，以便屏幕描述成为一个独立的对象并有自己的标识。可以声明一个基础屏幕定义，然后根据基础定义的衍生创建其它定义。如果基础屏幕的细节改变了，那么根据此扩展的其它屏幕定义也会改变。这就将面向对象的继承和封装原理带到了你的动态页面设计之中。

我们在 11.4 节讨论 Tile 定义。

定义是可选的。你也可以随时使用简单的 JSP 标记部署 Tile。

11.2.2. 部署Tiles模板

我们在 11.2.1 构建的布局模板定义了在何处放置 tile 但没有说明要使用什么 tile。那些细节——以及其它特别之处——会在部署时传递给布局模板。最简单的方式是从另一个 JSP 页面中调用布局模板。

清单 11.3 显示了使用一个 JSP 页面来向一个布局模板传递后者应该使用什么 tile。在 hello.jsp 被渲染时，它将返回返回 myLayout.jsp，连同特定 tile 的内容一起。

```
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<tiles:insert page="/layouts/myLayout.jsp" flush="true">
  <tiles:put name="title" value="Hello World" />
  <tiles:put name="header" value="/tiles/header.jsp" />
  <tiles:put name="footer" value="/tiles/footer.jsp" />
  <tiles:put name="menu" value="/tiles/menu.jsp" />
  <tiles:put name="body" value="/tiles/helloBody.jsp" />
</tiles:insert>
```

为了与不同的页面体使用同一个布局，我们只需简单地将标签

```
<tiles:put name="title" value="Hello World" />
<tiles:put name="body" value="/tiles/helloBody.jsp" />
```

替换为新的细节：

```
<tiles:put name="title" value="Hello Again" />
<tiles:put name="body" value="/tiles/pageTwo.jsp" />
```

新的页面将看起来更像原始的 hello.jsp， 除了不同的标题(Hello Again) 和不同的页面体 tile (pageTwo.jsp 的内容)。

你也可以按这种方式继续使用布局，只要根据需要替换不同的属性值就行。这种参数的传递可以用一个单一的基础模板来安排整个网站的所有页面。如果站点的布局需要改变，只需要更改一个基础模板。

然而，要完全使用这个方法，对每个新部署的页面你必须至少要两个 JSP 文件：一个文件用作新内容，而另一个文件用来插入模板并且包含第一个文件中的新内容。本章后面，我们将展示如何在 Struts 中使用 Tile 定义来避免使用第二个文件。下一节的**页体—包装（body-wrap）部署方式**也是避免第二个文件的一个方法。

Body-wrap部署

某些 tile 可以在一个应用中使用很多次。实践中，应用中的大部分 tile 可能都是内容 tile，它们提供了给定页面中截然不同的那个部分。典型地，页面的内容 tile 在应用中仅使用一次。页头，页脚，以及菜单 tile 可能会在各个页面中反复使用。但多次使用的 tile 往往是每个特定内容 tile 的装饰窗口。

在这种情况下，你可以简单地使用屏幕定义的剩余部分来包装内容 tile。窍门是只需要将 HTML 标记提供为 put 标签的值。如清单 11.4 所示，确保指定了 type="string"，以便 Tile 不会找不到页面的路径。

```
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<tiles:insert page="/layouts/myLayout.jsp" flush="true">
  <tiles:put name="title" value="Hello World" />
  <tiles:put name="header" value="/tiles/header.jsp" />
  <tiles:put name="body" type="string">
    <%-- Place the content from /tiles/PageTwo.jsp here --%>
  </tiles:put>
  <tiles:put name="footer" value="/tiles/footer.jsp" />
  <tiles:put name="menu" value="/tiles/menu.jsp" />
</tiles:insert>
```

这样就避免创建额外的文件。另一个效应是防止 body tile 被其它的页面使用。当 body 标记确实需要在一个以上的地方使用时，你就可能需要重构页面，以便使内容处在一个分离的 tile 之中。但几乎对所有的页面而言，内容 tile 都可能只需要使用一次。

body wrap 是一个非常有效率的方式。唯一不好的是屏幕定义散布在整个站点之中，这使得要进行全局的改变变得很困难。11.3 节要描述的 Tile 定义，提供了一个更加中心化的方法，它和 Struts 架构一起配合得很好。但是 bodywrap 部署模式对小的应用来说也是很好的选择。

11.2.3. 添加样式表

因为 tile 都是 JSP 页面，所以针对 JSP 的所有配置它都可以具有，包括 CSS[W3C, CSS]。在 tile 中使用样式表不是必需的，但会有所帮助。

虽然现今的浏览器并不全面支持 CSS，这仍然是一个定义颜色方案和其他关键属性的有用的方法。样式表可以确保这些漂亮的样式在 tile 之间保持一致。

如果要指定一个样式表，只需简单地插入一个标签。<html:base> 标签可以帮助浏览器解决到样式表或者其它资产的相对路径问题。当然最好使用<html:rewrite> 标记来渲染路径，并且进行 URL 编码。

清单11.5 显示了如何使用这两个标记。

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<HTML>
  <HEAD>
    <TITLE><tiles:getAsString name="title"/></TITLE>
    <html:base/>
    <LINK rel="stylesheet" type="text/css"
            ref="<html:rewrite page='/assets/styles/global.css'/>">
  </HEAD>
  <BODY>
```

清单 11.1 使用<html:base> 和<html:rewrite>

11.2.4. 模板和MVC

正确使用动态模板可以在 MVC 架构之下工作得很好。(参见第 2 章关于 MVC 的详细信息。)动态模板清晰地将标签和内容分开了。在实践中，页面实际的内容部分往往处于页面的中心，被其它标记或者导航所围绕。内容，标记，导航可以简单地映射为模型、视图和控制器。

通常，对于从一个现有的应用系统中创建一个 MVC 的模板系统，只需要使用标准的重构技术就可以了。页面的基础标记可以提取到一个主模板中。站点的页头和页脚分别提取到各自的文件中，容纳内容的区域也放到其自己的模板文件中。然后基础模板再将它们包含回来。象 CSS 之类的相关技术可以用来定义基础模板的基本格式，并可以无缝地应用于形成最后页面的其它部分。

DEFINITION Refactoring （重构）是一个通过对源代码的重新构建提高其清晰性和灵活性以及消除冗余和无用的代码的过程.

Tiles 包通过它的定义（Definition）特征来更加前进了一步。使用定义，我们可以串联应用需要的各种物理模板，并且将实现细节移到一个中心 JSP 页面，或者，最好是一个 XML 文档。

11.3. Tiles定义（Definition）

在清单11.3中，我们看到即使最简单的模板布局也需要大量的参数。我们也看到，在这些参数中，只有几个会在页面间发生变动。我们需要做的是，有一种方式来在一个可重用的资源束中定义所有这些特性和属性，而这个资源束可以在参数改变时重新装入，——这其实就是Tiles定义要做的事情。

11.3.1. 声明Definition

因为目的相似，编写 Tiles 定义和编写<tiles:insert> 标签非常相似，就象我们在 11.2.2 节中所做的一样。 一个定义需要如下的信息：

- 到基础模板文件的路径
- 一个列表，包含 0 个或者多个要传递到模板的属性 (名值对)
- 一个定义的标识符

如你所见，Definition 和 <tiles:insert> 标签之间真正的不同在于 Definition 可以命名。但是，尽管只加入了一个标识性到特性列表上，却打开了几扇门:

- 在部署时，一个模板 Definition 可以通过传递附加的或者替换的属性而重新装入
- 一个模板定义可以通过基础定义扩展到其他 Definition
- 一个模板 Definition 可以被存储在一个 JSP 中被重用，或者从一个 XML 文档中被装入
- 一个模板 Definition 可以是 Struts ActionForward 的目标

我们来看看指定 Tiles Definition 的两种方式，用 JSP 或者通过 XML 文档。

11.3.2. JSP声明

一个开始使用 Definition 的快速和容易的方式是在 JSP 中声明它。这并不减少你的应用所需的模板文件数，但是确实可以使你能够通过重新装入来进行重用。你可以声明一个基础定义，然后在其它定义中指定新的定义是如何不同于它们的前辈的。

这样要求一个存根（stub）文件用来在运行时部署定义。在 11.3.3 节，我们将看到将这些同一个定义放入到一个 XML 文档，然后直接从 Struts ActionForward 中进行部署。因为基本流程是一样的，让我们先来讨论 JSP 类型的声明。

在 JSP 页面中使用 Tiles Definition 的基本流程包括:

- 在 JSP 中声明一个 Definition
- 部署一个 JSP 声明的 Definition
- 重载一个 Definition
- 在 JSP 中重用 Definition

我们来依次讨论。

11.3.3. 通过JSP声明一个Definition

清单11.6 指定和清单 11.4一样的特点，但使用了Definition

```
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
```

```
<tiles:definition id="definitionName" page="/layouts/myLayout.jsp">
  <tiles:put name ="title" value="Hello World" />
  <tiles:put name ="header" value="/tiles/header.jsp" />
  <tiles:put name ="footer" value="/tiles/footer.jsp" />
  <tiles:put name ="menu" value="/tiles/menu.jsp" />
  <tiles:put name ="body" value="/tiles/helloBody.jsp" />
</tiles:definition>
```

清单 11.2 一个简单的定义

清单11.6 中的Definition将作为一个bean被保存在JSP 上下文范围中，使用 id 作为属性关键字。像许多Struts 标签一样，<tiles:definition> 标签支持scope 属性来指定一个特定的上下文 (应用，会话，请求，页面)。缺省的是页面上下文。这个定义仅对该JSP的余下部分有效。

部署JSP声明的Definition

要将一个定义放到应用中，你可以使用 `<tiles:insert>` 标签，指定bean的名称 (Definition的 ID)，如果需要也可以包括范围，如清单11.7所示：

```
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<tiles:definition id="definitionName" page="/layouts/myLayout.jsp">
  <tiles:put name="title" value="Hello World" />
  <tiles:put name="header" value="/tiles/header.jsp" />
  <tiles:put name="footer" value="/tiles/footer.jsp" />
  <tiles:put name="menu" value="/tiles/menu.jsp" />
  <tiles:put name="body" value="/tiles/helloBody.jsp" />
</tiles:definition>
<tiles:insert beanName="definitionName" flush="true/>
```

清单 11.3 部署一个定义

启动时， Tiles 使用 XML 元素的 id definitionName 名称创建一个定义对象 (或 bean)。运行时， <tiles:insert> 标记通过 beanName 属性引用定义 id。

重载一个定义

一旦我们已经声明了一个定义，我们就可以通过其名称来引用它，并且重载某些或全部属性。要重载属性，只需要重新表明以一个新值。要创建一个新的属性，请包含一个新的属性名称和值就可以了。这就允许你创建一个基础定义，然后标识那些需要改变的属性来创建一个新的页面，如清单 11.8 所示

```
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<tiles:definition id="definitionName" page="/layouts/myLayout.jsp">
```

```
<tiles:put name="title" value="Hello World" />
<tiles:put name="header" value="/tiles/header.jsp" />
<tiles:put name="footer" value="/tiles/footer.jsp" />
<tiles:put name="menu" value="/tiles/menu.jsp" />
<tiles:put name="body" value="/tiles/helloBody.jsp" />
</tiles:definition>
<tiles:insert beanName="definitionName" flush="true" >
  <tiles:put name="title" value="New PageTitle" />
  <tiles:put name="body" value="/tiles/anotherBody.jsp" />
  <tiles:put name="extra" value="/extra.jsp" />
</tiles:insert>
```

清单 11.4 重载一个定义

这儿，在声明了定义之后，我们的 <insert> 标签指定了一个新的页面 title 和另一个不同的 body，并添加了一个在原始定义中没有的新的 extra 属性。这意味着布局没有管理到这个 tile，但如果提供了也可以显示它。布局可以使用可选的 ignore 属性值。当 ignore 设置为 true，如果属性不存在，将不会报告错误。

这也是为什么 myLayout.jsp 相对于原始定义要添加一个 extra tile:

```
<tiles:insert attribute="extra" ignore="true" />
```

在JSP中重用定义

在前面的例子中，在重载某些属性之前，我们不得不重复原始定义。当然，这不是一个很好的重用实践方法。还有另一个好一些的方法，请看清单 11.9，是如何声明和重用的。它通过使用一个工具页面，然后在需要某些定义的时候包含它。

```
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<%@ include file="definitionsConfig.jsp" %>
<tiles:insert beanName="definitionName" beanScope="Request" >
  <tiles:put name="title" value="Another Page" />
  <tiles:put name="body" value="/tiles/anotherBody.jsp" />
</tiles:insert>
```

清单 11.5 包含一个定义

这里的关键之处在于，标准的 JSP include 标签将一个包括了我们的 Definition 的文件包含了进来。Definition 是正常创建的；你可以按通常的方式使用这些 Definition。这个例子也重载了 title 和 body 属性来定制化页面。

如果你的应用中只有少量的 Definition，这个方法工作得很好。但它扩展性不够。因为每次文件被包含进来后，所有的定义对象都要重建，这样影响了性能。在开发时，这仍然是一个好办法，因为对 Defintion 的修改在下次页面装入时就会体现出来。

在生产环境中，有一个方法是在应用范围中创建定义，然后用 Struts <logic:notPresent> 标签块（或者相等的标签）保护起来，如清单 11.10 所示：

```
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<logic:notPresent name="definitionName" scope="application">
  <tiles:definition
          id="definitionName"
          Page="/layouts/myLayout.jsp">
    <tiles:put name="title" value="Hello World" />
    <tiles:put name="header" value="/tiles/header.jsp" />
    <tiles:put name="footer" value="/tiles/footer.jsp" />
    <tiles:put name="menu" value="/tiles/menu.jsp" />
    <tiles:put name="body" value="/tiles/helloBody.jsp" />
  </tiles:definition>
  <%-- … other definitions … --%>
</logic:notPresent>
```

如果定义已经被创建而且存储在应用范围的上下文中，它们就不需要被重新创建。每个页面仍然需要包含定义，而且每个页面将查看它们是否存在，但至少它们不需要不断创建它。

Tiles 提供了一个更好的方法来装入和重用 Defintion。Definition 可以从 XML 文档声明和装入一次。在内部， Tiles 直接从 ActionForward 中渲染 Definition。这确实是一个管理 Tiles Defintion 的好办法，在下一章我们将讨论这个方法。

11.3.4. 配置文件声明

通过在一个 XML 配置文件中声明你的 Definition，你可以使你的应用在启动时装入这个文件，并且创建一个包含你的 Definition 的“ Definition 工厂 ”。 每个 Definition 用其名称属性来标识，并且要注意它们应该是唯一的。其它组件，如 ActionForward，就可以通过名称来引用定义。

在一个 XML 配置文件中声明 Tiles Definition 需要一些额外的设置，以允许在应用初始化时可以读取并装入配置文件。参见第 4 章，可以得到关于如何在 Struts 1.1 中安装 Tiles。

应用初始化时，XML 配置文件被读取，然后解析到一个定义工厂，该 Definition 工厂包含了每个声明过的 Defintion 的实例。每个 Definition 应该有其唯一的名称，以便它可以被 JSP 标签或者 Struts ActionForward 引用。Definition 的名称及用作内部引用。它不是一个 URI，不能直接被客户引用。

使用 XML 文档的 Definition 声明的流程和使用 JSP 的处理并没有什么太大的不同。主要的不同点是它们如何被创建，如何被扩展，以及 Definition 如何可以被 Struts ActionForward 使用。

创建配置

XML 配置文件的总体格式有点类似于 Struts 配置文件(因为它们都是正确的 XML 文档)。

令人惊讶的是，XML 配置文件文件之中使用的语法和 Tiles <definition> 标记相像，如清单 11.11 所示：

```
<!DOCTYPE tiles-definitions PUBLIC
"-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
"http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">
<tiles-definitions>
  <definition name="definitionName" page="/layouts/myLayout.jsp">
    <put name="title" value="Hello World" />
    <put name="header" value="/tiles/header.jsp" />
    <put name="footer" value="/tiles/footer.jsp" />
    <put name="menu" value="/tiles/menu.jsp" />
    <put name="body" value="/tiles/helloBody.jsp" />
  </definition>
  <!-- ... more definitions ... -->
</tiles-definitions>
```

在本书的网站[Husted]上和一个空白应用一起提供了一个空的Tiles Definition文件。

NOTE

Tiles Definition文件的名称和位置可以在web 应用部署描述符(web.xml)中指定。我们推荐你可以创建一个配置文件夹在 WEB-INF之下，用来存放在Struts应用中越来越多的配置文件.

扩展定义

一个 Definition 可以设置为另一个 Definition 的子类。这时，新的 Definition 继承了父亲 Definition 的所有特性和属性。 属性 extends 就是用来指示其父 Definition 的：

```
<definition name="portal.page" extends="portal.masterpage">
  <put name="title" value="Tiles 1.1 Portal" />
  <put name="body" value="portal.body" />
</definition>
```

在这个代码段中，我们配置了一个新的 Definition，名称为 portal.page，它扩展了 portal.masterpage 的定义。新的定义继承了父定义的所有特性和属性。在前面的片断中，属性 title 和 body 被重载了。这种继承能力允许我们声明一个根定义—设置缺省的属性—然后通过特定的属性来扩展定义 (如 title 和 body)。如果你的所有定义都是从一个根定义中扩展而来，那么在根定义中的修改就会影响到所有扩展至它的定义。扩展类似于重载，但增加了持久性。重载描述了我们在何处指定了一个定义，并且传递给它新的参数 (或者属性)的过程，就象调用一个方法，并传递给它参数一样。但是 <tiles:insert> 标记不能调用另一个<tiles:insert> 标记，所以重载的定义不能被引用和重用。通过使用 extends 属性，你可以创建一个新的定义。这个新的定义就可以被插入，被重载，甚至被其它定义扩展。它仍然通过继承树被连接回其祖先，如图 11.5 所示。

扩展和重载定义可以戏剧性地减少你的页面声明中的冗余信息。网站中的每个标签，导航和内容组件都只需要声明一次。组件可以被以被用在任何需要用到的地方。

虽然这的确很酷，我们仍然需要一个额外的页面来容纳定义。这意味着要添加一个新的内容的页面时，我们需要添加一个内容页面然后然后用一个页面来插入标识新内容的定义。常规应用将对 60 个内容页面有 60 个页面。而一个模板应用也至少有 120 页面来完成同样的事情。模板页面比其常规页面要小巧而简单，但如何管理文件却是个问题。

解决页面暴涨的方法使用 Struts ActionForward 来整合这些定义。.

11.3.5. 将Definition用作ActionForward

在一个 Struts 应用中，大部分页面都不是直接引用的，而是封装在一个 ActionForward 对象之中。 ActionForward 被给定一个唯一的逻辑名称，以及一个通常用来引用表现页面的 URI。Struts Action 选择并返回一个 ActionForward 给控制器 servlet。ActionServlet 然后将控制转发到由 ActionForward 的 path 属性指定的 URI。 (关于 ActionForward 的详细信息，参见本书第 1 部分)。

Tiles 包中包括一个 ActionServlet 子类，它会检查 path 属性是否匹配 Definition。如果定义的 id 和 ActionForward 的 path 属性匹配,那么定义 bean 就会被放入在请求范围上下文中，控制被转发到布局，装配好的模板就会显示。

因此，你可以配置一个使用 Definition 的名称而不是 URI 的 ActionForward：

```
<action
  path="/tutorial/testAction2"
  type="org.apache.struts.example.tiles.tutorial.ForwardExampleAction">
  <forward
    name="failure"
    path=".forward.example.failure.page"/>
  <forward
    name="success"
    path=".forward.example.success.page"/>
</action>
```

当然，你也可以使用传统的斜杠方式不是是点号方式来命名定义：

```
<action
  path="/tutorial/testAction2"
  type="org.apache.struts.example.tiles.tutorial.ForwardExampleAction">
  <forward
    name="failure"
    path="/forward/example/failure.page"/>
  <forward
    name="success"
```

```
    path="/forward/example/success.page"/>
</action>
```

然而，第二中命名方式可能和与 Action 和页面 URI 相关的标识符相冲突。一个好办法是在 Tiles 定义中使用点号，而在 ActionForward 中使用斜杠语法。

这样就确保了名称是不交叉的。Action 中的代码和以前一模一样。如果你要从 ActionForward 中的表现页面的 URI 切换到 Tiles 定义，大部分 Action 类都不需要重新构建。典型地，Action 类将忽略 ActionForward 路径，而只是根据名称来处理 ActionForward。

你可以在你的应用中混合使用常规的 ActionForward 和 Tiles-Definition-ActionForward。Tiles ActionServlet 会按自己的方式处理每个对其项目的请求。当以这种方式使用 ActionForward 时，应用中的模板页面数量会显著减少。为了整合 60 个内容页面，只需要加上少量的工具 tiles 来提供标准的导航和布局特征。但是创建第 61 个页面意味着只需要再创建一个仅有内容的 JSP 和一个 XML 定义，而后者经常只有一行。

将 Tiles 定义部署为 ActionForward 给了你动态模板更大的能力和灵活性，而没有多余的常规“官样形式”文件。

11.4. Tile属性（Attributess）

能扩展定义和重载特性是一个强大的特征。但到目前为止，我们仅仅展示了一些属性是被硬编码到页面中的静态值的例子。但究竟你需要在运行时指定一个属性吗？是否 Action 能够传递属性值给一个 tile 吗?

是的，可以。

秘诀是，Tiles 将属性存储在其自己的上下文中。 就象 JSP 存储属性到页面上下文中一样，Tiles 存储它的属性到与用户请求相关的 Tiles 上下文中。

对于 Tiles 上下文没有什么神秘的。它就是一个简单的集合，由 Tiles 在请求中创建以供其组件使用。特定的上下文是一种管理由组件创建的，并与其它组件共享的各种对象的通用技术， 特别是在控制可能被从应用的一层传递到另一层的时候。有好几个 Tiles 标签都是设计来使用 Tiles 上下文的，包括使用属性（Attribute）。

11.4.1. 使用属性

<tiles:useAttribute> 标签使得某个 Tiles 上下文属性在整个页面上下文中有效。这使得属性值对那些读这个页面上下文的标签有效，比如 Struts <bean:write> 标签。

相同的属性名称可以被用在上下文中

```
<tiles:useAttribute name="myAttribute" />
```

或者可以指定另外一个属性名称:

```
<tiles:useAttribute
        attributeName="anAttribute"
        name="myAttribute" />
```

一旦 Tiles 将属性放入页面范围，Struts bean 标签就可以以常规的方式引用这个信息：

```
<bean:write name="myAttribute" />
```

<useAttribute> 标签也可以用来声明一个脚本变量供 JSP scriptlet 使用:

```
<tiles:useAttribute
        id="list"
        name="myAttribute"
        className="java.util.List" />
```

通常， <useAttribute> 标签与 <useBean> 动作和 Struts <bean:define> 标签相对应，但是允许访问 Tiles 上下文中的属性。

注意，每个 tile 都是一个独立的 JSP，因此它们有其自己的页面上下文。为了输出一个属性，以便它对组成全部响应的页面都有效，要指定另一个范围。例如，

```
<tiles:useAttribute name="myAttribute" scope="request"/>
```

将属性放在请求范围，在那里其它 tile 中的标签才可以找到它们。

因为每个 tile 从技术上说都是一个不同的“页面”，每一个都在其自己的页面范围之内。如果你想避免和另一个 tile 的属性冲突，你可以使用页面范围。如果你想一个属性被另一个 tile 共享，你可以使用请求范围。

<useAttribute> 操作是按顺序进行的。页面中后面渲染的 tile 可能可以使用某个属性，而早先渲染的 Tile 却可能不能使用它，因为它还不存在。

11.4.2. 导入属性

缺省时，<tiles:importAttribute> 标签将全部 Tiles 上下文属性导入到页面上下文:

```
<tiles:importAttribute/>
```

存储在当前 Tiles 上下文中的部分或者全部属性会在整个标准的页面上下文中有效。

可选地，可以指定一个单独的属性或者另一个上下文：

```
<tiles:importAttribute name="myAttribute" scope="request"/>
```

但是，和<useAttribute>不同，<importAttribute> 不支持重命名属性，或者将它们暴露为一个脚本变量。

11.4.3. put

<tiles:put> 标记使用来将一个值和一个属性相关联。name 属性通常被指定为一个简单的字符串，但实质上却可以为各种可能的类型：标签属性，标签体（body），JavaBean 等等。

put为标签属性

当在 JSP 中使用时，这通常是一个 put 命令的最通用形式。值通常用一个简单字符串来设置，但也可以是一个运行时值（表达式）：

```
<tiles:put name="title" value="My first Page" />
```

或者

```
<tiles:put name="title" value="<%=myObject%>" />
```

put为标签体

就象许多 JSP 标记一样，值属性也可以设置为标签体:

```
<tiles:put name="title">My first Page</tiles:put>
```

这个方法也可以用来嵌套其它标记的输出：

```
<tiles:put name="title">
  <bean:write message="first.pageTitle"/>
</tiles:put>
```

put为在某个范围定义的bean

与其他许多 Struts 标签一样，属性可以通过一个 bean 来传递:

```
<tiles:put name="title" beanName="myBean" />
```

beanName 标识的对象被获取，然后其值被用作为属性值。如果 myBean 不是一个字符串，Struts 或 Tiles JSP 标签将会自动调用对象的缺省 toString() 方法，以便 myBean.toString() 的结果值可以被用来设置 value 属性。

缺省时，范围会按通常的顺序进行搜索，直到找到第一个 myBean 的实例。当然你也可以指定一个特定的范围:

```
<tiles:put name="title" beanName="myBean" beanScope="session"/>
```

这样可以忽略在页面或请求范围的一些实例，而直接查找会话范围。Tiles 添加了一个它自己的上下文，称之为 tiles，这可以在标准范围检测之后进行检测。beanScope 可以接受的值为：页面, 请求, 应用,以及 **tiles**。

put为在某个范围内定义的Bean的属性

还是和许多 Struts 标签一样，你也可以在 JavaBean 上指定一个特定的属性。 <put> 标签然后会调用这个方法来设置属性值:

```
<tiles:put name="title" beanName="myBean" beanProperty="myProeprty"/>
```

这相当于是调用 myBean.getMyProperty 设置 title 属性的值。

指定属性类型

<put> 标签用来设置一个可以被其它标签使用的属性，通常这些标签是<tiles:insert> 或者 <tiles:get>。对应的标签可以使用各种方式传递的值。它可以指定一个直接字符串，一个页

面 URL，或者需要插入的另一个定义。属性需要表达的数据的类型可以在 put 标记中用可选的 type 属性来指定。这可以帮助相应的标签根据需要插入值:

```
<tiles:put name="footer" value="/tiles/footer.jsp" type="page"/>
```

如果指定了这个属性，有效的数据类型可以是：string, page, 或者 definition。

表 11.2 提供了每一个类型的描述。

表格 11.2 put 标记的 type 属性的可用的 token

Token	说明
string	表示一个字符串的值属性
page	表示一个URL的值属性
definition	表示一个定义名称的值属性

指定安全角色

当使用基于容器的安全认证时，你也可以为 tile 指定一个安全角色。如果用户不是指定的角色，那么属性就不会被设置。

```
<tiles:put name="title" value="myValue" role="myManager"/>
<tiles:put name="title" value="myValue" role="myStaff"/>
```

这允许你为每个安全角色指定一个 tile，而让框架为当前的用户选择相应的 Tile。

如果 <put> 被用在<tiles:insert> 标签中，角色会立即被检测。如果<put> 被用在<tiles:definition> 标记中，角色在 Tiles 上下文初始化时检测。

11.4.4. putList 和 add

除了接受单一对象之外， Tiles 属性也可以是 java.util.List。你可以为一个属性指定一个序列对象，并且作为一个单一属性传递它。<tiles:add> 标记被嵌套在<tiles:putList> 中来指定放入列表中的项目：

```
<tiles:insert page="menu.jsp" >
  <tiles:putList name="items">
    <tiles:add value="home" />
    <tiles:add value="documentation"/>
  </tiles:putList>
</tiles:insert>
```

<putList> 标记经常和 <useAttribute> 或者 <importAttribute>标签一起使用，来使列表

可以被页面中的其他标签访问到：

```
<tiles:importAttribute/>
<table>
  <logic:iterate id="item" name="items" >
    <TR>
      <TD>
        <bean:write name="item">
      </TD>
    </TR>
  </logic:iterate>
</table>
```

11.5. 迁移一个应用到Tiles

在本章开头，我们描绘了 Tiles 是一个使你的应用达到良好一致性和易用性的手段。我们也说明了一致性是良好设计的特点之一。这是因为一致性意味着可重用。

重用可导致应用不容易崩溃并且易于维护。经常，你已经有了一个现存的应用，你想将其改变到 Tiles 之上。你也许是想改善外观效果，以使其看起来更加一致，或许你是想提高其功能设计——或者都想。改善现有软件的设计称之为重构（refactoring） [Fowler]。

如果你熟悉常规的重构技术，迁移一个应用到 Tiles 类似于提取（Extract）方法。 Tiles 等同于那些称之为渲染页面的方法。当我们完成后，页面被简化为一个布局以及一些它调用的 tiles。这个工作有点象一个类中的中心方法，该方法在调用一个方法后调用又另一个成员方法。

一旦我们提取出我们的 tile 并将页面简化为一个清单，我们便可以再进行下一步，将页面替换为 XML 元素。Struts 可以直接使用这个元素来渲染 tile，而不需要和其它插入页面打交道。但是，如其它所有的重构一样，我们先从最小一步开始。

11.5.1. 设置 Tiles 框架

首先对所有的东西做一个备份，不管你在常规方式下做了多少备份。迁移是一个麻烦而又实际的过程，在所有的东西都 OK 之前要通过好多道关。所以要准备不时地回滚。

Struts 1.0

如果你还没有做，要做的第一件事情就是安装 Tiles 1.0 包，然后通过应用的部署描述符来装入 Tiles servlet。1.0 版的 Artimus (见第 16 章)是基于 Tiles 的一个可工作的例子。然后测试你的应用确保它们工作得很正常。

另外，确保设置一个骨架 Tiles 定义的 XML 文档，以便你可以在重构过程时开始填充它。

Struts 1.1

Tiles 已经集成到 Struts 1.1 中。我们在 4.8 节中已经涉及到如何在 Struts1.1 中激活 Tiles 支

持。

11.5.2. 测试缺省配置

在部署描述符 (web.xml)中设置 debug 和 detail 参数为 level 2，然后重启动应用。仔细检查日志记录项，看是否有新的错误信息产生。 进行一些单元测试，然后点击整个应用确认看是否还是有效。

11.5.3. 评估页面

在 Tiles 引入和运行后，下一个艰苦的工作就是仔细查看你的页面以决定总体布局类型和每个布局的区域划分。

现在也是考虑命名方案的时候。每个组件都需要有其自己的标识符。

标识布局

当你彻底审视一个应用时，你会发现各种菜单和对话框页面—列表页面，视图页面，以及编辑页面—位于其他页面之中。现在注意力应该集中在页面包含了什么以及页面的各个部件如何一起配合。它有 header 或 footer 吗?它有菜单吗? 菜单是在旁边还是顶部？在这时，部件，比如导航条，的相对位置比在其在该导航条中的实际链接要重要得多。从这些组开始，试着标识一些公共的布局。再次提醒，注意力集中于整个应用使用的视觉布局，而不是页面内容。

标识tile

下来要仔细关注每一个公共布局，标识出独立的 tile。决定页面的部件如何配合在一起。然后看看页面本身，标识出独立的 tile。每个 tile 的表现代码可以存储为一个单独的文件。我们可以在多个布局中使用它，或者在不触及其他 tile 的情况下编辑它。

如前所述，这有点类似于对一个大型的 Java 类使用 Extract 方法。一个线索是在现有的代码中查寻注释和说明，比如 <!-- menu starts here -- >，以一段说明开始的语句块往往是 tile 的候选者，比如清单 11.12：

```
<%-- messages --%>
<TR class="message">
  <TD colspan="3">
    <logic:messagesPresent>
      <bean:message key="errors.header"/>
      <html:messages id="error">
        <LI><bean:write name="error"/></LI>
      </html:messages>
      <bean:message key="errors.footer"/>
    </logic:messagesPresent>
  </TD>
</TR>
```

清单 11.6A 一个很好的候选 tile

页面中的样式改变也是潜在 tile 的标志。如果页面设计员引入了一个样式用来分开一部分页面，这部分页面就是一个很好的 tile。

其实最好的候选是一段自包含的代码，具有单一和连贯的目的，再次说明，这有点像一个 Java 方法。

如果表现代码并没有这样的注解，那么最好是在分离它们为 tiles 之前，先加上一些注解。如果一个页面片段看起来像是一个 tile，除非每个页面都在 tile 上输出不同的内容，否则也不要绝望。Tiles 也可以传输字符串常量到一个 tile 中以便其它的标签能够重用它们。这时，只需要加入一个标记，象${subtitle}，它就可以被传入的字符串替换。

命名候选者

在开始时确定一个很好的命名方案是很有帮助的。我们对某个组件的命名应该能够明确其目的。我们也需要对各个区域，通用布局以及系统中的每个页面进行命名。

表 11.3 列出了这些东西的Tiles 术语：tile, layout, 和 definition。

Tile 是一些从现有页面中分离出来的连续的文本和标记的片断。Tile 可以表现静态内容或标记，或者兼而有之。

大部分tile 表现导航性控件和其它通用部件，并可以在页面之间重用。通用部件可能会包括一个在页面中某处显示logo的表格，或者一些个可以用在多个表单中的按钮，比如保存/取消什么的。如果某个样式已经和某个部件关联，可以考虑在样式之后命名tile。

表格 11.3 Tiles 术语

项目	说明
Tile	一个包含HTML 标记或者JSP 代码的页面碎片.
Layou	一个描述如何在页面中定位tiles的JSP文件。
Definition	一个代表一个特殊页面的JavaBean。Definition 由一个layout 加上许多tiles ，以及其他运行时产生独特页面的选项所组成。Definition 可以表现为一个JSP 或者包含在XML 配置文件中。

其他的 tile 将会包含在单个页面中使用的内容。这些 tile 通常可以在页面中央，被其它提供菜单条或者其它 HTML 装饰的 Tile 所包围。

你可以使用和命名 HTML 或者 JSP 同样的规范来命名 tile。最好是将共享的 tile 和那些专门为特定页面设计的内容分开。比如，有一个比较容易的方法是在你的页面文件夹之后再创建一个文件夹存放 tile.

```
/pages
./article/Form.jsp
./channel/Channels.jsp
/tiles
./header.jsp
./message.jsp
```

如果某个文件位于 tiles 文件夹，就意味着可能该文件会被用在多个 Definition 中。如果某个文件位于 pages 文件夹，意味着其包含着独特的内容，并且一般不希望被共享。

布局 (layout) 是 tiles 的容器，描述每个组件如何出现在页面上。Layout 可以在 Definition 间重用。为了集中管理，最好也创建一个 layout 文件夹在 tiles 文件夹之下:

```
/tiles
  ./header.jsp
  ./message.jsp
  ./layouts
    ./Base.jsp
    ./Remote.jsp
```

Definition 存储为一个 XML 文档的元素。每个 Definition 元素都需要其自己的名称。实践中，Definition 一般都和 ActionForward 共享命名空间。命名方案应该确保 ActionForward 和 Definition 的名称不会冲突。最好将 Definition 分组存放在目录/子目录形式的结构中，所以还需要一个命名分隔符。一个通常的约定是使用点号(.)来分隔 Definition 名称，而在 ActionForward 中使用斜杠(/)。为了确保没有和文件系统名称冲突，可以在通常需要前导斜杠的地方使用前导点号:

```
<definition name=".account.logon">
 . ..
</definition>
```

当然，也可以使用其他的命名规范和约定。比如你可以使用@ 号而不是点号，或者每个 Definition 前导以 TILES: 然后使用斜杠。因为这些都是逻辑引用，所以可以使用任何有效的 URI。

一旦你已经有了一个你需要些什么 tile, layout, 和 Definition 的大致概念，以及你将如何调用它们，下一步就是如何将它们分离出来，并重构页面。

11.5.4. 使用<tiles:insert>重构页面

就像其它重构一样，开始时最好放慢些，待一个小修改完全完成后再进行其它的。一旦过程可以继续，你可以进行更大的修改。

在大多数情况下，我们的目标是简化将页面简化为可以在 Tile 配置文件中声明并且可以在 ActionForward 中调用的 Definition 之中。这个方法节省了创建插入到 tiles 的一系些列文件。但是刚开始时，最简单的办法还是使用<tiles:insert> 标签构建页面。

在迈出右腿之前，请记住:

- 选择一个好的起始页面
- 仔细地从该页面中分离出 tile
- 按上述方法开发一系列好的东西出来。

选择好的起始页面

最好从一个简单页面开始，分离出公共组件，并且将它们其中之一插入到原来的页面中。你

的应用欢迎页面或者登录页面可能是个比较好的选择。这些都是相对简单并且有很好的可重用内容和定制内容的页面。如何没包含太多的装饰组件，一个内部组件也是一个很好的选择。清单 11.13 显示的是一个来自于 Artimus 示例应用 [ASF, Artimus]的内部页面，并在迁移到 Tiles 之前的样子。

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<%@ taglib uri="/tags/request" prefix="req" %>
<!-- HEADER -->
<HTML>
  <HEAD>
    <html:base/>
    <LINK rel="stylesheet" type="text/css" href="<html:rewrite
         forward='baseStyle'/>">
    <TITLE>Artimus - Article</TITLE>
  </HEAD>
  <BODY>
    <Table class="outer">
      <TR>
        <TD>
          <Table class="inner">
            <!-- MESSAGE -->
            <TR>
              <TD class="message" colspan="3"
                  width="100%"><html:errors/></TD>
            </TR>
            <TR>
              <TD class="heading" colspan="3">
                <H2><bean:write name="articleForm"
                           property="title"/>
                </H2>
              </TD>
            </TR>
            <TR>
              <TD class="author" colspan="3">by
                   <bean:write name="articleForm"
```

```
                    property="creator"/>
          </TD>
        </TR>
        <TR>
          <TD class="article" colspan="3">
            <bean:write name="articleForm" property="content"
                 filter="false"/></TD>
          </TR>
          <%-- CONTRIBUTOR PANEL --%>
          <req:isUserInRole role="contributor">
            <TR>
              <TD colspan="3"><HR /></TD>
            </TR>
            <TR>
            <%-- DELETE --%>
            <logic:equal name="articleForm"
                         property="marked" value="0">
              <html:form action="/admin/article/Delete">
                <TD class="input">
                    <html:submit >DELETE</html:submit>
                </TD>
                <html:hidden property="article"/>
              </html:form>
            </logic:equal>
            <%-- RESTORE --%>
            <logic:equal name="articleForm"
                    property="marked" value="1">
              <html:form action="/admin/article/Restore">
                <TD class="input">
                  <html:submit>RESTORE</html:submit>
                </TD>
                <html:hidden property="article"/>
              </html:form>
            </logic:equal>
            <html:form action="/admin/article/Edit">
              <TD class="button" colspan="2">
```

```
                <html:hidden property="article"/>
                <html:submit>EDIT</html:submit>
                <html:cancel>CANCEL</html:cancel>
              </TD>
            </html:form>
          </TR>
          </req:isUserrInRole>
          <!-- NAVBAR -->
        </Table>
      </TD>
    </TR>
    <TR>
      <TD class="navbar">
        <html:link forward="done">DONE</html:link>
      </TD>
    </TR>
  </Table>
</BODY>
</HTML>
```

一旦选择一个开始页面。你就可以使用它分离出每一个逻辑块到一个 tile 中并插入回去。每次分离之后，都要测试一下是否该页面是否能够被正确渲染加工。清单 11.14 显示了一个分离到 tile 中的碎片。

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<HTML>
  <HEAD>
    <html:base/>
    <LINK rel="stylesheet" type="text/css" href="<html:rewrite
         forward='baseStyle'/>">
    <TITLE>Artimus - View Article</TITLE>
  </HEAD>
  <BODY onload="document.forms[0].elements[0].focus();">
    <!-- OUTER Table -->
    <Table class="outer">
      <TR>
        <TD align="center">
```

```
        <!-- INNER Table -->
        <Table class="inner">
          <TR>
            <TD class="navbar" colspan="3">View Article</TD>
          </TR>
```

清单 11.15 显示了 View.jsp 如何能够在分离部分内容出去之后重新包含 Header tile。

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<%@ taglib uri="/tags/request" prefix="req" %>
<!-- HEAD -->
<tiles:insert page="/tiles/header.jsp"/>
<!-- MESSAGE -->
<TR>
  <TD class="message" colspan="3" width="100%"><html:errors/></TD>
</TR>
<!-- ... -->
</HTML>
```

完成你的第一个 tile 分离出来并插入回去之后，请确保你在开始下一个之前打开浏览器测试一下。

当所有的流程都完成后，该页面将会包含一系列插入的 tiles，如清单 11.16 所示。

```
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<tiles:insert page="/tiles/header.jsp"/>
<tiles:insert page="/tiles/message.jsp"/>
<tiles:insert page="/tiles/view.jsp"/>
<tiles:insert page="/tiles/navbar.jsp"/>
```

如果位于 tile 的某些文本需要定制，那么对于页面 title，你可以使用<tiles:put> 标签来将定制值传递给 tile。<tiles: getAsString> 标签会在页面渲染的时候显示输出它。清单 11.17 显示了发送文本给 tile；清单 11.18 显示了如何重新输出动态文本。

```
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<tiles:insert page="/tiles/header.jsp">
  <tiles:put name="title" value ="Artimus - View Article"/>
  <tiles:put name="subtitle" value ="View Article"/>
```

```
</tiles:insert>
<tiles:insert page="/tiles/message.jsp"/>
<tiles:insert page="/tiles/view.jsp"/>
<tiles:insert page="/tiles/navbar.jsp"/>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<HTML>
  <HEAD>
    <html:base/>
    <LINK rel="stylesheet" type="text/css" href="<html:rewrite
         forward='baseStyle'/>">
    <TITLE><tiles:getAsString name="title"/></TITLE>
  </HEAD>
  <BODY onload="document.forms[0].elements[0].focus();">
    <!-- OUTER Table -->
    <Table class="outer">
      <TR>
        <TD align="center">
        <!-- INNER Table -->
          <Table class="inner">
            <TR>
              <TD class="navbar" colspan="3">
                <tiles:getAsString name="subtitle"/>
              </TD>
            </TR>
```

如果你喜欢，Struts <bean:write> 或者 <bean:message> 标签也可以用来替代 <tiles:getAsString>。这样，属性会被存放在标准的上下文，其他标准的标签也会一起正常工作。

分离tile

重构的主要工作是决定页面的哪一个部分是哪一个 tile 的一部分，将其分离出来作为一个单独的 tile 文件然后再插入回去。

下面是一个步骤检查清单:

- 选择并剪切一个块
- 打开一个新文件
- 粘贴代码块

- 存储并命名为一个 JSP (或者 HTML 文件)。
- 插入标签库导入语句
- 关闭新文件
- 放置 <tile:insert page="/path/to/new/page"/> 标签到片断原来的地方

分离实践

下面是一些关于分离过程的告诫和实践提示。因为这可能会成为一个例行程序，重要的是明白实践经验和从过去的错误中学习:

- 所有 tile 使用的定制标签都要导入到 tile.
- 所有定制标签元素的开始和结尾必须位于同一个 tile.
- 避免传递 HTML 元素到另一个 tile.
- 根据需要使用标记
- 考虑到折衷问题
- 使用一些技巧

Tile 使用的所有定制标签都必须导入到 tile 中。 Tile 可以继承 HTML 资产，象样式表什么的。但它不能继承对 JSP 资产的引用，如标签库。web 浏览器在渲染页面的时候才使用样式表，但是因为每一个 tile 实际上是一个独立的 JSP servlet，它需要引用其自己的 JSP 资源引用。

所有的定制标签元素必须在同一个 tile 中开始和结束。 如果你在你的文件中使用<html:html>标签， 将该元素的标签放置在 layout 的开头和结尾。这一限制不适用于 HTML 元素。例如，你可以在 header tile 中打开<BODY> 元素，然后在 footer tile 中关闭它。但是定制标签是在编译时生效的。所以， JSP 标签必须在同一个文件中打开和关闭。

避免 HTML 元素跨越多个 tile。 实际应用中，你可能决定一个 tile 打开一个元素，比如一个表格；第二个 tile 提供内容，比如表中的行；第 3 个 tile 关闭这个元素。如果这种模式有用，你可以使用它。但是最好还是在一个 tile 中打开和关闭元素。 这样可以更容易发现标记错误。即便中间的 tile 仅仅提供表格的行，它也可以这样使用完整的<TR>...</TR> 标记。

按契约进行标记。 当你决定使用诸如 Tiles 的东西时，你也决定了象处理代码一样处理标记。因此，所有通常的模式和范例都适用，比如断言和编程等等。在设计你的 tiles 时，可以考虑每个 tile 的前置条件和后置条件，就像你使用类中的方法一样。尽可以写下每个 tile 的前置条件，就像针对一个方法一样。一个好办法是在 JSP 中使用标准的 JavaDoc 约定，可以避免新的负担:

```
<%--
/**
* Global page header, without automatic form select.
* See headerForm.jsp for version with form select.
* Imports global style sheet.
* Opens HEAD, BODY, and Talble. Another tile must close these.
* @author Ted Husted
```

```
* @license ASF 1.0
*/
--%>
```

对于 HTML tiles，你可以使用标准的 HTML 注释。和 JSP 注释不同，HTML 注释在页面的源代码中是可见的。你可能希望在使用 HTML 注释时，尽量小心而简短。

要考虑到折衷。分离页面到 tiles 的工作很像是数据库的规格化。

你可以完全剥离所有的冗余信息，但是如果这样做，某些部分会变得更加小，甚至出现维护和性能问题。实践中，你可能对 header 或者 footer 文件有两三个变体。但是如果标记改变，与修改 30 甚至 300 个文件相比，修改 3 个文件仍然是容易得多。因为对于很多编程任务来说，通用的折衷代价是适用的。

使用一些技巧。如果你同时需要样式表和其它标记的改变，完全可以创建一个骨架文件并使用查找替换来完成这个工作。这会更加容易忘记你可以保存它们多少次。

11.5.5. 分离<tiles:insert> 标签到Definition中

当你完成 11.5.3 的过程，并且确定它会正常工作之后，你可以继续将<tiles:insert> 标签分离到一个 XML Definition 中来完成这个工作。这项任务有四个步骤：

- 将页面移到 layouts 文件夹
- 重命名 body tile
- 将 insert 标签转换为 layout 和 Definition
- 更新 ActionForward

将页面移到 layouts 文件夹

将重构的页面移动到你存储 layout tiles 的地方，比如 /tiles/layouts。此时，重构的页面可能具有一系列<tile:insert> 标签，而页面的原始内容已经被裁减到其它 tile 中了。

重命名body tile

其中一个分离的 tile 可能会标识的是原始内容的 body 页面。如果是这样，请开始将它重新命名为你刚移动的原始页面。这意味着原始页面的核心内容仍然在其原来的地方。这有助于最小化变更。如果需要编辑页面的内容，内容应该仍然在其原来的地方。如果你这样做了，请在移动和重命名文件之后更新<tiles:insert> 标签。对于清单 11.16, 我们可以将

```
<tiles:insert page="/tiles/view.jsp"/>
```

修改为

```
<tiles:insert page=”/pages/view.jsp"/>
```

将insert标签转换为layout 和 Definition

对每一个 insert 标签添加一个名称属性。这通常会匹配于 JSP 的名称。异常情况可能是表现页面原始内容的 tile 。这也通常具有一个更加通用的名称，比如内容。

将<tiles:insert> 语句从重构页面拷贝到本节开头摄制的 tiles.xml 配置文件，并将它们置于<definition> 元素之中。如果你使用了一些<tiles:put> 元素，你可以将它们放在元素开头：

```
<definition>
  <tiles:insert put="title" value ="Artimus - View Article"/>
  <tiles:insert put="subtitle" value ="View Article"/>
  <tiles:insert name="header" page="/tiles/header.jsp"/>
  <tiles:insert name="message" page="/tiles/message.jsp"/>
  <tiles:insert name="content" page="/pages/view.jsp"/>
  <tiles:insert name="navbar" page="/tiles/navbar.jsp"/>
</definition>
```

然后，将 <tiles:insert> 标签重命名为 <tiles:put> 元素，并且 page 属性为值属性。对于<definition> 元素，添加 name 和 path 属性。Path 属性应该是你的 layout 页面 (见步骤 1)。Name 属性对应的是你的原始页面的名称，但是将斜杠替换为点号。清单 11.19 显示了我们的完整的 tiles.xml 文件.

```
<!DOCTYPE tiles-definitions PUBLIC
"-//Apache Software Foundation//DTD Tiles Configuration//EN"
"http://jakarta.apache.org/struts/dtds/tiles-config.dtd">
<tiles-definitions>
  <definition name=".article.view"
              path="/pages/tiles/layouts/Base.jsp>
    <tiles:put name="title" value ="Artimus - View Article"/>
    <tiles:put name="subtitle" value="View Article"/>
    <tiles:put name="header" value="/tiles/header.jsp"/>
    <tiles:put name="message" value="/tiles/message.jsp"/>
    <tiles:put name="content" value="/pages/view.jsp"/>
    <tiles:put name="navbar" value="/tiles/navbar.jsp"/>
  </definition>
</tiles-definitions>
```

现在，在你的 layout 页面中，将<tiles:insert> 标签改为<tiles:get> 标签，并删除 page 属性(因为现在它是 Definition 的一部分)。某些<tiles:put> 标签可以改为<tiles:useAttribute> 标签。保留 name 属性，但是删除 value 属性(因为 value 也是 Definition 的一部分了)。

清单 11.20 是一个完整的 layout JSP.

```
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<tiles:useAttribute name="title"/>
<tiles:useAttribute name="subtitle"/>
```

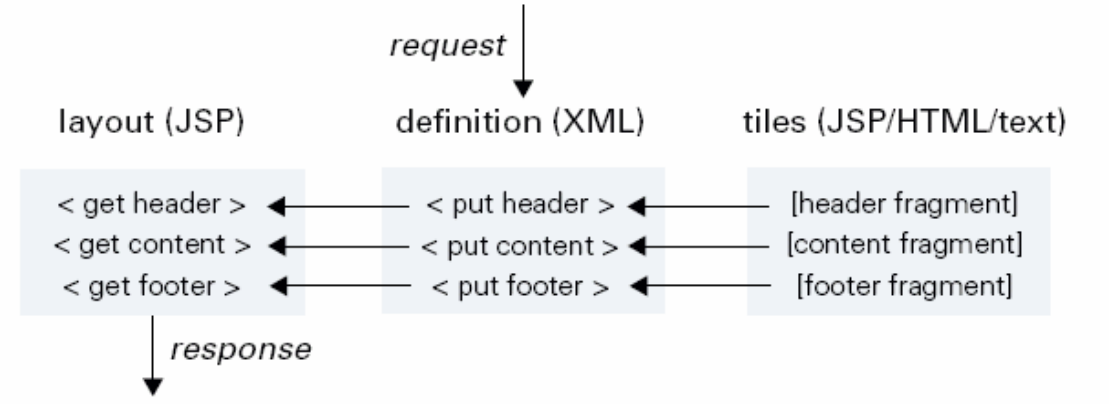
```
<tiles:get name="header">
<tiles:get name="message"/>
<tiles:get name="content"/>
<tiles:get name="navbar"/>
```

更新ActionForward

最后，将引用到原来 JSP 的 ActionForward 替换为引用到 Tiles Definition:

```
<action
  path="/article/View"
  type="org.apache.scaffold.struts.ProcessAction"
  parameter="org.apache.artimus.article.FindByArticle"
  name="articleForm"
  scope="request"
  validate="false">
  <forward
    name="success"
    path=".article.View"/>
</action>
```

如图 11.6 所示，Tiles ActionServlet 将会解释这些引用，并对 Layout 页面使用正确的 Definition。



如果你先前是直接转发到JSP,你可以使用标准的Scaffold SuccessAction 将控制路由到控制器，所以Definition 能够被正确选择和用来加工你的layout页面。你可以象使用JSP的系统路径一样来使用Definition 。如果有很多的引用，你可以在编辑器中使用查找替换功能。

在运行时，Tiles ActionServlet 将解释ActionForward 并且检查其路径是否符合tiles配置中的 Definitions 。如果找到匹配，它就会在相应中包含Definition 中的每个tile。容器然后正常处理每个包含的tile。HTML tiles 是由容器的 HTML 服务处理的， JSP tiles 则由容器的 JSP 服务处理。

如果 ActionForward 路径不是一个Definition的名称，ActionServlet 将其处理为一个正常的URI，象往常一样处理。

NOTE

如果你的重构页面和原来显示的不一样，首先请确保在tiles导入了所有必需的标签库. 如果必要的标签库没有包含，标签将不能被渲染，并且会被浏览器所忽略 (你可以在HTML页面代码中看到)。如果这不是问题，请重新创建一个新页面，然后通过手动来改正这个问题。大多数情况下，你会发现都是s因为不正确的打开或者关闭了某个元素所致。

另一个API约定是检测ActionMapping的 input 属性的Path值。这应该也指向Definition的名称而不是物理的JSP页面。

为了测试你的修改，请确保重新载入应用，以便当前的 Struts 和 Tiles 配置装入内存。万车这些后，你就可以测试它们了。

一旦你你将页面通过<tiles:insert> 重构完成，并将其转换为 Definition，你可能希望将其他页面直接转换为 Definition。为了这样，你可以

- 拷贝一个已有的 Definition ，并使用相同的 layout，然后取一个新的名字；
- 剪切并存储正在重构的页面的代码片断；
- 修改新的 Definition 以引用刚刚存储和转换的代码片断；
- 测试。重复。

NOTE

当你第一次转换一些页面为 Tiles，在页面第一次渲染之前可能会多花一点时间。这是因为使用 JSP 作为新创建的 tiles，以及需要一些额外的内容。一旦每个 Tiles 的 JSP 创建完成，那么到下次被修改之前，他都不需要重新创建，一切将归于正常。

11.5.6. 规格化基本布局

我们回头看看代码清单 11.20, 其中的 layout 文件实际上是一系列<tiles:insert> 标签。如果你喜欢，你也在可以在你的 Layout 的代码中使用常规的 HTML 和 JSP 代码。这是放置最顶层标签的好地方，比如<HTML> 或者 <html:html>, 以便它们不会被放入其它其他 tiles 之中, 而这些 Tiles 可能需要作其它事情。

清单 11.21 显示了修订后的布局页面，它从 header 和导航条 tiles 中分离出顶层元素并放入到基本布局中。我们也将它的名称从 Base.jsp 变更为 Article.jsp 以指示它的角色为文章的布局。而应用的其他页面可能需要使用不同的布局。

```
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<html:html>
  <HEAD>
    <html:base/>
```

```
    <LINK rel="stylesheet" type="text/css" href="<html:rewrite
        forward='baseStyle'/>">
    <TITLE>Artimus - <bean:write name="title"/></TITLE>
  </HEAD>
  <tiles:UserAttribute name="title"/>
  <tiles:get name="header"/>
  <tiles:get name="message"/>
  <tiles:get name="content"/>
  <tiles:get name="navbar"/>
  </BODY>
</html:html>
```

11.5.7. 将Definition提炼到基本和扩展类之中

在你将你的页面转换为布局和 Definition 的时候，最后可能很容易成为这样：:

```
<definition name=".article.View" path="/tiles/layouts/Article.jsp">
  <put name="title" value="View Article" />
  <put name="header" value="/tiles/header.jsp" />
  <put name="messages" value="/tiles/messages.jsp" />
  <put name="content" value="/pages/articles/view.jsp" />
  <put name="navbar" value="/tiles/navbar.jsp" />
</definition>
<definition name=".article.View" path="/tiles/layouts/Article.jsp">
  <put name="title" value="Search Result" />
  <put name="header" value="/tiles/header.jsp" />
  <put name="messages" value="/tiles/messages.jsp" />
  <put name="content" value="/pages/articles/result.jsp" />
  <put name="navbar" value="/tiles/navbar.jsp" />
</definition>
```

如果仔细看，会发现第 1 个和第 3 个不同，但是其他的都是一样。一个更好的编写办法是创建一个基础 Definition。

Tiles Definition 支持扩展属性属性，这使得 Definition 可以继承属性而只需要重载改变的那些。这里，我们扩展了.article.Base 并且重载了 title 和内容:

```
<definition name=".article.Base" path="/tiles/layouts/Article.jsp">
  <put name="title" value="${title}"/>
```

```
  <put name="header" value="/tiles/header.jsp"/>
  <put name="message" value="/tiles/message.jsp"/>
  <put name="content" value="${content}"/>
  <put name="navbar" value="/tiles/navbar.jsp"/>
</definition>
<definition name=".article.View" extends=".article.Base">
  <put name="title" value="View Article"/>
  <put name="content" value="/pages/article/view.jsp"/>
</definition>
<definition name=".article.Result" extends=".article.Base">
  <put name="title" value ="Article Search Result"/>
  <put name="content" value="/pages/article/result.jsp"/>
</definition>
```

因为使用了 base Definition，现在我们仅需要在每个子定义中提供两行代码。其他设定都直接使用不需要再来标明了。

如果有一些属性需要在整个站点中使用，你可以将其放置在 base Definition 中，而只需要扩展其他的属性即可。那么，如果这些基础属性需要改变，你只需要在一个地方就可以修改了。

根据约定，我们为第一条和第 3 条项目的值（标题和内容）加入标记以指示这里是一个子定义需要重载的扩展点。如果基础定义可以直接使用，那么这些标记将仅仅输出字面的样子。${}标记对 Tiles 来说没有特殊的含义。

这里要说的另一个约定是在 layout JSP 中使用大写字母而在 tile JSP 中使用小写字母。这样来指示 layout 页面可以被直接调用，因为它们可以是完全正式的 JSP 类。而 tile 页面就像是被 layout JSP 调用的方法，所以使用和方法一样的命名约定。但这仅仅是一个约定；其它一致性的命名方案也是可以使用的。

11.5.8. 开发过程

在首次的几个页面完成后，你应该能够通过以下方法来开发程序：

- 创建一个新的 Definition (在 tag.xml 中), 通常可以拷贝相似的代码
- 用现有的页面、页面 title, 以及一些其他定制信息的路径更新 Definition
- 打开现有的页面
- 删除顶部和底部内容，保留核心内容和 tag 导入语句
- 检查和修改核心内容，以确保其标记符合与在定义中之前和之后的的 tiles 的 API 合约。一个 tile 可能需要打开一个元素，比如<table>, 而另一个 tile 则可能需要关闭它。删除一些不需要的标签导入语句。当然，也可以加上一些注释块
- 更新 Struts 配置 (struts-config.xml) 以引用新的定义，包括某些 input 属性.

- 重新装入 tag 配置 和 Struts 配置.
- 测试页面.
- 反复直至满意

首先，你将可能会从出现在你的应用流中的页面开始。一旦你完成这个过程，就很容易的处理页面树，并且依次来重构它们。这将确保你不会遗留某些东西。这样也可以发现先前的工作中造成的一些废旧页面。

定义

对于某个方法的方法体的一个通用看法是，它是方法和其调用者之间的协约。调用者同意提供特定的参数而方法则同意提供基于特定参数的结果。因为 API实际上是一系列方法体（签名）的集合，将组件间的交互看作是一个榜定的协议通常称之为是API 合约.

11.5.9. 管理迁移

迁移一个应用到 Tiles 不是一件复杂的事情，但是也不要小看它。对项目要确保有足够的时间规划。开始时的一些页面可能会花较多的时间，但是一旦建立了模式，其他的每个页面只是花费你几分钟而已。

如果 你也需要给你的站点一个新的外观，你也要先将其转换到 Tiles 之后才来进行，免得你同时进行两项新的任务。一旦布局迁移到 Tiles,站点外观的新设计将十分的快速和容易。

开始迁移的最好的时间是在你知道将要对站点进行改版，而手头有没有新的设计的时候。如果你在新的改版设计定下来的时候已经将站点迁移到 Tiles 上，那么改版工作将会非常顺利。不推荐同时做这两个工作，特别是第一次迁移。

那么什么是迁移的底线？工作量各不相同，但是对于一个有 25 个表现页面，140,000 kb 标记代码的小应用来说，可能会迁移到 55 个 tiles，大约 120,000 kb 标记代码，大约有 15% 的冗余标记被删除。

而且，应用的新页面可以很快速容易的创建并且与现有的页面保持一致。要改变整个站点的布局，你只需要改变整体布局或者仅仅修改少数几个 tiles 而不是整个站点的页面。

11.6. 小结

动态模版系统，比如 Tiles 框架，可以将一些常见的编程模式引入到 Web 应用的表现层之中。它们将散碎的 HTML 标记和 JSP 命令变成易于管理的组件。我们可以通过调用单独的小块，或者说是 tiles 来组装页面，就像我们调用 Java 方法来执行一个大的处理过程一样。tile 封装是一个标记块，非常象封装了代码块的方法。

Web 应用的页面应该具有一致的外观和感觉，或者布局，这使得用户很容易在站点中导航。在我们使用 Tiles 组装页面时，我们从定义每一个 tiles 的位置和给它们一个逻辑名城的基础布局（base layout）开始。Tiles 的路径可以在运行时动态传递。一个典型的页面可以使用 5 个或者 6 个 tiles，每一个页面间通常只有一两个 tiles 的差异。

在 Tiles 中, 一个完整的页面，包括布局和对它们的 tiles 的路径，可以表现为一个称为是定义（Definition）的对象。为了组装一个特定页面，我们可以简单的引用它的 Tiles Definition。就像其他 Struts 框架组件，Definitions 可以通过一个 XML 文档来进行配置，在启动予以装入。

Struts 框架使用 ActionForwards 来封装对系统资源的路径，包括表现页面和 Action 类。框架的控制器，ActionServlet，使用 ActionForwards 来路由控制。其他组件通过名称引用 ActionForward，并依赖控制器来调用 forward 的路径指明的资源。

对控制器的标准扩展允许 Definitions 被用作是 ActionForward 的路径。当一个 Definition 是某个 ActionForward 的目标，控制器就会在响应中包含指明的页面碎片。而每个页面碎片的标准服务就会完成相应的工作。

当一个应用被迁移到 Struts 时，其结果是系统资源路径一般都封装到了 Struts 配置之中。表现页面然后就可以引用到其他资源，使用的是逻辑名称而不是实际的路径或者 URI。这使得你可以关注于页面需要做的事情而不是如何做。

在同一种情况下，当一个应用被迁移到 Tiles 中，配置也封装了系统路径。Struts 然后就可以通过名称引用页面定义，而将组装页面的特殊工作留给 Tiles 框架来处理。

Tiles 完全可以使用在具有成百上千个页面的大型系统中。

就像将流程分解为单独的方法，使用 Tiles 重构一个应用可能会创建更多组件，但是每个组件都会更小并且更易于维护。更重要的是，每个单独的 tiles 都可以被重用，避免在多个页面中进行相同的修改。随着应用增长，消除荣誉的需要变得更加重要。

本章关注的是向你提供关于使你的应用使用或者迁移到 Tiles 的详细信息。但是它绝不是涉及到 Tiles 框架可能会遇到的所有问题。第 15 章的 Artimus 示例应用就是基于 Tiles 并且展示了本章所讲述的一些最佳实践。

在下一章，我们将探索另一个可选组件， Struts Validator。

12. 用户输入校验

David Winterfeldt 和Ted Husted合著

本章内容

- 理解校验数据的需要
- 配置和使用 Commons 校验器
- 使用多页面和本地化校验
- 编写自己的校验器

Us: Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.

—Rich Cook

Them: I never know how much of what I say is true.

—Bette Midler

12.1. 看到时我就认识它

大多数应用都需要从用户处收集数据。这些输入可能来自于非常灵活的文本字段或者GUI元素，比如菜单列表，单选按钮，或者复选框。在实践中，用户的输入并不是总是有意义的。一些菜单选项可能是互斥的。电话号码可能还缺少一位数字。在数字字段中可能输入了字母。而字母字段中又输入了数字。这是因为数据输入表单并不清楚，或者因为用户并没有注意。但是，在其发生的很多时候，是具有很大的规律性的。

没有什么比应用输出了莫名其妙的垃圾结果更能令用户丧气，即使是对那些输入了垃圾数据的用户而言。一个谨慎的应用应该细察所有的输入，防止所有可预知的错误，以及保护用户自己避免犯错。而且，毕竟，如果它们确实发生了，我们也还要修正它的。

12.1.1. 不能拒绝的输入

在一个常规应用中，数据输入控件可以简单地拒绝不正确的输入，但是要它们是以模态的方式进行却显得很奢侈。

定义	当一个用户接口元素收集一个应用的所有输入时，我们称它为模态的。其它元素在它消失之前不能被访问。为了继续，用户必须完成该对话框，或者关闭应用。大多数用户接口都是非模态的。

天生就是非模态的Web应用就少有选择。缺省地，浏览器显示的HTML元素将接受输入到它们之中的任何东西。一个元素没有能力知道在表单输入了什么。我们可以用JavaScript来做一些技巧性处理，但是并没有谁能保证用户将浏览器的JavaScript支持选项打开。

当然，我们可以在数据达到业务层时校验它们。(关于应用层和业务对象，参见第2章) 许多业务逻辑对象确实具有一些内置的校验，但是大多数业务对象在接收数据之前并不检查它们。业务层方法都趋向于信任数据的创建者。它们天真地期望对象能够被提供以合理的数据，而自己只是做自己被赋予的任务而已。即使业务对象再悲观一些，通常它们能做的所有事情就是抛出一个异常。对于业务对象来说，显示一个对话框让用户纠正输入错误的数据并不是它们的职责。

当然，在上下文中校验数据却是业务对象的职责。让我们看看一些例子，例如，输入的用户名称和密码是否匹配。但是仍然有一些有目的校验规则可以在数据被传输到业务层之前对其进行校验。在一个分布式应用中，业务对象可能会驻留在远程机器上，这样创建往复的解决输入错误问题的方式的代价就显得比较高昂。

12.1.2. Web层校验

在现实生活中，通常由web应用框架来提供有目的的校验方法，从而窄化模型和视图之前的距离。在一个非模态的分布式环境中，我们需要能够进行下列事项的校验方法：

- 要求特定的字段具有值
- 确认输入的值是否是期望的格式或者出于某个范围之内
- 一次性检查整个表单并返回一个消息列表
- 比较字段之间的值
- 返回原始输入以便纠正
- 需要时，显示本地化信息
- 如果JavaScript 被禁止，执行服务器端校验

还有两个其它的重要的特点是，松散耦合和可选的客户端校验。

松散耦合

在实际情况下，输入需要被控制器校验，但是业务校验是紧密结合到业务层的。这意味着校验规则应该与标记或者Java代码中单独分开存储，以便它们能够在不改变其它源代码的情况下被重新评审和修改。保证校验规则的松散耦合使得保持校验和业务需求的同步更加容易。

定义	耦合（*coupling* ）程度意味着两个组件之间的连接强度。耦合是内聚的互补。内聚（cohesion）描述的是一个组件的内部内容彼此之间的相关强度。我们的目标是要创建具有内部完整性(强内聚)的和与其他组件具有更小的、直接的，可见的，和灵活的关系(松散耦合)的组件。[McConnell]

某些校验规则可能也需要本地化。当添加了对一个新的场所的支持时，我们应该能够象更新资源束一样容易地更新校验规则。

虽然校验规则能够为表现层提供方便，但是要认识到它们其实是属于业务层。

校验规则也不应该和表现源代码相混合。

客户端校验

客户端校验天生是不安全的。可以很容易地欺骗一个正在提交的web 页面并且绕过原始页面中的任何脚本。尽管我们不能完全依赖于客户端 JavaScript 校验，但它还是有用的。 其优点是对用户的立即反馈，避免了一次服务器往返，节省了时间和带宽。所以，理想情况是，基于同一套规则产生JavaScript 和 服务器端校验。当JavaScript 激活时，输入可以在提交之前在得到客户端校验。如果没有激活，输入仍然在服务器端进行校验以确保没有任何错误。

12.1.3. 校验器的地位

使用Jakarta Commons Validator [ASF, Validator] 可以带来以下结果：

- 校验器是一个符合这些需求---或者更多，的框架组件
- 校验器通过一个能够产生针对表单字段的校验规则的XML文件进行配置
- 规则也是通过XML配置的校验器定义的
- 针对基本类型，比如日期和整数的校验器，已经内置提供。如果需要，你可以创建你自己的校验器
- 正则表达式可以用于基于模式的校验，比如邮政编码和电话号码

- 支持多页面和本地化校验，所以你可以用任何语言编写wizard。

定义

正则表达式（*regular expression*） *是一个用来使字符创匹配某个模式的公司。正则表达式经常用于许多* Unix 命令行和编程工具中。关于正则表达式的详细信息，参见 Stephen Ramsay 的 “Using Regular Expressions” web 页面。[Ramsay]

在你的应用中使用Jakarta Commons Validator有诸多好处：

- 优化资源的使用：当允许时提供 JavaScript 校验，并且服务器端校验也是可以保证的；
- 单一维护点：客户端和服务器端校验都来自同一配置产生；
- 可扩展性：定制校验可以通过正则表达式或者在Java 代码中产生；
- 可维护性：它和应用之间是松散耦合的，并且可以在不修改任何代码的情况下进行维护；
- 本地化：校验可以进在需要时和需要的地方才进行定义；
- 与Struts的集成，默认情况下，校验会共享Struts消息资源束。本地化文本可以集中管理并可重用；
- 服务器端校验的容易部署性：为了使用服务器端 校验，你的Struts ActionForm 可以简单扩展ValidatorForm 或者ValidatorActionForm 类。剩下的都是自动的。
- 客户端校验的容易部署性：为了使用客户端校验，你只需要添加一个JSP 标签就可以产生校验脚本，并且使用该脚本来提交表单；
- 容易配置：校验器使用 XML 文件来进行配置，就象web 应用部署描述符合Struts配置一样。

当然，但是，它也有一些缺点：

- 非模态的客户端校验：产生的JavaScript 是非模态的；直到表单被提交之前它并不参与；
- 依赖性：校验和字段和ActionForm 属性是分离的。页面标记，ActionForm，以及校验器和Struts 配置文件之间必须保持同步；
- 缺乏数据转换和变换：包并未提供数据转换和变换功能。如需要，转换和变换必须自己单独编程。

请注意将Jakarta Commons Validator应用在你的应用中并不是万能灵药。某些校验只可能在服务器断进行。如果失败，错误消息可以根据情况显示与JavaScript 消息不同的内容。接口的不一致会使用户混淆。

定义

数据转换（Convertion）是将一种数据类型转换为另一种类型，比方说从 String 到 Integer。数据变换(Transformation)则是数据内部格式的变化，比如在一个字符串中添加一个标点，或者在其被保存之前去除一个不想要的标点。本地化需要将数据变换成显示格式。

在这一章，我们将向你展示如何在你的应用中最好地使用Commons Validator框架。我们将涉及到校验器的总体设计，并给出一个简单的例子。然后我们会随同经常需要的技术逐个审

视校验器的每一个组件，比如覆写缺省消息，取消校验，使用多页面工作流，校验集合，等等。

要强调的是，有目的的，输入输入校验不是一个综合性的方案。不用访问模型层就可以发现许多种错误。我们可以看看用户名和口令在长度和组成方面是否符合业务需求。但是如果用户名称和口令是有效的，我们就需要连接到业务层并和数据数据通信。但是，通过在实际请求之前检查数据是否可能有效，我们可以消除不必要的数据访问事务，这对谁都有好处。

NOTE	你可能想知道“那我将使用 Struts 框架来构建我的应用还是 Struts 校验器框架呢?”实际上，是两者。 大多数应用都使用多个框架组件来构建，包括某些团队自产的框架。Struts 构建在 Sun 的 Java J2SE 框架之上。同时，Struts 校验器构建在 Struts 框架之上。所以，在你的应用使用多个包中的多个类，它可能也使用了多个框架。关于使用框架架构的信息，参见第 2 章。

第4章我们讨论了Struts 1.1中校验器的设置。这一章是在应用中具体使用校验器的开发人员指南。

12.2. Struts 校验器概述

我们来看看Struts 校验器是如何同其他组件交互来根据同一套校验规则来提供服务器端和客户端校验的。

一旦校验器的各个部分通力合作，你会惊讶校验数据居然是这么的容易。表 12.1 列出了组成Struts 校验器的各个组成部分。

起初， Commons Validator是作为Struts 框架的一个扩展创建的。但是自从它也被用于框架之外后，开发者就将它贡献为Jakarta 的另一个子项目Commons下的一个组件。

表格 12.1 主要的 Struts 校验器组件

组件	说明
校验器	**处理原生和其它通用类型。基本校验器包括 required，mask (匹配正则表达式)，minLength，maxLength，range，native types， date，email，和 creditCard。也可以定义定制 (或者插件) 校验器。**
资源束	**提供(本地化的)标注和消息。默认与 Struts 共享消息资源。**
XML 配置文件	**根据需要定义针对字段的表单集和校验。校验器可以在一个单独的文件中定义。**

JSP 标签	**对给定的表单或 Action 路径产生 JavaScript 校验器 。**
ValidatorForm	**根据 FormBean 的名称自动校验属性（在运行时通过 ActionMapping 参数传到 validate 方法)。必须被扩展才能提供表单之上的期望属性的校验。**
ValidatorActionForm	**基于 action 路径自动校验属性(在运行时通过 ActionMapping 参数传到 validate 方法)。必须被扩展才能提供表单之上的期望属性的校验。**

Struts 分发包中包括校验器包，以及将Commons Validator与 Struts集成的类。这个包，以及其扩展的Commons Validator包，组成了Struts 校验器。在本章后面，我们将Struts 校验器看作是Commons Validator的一个超集。

校验器包实际上是一个多个Validator对象的集合，用Java写成。每一个Validator对象都会把一个有关属性的规则强加于另一个对象之上。在Struts 校验器中，这些对象是ActionForm。校验器具有标准的进入方法，象Struts Action一样，用来在需要时调用校验器。校验器配置文件使你能够将一个或者多个校验器关联到表单中的每一个属性。

在实践中，大多数应用都需要执行大量的公共校验。

有些字段可能必须要求有数据输入。而邮政编码总是具有已知的长度。其它公共字段类型包括数值、日期、信用卡号码等等。

校验器本身具有一些基本的校验器来处理这些公共需要，当然还有其它一些需要。如果你的校验不能被基本校验器或者正则表达式满足，你可以开发你自己的校验器并插入到包中。基本校验器支持其自身附带的基本插件。

你的定制校验器可以做基本校验器能做的任何事情，甚至更多。

你的应用所需使用的校验器，基本的或者定制的，都可以在一个XML配置文件中指定，通常配置文件的名称为 validation.xml。为了更易维护，你可以在一个单独的文件中指定将校验器关联到你的ActionForm的属性的规则，该文件通常名为 validator-ruless.xml。

1.0 vs 1.1

Struts 1.0 中的校验器使用一个单一的 validation.xml 文件来包含校验器定义和表单校验规则。

Struts 1.1 允许你将这些组件分到单独的文件中。在本章，我们将使用 Struts 1.1 所用的文件。如果你使用 Struts 1.0，所有配置元素都可以保持在一个 validat.xml 文件中。

Validator-rules文件有一个`<form>` 元素，通常对应于Struts应用中的`<form-bean>` 元素。`<form>` 元素然后又具有`<field>` 子元素。每一个`<field>` 都可以指定它必须成功通过一个或者多个校验器。如果有一个校验器失败，它便会回送一个指向应用资源中某个消息的关键字，连同一些替换参数。Struts 使用这个关键字和参数来产生一个本地化错误消息。如果还使用了客户端校验，同样的消息可以被显示在一个JavaScript 窗口中，如图12.1所示。

校验文件就是插入你的校验器规则所需的基本或者定制校验器的地方。在这里你可以指定需要使用哪个校验器类，以及可选的客户端 JavaScript 校验设置。当使用客户端校验时，JavaScript 校验必须在表单被提交到应用之前首先通过校验。

除了校验器配置文件中的JavaScript 元素之外，客户端校验的其它配置块就是 `<validator:javascript>` 标签 (Struts 1.0) 或者`<html:javascript>` 标签 (Struts 1.1)。 如果你在JSP中的某处放置了这些标签，它就会将针对校验器的所有有关JavaScript综合到一个可以从一个标准进入方法中调用的 script 。然后你便可以使用`<html:form>`标签的`onsubmit` 属性调用进入方法。如果 JavaScript 校验通过，表单就可以被提交。如果没有，将弹出一个窗口显示本地化的错误消息。

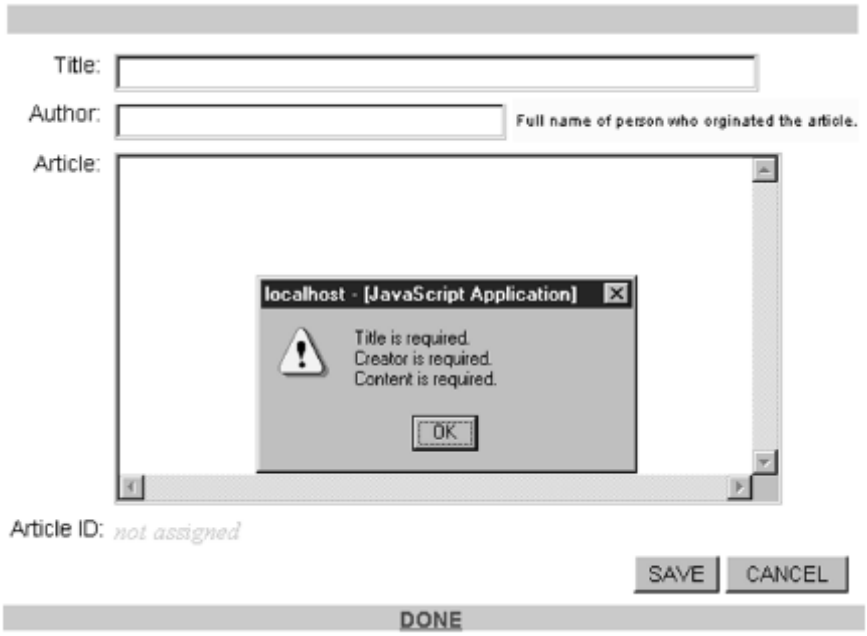


图 12-1 Struts Validator 产生的一个 JavaScript 消息窗口

为了激活服务器端校验，你只需要将你的ActionForm 扩展自Struts Validator包中的一个或者多个基类。`ValidatorForm` (`org.apache.struts.validator.ValidatorForm`) 类就对应于标准的`ActionForm`。而`ValidatorDynaForm` 类 (`org.apache.struts.validator.DynaValidatorForm`) 则对应于DynaActionForm。

缺省情况下，校验器的 `<form>` 元素是使用属性（`Attribute`）名称或者`form bean`名称来匹配ActionForm的。另外，你还可以使用ValidatorActionForm (`org.apache.struts.validator.ValidatorActionForm`) 或者DynaValidatorActionForm (`org.apache.struts.validator.DynaValidatorActionForm`) 来使用ActionMapping 匹配 `<form>` 元素。

在下一节，我们将使用logon应用中的客户端服务器端校验作为背景，向你展示如何将这些东西整合在一起。

1.0 vs 在 1.1 版中， Struts 校验器是包装到 Struts JAR 中并且作为正式发布版的可选组件的。可能会有一些微小的实现细节改变，但是包基本没有改变。如果

1.1	有实现差异，我们会针对每一个发布版列出一个单独的列表。.

12.2.1. Logon 示例

我们采用来自于第3章的logon 应用作为Struts 校验器的背景示例，以帮助我们展示如何将这些组件整合到一起。然后在第12.3节，我们将深入讨论每一个组件。

在设置好包之后(见第4章)，下一步就是确保你所需的校验器可用。它们可以在一个文件中设置，默认下是validator-rules.xml 。我们的例子仅使用了`required` 校验器，虽然大多数应用还可能使用其它多个校验器。

在接下来的小节中，我们将看到为了使用Struts 校验器校验用户名和口令我们需要哪些东西。

validator-rules.xml

Struts Validator 分发包包括针对原生类型和其它公共需要的校验器，比如e-mail 和信用卡号码校验。在实践中，可能使用基本校验器就会满足你的需要。如果不能，你也可以编写你自己的校验器，然后在validator-rules.xml 文件中连同其他基本校验器一起来定义它。(我们说过，在Struts 1.0中，校验器和表单校验都是在一个单独的validation.xml 文件中定义的。)

清单 12.1 和12.2 就展示了Struts 1.1中的`required` 校验器的Java 和XML 源代码。关于如何编写你自己的可插入校验器，参见第12.9节。

清单 12.1 required validato 的 XML 定义源代码(Struts 1.1)

```
<validator name="required"
        <!—①-->
        className="org.apache.struts.util.StrutsValidator"
        method="validateRequired"
        <!—② -->
        methodParams="java.lang.Object,
                      org.apache.commons.validator.validatorAction,
                      org.apache.commons.validator.Field,
                      org.apache.struts.action.ActionErrors,
                      javax.servlet.http.HttpServletRequest"
        msg="errors.required">
        <!—③ -->
        <javascript><![CDATA[
              function validateRequired(form) {
                  var bValid = true;
                  var focusField = null;
                  var i = 0;
                  var fields = new Array();
```

```
                    oRequired = new required();
                    for (x in oRequired) {
                      if ((form[oRequired[x][0]].type == 'text' ||
                          form[oRequired[x][0]].type == 'textarea' ||
                          form[oRequired[x][0]].type == 'select-one' ||
                          form[oRequired[x][0]].type == 'radio' ||
                          form[oRequired[x][0]].type == 'password') &&
                          form[oRequired[x][0]].value == '') {
                          if(i == 0)
                             focusField = form[oRequired[x][0]];
                          fields[i++] = oRequired[x][1];
                          bValid = false;
                        }
                      }
                      If (fields.length 0) {
                           focusField.focus();
                           alert(fields.join('\n'));
                      }
                      Reurn bValid;
                    }
                  ]]>
            </javascript>
</validator>
```

①这一段包括了对服务器端校验器 (见清单12.2)的引用。

② methodParams 仅用在Struts 1.1中。

③客户端 JavaScript 包括在XML 元素中。

如上所示，validator元素在两个部分定义：

- 一个用作服务器端 校验的Java 类和方法
- 用作客户端校验的JavaScript

required 校验器调用的Java 方法如清单 12.2所示：

清单 12.2 validateRequired 方法的 Java 代码(Struts 1.1)

```
public static boolean validateRequired(Object bean,
                                ValidatorAction va, Field field,
                                ActionErrors errors,
                                HttpServletRequest request) {
   String value = null;
   If (isString(bean)) {
     value = (String) bean;
```

```
    } else {
      value = ValidatorUtil.getValueAsString(bean, field.getProperty());
    }
    If (GenericValidator.isBlankOrNull(value)) {
      errors.add(field.getKey(),
      StrutsValidatorUtil.getActionError(request, va, field));
      Return false;
    } else {
      Renturn true;
    }
}
```

应用属性

如果一个校验器失败，它将回送一个消息关键字和替换参数，这些就可以被标准资源束使用。缺省下， Struts 校验器共享应用其他部分所用的资源束。这通常名为ApplicationResources.property，或者干脆就是application.property。当没有指定另外一个消息资源时， Struts 校验器将自动从缺省资源束查找消息。将errors，点号和校验器名称连接起来通常就可以创建一个校验消息的关键字。下面就是我们的required校验器的消息资源条目的样子：

```
errors.required={0} is required.
```

当我们配置一个字段使用required 校验器时，我们也会将字段的标注作为可替换参数传入。然后校验器就可以对所有必需要求的字段重用同一个消息。此外，你也可以定义你自己的消息来有选择地覆盖缺省消息。(见第12.4.3节。)

资源束也将包含你的应用所需的其它消息，以及Struts 校验器所需的特定标签和消息。

下面就是针对 username 和login 校验的消息：

```
# -- logon --
logon.userName.maskmsg=UserName must be letters and numbers, no spaces.
logon.password.maskmsg=Password must be five characters long and contain a
special character or numeral.
```

我们还需要username 和password 字段的标注。然而，如果你的应用已经本地化，这些可能已经提供了：

```
logon.userName.displayName=UserName
logon.password.displayName=Password
```

注意，我们在标注和消息前面加上了logon 作为前缀。给每个表单一个自己的名字空间，在应用增长时可以避免很多冲突。

validator.xml

在validator.xml 文件中定义formset 元素时，需要使用校验器和消息关键字，如清单12.3

所示。

清单 12.3 formset 元素

```
<!—① -->
<formset>
  <!—②-->
  <form name="logonForm">
    <!—③,④ -->
    <field
          property="username"
          depends="required,mask">
      <!—⑤-->
      <msgname key="logon.username.maskmsg"/>
      <!—⑥-->
      <arg0 key="logon.userName.displayName"/>
      <!—⑦,⑧-->
      <var>
        <var-name>mask</var-name>
        <var-value>^[a-zA-Z0-9]*$</var-value>
      </var>
     </field>
    <!—⑨ -->
   <field property="password"
          depends="required,minlength">
      <arg0 key="logon.password.displayName"/>
      <var>
         <var-name>minlength</var-name>
         <var-value>5</var-value>
      </var>
    </field>
  </form>
  <!-- ... -->
</formset>
```

① formset 是一个或者多个form的包装器.

② 每个form 元素都要给定其自身的名称。这应该对应于Struts 配置中的form bean 名称或者 action 的路径。

③ 每个form 元素有一些field 元素组成

④ field 元素通过depends 属性指出要使用哪个校验器。

⑤ 可选的msg 元素允许你指定针对一个校验器的定制消息和用于替换参数的消息关键字

⑥ arg0 元素指定了那些需要它们的消息所用的第一个替换参数

⑦ var 元素用于传递变量属性到校验器中。

⑧ 这里传递了一个正则表达式到mask 校验器。表达式意思是用户名称只可以包含字母和数字。.

⑨ 这里我们设置password 是必须的，并且至少有5个字符的长度。password 长度是一个业务需求，帮助使用户账户更加安全。

password 校验消息使用了缺省 minlength or required 消息，该消息定义于validation-rules.xml (errors.minlength and errors.required)。

JSP标签 / logon.jsp

JavaScript 校验是可选的，但是如果你喜欢在你的应用中使用，也是很容易实现的。清单2.4就展示了修改后使用了JavaScript 的logon.jsp：

清单 12.4 准备进行 JavaScript 校验的 logon.jsp

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<!-- ① -->
<%@ taglib uri="/tags/struts-validator" prefix="validator" %>
<HTML>
  <HEAD>
    <TITLE>Sign in, Please!</TITLE>
  </HEAD>
  <BODY>
  <!-- ② -->
   <html:form action="/logonSubmit" focus="username"
     onsubmit="validateLogonForm(this)">
   <table border="0" width="100%">
     <TR><TH align="right">username:</TH>
       <TD align="left"><html:text property="username"/></TD>
     </TR>
     <TR><th align="right">Password:</TH>
       <TD align="left"><html:password property="password"/></TD>
     </TR>
     <TR>
       <TD align="right"><html:submit property="submit" value="Submit"/></TD>
       <TD align="left"><html:reset/></TD>
     </TR>
   </table>
  </html:form>
```

```
  <!-- ③ -->
  <validator:javascript formname="logonForm"/>
</BODY>
</HTML>
```

① 这里我们导入validator标签库。

② 这一段调用校验脚本，然后提交表单。

③ 这里，我们添加了标签来输出JavaScript到页面中。

validate 方法

为了使用服务器端校验，只需要使form bean 扩展ValidatorForm 而不是ActionForm

```
public final class LogonForm extends
org.apache.struts.validator.action.ValidatorForm {
```

并且删除原有的validate 方法。当控制器调用validate方法时，ValidatorForm 方法就会介入并且按照validation.xml 文件中定义的规则进行校验。

12.3. 基本校验器

如表 12.2 所示，Struts 校验器附带了14 中基本校验器。它们应该能够满足大多数应用的需要了。如果还有新的需要，如你将在第12.9节所见，你可以添加定制的，或者可插入的插件式，校验器。

12.3.1. required 校验器

required 校验器是最简单也是最常用的校验器：

```
<field
      property="customerId"
      depends="required"/>
```

如果一个字段没有输入，或者仅有空格输入，那么这个校验就会失败，一个错误就会发回给框架。否则，校验成功通过。为了决定字段是否只包含空格，将调用标准的 String.trim() 方法(value.trim().length() == 0)。

因为浏览器根本不会提交空字段，所以如果字段为null或者为0长度，所有不是必须的（required）字段都会跳过全部校验。

12.3.2. mask 校验器

mask 校验器检查校验值是否符合正则表达式，如果匹配，则成功通过

```
<field property="postalCode" depends="mask">
  <arg0 key="registrationForm.postalCode.displayName"/>
  <var>
    <var-name>mask</var-name>
    <var-value>^\d{5}\d*$</var-value>
  </var>
</field>
```

表格 12.2　表 12.2 基本校验器

校验器	目的
Required	**如果字段包含除空格之外的其它字符则成功。**
Mask	**如果值匹配 mask 属性给定的正则表达式则成功。**
range	**如果值位于 min 和 max 属性给定的范围则成功 ((value >= min) & (value <= max)).**
maxLength	**如果字段长度小于或者等于 max 属性则成功。**
minLength	**如果字段长度大于或等于 min 属性则成功。**
byte, short, integer, long, float, double	**如果字段可以转换为对应的基本类型，则成功。**
Date	**如果值能够表示一个有效的日期，则成功。可以提供日期模板。**
creditCard	**如果值是一个有效的信用卡号码则成功。**
email	**如果值是一个有效的 e-mail 地址则成功。**

Jakarta RegExp 包 [ASF, Regexp] 可用来解析正则表达式。如果一个表达式需要被多个字段使用，也可以将它在validation.xml 文件中定义为一个常数，例如：

```
<constant>
  <constant-name>zip</constant-name>
  <constant-value>^\d{5}\d*$</constant-value>
</constant>
```

和其他大多数标准校验器一样，mask 校验器的声明是要依赖required 校验器。因此，如果一个字段同时依赖required 和 mask，那么 required 校验器必须在mask 校验器被应用之前成功通过。

12.3.3. range 校验器

range 校验器检查值是否落于最大值和最小值指定的范围之中：

```
<field property="priority"
       depends="required,integer,range">
  <arg0 key="ResponseForm.priority.displayName"/>
  <var>
    <var-name>min</var-name>
    <var-value>1</var-value>
  </var>
  <var>
    <var-name>max</var-name>
    <var-value>4</var-value>
  </var>
</field>
```

如果在字段中输入数字1,2,3或者4，这个校验器会成功。在实际应用中，错误消息应该显示范围的最大值和最小值，以便能够帮助用户正确输入。你可以使arg 元素来通过引用将min 和max 变量包含在消息中：

```
<field property="priority"
    depends="required,integer,range">
  <arg0 key="ResponseForm.priority.displayName"/>
  <arg1 name="range" key="${var:min}" resource="false"/>
  <arg2 name="range" key="${var:max}" resource="false"/>
  <var>
    <var-name>min</var-name>
    <var-value>1</var-value>
  </var>
  <var>
    <var-name>max</var-name>
    <var-value>4</var-value>
  </var>
</field>
```

这意味着范围消息的模板可能是这样的东西：

```
errors.range=Please enter a value between {1} and {2}.
```

所以，如果priority字段的 range 校验器失败，那么校验消息将会是：

```
Please enter a value between 1 and 4.
```

缺省下，校验器假定arg 元素的关键字匹配资源束中的关键字， 并且将替换资源条目的值为key 属性的值。resource=false 开关告诉校验器照原样使用值。如果值是被一个select 元素渲染的，你可能会希望将消息映射到选择列表中的第一个和最后一个选项：

```
<arg1 name="range" key="priority.range.first"/>
<arg2 name="range" key="priority.range.last"/>
```

这意味着在资源束中也有这样的条目：

```
priority.range.first=do-it-now
priority.range.last=forget-about-it
```

如果校验失败，针对priority的消息将会是：

```
Please enter a value between do-it-now and forget-about it.
```

12.3.4. maxLength 校验器

maxLength 校验器只检查范围的高端；如果字段的长度小于或者等于max属性则成功：

```
<field property="remarks"
       depends="maxlength">
  <arg0 key="responseForm.remarks.displayName"/>
  <arg1 name="maxlength" key="${var:maxlength}" resource="false"/>
  <var>
    <var-name>maxlength</var-name>
    <var-value>1000</var-value>
  </var>
</sfield>
```

这里field 元素确保remarks (也许是一个文本区字段) 的长度不超过1000 个字符。注意，我们将长度作为一个校验消息的参数传入，就如我们在range 校验器所作的一样。

12.3.5. minLength 校验器

minLength 校验器检查范围的低端；如果字段的长度大于或者等于min属性则成功：

```
<field property="password"
       depends="required,minLength">
```

```
  <arg0 key="logon.password.displayName"/>
  <arg1 name="minlength" key="${var:minlength}" resource="false"/>
  <var>
    <var-name>minlength</var-name>
    <var-value>5</var-value>
  </var>
</field
```

field 元素说明， password 必须输入并且其长度必须至少是5个字符。

12.3.6. byte, short, integer, long, float, 和 double 校验器

这些校验器都对校验值应用标准的 type.parseType 方法。如果捕捉到一个异常，校验器 便返回false。否则，校验成功：

```
<field property="amount"
       depends="required,double">
  <arg0 key="typeForm.amount.displayName"/>
</field>
```

12.3.7. date 校验器

date 校验器检查校验值，看其是否能够表示一个有效的日期：

```
<field property="date"
   depends="required,date">
  <arg0 key="typeForm.date.displayName"/>
</field>
```

校验器将框架维护的标准 Locale 对象 (java.util.Locale)传递给日期工具，所以结果将自三动本地化。datePattern 属性将传递一个标准日起模板给一个 java.text.SimpleDateFormat 对象：

```
<var>
    <var-name>datePattern</var-name>
    <var-value>MM/dd/yyyy</var-value>
</var>
```

在内部， datePattern 属性用在 impleDateFormat 的构造器之中，并且用来解析校验值：

```
SimpleDateFormat formatter = new SimpleDateFormat(datePattern);
```

```
Date date = formatter.parse(value);
```

如果解析成功，校验器便成功。

如果设置的是 datePatternStrict 属性，则还要检查长度已确保在适当的时候包含必要的前导0:

```
<var>
  <var-name>datePatternStrict</var-name>
  <var-value>MM/dd/yyyy</var-value>
</var>
```

如果没有指定任何模板，则会使用用户场所的 DateFormat.SHORT 格式。如果连Struts Locale 对象都无效，则会使用服务器的缺省场所。

对所有日期转换， setLenient 方法都设置为false。

12.3.8. creditCard 校验器

creditCard 校验器将分析校验值看其是否可能是一个信用卡号码：

```
<field property="creditCard"
       depends="required,creditCard">
  <arg0 key="typeForm.creditCard.displayName"/>
</field>
```

信用卡号码包含一个奇偶校验位。校验器将检查该位，以及其他其他业务规则，比如卡号的前缀是否是匹配某一个服务商 (American Express, Discover, MasterCard, VISA) 以及卡号的长度是否对制定的服务商来说的正确的长度。

12.3.9. email 校验器

email 校验器会对需要校验的email地址做一个广泛的检查，以确保它符合发布的email地址规范：

```
<field property="email"
       depends="required,email">
  <arg0 key="typeForm.email.displayName"/>
</field>
```

12.4. 资源束

校验的基本目的是使用户纠正输入问题，并且重新输入。因为该处理过程设计到显示字段标注和消息，Struts 校验器便很好的利用了Java本地化特征和框架对这些特征的支持。(Java 本地化特征在第13章讨论。)

当然，框架也需要为字段提供本地化的标注。

因为Struts 资源束是提供为一个应用级的资源，校验器将能够和框架共享同一资源束，所以你可以将所有字段标注和消息放在一起。在校验一个特定的字段时，你只需要为校验器添加缺省的消息和一些定制消息。

12.4.1. 缺省资源束

Struts 资源束是通过部署描述符（见第 4 章）配置的。它通常名为 ApplicationResources.property 或者干脆是 Application.property。在我们的示例应用中，资源束是保存在位于 /WEB-INF/src/java/resources 之下的一个包中。我们的缺省属性文件（`java.util.Property`）名称为 Application.property。支持场所的文件名称是 Application_*en*.property, Application_*e s*.property，Application_*fr*.property，等等。(再提一下，第 13 章会讨论更多的本地化的内容)。

在校验器中使用Struts资源束并不需要做什么特别的事情。仅需要添加你所需要的标注和消息，并且在校验器配置文件中引用消息资源。如果应用已经被本地化，那么字段标注应该已经提供，那么你就可以在框架的其他地方共享它们，如下所示：

```
<field property="lastName"
       depends="required">
  <arg0 key="registrationForm.lastName.displayName"/>
</field>
```

12.4.2. 缺省校验器消息

如果一个field元素没有提供定制消息，校验失败时便会使用校验器的缺省消息。校验器的缺省消息的关键字是在其被定义时指定的。通常的惯例是在校验器的名称之前添加一个 `errors.` 前缀。所以`required` 校验器的缺省消息关键字就成为`errors.required`。

这个约定于Struts `<html:error>` 标签相吻合，它会查找诸如*errors.header* 和 *errors.footer* 之类的条目。清单12.5 就列出了你可以针对基本校验器添加到English 资源束中的关键字和消息模板。

清单 12.5 基本校验器的默认消息

```
# Struts Validator Basic Error Messages
errors.required={0} is required.
errors.minlength={0} cannot be less than {1} characters.
errors.maxlength={0} cannot be greater than {1} characters.
errors.invalid={0} is invalid.
errors.byte={0} must be a byte.
errors.short={0} must be a short.
errors.integer={0} must be an integer.
errors.long={0} must be a long.
errors.float={0} must be a float.
errors.double={0} must be a double.
errors.date={0} is not a date.
```

```
errors.range={0} is not in the range {1} through {2}.
errors.creditcard={0} is not a valid credit card number.
errors.email={0} is not a valid e-mail address.
```

你可能注意到，这里没有针对errors.mask的条目。处于历史原因，针对mask 校验器的消息是errors.invalid 而不是errors.mask。

每一个校验器消息都可以使用四个替换参数，它们都可以在字段定义中指定为特定的arg 元素。第一个参数，arg0 或 {0}，通常是字段标注的关键字。校验器然后会从资源束中查找标注的显示文本，并且合并本地化文本到消息模板中。

12.4.3. 定制校验器消息

一个字段也可以指定一个定制校验器而不是默认校验器。

这通常会发生在使用mask 校验器的时候，以便你可以解释正则表达式期望什么样的模式。消息关键字在一个 msg 元素中给出。因为一个字段可以使用不止一个校验器，校验器的名称 便包含在msg 元素的name 属性中：

```
<field
       property="username"
       depends="required,mask">
  <msg name="mask"
       key="logon.username.maskmsg"/>
   <arg0
        key="logon.username.displayName"/>
   <var>
     <var-name>mask</var-name>
     <var-value>^[a-zA-Z0-9]*$</var-value>
   </var>
</field>
```

在属性文件中，我们就可以这样来替换关键字/模板条目：

```
logon.username.maskmsg={0} must be letters and numbers, no spaces.
```

12.5. 配置文件

Struts 校验器的强项是，校验是在应用源代码之外使用一个 XML 配置文件来设置的。配置指定表单中的哪个文件需要校验，字段所使用的校验器，一些字段可能使用的某些特殊设置。可以针对不同的场所配置不同的 formsets 并且覆盖场所敏感的校验。

框架使用的所有校验器都通过 XML 来配置，包括包所附带的基本校验器。你可以省略你的应用不需要的校验器并且插入你自己的框架自带的校验器之外的定制校验器。这有利于形成一个非常灵活的包，但是将这些所有配置都综合到一个文件可能会导致一个非常冗长的文

档。如表 12.3 所示，Struts 校验器实际上可以使用两个 XML 文件：一个用作设置校验器，而；另一个用作应用设置。(如注， Struts 1.0 使用一个配置文件。)

这将导致非常方便的管理。通过这种方式，可以很容易在应用间拷贝标准的 校验规则集，然后再定制特殊的 validator.xml 文件。或者，如果你愿意，你仍然可以将所有元素综合到一个 validation.xml 文件中。

在 Struts 1.1 中，valiudator 配置文件的路径是在 struts-config.xml 文件中声明的，如下所示：

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
       property="pathNames"
        value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```

表格 12.3 Struts 校验器 配置 文件 s

文件名称	说明
validator-rules.xml	校验器配置文件
validation.xml	配置应用的校验

关于如何安装Struts 校验器 1.1 组件到你的应用中参见第4章。

12.6. 校验器 JSP 标签

Struts 校验器框架将 (可选的) 需要校验一个表单所需的JavaScripts综合到一个脚本中。比拟可以通过定制标签来将脚本插入到JSP 页面中。然后脚本就可以在表单被提交时被调用。如果校验失败，脚本将显示一个错误窗口，如图12.1，并且提交失败。否则，提交成功，而控制转到Struts ActionForm 的`validate` 方法。这样确保校验即便在JavaScript被禁止时也会被触发。

在清单12.4中，我们引入了`<javascript>` 标签。为了简化，代码清单中保留了原始的`<html:errors>` 标签。原始`<errors>` 标签可以非常容易地用在页面中但是需要你将标记混合在其它消息中。否则，如果消息被表示为一个块，它们全部都会渲染到一个单一的段落中。

在Struts 校验器框架之下，错误消息是和JavaScript 校验共享的。在实践中，在两种情况下使用同样的标记可能会有问题。稍好一些的方式是到处都使用纯文本的消息。

Struts 校验器标签库提供还提供了一些帮助解决这些问题的额外标签。

`<errorsPresent>` 或者 `<messagesPresent>` 标签报告是否所有的消息都是未决的。`<messages>` 标签则类似于一个迭代器，所以你的页面可以遍历整个队列，提供整个方式所需的标记条目。

清单12.6 再次展示了清单 12.4中的代码，这次配备了消息标签。清单12.7展示了这段代码

的 Struts 1.1版本。

清单 12.6 具有校验和附加 JSP 标签(1.0)的 Logon 页面

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<!-- ① -->
<%@ taglib uri="/tags/struts-validator" prefix="validator" %>
<HTML><HEAD><TITLE>Sign in, Please!</TITLE></HEAD>
<BODY>
  <!-- ② -->
  <validator:errorsPresent>
    <UL>
      <!-- ③ -->
      <validator:errors id="error">
       <LI><bean:write name="error"/></LI>
     </validator:errors>
</UL>
  </validator:errorsPresent>
    <!-- ④ -->
  <html:form action="/logonSubmit" focus="username"
          onsubmit="validateLogonForm(this)">
  <table border="0" width="100%">
     <TR><TH align="right">UserName:</th>
        <TD align="left"><html:text property="username"/></TD>
     </TR>
    <TR><th align="right">Password:</TH>
        <TD align="left"><html:password property="password"/></TD>
    </TR>
    <TR>
        <TD align="right"><html:submit 属性="submit" value="Submit"/></TD>
        <TD align="left"><html:reset/></TD>
        <!-- ⑤ -->
       <TD align="left"><html:cancel onclick="bCancel=true"/></TD>
     </TR>
   </table>
 </html:form>
 <!-- ⑥ -->
 <validator:javascript formName="logonForm"/>
</BODY>
</HTML>
```

清单 12.7 具有校验和附加 JSP 标签 JSP 标签(1.1)的 Logon 页面

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<!-- ① -->
<%@ taglib uri="/tags/struts-validator" prefix="validator" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<HTML>
  <HEAD>
    <TITLE>Sign in, Please!</TITLE>
  </HEAD>
  <BODY>
  <!-- ② -->
  <logic:messagesPresent>
    <UL>
      <!-- ③ -->
      <lvalidator:messages id="error">
        <LI><bean:write name="error"/></LI>
      </validator:messages>
    </UL>
  </logic:messagesPresent>
  <!-- ④ -->
  <html:form action="/logonSubmit" focus="username"
          onsubmit="validateLogonForm(this)">
    <table border="0" width="100%">
      <TR><TH align="right">Username:</th>
        <TD align="left"><html:text property="username"/></TD>
      </TR>
      <TR><th align="right">Password:</TH>
        <TD align="left"><html:password property="password"/></TD>
      </TR>
      <TR>
        <TD align="right"><html:submit property="submit" value="Submit"/></TD>
        <TD align="left"><html:reset/></TD>
        <!-- ⑤ -->
        <TD align="left"><html:cancel onclick="bCancel=true"/></TD>
      </TR>
    </table>
  </html:form>
  <!-- ⑥-->
  <validator:javascript formName="logonForm"/>
</BODY>
```

```
</HTML>
```

下面是对清单 12.6 和12.7都一样的一些备注说明：

①因为我们要使用Struts 校验器标签，所以需要导入该标签库。在Struts 1.1中，我们还需要导入logic 标签库，因为它已经包含了我们可以在这里使用的标签了。

②Struts 1.0 的error 标签，或者Struts 1.1 的消息标签，我们将HTML 保持在页面中，而不是在错误消息资源中。如果没有消息，整个快将会被跳过。

③ `<errors>` 或 `<messages>` 标签类似于一个迭代器。该标签会将每一个消息暴露在 `id errors` 之下，以便bean 标签可以依次输出它们。

④一个 JavaScript `onsubmit` 事件被添加到`<html:form>` 标签中。这将在提交按钮被点击时，或者J avaScript `submit`事件被触发时，调用我们的校验器 JavaScript。

⑤为了允许用户能够取消和饶过校验，可以设置一个JavaScript 旗标。如果Submit 按钮设置 `bCancel` 为true，那么Struts 校验器将把控制转给Action。

⑥最后，是实际的 `<javascript>` 标签。在运行时，这将会被一个综合了针对该表单的JavaScript的脚本所代替。该脚本，类似于ActionForm, 是在`action-mapping` 的属性名称后命名的，这里是`logonForm`。

框架提供的基本校验器都包括JavaScript 校验。

如果一个可插入校验器 (第12.9节) 提供了一个JavaScript 校验，它也可被包括在这里。脚本要成功，所有校验都必须成功。

产生的JavaScript 观察ValidatorForm 的`page` 属性，并且仅对页面编号小于或等于表单的页面编号的字段产生校验器。缺省的页面编号都是0。这对多页面的向导表单是有用的。 (见第12.10.1节。)

12.7. ValidatorForm 和 ValidatorActionForm

为了在 Struts1.1 中使用 Struts 校验器，只需要遵循第 4 章中的安装指导并且使你的 ActionForm 扩展 ValidatorForm 和 ValidatorActionForm。ValidatorForm 将匹配 `formset` 名称和 `form-bean` 名称。ValidatorActionForm 将匹配 `formset` 名称和 `actionmapping` 路径。

大多数情况下，Struts 校验器可以完全替换掉编写 ActionForm 中的定制 `validate` 方法的需要。然而，如果你仍然需要一个 `validate` 方法，它仍旧可以在校验框架之外正常工作。在大多数情况下，你可能想要首先调用校验器框架，然后在它们通过之后再调用你自己的校验。清单 12.8 展示了如何完成这个任务。

清单 12.8 Validate 方法

```
public ActionErrors validate(
             ActionMapping mapping,
             HttpServletRequest request) {
```

```
  //①
  ActionErrors errors = super.validate(mapping, request);
  // ②
  if (errors==null) errors = new ActionErrors();
  if  (errors.empty()) {
    // ③
    if  (notGood(mapping,request))
        errors.add(ActionErrors.GLOBAL_ERROR,
                   new ActionError("errors.notGood","goodproperty"));
    }
    if  (errors.empty())
        return null;
    renturn errors;
  renturn (errors);
}
```

①首先，我们调用祖先 `validate` 方法。在这里，我们是调用的校验器框架。

②因为祖先方法可能会返回null，我们会首先检查null。因为我们想要运行我们自己的校验，它可能需要提交错误，如果没有返回一个错误我们将创建一个新的`ActionErrors` 对象。如果祖先返回错误，这个实现将不会运行我们自己的校验，虽然你也可以在其适合的情况下很容易的运行你的校验。

③我们的示例校验调用一个名为`notGood`的助手方法，它返回我们的定制校验的结果。我们从`validate` 方法传递参数，以确保`notGood` 知道`validate` 所知的任何事情。

12.8. 本地化的校验

校验器配置文件包含一个`formset` 元素。formset 是一个共享公共场所设置的所有表单集合的包装器。虽然有些表单中的一些字段的确需要本地化。但是大部分还是不需要的。

Struts 校验器允许你本地化选择的字段并且对余下的字段使用缺省校验。缺省formset将忽略language, country, 和variant 属性。本地化是完全有范围的；你可以定义一个格式来仅仅覆盖language, 或者 country，如果需要两者都可以。关于如何国际化你的应用，参见第13章。

12.9. 可插入校验器

每一个field 元素都可以指定一个其依赖的校验器列表。有些应用使用极少的校验器；而有些则使用很多。为了允许开发人员仅仅装载他们的应用所需的校验器以及使得装载定制对象更加容易，校验器可以从配置文件中声明，本质上令它们都是可插入的。

定义	*Pluggable* 是一种面向对象的设计策略，允许对象独立于应用进行开发，然后和应用整合在一起而不用修改代码基。可插入组件通常由第 3 方创建。

12.9.1. 创建可插入校验器

创建你自己的校验器并且插入到应用中有两个步骤：

- 在Java类中创建一个方法来处理服务器端校验
- 修改validator-rules.xml 文件个中针对你的校验器的元素。如果你的校验器将具有一个户端 JavaScript 组件，你可以将此作为 `validator` 元素的一部分。.

创建一个validate方法

任何类都可以用来保存你的 `validate` 方法。在Struts 1.0中，方法必须遵循特定的签名：

```
public static boolean validateMyValidator(
    Object bean,
    ValidatorAction va,
    Field field,
    ActionErrors errors,
    HttpServletRequest request);
```

表 12.4 提供了指向传递给你的 `validate` 方法的参数的关键字。

在 Struts 1.1 中，你可以将你的方法的签名指定为配置的一部分。

关于如何使你的可插入校验器完成你所需的工作的编程提示，参见 `org.apache.struts.validator.util.StrutsValidator` 类。这个类中包含了针对校验包所附带的基本校验器的方法。你的校验应该实际上是和基本校验器相同的工作方式，因为它们本质上都是插件。

STRUTS TIP

在开发可插入性校验器时，请关注 log 文件。许多编程错误将只是被记录，而不会在应用中暴露。一个恰当的例子是在 validator-rules.xml 文件中的包名称错误。一个 `ClassNotFound` 异常将会被记录，但是如果校验成功，应用将继续进行。

表格 12.4 清单 12.3 校验器 方法 属性

属性	说明
Object bean	校验将在其上执行的 bean。
ValidatorAction va	这是一个表示针对来自于 validator-rules.xml 中的这个校验器的 validator 元素的 JavaBean。这个 bean 是在 the org.apache.commons.Validator 包中声明的。
Field field	这个 JavaBean 表示针对我们将要校验的字段的元素。这个 bean 是在 org.apache.commons. Validator 包中声明的。

ActionErrors errors	一个接受任何可能发生的校验错误的 ActionErrors 对象。
HttpServletRequest request	此次操作之下的当前请求对象

声明一个validator元素

清单 12.1 和 12.2 展示了配置元素，这和用于你自己的 plug-in 所用的配置是一样的。required 部分很简单：

```
<validator
    name="required"
    className="org.apache.struts.validator.util.StrutsValidator"
    method="validateRequired"
    msg="errors.required"/>
```

这个元素告诉 Struts 校验器哪个方法调用那一个类，以及校验失败时所用的消息关键字。(嘿，所时候它不得不失败；否则你到底要说什么?)

在 Struts 1.1 中，你也可以指定你的 validate 方法将使用的参数：

```
<validator
    name="required"
    className="org.apache.struts.util.StrutsValidator"
    method="validateRequired"
    methodParams="java.lang.Object,
                  org.apache.commons.validator.ValidatorAction,
                  org.apache.commons.validator.Field,
                  org.apache.struts.action.ActionErrors,
                 javax.servlet.http.HttpServletRequest"
                  msg="errors.required"/>
```

除了声明服务器端校验之外，你可以指定你的校验器一起使用客户端 JavaScript 校验。常被的validator-rules.xml 中的各种JavaScript 元素提供提供了多个如何编写复杂脚本的例子。

12.10. 技术

除了我们前面所述的日常使用之外，在Struts校验器中你还可以使用多种特殊的技术：

- 多页面校验
- 取消按钮
- 定制消息
- 交叉相关按钮

- 通过validate 方法综合校验器

12.10.1. 多页面校验

许多开发人员都喜欢使用多页面表单向导。一个向导会通过多个不同的表单收集某个操作所需的信息。然后，它在最后综合结果。因为不是一次提供太多的信息，它能使用户更易完成较大的表单。有些开发人员对每个页面使用不同的表单。而其他一些则喜欢使用一个大表单，但是每次只暴露一个部分。

如果你使用的是一个大表单向导的方式，Struts 校验器将在 field 元素中包括一个 page 属性，并且在 ValidatorForm 中提供一个对应的 page 属性。在执行对某个字段的校验之前，框架将进行检查看 field 页面是否小于或者等于 ValidatorForm 页面。这就是说作为校验第 3 个页面的一部分，我们还会再次对页面 1 和 2 的校验进行检查。这样将会有助于防止用户跳过页面。

如果你的一个大表单向导具有一个 reset 方法，你也可以使用 page 属性来重设仅针对当前页面的那些值：

```
If (page==1) {
  // 重设页面 1 属性
}
if (page==2) {
  // 重设页面 2 属性
}
```

12.10.2. 取消按钮

大多数表单都给用户一个机会能够一齐取消操作，通常是通过一个取消按钮。这里的问题是取消按钮提交表单—所以 JavaScript 试图校验它。服务器端部分可以做来知道取消按钮，但是 JavaScript 是在客户端，并且对框架一无所知。所以，为了取消一个表单，用户首先安抚好 JavaScript 校验。.这可不行。

为了解决这个问题， Struts 校验器提供了一个 bCancel JavaScript 变量。

如果 bCancel 被设置为 true，JavaScript 校验也将返回 true，允许取消请求直接传递到容器。 服务器端校验知道 Struts <cancel> 按钮，所以如果它们看到请求中有取消请求，就不会触发服务器端校验。剩下的就在于 Action 了，它在提交任何用户可能传递的操作之前应该检查其自己的 isCancelled 方法：

```
<html:cancel onclick="bCancel=true;">
<bean:message key="button.cancel"/>
</html:cancel>
```

12.10.3. 定制消息

每个校验器都可以定义一个缺省的在校验失败时所用的错误消息。例如，

```
Errors.integer={0} must be an integer.
```

在有人试图往一个标注Quantity的字段中输入一个字母时，将自动显示：

```
Quantity must be an integer.
```

对大多数校验器来说，这会工作得很好，并且缺省消息也正是你所需的。但有个例外是mask校验器，它被和正则表达式一起使用。

这里，缺省消息是

```
Errors.invalid={0} is invalid
```

如果在表单中的一个至少需要5个字符长度的password字段中使用 mask 校验，缺省的消息将会是：

```
Password is invalid.
```

这并没有告诉用户他们该如何来纠正问题。稍微好一些的消息可能是：

```
Password must be five or more characters long.
```

当然，该消息将不能用于其它使用不同正则表达式的mask 校验器的字段中。

在大多数情况下，在你使用 mask 校验器时，你还应该指定一个解释正则表达式所期望的内容的定制错误消息：

```
<field property="password" depends="required,mask">
  <msg name="mask" key="accountForm.password.mask"/>
  <arg0 key="NameForm.password.displayName"/>
  <var>
    <var-name>mask</var-name>
    <var-value>^{5}*$</var-value>
  </var>
</field>
```

并且在应用资源中有：

```
accountForm.password.mask ={0} must be five or more characters long.
```

虽然消息属性最常用在mask 校验器中，你也可以在逐字段的基础上覆盖针对任何校验器的消息。

12.10.4. 交叉相关的字段

如果用户要修改他们的口令， 对于程序来说，要求用户输入新口令两次以帮助他们正确输入，是很常见的事情。表单中的许多其它字段可能也会以某种方式交叉相关。你可以通过定义你自己的plug-in 校验器以你喜欢的方式对它们进行比较。

在同一应用中同时使用它们，示于清单12.9，12.10 和12.11的组件创建了一个 比较两个字段以保证它们相等的 plug-in 校验器：

清单 12.9 Validator-Rules.xml

```
<validator name="identical"
  className="com.mysite.StrutsValidator"
  method="validateIdentical"
  depends="required"
  msg="errors.identical"/>
```

清单 12.10 validate.xml

```
<field property="password"
        depends="required,identical">
  <arg0 key="accountForm.password.displayName"/>
  <var>
    <var-name>secondProperty</var-name>
    <var-value>password2</var-value>
  </var>
</field>
```

清单 12.11 apps.ValidateUtils

```
public static boolean validateIdentical(
                  Object bean,
                  ValidatorAction va,
                  Field field,
                  ActionErrors errors,
                  HttpServletRequest request) {
  String value = ValidatorUtil.getValueAsString(bean,field.getProperty());
  String sProperty2 = field.getVarValue("secondProperty");
  String value2 = ValidatorUtil.getValueAsString(bean, sProperty2);
  If  (!GenericValidator.isBlankOrNull(value)) {
    try {
      if (!value.equals(value2)) {
        errors.add(field.getKey(),
               ValidatorUtil.getActionError(application, request, va, field));
      return false;
      }
    }
    catch (Exception e) {
      errors.add(field.getKey(),
                 ValidatorUtil.getActionError(application,
                                              request, va, field));
```

```
      return false;
    }
  }
  returen true;
}
```

12.10.5. 综合使用校验器和 validate 方法

Plug-in 校验器，比如第 12.10.4 节所示的那个，可以用于确保字段之间更加复杂的关系得到维护。例如，如果有些东西可以车运或者船运，而用户选择了船，你应该插入一个校验器以确定选择了船运选项。

插件校验器最好的候选者就是那些你可以在多个表单间重用的。如果校验不能被重用，你也可以覆盖 `validate` 方法并且按常规的方式使用它。只是需要确保也调用了祖先的 `validate` 方法，以确保全部框架校验都被触发。清单 12.12 就展示了如何综合 Struts 校验器和一个定制的 `validate` 方法。

清单 12.12 综合使用校验器和校验方法

```
public ActionErrors validate(ActionMapping mapping,
                             HttpServletRequest request) {
    ActionErrors errors = super.validate(mapping, reuest);
    if (errors==null) errors = new ActionErrors();
    // if selects shipping
    if ("S".equals(deliveryType)) {
       // Vendor required
       If ("".equals(getShipVendor().trim())) {
          errors.add(ActionErrors.GLOBAL_ERROR,
                     new ActionError("item.shipVendor.maskmsg"));
        }
       if (errors.empty()) return null;
    return errors;
}
```

12.11. 迁移一个应用到 Struts 校验器

虽然最近大多数 Struts 应用都编写来使用 Struts 校验器，但是大多数在转移到校验器制钱都是使用自己的程序进行校验的。在这一节，我们将经历将一个简单的 ActionForm `validate` 方法迁移到其 Struts 校验器版本的过程。练习点并不是 `validate` 方法的例子，这微乎其微，但是重点是将方法转移到 Struts 校验器的过程。

在第 11 章，我们将遵循类似的过程来将一些示例页面迁移到 Tiles。

12.11.1. 设置校验器框架

设置校验器在Struts 1.0 和1.1中稍微有些不同，但是but works just as well with either version.

NOTE	在开始之前，最好做一次额外的备份，不要管常规下已经做了多少次备份了。迁移到校验器在起初可能是很灵活的，并且实际上，在一切 OK 之前，你需要搞定的不只是一件事情那么简单。所以，应该准备好回滚过去，然后再试。'Nuff 说。

Struts 1.0

如果你还没有这样做，第一步就是安装校验器包，并且通过你的应用部署描述符装载校验器servlet。(校验器servlet只是一个资源装载器，所以不会与Tiles servlet相互冲突。)

然后，通过测试点击几个页面测试你的应用是否正常。

本书网站[Husted]上的空白Struts 1.0应用就包括了一个空的校验器配置文件以及一个示例设置。

Struts 1.1

校验器集成到Struts 1.1中。在Struts 1.1中激活校验器包含在本书的第4.9节。

12.11.2. 测试缺省配置

在部署描述符 (web.xml) 中将 `debug` 和 `detail` 参数设置为 level 2，重新启动应用。仔细检查 log 文件是否有新的错误消息。运行一些单元测试，并且点击测试应用确认操作仍然正常。

12.11.3. 重审你的校验

设置好校验器并开始运行后，下一步就是仔细研究你的 `validate` 方法。 标识出那些可以对应于一个标准的 Struts 校验器校验，而那些必须通过定制校验来处理。校验器并不是一个或然命题。你可以继续使用 ActionForm 的 `validate` 方法来处理一些事情，而校验器笮处理另一些事情。

清单 12.13 展示了一个调用 Struts 校验器然后调用一些定制校验的代码骨架：

清单 12.13 调用校验器和你的定制定制程序

```
public ActionErrors validate(
             ActionMapping mapping,
             HttpServletRequest request{
 // ①
 ActionErrors errors = super.validate(mapping, request);
 // ②
 if  (null==errors) errors = new ActionErrors();
 /* ③
 if (!(dateCheck()))
   errors.add(ActionErrors.GLOBAL_ERROR,
          new ActionError("errors.invalid","Expiration Date"));
*/
```

```
// ④
// ⑤
if (errors.empty()) return null;
 return errors;
} // end validate
```

.

① 此方法是想要用于一个 Validator Form 子类。通过调用和捕获超类的 validate 方法，我们可以运行任何 Struts 校验器校验但是同时又保留了调用我们自己的定制校验器的空间。

② 如果超类校验全部通过， validate 将返回 null。我们仍然可以找到我们自己的校验过程产生的错误，所以我们创建了一个 ActionErrors 集合来持有它们。如果 ValidatorForm 超类返回一些错误，我们仍然可以继续使用同一个集合来持有它们。这将提供一个统一的错误消息集合给用户，而不管是哪一个 validate 方法运行的校验。

③这个校验已经被注释掉，而被 Struts 校验器所替代。一旦方法全部被测试，它将被删除。

④其他还没有被转移到 Struts 校验的校验程序可以在此运行。

⑤如果没发生错误，代码将返回 null ;否则，它将返回一个错误的综合列表—包括针对 Struts 校验器和我们自己的校验的错误。

此方法的精神何在？如果一个标准校验不能够处理一些你自己的校验过程，就可以计划暂时保留它们。你可能还想要在后面使用定制校验器替换这些过程，但是请首先摘下够得着的果子，而使标准版本首先工作起来。

12.11.4. 扩展ValidatorForm 或者Scaffold BaseForm

在对你的 ActionForm 进行修改之前，请确保它扩展 ValidatorForm 或者 Scaffold BaseForm：

```
import com.wintecinc.struts.action.ValidatorForm; // Struts 1.0.x
// import org.apache.struts.validator.ValidatorForm; // Struts 1.1
public class ActionForm extends ValidatorForm {
// . . .
}
```

或者

```
import org.apache.scaffold.struts.BaseForm;
public class ActionForm extends BaseForm {
// . . .
}
```

在这两种情况下，你都可以毫无错误的重建和测试你的应用。这两个类都不会改变缺省的

ActionForm 功能。如果此时你的确遇到了一些错误，在继续之前请先解决它们。

12.11.5. 选择一个校验来迁移

我们来看一下一个来自于Artimus 示例应用 [ASF, Artimus]早期版本的校验例子。在保存一篇文章之前它会检测看一些必须的字段是否已经提供了。清单12.14 展示了类的完整代码：

```
package org.apache.artimus.article.struts;
import javax.servlet.http.HttpServlet请求;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;
import org.apache.artimus.struts.Form;
public class ArticleForm extends Form {
// ①
private boolean isNotPresent(String field) {
 return ((null==field) || ("".equals(field)));
}

public ActionErrors validate(ActionMapping mapping,
                             HttpServletRequest request) {
 // ②
  ActionErrors errors = new ActionErrors();
 // ③
  String title = getTitle();
  if (isNotPresent(title)) {
        errors.add(ActionErrors.GLOBAL_ERROR,
                    new ActionError("errors.required","Title"));
  }
  String content = getContent();
  if (isNotPresent(content)) {
     errors.add(ActionErrors.GLOBAL_ERROR,
          new ActionError("errors.required","Article text"));
  }
  String creator = getCreator();
  if (isNotPresent(creatoror)) {
      errors.add(ActionErrors.GLOBAL_ERROR,
                 new ActionError("errors.required","Author"));
  }
  if (errors.empty()) return null;
 rerurn errors;
} // end validate
// end ArticleForm
```

①isNotsPresent 是一个简单的工具方法来测试一个字段是否为null或者为空。

②这段代码创建一个ActionErrors 集合，以便随后可以用得着。

③这里运行我们的3个简单校验。如果有某个失败，就会创建一个ActionError。errors.required 模板被用于创建ActionError，并且字段被合并到消息中。

校验运行时，没什么大问题，但是最好还是从一些简单的事情开始，并且使你的工作先运转起来。我们来看看title 校验过程怎样迁移到Struts 校验器。

12.11.6. 添加formset, form，和 field 元素

首先，添加一个缺省的 <formset> 元素到校验器配置文件 (validation.xml)中，并且向其中添加一个针对表单的 form 元素和一个针对属性的 field 元素。

清单 12.15 列出了用于我们的 articleForm 例子的初始<formset> 设置。

清单 12.14 Struts Validator 的一个初始<formset> 设置 s

```
<formset>
  <form name="articleForm">
   <field
         property="title"
         depends="required">
      <arg0 key="Title" resource="false"/>
    </field>
  </form>
</formset>
```

清单 12.11 中的 title 校验值只是对其必需性的检查。当然，Struts 校验器有一个标准校验器来做这种事情。在清单 12.15 中，我们指定了在表单 articleForm 中，字段 title d 依赖于 required 校验器。

我们的错误消息包括字段的名称，并将其作为{0} 替换参数，所以我们可以将其作为<arg0> 元素传递给字段。

12.11.7. 向ApplicationResources中加入新的条目

清单 12.15 中的<field> 元素使用了 resource="false" 属性来传递一个字面 String。在这种情况下，最好遵循正确的路线，将语言元素提取到 ApplicationResources 资源束中。在你编写你自己的校验方法时，很容易将这些语言元素嵌入到 Java 代码中。但是在你使用 Struts 校验器的时候，也很容易将它们放入到 ApplicationResources 资源束中(它们本来就属于那里)。清单 12.16 列出了引用了 ApplicationResource 资源束的初始<formset> 设置：

清单 12.15 Struts Validator 的一个初始<formset>设置

```
<formset>
```

```
  <form Name="articleForm">
    <field
           property="title"
           depends="required">
      <arg0 key="article.title.displayName"/>
    </field>
  </form>
</formset>
```

然后我们还需要在我们的 ApplicationResources 文件中加入适当的条目(/WEB-INF/src/java/resources/Appplication.property):

```
<formset>
  <form name="articleForm">
    <field property="title"
           depends="required">
      <arg0 key="article.title.displayName"/>
    </field>
    <field property="creator"
           depends="required">
      <arg0 key="article.creatoror.displayName"/>
    </field>
    <field property="contentDisplayHtml"
           depends="required">
      <arg0 key="article.content.displayName"/>
    </field>
  </form>
</sformset>
```

12.11.8. 调用Struts 校验器

在 ArticleForm 类的 Java 源代码文件中，我们如今可以调用 ValidatorForm 超类来校验我们的 `title` 字段。一开始，我们可以将原来的校验注释掉而保持其余部分不作变动。清单 12.17 展示了修改后的针对 `articleForm` 的 `validate` 方法。这是来自于清单 12.13 中而应用到清单 12.16 的骨架代码。

清单 12.16 修改后的 validate 方法

```
public ActionErrors validate(ActionMapping mapping,
                             HttpServletRequest request) {
 ActionErrors errors = super.validate(mapping, request);
 if (null==errors) errors = new ActionErrors();
```

```
  /*
String title = getTitle();
if (isNotPresent(title)) {
   errors.add(ActionErrors.GLOBAL_ERROR,
                        new ActionError("errors.required","Title"));
}
*/
String content = getContent();
if (isNotPresent(content)) {
    errors.add(ActionErrors.GLOBAL_ERROR,
                        new ActionError("errors.required","Article text"));
}
String creatoror = getCreatoror();
if ((isNotPresent(creator)) {
  errors.add(ActionErrors.GLOBAL_ERROR,
                       new ActionError("errors.required","Author"));
}
if (errors.empty()) return null;
return (errors);
} // end validate
// end ArticleForm
```

12.11.9. 测试并重复

重新构建并重新装入应用以确保任何东西的最新版本都会起作用。试着打败校验。当运行确认正常时，继续下一个校验过程。在我们的例子中，最终会是：

```
<formset>
  <form name="articleForm">
    <field property="title"
           depends="required">
      <arg0 key="article.title.displayName"/>
    </field>
    <field property="creatoror"
           depends="required">
      <arg0 key="article.creatoror.displayName"/>
    </field>
    <field property="contentDisplayHtml"
           depends="required">
      <arg0 key="article.content.displayName"/>
    </field>
  </form>
```

```
</formset>
```

12.11.10. 删除ActionForm 超类

我们的示例 ActionForm 子类化了一个粗糙的 ActionForm (第 5 章)。

ArticleForm 类自身除了处理继承的属性的 `validate` 方法之外并没有提供什么。Bean 的所有属性都定义在一个基类中。

在这里，一旦所有的校验都转换到 Struts 校验器，我们就可以删除类而将`<form-bean>` 元素修改为使用基类。

即，原来这样：

```
<form-beans>
  <form-bean
    name="baseForm"
    type="org.apache.artimus.struts.Form"/>
  <form-bean
    name="articleForm"
    type="org.apache.artimus.article.struts.ActicleForm"/>
  <!-- ... -->
</form-beans>
```

如今可以为：

```
<form-beans>
  <form-bean
    name="baseForm"
    type="org.apache.artimus.struts.Form"/>
  <form-bean
    name="articleForm"
    type="org.apache.artimus.article.struts.ActicleForm"/>
    <!-- ... -->
</form-beans>
```

同一个 ActionForm 可以被人和数量的 form bean 所用，就象一个 Action 类可以被任意数量的 action mapping 所用一样。每一个实例只是给定一个不同的 attribute 名称，而 Struts 校验器将根据匹 attribute 名称来配 `<form>` 元素。通过一个粗粒度的 ActionForm 来定义我们的属性, Struts 校验器 `<form>` 元素就能完成子类的工作。

在 Struts 1.1 中，你可以使用 DynaValidatorForm 类来避免声明新的 ActionForm 类。你可以不讲你的 form 所需的简单属性声明为`<form-bean>` 元素的一部分。

下面是 Artimus 1.1 ArticleForm 的`<form-bean>` 配置：

```
<form-bean
  name="articleForm"
  type="org.apache.struts.validator.DynaValidatorForm">
  <form-property
    name="keyName"
    type="java.lang.String"/>
  <form-property
    name="keyValue"
    type="java.lang.String"/>
  <form-property
    name="marked"
    type="java.lang.String"
    initialValue="0"/>
  <form-property
    name="hours"
    type="java.lang.String"/>
  <form-property
    name="articles"
    type="java.lang.String"/>
  <form-property
    name="article"
    type="java.lang.String"/>
  <form-property
    name="contributor"
    type="java.lang.String"/>
  <form-property
    name="contributedDisplay"
    type="java.lang.String"/>
  <form-property
    name="creator"
    type="java.lang.String"/>
  <form-property
    name="title"
    type="java.lang.String"/>
  <form-property
    name="contentDisplayHtml"
    type="java.lang.String"/>
</form-bean>
```

这个 DynaBean 的校验器 `<form>` 示于第 12.11.9 节。

这两个 XML 构造就可以替代编写常规的 ActionForm 类和其 `Validate` 方法。

12.12. 小结

Struts 校验器是对框架的一个强大的补充。它允许将校验放入一个单独的配置文件中进行管理，它那里，它们可以被评审和修改而不用改变任何 Java 或者 JSP 页面代码。这很重要，因为象本地化一样，校验是连接到业务层，因为不应该与表现代码混合在一起。

框架附带了几个可以满足你的大部分常规需要的基本校验器。你可以很容易地针对特殊需求添加定制校验器。如果需要，原始的 Struts `validate` 方法可以和 Struts 校验器混合使用一确保满足你所有的要求。

和 Struts 框架主框架一样，Struts 校验器可以从根本上进行本地化。它甚至可以和主框架共享标准消息资源文件，为你的翻译提供无缝的解决方案。

校验输入是一个 web 应用框架必须提供的基本服务，而 Struts 校验器是一个可以扩展至满足甚至最复杂的应用的灵活的解决方案。

13. 本地化

本章内容

- *理解本地化的重要性*
- *使用 Struts 的国际化特征*
- *本地化你的应用*

Life is a foreign language; all men mispronounce it.

—Christopher Morley (1890–1957)

13.1. 以另外的名称

Struts 框架的一个关键特征就是它从用户收集输入，校验输入的形式，以及(如果必需) 回显信息给用户进行纠正。为了帮助将输入重新显示给用户， Struts 提供一些定制标签来创建HTML控件，并且从JavaBean [Sun, JBS]中组装数据。为了提醒用户什么数据需要纠正，框架还提供一个后处理系统。

虽然也可以很容易地使这些组件使用硬编码的字符串，但是这会使得文本非常难以进行改变。为了替换某个标注或者文本信息，需要对表现代码和Java代码都进行修改。这也意味着同一个应用如果不经过重新编译，将不能用在不同的语言环境中。

为了避免这些问题， Struts框架让开发人员在一个单独的文件中定义标注和信息，这个文件称为是*resource bundle* (`java.util.ResourceBundle`)。 当需要输出一个消息或者标注时，可以通过关键字来引用之。框架会在运行时自动根据关键字检索响应的信息。

图像或者图像按钮的源路径以及其替换文本也可以从资源束中来读取。资源束是Java的国际化特征的一部分。像Java一样, Struts 是从头彻底进行国际化的。

这也是本框架风靡Java世界的原因。

定义

Internationalization 就是设计一个应用的流程，以使之能够不经过工程修改就可以使用与各种不同等额语言和区域场所。

有时，国际化缩写为 i18n，因为在第一个字母 i 和最后一个字母 n 之间有 18 个字母.

Struts 直接构建在Java 平台提供的标准特征之上。所以开发人员可以一致地使用相同的技术来本地化他们的组件而不是直接依赖于Struts 框架。

定义

本地化（Localization）通过添加特定场所的组建和翻译文本来使软件适用于特定的场所和语言的过程。这个词汇一般缩写为 l10n，因为在第一个字母 l 和最后一个字母 n 之间有 10 个字母。通常，本地化最费时间的部分是翻译消息文本。其他类型的数据,如声音和图像等等，如果是文化敏感的，可能也需要进行本地化。本地化也包括对货币符号、时间格式、数字格式、日期格式等的修改。

在这一章，我们首先来看，为什么有那么多开发人员要本地化他们的应用，以及Java 的国际化是如何工作的。我们也讨论Struts i18n 组件，以及如何利用它来本地化你的应用。一些常用的Struts 插件，如Struts Validator 和Tiles，也可以被本地化，连同你可以传递给HTML控件元素的集合。最后，我们会看看如何来对这些组件进行本地化。

13.1.1. 为什么要本地化？

我们花费了大量的精力来开发web应用。每个组织都希望在应用的开发上得到更大的投资回报。因为互联网连接着世界各地，每个应用都可以面对国际的受众。如果某个应用可以被本地化，那么它就可以被更多的访问者所访问。特别是在多语言的国家，比如比利时、加拿大、瑞士，甚至美国。

不幸的是，许多应用都不是国际化的，要完成这个任务需要大量的精力。然而，因为Struts具有内建的国际化特征，因此大部分基于Struts的应用都可以很顺利的进行国际化。如果一个 Struts 应用从一开始就构建来使用国际化特征，本地化仅仅是需要翻译一个单独的消息文件而已。

即使不需要本地化，妥善使用这里介绍的技术也可以是你构建一个更加鲁棒的而且易于维护的应用。本地化将各种公共资源集中起来，可以一起进行评审和修改。难以进行本地化的应用多半也可能还有一些其它毛病。许多国际化的技术都基于通用的最佳实践，比如封装、模块化、以及内容分离。

虽然语言是本地化中最重要的一个部分，但却不是唯一需要考虑的因素。日期、货币、图像等资源通常也需要进行本地化考虑。Java 平台有内置的特征可以处理日期、时间和货币，Struts 又添加了选择本地化图像的方便的支持特征。

13.1.2. Java国际化是如何工作的

要被国际化，一个应用应该能够：

- 区分不同的场所（Local）；
- 使用同一个可执行包根据不同的场所显示相应的信息和标签；
- 在不重新编译的情况下提供对新的场所的支持；
- 自动格式化场所敏感的条目，比如日期和货币，为相应的场所和语言支持的格式。Java 可以通过使用3个关键的类: Locale, ResourceBundle, 和MessageFormat来帮助应用进行国际化，如表 13.1。

表格 13.1 Java 国际化的关键类

国际化需求	Java 类
区别不同的场所	java.util.Locale
使用同一个可执行包根据不同的场所显示相应的信息和标签；	java.util.ResourceBundle
自动格式化场所敏感的条目，比如日期和货币，为相应的场所和语言支持的格式。	java.text.MessageFormat

我们来逐一进行讨论。

13.1.3. 场所（Locale）

Java对国际化的支持的绝对核心是Locale 对象(`java.util.Locale`)。这个看似简单的对象使你可以结合语言、国家以及一些可选的变量到一个单一的场所实体中。语言和国家代码

通过ISO 标准来定义[IS0-3166, ISO-639].。表 13.2列出一部分国家和语言代码。

表格 13.2 示例语言和国家代码

语言代码	说明	国家代码	说明
De	German	CN	China
es	Spanish	CA	Canada
En	English	DE	Germany
Fr	French	FR	France
Ja	Japanese	IN	India
Jw	Javanese	US	United State
ko	Korean		
Zh	Chinese		

可选的variant 通常用来表示某种方言 (或许表示需要使用哪一种浏览器)。Struts 框架没有使用variant 字段。但是因为使用了标准库，如果需要，也可以使用variant 字段。

为了创建一个 Locale 对象，可以传递语言和国家代码到它的构造器中。比如，对讲法语的加拿大人可以创建这样的场所：

```
Locale = new Locale("fr", "CA");
```

通常，Locale 对象会被传递给工具方法来根据Locale 对象的设置对输出进行格式化。可以根据场所改变其行为的组件通常称为是场所敏感的。场所仅仅是一个提供信息目的的简单标识。它指示场所敏感方法来完成实际的本地化工作。

JVM 有一些方法都是场所敏感的，它们可以根据给定场所的习惯来格式化数字、日期和货币等内容。

比如：

```
NumberFormat.getInstance(myLocale)
NumberFormat.getCurrencyInstance(myLocale)
NumberFormat.getPercentInstance(myLocale)
```

有很多Struts 组件也是场所敏感的，它们使用框架为每个用户管理Locale对象。关于Struts i18n 组件，参见13.2。

Locale 对象也可以产生一个用户友好的设置描述。这在你准备向用户显示当前场所设置或者产生一个可支持的场所列表时非常有用。有好多方法可以用来处理这种说明，以便很好地满足Locale对象作为标识符的角色。这些方法中大部分可以重载，以便你可以在其它Locale对象中显示这些描述。.

表格 13.3 处理场所描述的两个重载方法

方法	Purpose
getDisplayName()	**返回适合于向用户显示的场所名称**
getDisplayName(Locale)	**返回适合于向用户显示的场所名称**

Locale 对象是不可修改的。如果你需要修改场所，必须创建一个新的Locale对象，用来替换原来的。在JVM 中有一些常用的现成对象供你使用，参见Java API获得更多信息(`java.util.Locale`).

13.1.4. ResourceBundle

在设计一个国际化的应用时，你首先应该将注意力集中在用户接口上。应用所使用的各种各样的菜单、对话框通常称为是程序资源。许多编程环境的设计都允许用户接口可以从一个单独的文件中来导入，或者用 Java 的术语来说，是资源束（resource bundle）。

Java ResourceBundle 类(java.util.ResourceBundle)并不是设计来读入整个用户接口，却可以用来存储用户接口元素所需的文本和消息资源。你的应用可以请求场所相关的资源，并从中间所相应的项目。对每个元素来说都使用相同的关键字，而不管具体的场所，但是其返回的 String 或者对象可能不同。要支持其他场所，仅需要添加该场所的一个资源文件到资源束中。

定义

资源束 (java.util.ResourceBundle) 是一个属性对象 (java.util.property) 的集合。每个属性对象都以关键字对应特定的地点。地点是通过区域和语言用 Locale 对象 (java.util.Locale)来表示的。当请求一个关键字时，也就指明了某个场所。如果场所的一个属性对象在包中没有找到，则会从最接近的场所中返回值。在实际应用中，ResourceBundles 使用继承的原理。如果最接近的场所没有找到，则返回下一个最接近的，依次类推。资源束可以包含一个缺省的属性对象，用在找不到更好的匹配的时候。

ResourceBundle是一个抽象类，具有两个原始实现，即ListResourceBundle 和 PropertyResourceBundle类。

PropertyResourceBundle 类 (`java.util.propertyResourceBundle`) 用来管理一个文本消息集合，它最常用于和Struts应用中结合使用。消息可以从一个属性文件(`java.util.property`)中载入，通常的做法是调用一个静态方法，像这样：

```
message = ResourceBundle.getBundle("application",userLocale);
```

属性文件是一个简单的文本文件，可以在许多普通的平面文本编辑器中进行创建。

而每个文本对应的场所则可以用一种缩写习惯所命名。语言和国家代码被添加到文件名称的末尾，扩展名之前。前述示例代码、应用中的第一个参数会应用到一系列属性文件，象这样：

```
Applicatiopn_es_ES.property
Application_fr_FR.property
Application.property
```

属性文件包含关键字-值对，称为是条目（*entries*）。关键字是一个你可以在应用中用来检索特定信息的记号。每个场所对应的文件包含根据各国家和语言的翻译过的信息。

对于场所es_ES(Spanish/Spain)的属性文件可能看起来是这样：

```
greetings = Hola
farewell = Adios
inquiry = Como estas?
```

经过翻译后，对于fr_FR (French/France)场所的属性集可能看起来是这样：

```
greetings = Bonjour.
farewell = Au revoir.
inquiry = Comment allez-vous?
```

在不能找到请求的场所时，将使用缺省的资源束。这意味着不需要向服务器的缺省场所提供额外的资源文件。如果服务器的缺省场所是en_US，你可以提供一个application.property文件，以及一个名为Application_en_US.property的副本文件。你可以将它用作是缺省的Application.property文件:

```
greetings = Hello
farewell = Goodbye
inquiry = How are you?
```

Struts框架将自动在其场所敏感的组件中使用缺省的ResourceBundle。参见13.3 ，关于如何载入缺省资源束。

ListResourceBundle 类 (java.util.ListResourceBundle) 可以用来载入任意的非字符串对象。ListResourceBundle的使用已经超出本章范围之外，但是如果要使用它，则该对象可以使用已经由框架管理的场所对象。 对标准的Java场所敏感的对象也是如此。

NOTE

因为历史的原因，Struts 1.0 和 Struts 1.1 没有使用来自于 java.util 包的实际 ResourceBundle 类。但是，Struts 的版本的工作方式是一样的。在开发你自己的应用时，你可以考虑类的互换。

13.1.5. MessageFormat

为了更加有用，许多消息必须包括运行时信息或者访问用户特定的数据。这些可能是记录数，当前日期、或者数量。在一个非国际化的应用中，我们通常将字符串和运行时变量合并以创

建这些消息，以及进行格式化。

对国际化的应用，简单的连接并不是一个好的选择。将要合并的语言条目可能跟据不同的用户而不同，不能进行硬编码。而是要将消息模版从资源束中载入，并且与其它信息合并在一起。*MessageFormat*类（`java.text.MessageFormat`）用来帮助你将消息模版和替换参数在运行时合并起来。

比如一个运行时信息像这样：

```
The disk named MyDisk contains 300 files.
```

可以基于这样的模版

```
The disk named {0} contains {1} files.
```

在运行时，应用会传递一组对象给合并它们的格式化器：

```
diskParameters[0] = (object) diskName;
diskParameters[1] = (object) fileCount;
formatter.applyPattern(messages.getString("disk.inventory"));
String output = formatter.format(diskParameters);
```

数组中的第一个对象和`{0}`合并，第二个与`{1}`标记合并，以此类推。

我们将在13.2中会讨论，Struts框架提供一些组件可以有助于你完成以下任务：

- 为每个用户管理场所对象
- 自动为每个支持的场所载入资源信息.
- 合并运行时参数到消息模版中

13.2. Struts的国际化组件

在创建Struts之前，Craig McClanahan 花了两年时间在比利时电信管理一个软件开发项目。许多Struts 的基本理念都发源于该项目，包括对国际化的基本支持。Struts的流行也体现在其开发团队之中，这些成员来自于Australia, France, Russia, Scotland, 以及美国。令人感兴趣的是，本书也描写了波尔多地区的特征，它离布鲁塞尔，Struts的祖籍，并不很远。

如表13.4，Struts 提供了一些国家化组件来帮助你本地化你的应用。

表格 13.4 Struts 国际化组件

每个用户的标准场所对象	会话场所属性
处理消息资源的框架类	MessageResources
自动载入应用的消息模版	缺省资源束
对错误和其他消息的特殊类进行排队	ActionErrors 和 ActionMessages
场所敏感的视图组件	JSP 标签

我们来看看它们如何一起工作的。

13.2.1. 会话场所属性

如我们在13.1.2种所见，所有标准的Java本地化工具都依赖于场所（Locale）对象。所以本地化的窍门就在于为每个用户维护场所对象。

Java servlet 框架提供了一个手段来为每个用户暂时存储对象，称之为会话上下文。默认情况下，Struts 将为每个用户存储 `Locale`属性在它们的会话上下文中，并在一个已知的关键字下。一开始，这个属性被设置为服务器的缺省locale。如果你的应用用某个用户场所替换了这个对象，框架中的场所敏感对象将自动根据用户场所选择消息而不是根据缺省场所。

NOTE

但是 ServletRequest.getLocale()如何呢?它返回用户的浏览器设置的场所(通过 HTTP Accept-Language header 传递).

不幸的是，这一设置超出了应用的直接控制，并不能保证它被设置正确。作为备选方案，Struts 在会话上下文中提供了一个 locale 对象，你可以在应用中进行控制。

当然，本地化并不是很神奇的。应用必须为每个场所提供一套消息资源，并且提供选择和更新用户场所的机制。

13.2.2. MessageResources

为了获取消息，Struts提供了一个*MessageResources*类(`org.apache.struts.util.MessageResources`)。Struts 开发人员很少直接调用这个对象，但是可以通过其他类来幕后使用 MessageResources 返回相应的消息。

MessageResources 描述了一个检索场所敏感消息的API。API 也可以使用MessageFormat 类(`java.text.MessageFormat`)来在运行时合并参数和格式化消息。MessageResources 接口并没有指定如何以及在何处存放消息，仅仅包括如何检索它们。其缺省实现使用标准的`PropetyResourceBundle`，但是，你也可以使用其他的存储机制，比如XML 文件或者SQL数据库，都可以使用。

典型地，消息可以通过JSP 标签或者其他表现系统来进行检索。Action 也可以通过调用消息资源来准备一个本地化消息。下面是一个例子，它从会话上下文中检索用户场所，传递到

MessageResources以获得场所的相应消息，并将其存放在一个bean 中以备后用。

```
Locale locale = (Locale) session.getAttribute(Action.LOCALE_KEY);
MessageResources messages = servlet.getResources();
String message = getMessage(locale,"important.message");
((messageForm) form).setMessage(message);
```

getMessages 方法签名被重载，以便能够进行5个参数替换，参数s({0}...{4}:

```
String message = getMessage(locale,"important.message",new Date());
```

参数通常是字符串，但是也可以是任何Java 对象。当使用其他对象时，MessageFormat 类也可以有助于格式化这些对象。参见13.3获取更多信息。

如果getMessage 方法中忽略了Locale 对象，这时，将使用服务器的缺省 locale对象。

13.2.3. 缺省资源束

框架使用MessageResources 的一个实例来提供对缺省资源束的访问，该实例由控制器servlet自动载入。包中的每个资源在第一次需要的时候被载入，然后保留在内存之中。其他 Struts i18n 组件将从这些包中检索消息模版，除非另有指定。

因为缺省情况下，Struts使用标准的属性文件和标准的PropertyResourceBundle，所有的常用特征都可以使用于格式化你的消息字符串，如表13.5所示。

13.2.4. 格式化消息

因为使用标准Java库，格式化数字、日期、和货币的常规技术都可以用在你的Struts 消息的格式化中。例如，如果一条消息涉及到日期，资源束的属性文件可以读取：

```
detected.ufo = At {2,time,short} on {2,date,long}, we detected
{1,number,integer} spaceships on the planet {0}.
```

一个单独的 date 属性会被传递到消息模版作为第2个参数。然后被用来在另外的地方显示日期和时间。

表格 13.5 资源属性文件的格式化特征

特征	模版	例子
字面字符	****	**WEB INF\\lib**
单引号	**''**	**Struts'' ancestral home**
特殊符号	**{#,type,style}**	**{2},real,currency**

如果用户的场所是en_US:

```
At 1:15 PM on April 13, 1998, we detected 7 spaceships on the planet
```

```
Mars.
```

如果有哪个户的场所是de_DE:

```
Um 13.15 Uhr am 13. April 1998 haben wir 7 Raumsch如果fe auf dem Planeten
Mars entdeckt.
```

在Action 类中，Date 对象可以传递给ActionError或者ActionMessage:

```
Date today = new Date(System.currentTimeMillis());
messages.addActionMessages.GLOBAL_MESSAGE,
new ActionMessage("detected.ufo",ufoCount,today,ufoPlanet));
saveMessages(requestmessages);
```

13.2.5. 显示特殊字符串

Java 平台是完全支持基于Unicode的，可以处理任何语言的字符。但是，Java编译器和其他 一些Java 工具则只能够处理包含Latin-1 [ISO 8859-1] 字符或者Unicode编码字符的文件。Unicode编码的字符表示为 /u 开头的转义符号，后面跟着字符的Unicode 字符集编号—例如， /u0084。

定义

Unicode 提供为每个字符提供了一个唯一的编码，不管是什么平台，什么编程环境和语言。在现代很多操作系统、浏览器和许多其他产品都支持这种编码。Unicode 标准的出现，以及支持他的相关工具，是软件全球化的一个趋势体现。 [Sun, i18n]

为了帮助转换非Latin-1 编码的文件，JDK 提供了一个工具*native2ascii* 。该工具转换一个原生编码的文件(非-Latin 1 和非Unicode编码)为一个Unicode编码的文件。尽管Unicode编码对人来说是没有意义的（不可读），Java 组件并没有给与它更多的用途。

你也可以使用Java 国际化和本地化工具包Toolkit 2.0 [Sun, JILKIT]中的消息工具来做同样的事情。工具包中还有一些其他的工具，你会发现在你大规模翻译文件时非常有用。如果你准备管理大量的本地化资源文件，最好下载它们。

13.2.6. ActionErrors

就如在本章开头的介绍中所讲， Struts 工作流(提交/校验/纠正/提交) 是框架的一个基本组成部分。当进行校验时，验证方法需要一种途径可以将错误消息传递给输入表单，从而显示给用户。*ActionErrors* (org.apache.struts.Action.ActionErrors) 对象就是用来携带这些消息的。

因为在验证时可能会产生不止一个错误，ActionErrors 是一个集合。可以在其中排列多个错误，并一次性显示给用户。每个消息可以有选择的和表单中的某个项目相关。这给了开发人员已将错误信息显示在相应属性旁边的机会。

当然，ActionErrors 是一个ActionError 对象(org.apache.struts.Action.ActionError)的集合。每个错误对象都有其消息关键字以及可选的property名称。关键字用作来从资源束中查询消息(见13.1.2)。要添加一个错误消息到队列中，可以使用这样的调用：

```
Errors.add(ActionErrors.GLOBAL_ERROR, new
                          ActionError("prospect.statusDate"));
```

其中，prospect.statusDate 是资源束中的关键字。也可以在参数中传递和合并消息字符串：

```
Errors.add(ActionErrors.GLOBAL_ERROR,new
                       ActionError("record.updated",recordId));
```

当添加了所有的错误后，集合可以被在一个已知的关键字下存储到请求上下文中。Action的saveErrors 方法可以完成这个任务：

```
saveErrors(request,errors);
```

因为使用关键字和资源束，ActionErrors 的本地化支持是内置的，它可以自动根据用户的场所显示正确的信息。

13.2.7. ActionMessages

ActionMessages (org.apache.struts.Action.ActionMessages) 是在 Struts 1.1 中连同一个对应的 JSP 标签引入的。ActionMessages 旨在代替原本的 ActionErrors，但，为了向后兼容，原本的 ActionError 类也保留了下来。

那么，ActionError 有什么问题呢? <html:errors>标签的设计是鼓励开发人员将 HTML 标记放在消息文件中。但这会在消息被用在不同的上下文时可能会出现问题，比如在 JavaScript 中。它也往往会把翻译资源文件的翻译们搞糊涂。

ActionMessages 标签 (org.apache.struts.taglib.html.MessagesTag) 提供了一个附加功能，使开发人员并不需要将标记和消息混合起来。 (参见第 10 章，获取关于 Struts JSP 标签的更多信息)。

因为它们实际上是ActionErrors 集合的增强版本，缺省情况下，消息标签将使用ActionErrors队列。所以非常容易在原先使用ActionErrors的地方使用ActionMessages 。

如果你喜欢传递消息和错误，也可以使用单独的ActionMessages队列。要访问这个额外的队列，只需要调用saveMessages而不是saveErrors：

```
saveMessages(request,messages);
```

然后在JSP标签中指定messages=true:

```
<html:messages id="message" messages="true">
   <bean:write name="message"/><BR>
</html:messages>
```

消息队列也可以用来向页面发送确认提示信息，比如“ 记录已插入 ”或者“ 消息已发出 ”等等。这使得错误队列打开一个实际的“ *Houston, we have a problem* ”的提示。同一个页面也

可以使用消息标签(见前面的代码片段)，也可以使用下面的标记在页面的其他地方显示错误提示：

```
<html:messages id="error" messages="false">
   <bean:write name="error"/><BR>
</html:messages>
```

在这个例子中，请注意当使用<bean:write> 标签来输出本地化数据时，它并不是场所敏感的组件。<html:messages> 标签从ActionMessages或者ActionErrors 对象中取出本地化数据，它从缺省的MessageResource 对象(ResourceBundle)中依次检索它们。

在13.3中，我们看到了可以使用标准组件来输出本地化数据的其它方法。有好些Struts 标签都具有场所敏感的特性，我们在下节13.2.6中讨论。

13.2.8. 场所敏感的Struts JSP标签

我们在第10章中讨论了使用Struts JSP 标签的基础。如果你是刚开始接触Struts标签，请回过头去看看第10章。这里我们讨论的是Strut 标签的本地化特征。

如表 13.6所示，Struts 标记库包括各种场所敏感的标签。这些标签可以从资源束中检索文本消息，并为实时用户显示定制的信息表现。

表格 13.6 场所敏感标签

bean:message
bean:write
html:errors
html:html
html:image
html:img
html:messages
html:option
html:button
html:cancel
html:checkbox
html:file
html:hidden
html:multibox
html:password
html:radio
html:select
html:submit
html:text
html:textarea
html:form
html:link

这些标签都简单地提供了关键字，以便从资源束中检索替换文本和项目的建议标题。图像相关的标签接受额外的关键字属性，以便允许图像的来源和替换文本都可以被本地化。

表 13.7 总结了这些场所敏感标签中的通用的属性名称。使用这些属性的标签列出为说明的一部分：

表格 13.7 场所敏感标签中的公共属性名称

属性	说明和标签
arg0 **...** **arg4**	**数字化的替换值，如果有的话[message write]**
Bundle	**应用范围的 bean 的名称，在其下存储了包含消息的 MessageResources 对象。如果没有指定，将使用缺省值. [message write image img option errors messages]**
key	**被请求的消息的关键字,必须在消息资源中有一个对应的值。如果没有指定，关键字将从 name 和 property 属性中获取。[message write image img option]**
Locale	**会话范围 bean 的名称,在其下存储了当前选择的场所对象。如果没有指定，将使用缺省值 [message write image img errorsmessages]**
altKey,titleKey	**替换文本和建议项目标题的消息资源关键字。[button cancel file hidden image img multibox password radio reset select submit text textarea]**
titleKey	**建议标题的消息资源关键字(无替换文本) [form link]**
pageKey,srcKey	**标识输入标签图像、或者应用相关的路径、或者源 URL 的消息资源关键字. [image img]**
formatKey	**为一个格式化字符串标识一个关键字，以便能够从应用资源中检索 [write]**

我们来仔细看看几个有趣一些的标签`<bean:message>`, `<bean:write>`,`<html:errors>`, `<html:messages>`, `<html:html>`, `<html:image>`, `<html:img>`, 以及`<html:option>`。

<bean:message>

`<bean:message>` 是处理本地化消息和标注的主要标签。虽然其他标签也可以处理错误和确认消息， `<bean:message>` 是最常用在HTML 表单中的标签:

```
<html:cancel>
    <bean:message key="button.cancel"/>
</html:cancel>
```

<bean:write>

在 Struts 1.1 中， <bean:write> 增加了 format 属性。这样就提供了应用标准化模版到将要输出的值的机制。

```
<bean:write name="inputForm"
            property="amount"
            format="$#.##"/>
```

可以提供一个formatKey 属性来从资源束中读出模版：

```
<bean:write name="inputForm"
            property="amount"
            formatKey="pattern.currency"/>
```

如果要使用另一个场所对象而不是框架默认的场所，可以提供一个locale 属性。locale 属性指定了会话范围属性的名称：

```
<bean:write name="inputForm"
            propety="amount"
            formatKey="pattern.currency"
            locale="myLocale"/>
```

STRUTS TIP

不要使用定制标签来格式化数据。相反，可以在 ActionForm 或者(最好)在业务层中本地化数据。你可以轻易地将 Struts Locale 对象传递给 ActionForm (或者你的业务 bean)然后在 getter 方法中进行本地化。要在 ActionForm 中存储 Local 对象，添加一个 locale 属性并将下面的代码添加到 reset 方法中:

```
HttpSession session = request.getSession();
If (session!=null) {
  setLocale((Locale)
  session.getAttribute(Action.LOCALE_KEY));
}
else {
  setLocale(Locale.getDefault());
}
```

你就可以和标准的 MessageFormat 对象一起使用 Local 对象了。关于如何使用显示属性到转换输出，参见第 5 章.

<html:errors> and <html:messages>

<html:errors> 和<html:messages> 将自动根据用户场所显示未决的错误信息。本地化这些信息不需要额外的精力，它已经可以自己做好这些事情。关于使用<html:errors> 和<html:messages> 标签，参见第 10 章。.

<html:html>

许多浏览器（或者用户代理）也支持本地化。国际化的用户端代理可以本地化的东西包括：

- 高质量版式的象形变化
- 引用标号
- 决定连字符和空格等

为了指示用户代理页面中正在使用何种语言，你可以在<HTML> 标记中提供一个lang 元素。相应的<html:html> JSP 标签会在Struts 本地化激活时自动创建这些标记。

这可能会导致

```
<html:html> -> <html lang="en">
```

或者

```
<html:html> -> <html lang="fr">
```

依赖于用户的具体场所。

特殊情况下，你的页面应该没有这些标签也可以本地化。但包含这些标签也不是坏事，可以为你节省一些麻烦。

<html:image> and <html:img>

web 应用的图像也可能会包含一些语言信息。图像本身也可能和某种特定的区域和文化相关。<html:image> 和 <html:img> 标签提供了有助于本地化应用中的图像和图像按钮的属性。

在本地化图像和图像按钮时，需要考虑以下3个主要问题：

- 二进制通图像文件的源地址
- 图像的替换文本
- 项目的建议标题

NOTE 当应用于图像时，alt 和 title 属性是交替使用的。参考 HTML 规范 [W3C, HTML]获取关于如何使用这些属性的指南。

上面每一个项目都有一个属性来在 <image> 和<img> 标签中表现，如表13.8所示。page 属性允许以应用相关的格式给出图像的URL。

表格 13.8 alt, title, src, 和 page 属性

属性	说明

alt	**元素的替换文本**
Title	**元素的建议标题**
src	**图像的 URL 源**
page	**图像的应用相关 URI**

为了支持本地化，有一套属性可以用来从资源文件中根据关键字读取文本和路径。表 13.9 列出了这些属性。

表格 13.9 altKey, titleKey, srcKey, 和 pageKey 属性

属性	说明
altKey	**元素替换文本的消息资源关键字**
titleKey	**元素建议标题的消息资源关键字**
srcKey	**图像 URL 的消息资源关键字**
pageKey	**图像应用相关 URI 的消息资源关键字**

这些属性可以象相应的属性一样被指定:

```
<html:image pageKey="images.sign"
            altKey="images.sign.alt"
            titleKey="images.sign.title">
```

这一标签将根据用户的场所自动从消息资源中按给定的关键字插入相应的属性。这使得你可以为每个场所指定相应的图像文件，以及翻译过的`alt` 和 `title` 元素。

<html:option>

<html:option> 标签通常接受一个 text 和 a value 属性。text 属性被显示给用户。如果选择框被选择，则 value 会被在请求中提交。为了本地化 <option> 标签，你可以指定一个关键字来替换文本中的位置。这个关键字会自动检索资源文件然后插入用户场所相对应的消息。

其他属性

这些标签（`message`, `write`, `errors`, `messages`, `image`, `img`, 和`option`）都还可以接受另外两个属性。 (如表 13.10)， 可以用来优化本地化的效果。

提供这些属性是因为有时候开发人员会为不同的项目使用不同的资源束，比如图像。有时候也可以使用替换的场所。另外，这些属性在开发人员从其他本地化应用中迁移到Struts上时比较有用。

表格 13.10 bundle 和 locale 属性

属性	说明
bundle	**一个应用范围的针对可选 MessageResources 对象的属性名称**
Locale	**会话范围的针对可选场所对象的属性名称**

13.3. 本地化Struts应用

我们现在来综合所有东西来看看本地化一个Struts 应用的流程。

13.3.1. 激活本地化

Struts 本地化要检查 3 个地方：

- Servlet 的 locale 参数是否设置正确？
- 缺省的应用资源束是否设置正确？
- 是否所有本地化的页面都使用了<html:html> 标签？

13.3.2. 设置 locale servlet 参数

缺省情况下，Struts 框架将为每个用户会话提供一个Locale 对象，这样可以被那些场所敏感的组件所使用。这些对象是否自动创建，受到的locale 参数的控制，该参数可以在web应用部署描述符中配置 (见第4章)：

```
<init-param>
  <param-name>locale</param-name>
  <param-value>true</param-value>
</init-param>
```

缺省设置为 true，允许本地化特征。

13.3.3. 设置应用资源束参数

Struts 的场所敏感组件依赖于MessageResources类，该类又依赖于框架的 Locale 对象和缺省的应用资源束。应用资源文件的位置也作为是Struts ActionServlet的参数之一。缺省值是空字符串，所以指定这个参数非常重要。

```
<init-param>
  <param-name>
    application
  </param-name>
  <param-value>application</param-value>
```

```
</init-param>
```

这就告诉 ActionServlet 去在 CLASSPATH 中查找名为 application.property 的文件。如果某个特定点场所被请求， ActionServlet 就会查找名为 application_xx_XX.property 的文件，这里_xx_XX 就是给定的场所语言和国家代码，比如 _fr_CA 或者 _es_US (参见 13.2).

你也可以指定缺省的资源束为某个java包的一部分：

```
<init-param>
  <param-name>applicatipn</param-name>
  <param-value>resources.application</param-value>
</init-param>
```

这意味着你的资源文件可以在下面位置找到：

```
<app-context>/WEB-INF/classes/resources/application.property
<app-context>/WEB-INF/classes/resources/application_fr_CA.property
<app-context>/WEB-INF/classes/resources/application_es_US.property
```

或者在 WEB-INF/lib 文件夹中的JAR包中。

通常，你可能要将它们和其它的资源文件放在一起，你应该将拷贝这些文件到classes 目录中作为Ant 构建过程的一部分。如果你的源文件在 /WEB-INF/src/java下面，Ant 任务就可能看来是这样：

```
<target name="resources">
  <copy todir="classes" includeEmptyDirs="no">
    <fileset dir="src/java">
      <patternset>
        <include name="**/*.property"/>
      </patternset>
    </fileset>
  </copy>
</target>
```

这就将所有源文件包中找到的 *.property文件拷贝到classes文件夹中。

如果property文件放在/WEB-INF/之外，如果对资源属性文件发生了修改，一定要记着重新构建应用。更新的文件必须部署在应用能够找到类的地方。

而更新了资源束后，是否会自动载入新的资源束依赖于你的容器。

STRUTS 使用 Ant 构建和部署你的应用。除了编译 Java classes, Ant 也可以拷贝资源，配置,以及 JSP 文件从源代码树到部署的目标书中。参见第 15 章的

TIP Artimus 应用。

参见第4章，有关于 ActionServlet 参数和Ant构建文件的信息。

13.3.4. 使用框架的Locale对象

当本地化激活时， Struts ActionServlet放置一个 Locale 对象在用户的会话上下文中。对象被放于此处是为了整个应用都访问。

我们在13.2种解释过，Struts i18n 组件将自动使用这个对象，你自己的组件也可以。

13.3.5. 检测用户场所

你可以在Action中获取框架的Locale对象，如

```
Locale locale = request.getSession()
                        .getAttribute(Action.LOCALE_KEY);
```

如果你的Action 是基于Scaffold 包中的BaseAction 类，你也可以这样使用：

```
Locale locale = getLocale(request); // BaseForm 版本
```

你可以将这个对象传递给任何场所敏感的方法，包括在标准的`java.text` 包中的方法。这使得很容易在Action中创建本地化数据并将它们传递到表现层，以准备显示。

13.3.6. 改变用户场所

在用户会话首次被创建时，ActionServlet将会为用户简单地创建一个缺省的 Locale 对象。如果用户不在应用服务器的缺省场所之中，你的应用将需要替换这个对象。因为Locale 对象是不可修改的，所以必须重新用一个新的对象来替换它。当然，最好的地方是在Action中来完成，可以使用这样的代码：

```
Locale locale = new
             Locale(myForm.getLanguage(),myForm.getCountry());
HttpSession session = request.getSession(true);
session.setAttribute(Action.LOCALE_KEY,locale);
```

如果你的Action 是基于Scaffold 的BaseAction，你可以这样：

```
Locale locale = new
              Locale(myForm.getLanguage(),myForm.getCountry());
setLocale(request,locale);
```

如果 你的应用仅仅只须本地化语言，可以让国家参数空着。

13.3.7. 使用Struts场所敏感组件

Struts 本地化特征激活时，并且用户的框架 Locale 对象属性设置好了时，Struts 就可以自动地本地化相关的数据。参见13.2 关于使用这些组件的详细信息。

13.3.8. 将标志和消息放在资源属性文件中

缺省下，Struts 使用标准的属性文件来包含应用中使用的消息和标注。这是一些简单的文本文件，可以在任何文本编辑器中进行编辑。

文件本身是一个名值对的列表。关键字就是应用传递到其请求之中的东西。值就是返回的标注或者消息。

13.3.9. 创建特定语言的属性文件

你的应用应该为每一个支持的场所准备一个属性文件。如果命名正确，这些文件会被自动载入。你只需要将各条目翻译后，存储在应用能够找到的一个新文件中。如果应用不能在用户场所的资源中找到某个关键字，它会使用却身资源束中的来代替。

13.3.10. 在本地化感知的组件中指定一个相应的关键字

如13.2节中所述，Struts 的场所敏感组件可以接受一个到资源束中的关键字，并输出当前场所的相应标志和消息。更多信息，参见13.2.

13.3.11. 使用<bean:message>和其他组件

对那些没有内置本地化的组件，你通常使用`<bean:message>`来提供本地化消息。例如，为了创建一个本地化的取消按钮，可以使`<bean:message>` 来提供值:

```
<html:cancel>
   <bean:message key="buttons.cancel">
</html:cancel>
```

对于德国的用户，按钮的文本会处理成*Abbrechen.*，而对于挪威的用户，则可能是*Kanseller*。

13.4. 本地化其他组件

为完成你的国际化任务，你必须得本地化应用的其余部分。包括Struts 插件，比如 Struts Validator和Tiles，以及你的Action 可能会传递给HTML 元素的集合。

13.4.1. 本地化Struts Validator

在 第 12 章所讲的 Struts Validator 中，用户使用一个 XML 配置 文件来校验用户输入。这个配置可以用来通过 JSP 标签产生 JavaScript 。相同的配置也可以用在 Struts ActionForm 的 validate 方法中。

Validator 的配置文件由一些 FormSet 元素组成，对应于 Struts 配置中的 form-bean 元素。每个 FormSet 有一个字段集合组成。每个字段元素可以有其自己的校验设置。这些设置包括当字段元素校验失败时要显示的标注和消息。

所有Validator使用的标注和消息都可以连接到Struts消息资源，并可以自动进行本地化。如

果需要额外的本地化，比如验证邮政编码或者电话号码，你也可以定义一个特定场所的 FormSet 元素。一个场所特定的FormSet 元素就像缺省 FormSet的一个子集—你可以仅仅定义需要改变的字段。

如果一个缺省的 FormSet 是这样：

```
<form name="registrationForm">
  <field property="Name" depends="required">
    <arg0 key="registrationForm.name.displayName"/>
  </field>
  <field property="address" depends="required">
    <arg0 key="registrationForm.address.displayName"/>
  </field>
  <field property="postOffice" depends="required">
    <arg0 key="registrationForm.postOffice.displayName"/>
  </field>
  <field property="postalCode" depends="required,mask">
    <arg0 key="registrationForm.postalCode.displayName"/>
    <var>
      <var-Name>mask</var-Name>
      <var-value>^\d{5}\d*$</var-value>
    </var>
  </field>
</form>
```

应用可以为 postalCode 字段提供一个预备的校验器，就像这样：

```
<form name="registrationForm" locale="fr" country="CA">
  <field property="postalCode"
    depends="required,mask">
    <arg0 key="registrationForm.postalCode.displayName"/>
    <var>
      <var-name>mask</var-name>
      <var-value>^[a-zA-Z]*$</var-value>
    </var>
  </field>
</form>
```

关于Struts Validator的详细信息，参见第12章。

13.4.2. 本地化Tiles

我们已经在第11章讨论了Tiles 框架。你已经看到，强大的Tiles 定义特征允许你从更小的页面碎片中创建高阶的应用页面描述。你可以将定义存储在一个XML 配置文件之中，就像Struts的配置文件或者Struts Validator的配置文件一样。

就如同你可以为某个特定的场所创建一个属性文件，你也可以为每一个场所创建一个Tiles配置文件。它们都遵循相同的约定。如果你需要创建一个针对加拿大法语场所的Tiles配置文件，可以命名为 `tiles_fr_CA.xml`，并存储在缺省的 tiles.xml 文件的地方。

在针对特定场所的文件中，你只需要替换改变的定义部分就行了。

如果应用在配置文件中没有发现针对用户场所的定义，将使用缺省配置文件中的定义。

这种特征使得非常容易创建整个子站点，而它们都是场所特定的。这时，你只需要为每一个场所创建一个 Tiles 定义文件，然后修改定义的路径以指向每个Tile的本地化版本。这样，你可以重用布局（layout）tiles，以便能使整个站点具有一致的外观和感觉，但是需要改变所有的内容tile，以便能在每个场所下都能正确访问。

13.4.3. 本地化集合

有些 Struts JSP 标签基于集合向用户显示选择列表。因为这些集合通常从Action传递给页面，Action就有机会能够在将其传出之前对它们进行本地化。

<html:options>

`<html:options>` 标签支持两个并行的列表：一个是值（可以被该字段返回给服务器）而另一个是标注(即用户在选择框中见到的)。问题是如何以一种场所敏感的方式选择不同的标号而不需要修改其值。

在Action中，为每个value 组件在Request属性下创建一个名为`values`的数组，或者ArrayList (`java.util.ArrayList`) 。再构建一个对应的`labels` 数组或者ArrayList，用来根据用户场所查找MessageResource 。

MessageResource 可以从servlet中获取：

```
org.apache.struts.utils.MessageResources resources =
                                          servlet.getResources();
```

而场所则可以从请求中获取：

```
Locale locale = equest.getSession().getAttribute(Action.LOCALE_KEY);
```

它们一起使用来为每个值查找本地化消息：

```
String[] messages = new String[keys.length];
for (int i=0; i<keys.length; i++) {
  messages[i] = resources.getMessage(locale,keys[i]);
}
```

为了减少编程麻烦，Scaffold 包在MessageUtils 类

(org.apache.scaffold.text.MessageUtils)中提供了一个 getMessages 方法来做同样的事情：

```
String[] messages = MessageUtils.
                            getMessages(resources,locale,keys);
```

如果你想有另外的方法，也可以使用一个类似的方法，可以放回一个LabelValueBeans的集合：

```
ArrayList labelValueBeans =MessageUtils.
                      getLabelValueBeans(resources,locale,keys);
```

关于 <options> 标签和LabelValueBeans，参见第10章。

<html:multibox>

<html:multibox> 标签也使用集合得知来产生一系列checkbox 元素。如果值和标号匹配，你可以这样来产生一系列checkboxes：

```
<logic:iterate id="item" property="items">
  <html:multibox property="selectedItems">
    <bean:write name="item"/>
  </html:multibox>
  <bean:write name="item"/>
</logic:iterate>
```

如果 需要为每个checkboxes本地化标注，你也可以使用 MessageUtils.getLabelValueBean() 来创建本地化标签，可以这样使用：

```
<logic:iterate id="item" property="labelValueBeans">
  <html:multibox property="selectedItems">
    <bean:write name="item" property="value"/>
  </html:multibox>
  <bean:write name="item" property="label"/>
</logic:iterate>
```

13.5. 小结

在本章，你学到了为什么有那么多开发人员要本地化它们的应用，以及Java国际化是如何工作的。因为Struts直接构建在Java的国际化特征之上，本地化Struts 组件只需要费非常小的精力。流行的Struts 插件，Jakarta Tiles 和 Struts Validator也可以很容易地本地化。同样的技术也可以用来本地化其它Struts 组件和对象，比如集合Collection之上。

本章仅包含了Java 国际化特征的主要内容，关于详细的信息，参见Sun Java Tutorial, Internationalization Trail [Sun, i18n]。

14. 在 Struts 中使用数据服务

本章包括

- 理解为什么以及应用如何使用数据库以及其它数据服务
- 应用中集成数据服务
- 连接到数据服务时使用层
- 定义应用的业务层
- 连接数据访问组件到业务层
- 连接其它数据服务，比如搜索引擎和内容聚集到业务层

There is a tendency to mistake data for wisdom.

—Norman Cousins (1912–1990), American editor, author

14.1. 加快步伐

为了保存和检索数据，现在的应用都需要访问各种的其它系统维护的数据。每个系统都有其自身的方法，协议来进行数据传输和存储。还包括使用 LDAP 进行认证以及使用 XML 进行内容发布等等。

今天的应用也可能需要针对统一数据提供不同的透视图。除了标准的数据库检索之外，现在的应用可能还需要进行全文检索能力，以便用户可以大海中捞针，找到湮没于森林中的一棵树。应用也可能希望与其它系统共享其珍贵的内容，通过 RSS 技术则可以办到。

本章向你展示了你的应用如何可以连接到上述的各种数据服务，而不用对应用设计做出任何折衷。我们在一个实例应用中展示了各种特殊的例子，以使用数据库，搜索引擎，内容聚集等等技术，但是这些真实的技术可以适用于其它数据服务。这些策略遵循在整个 Struts 框架中使用的经典的层模式[POSA] 。

我们来看看我们所熟悉的二人转——JDBC 驱动和数据库——如何符合层模式。

14.1.1. 从模式的角度来看 JDBC

从模式的角度来看，常规数据库使用的 JDBC driver 是集成层的一部分，虽然实际的数据库是在资源层。

对开发人员来说，驱动和数据库之间的交换是透明的。这看起来好像应用是直接和数据库交谈，但是事实上，Java 在这两个组件之间提供了一个很薄的层。这种安排示于图 14.1。此方法的优点是开发人员可以改变数据库而不用关心数据库如何同驱动通信。所有 JDBC drivers 都以相同的方式接受 SQL 语句，即使其下层的数据库能够执行。

14-1 应用和驱动通信，然后驱动和数据库通信

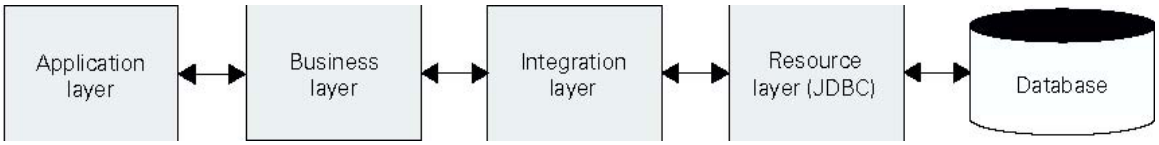
揭示业务层

JDBC 驱动所用的 *driver:device*, 或者 *façade* 模式是一种策略，你也可以在集成数据服务时使用。通常，你的应用所想说的就是，“*我有这些数据，请给我细心照管*”或者“*此标识符下还有那些记录*?”设计目标是提供一个应用可以使用的接口（或者驱动），而不用知晓太多数据接口后面所隐藏的细节。

应用中的那些经常说“这是我所有的，以及这是我想要的”的部分通常称为是业务层，或者程为模型（Model） (MVC 中的 M)。这也是你的应用所不同于其它应用的本质部分。

正如一个连接到 JDBC driver (其连接到数据库)的 Java 应用，大多数学者[Go3] 建议你应该使你的应用层连接到业务层，然后再由业务层连接到集成层。集成层访问数据库，返回结果给业务层，如图 14.2。

图表 7 Jack 构建的[Jack]应用。应用连接到业务层，而业务层连接到集成层，集成层又连接到资源层 (JDBC)，最后资源层连接到数据库。



层模式的特点是，位于某个层的类只能和位于该层的类或者其相邻层中的类进行交互。通过对代码进行分层，减小了应用间的耦合，应用系统显示更加容易维护和增强[Ambler]。我们在此章中将这个原理应用到实践中，将不同的数据服务连接到同一个应用。

当然，就像许多很好的建议一样，创建分离的业务层和集成层通常在实际应用中被忽略。许多应用都直接连接到资源层。业务层和集成层仍旧在此，只是淹没在了 SQL 语句之中。结果只见树木不见森林。

14.1.2. 数据服务介绍

基于本章的目的，我们会遵循最佳实践，使用层模式使 Struts 连接到不同的数据服务。我们从业务层的运行示例开始，然后将其连接到集成和资源层的不同实现之上。

首先，我们使用来自于 Scaffold 包的 StatementUtils 和 ResultSetUtils 类连接到 JDBC 数据库服务。这是一种简单的基线策略，它使用反射和元数据来传输一个 SQL ResultSet 到一个 f JavaBean 的 collection 中。你必须自己定义 JavaBean 和 SQL 语句，但是系统是很容易理解和使用的。

然后，我们将步出“数据库圈子”然后连接到 Lucene，一个流行的全文检索引擎。Lucene 是 Jakarta 的另一个开源项目。将 Lucene 添加到资源层是一个重要的概念证据。我们这里展示一个良好的分层设计使你能够将新的服务集成到你的应用中而不会对其有所影响。

最后我们会展示继续添加其它服务，比如 Rich Site Summary (RSS)，到应用中是多么的容易。RSS 对你的应用有选择地和其它应用共享信息是一个非常好的方式，并且已经是一个 web services 标准。

我们整个这章的主要精力是用于将这些产品整合到你的业务层的技术，而不是产品本身。Struts 框架是模型中立的，可以用于任何通用的数据访问工具。你可以根据原样部署这些例子，或者将它们作为联结你自己的数据访问服务方案的指南。

14.2. 业务层详解

在本书的前言部分，我们讨论过 Model-View-Controller (MVC) 模式。使用这个术语，业务层是包括前述资源层和集成层的模型的一部分。开发项目开始于设计各种类型的业务层(即使没有实现它)然后连接到资源层和集成层(即使它们全部混杂在一起)。因此，我们首先讨论业务层的输入输出然后将其连接到数据库系统。

14.2.1. Struts—拿出你自己的模型

我喜欢说 Struts 是一个 BYOM (Bring Your Own Model) 框架。它将数据从 HTTP 层带出后，就不管了。将数据带到模型，再返回来则是框架留给开发人员的作业。Struts User Guide [ASF, Struts] 推荐我们使用 *business logic bean，也称业务对象，* 填补这个空间。但是，什么是业务对象呢？在哪里？不要惊奇，它们是业务层的鼓吹者和宣扬者。

14.2.2. 定义业务对象

我们已经在第 8 章接触过业务对象了，作为 Action 对象讨论的一部分。通常，业务对象可能是应用中最难以描述的部分，因为它们就是应用。它么描述了你的应用之所以和其他任何应用不相同的逻辑：你的业务 API。

为了定义业务对象,你必须首先隔离你的应用的核心业务逻辑。如果你的应用依赖一个 JDBC 数据库，那么你的大部分 API 可以通过你的 SQL 查询来进行表达。如果这样，你的业务对象将看起来就像是一个你的应用使用的（或者将使用的）查询的清单。如果你是用预先准备的语句，在语句的可替换参数和到你的 Bean 的方法的参数之间也具有很紧密的关系。

下面是一个来自于 Poll 应用的示例查询。它用于区分来自某一个给定 IP (host)的人是否已经在在线 poll 中投过票了:

```
public static final String RESP_VOTED =
"SELECT COUNT(*),poll,host FROM polls_resp " +
"WHERE poll=? AND marked=0 AND host=? GROUP by poll;";
```

然而，从业务逻辑的角度，我们并不真正关心 SQL 命令看起来像是什么。我们只是想知道某一个投票者是否有资格对一个给定的调查投票。判断投票是否合格的方法的方法体可能是这样：

```
boolean isEligible(String poll, String host);
```

这个方法只使用两个`String`,一个用来标识 poll 而另一个用来标识投票者。这里没有提到 SQL 或 HTTP，只是说明了谁（Who）以及什么(What)的问题。

从 web 应用的角度来看， *who* 通常是一个 IP 地址(或者 RemoteHost)。而 *what* 则是可以在一个数据输入表单中传入的什么东西(可以表达为一个 Struts ActionForm)。这里是 Struts 应用可以如何调用 `isEligible()` 业务方法。记住该方法只是查找字符串；但它并不关心它们来

自于何处：

```
boolean voted = Access.isEligible(
   pollForm.getPoll(), request.getRemoteHost()
);
```

这个调用可以发生于 Struts Action 类中。Action 只是从 ActionForm 和 request 中获取需要的数据，它们都属于 web 层，然后将数据传递给业务对象。Action 对于业务对象将用这些数据来干什么或者“eligible” 到底是什么意思并不清楚。它只知道它们需要传输这些 `String`，捕捉 `boolean` 结果，并且观察是否出现异常。如果方法返回 false，Action 可以做某一件事情。如果方法返回 true，Action 则可做另外一些事情。但是这就是 Action 所需要知道的全部内容。

同样的模式也适用于任何使用这个业务对象的客户。客户采集数据并且解释结果，但是有关对结果的决定则委托给另一个对象。

*嗯，这就是**业务代表模式（** Business Delegate pattern）吗?* 是的，这种策略就是在 *Core J2EE Patterns* [Go3]中描述的业务代表模式的例子。

那么它适用于 EJB 吗？ 是的。你可以改变业务方法的实现来使用 EJB，而应用的其它部分并不知道这些不同。而业务方法的数据需求是相同的，不管使用何种技术来访问持久性数据。

14.2.3. 设计业务对象

作为一个有效的中间人，业务对象应该避免对其它类的依赖，并且应该尽可能地自包含(比如，它应该是弱耦合的)。理想情况下，业务方法应该使用普通的 Java 对象（POJO）接受和返回属性，并且抛出自己的一套异常。

所以，业务逻辑 bean 应该：

- 表达应用系统的核心逻辑—其 API
- 尽可能的使用原生 Java 类型和类作为参数
- 定义其自己的异常类
- 暴露最小的对其它类的依赖性

为什么是 bean 呢? 业务对象并不是一定要真的遵循 JavaBean 规范。设计的其它组件远比对象的调用规范更加重要。但是如果你目前创建了一个旧式的类，那么要使其遵循 JavaBean 设计模式是很容易的，这可以达到事半功倍的效果。关于设计 JavaBean 的更多信息，参见 Sun 的 JavaBean Tutorial Trail [Sun, Trails] 和 JavaBean Specification [Sun, JBS]。

为何要抛出自己的异常? 首先，是为了提供封装性。如果我们抛出一个 SQL 异常，我们就暴露了对 SQL 的依赖性，即使我们此后会决定使用其他技术。其次，是为了提供控制。通过抛出自己的异常，业务对象可以解释异常，并提供更加有意义的响应。通过使用链式的异常处理(见第 8 章)，业务对象可以根据来自于其它包的底层异常抛出自己的异常。关于编写异常类的信息，参见在线文章“Exceptional Practices” [Goetz]。

关于业务逻辑和业务对象模式的更多信息，参见 J2EE Blueprints [Sun, Blueprints]的第 14.1 节。

14.2.4. 设计结果

表 14.1 中描述了在你的应用中使用一个业务层和业务对象的设计结果。

表格 14.1 分层概念的设计结果

结果	**说明**
减小耦合，提高可管理性	业务对象减小了客户和实现细节之间的耦合。使得管理实现的变更更加容易，因为细节被封装在业务对象之中。
引入一个额外的层	某些人可能认为集成层没有必要而想要直接连接到资源
提供一个简单的统一接口	业务对象的签名实际上反映了它实际需要什么输入。
可以隐藏低效率的东西	由于隐藏了实现，开发人员可以不用知道正在访问的远程资源，正在进行的具体查询，也不用在意业务对象被调用的频度。

14.2.5. 将业务与 Action 混合 (不)

如果**没有**使用业务层，那么就有很多 Struts 特定的后果需要考虑：

- 在人们编写他们的第一个 Struts 应用时，通常他们会直接从一个 Action 类中直接访问数据库。这当然可以工作，但是这种方式的应用其设计将会变得难以扩展和维护。其问题和我们在讨论 JSP 的模型 1 设计的时候发现的一样。
- 所有实现在 Action 中的业务逻辑不能够在 Struts 框架之外被访问，或者使用 JUnit [JUnit]之类的工具进行正确测试。
- 数据访问、错误处理、以及其它资源层代码可能在 Action 之间重复，只能通过拷贝粘贴进行重用。
- 关键内容，比如 SQL 命令，可能会淹没在代码的海洋之中难以评审和维护。
- Action 实际上成为了应用的接口，鼓励开发人员在 Action 之间转发请求。
- 当 Action 成为 API 时，就需要新的机制，包括创建中间查询字符串，或者使 Action 检查请求中的其余对象。
- Action 会变得复杂和臃肿，因为它们试图处理来自于客户和其它 Action 的业务逻辑请求。
- 因为 Action 变得交叉依赖和互相粘附，应用将陷入到细碎的面条式代码的泥潭之中，就如同我们在 Model 1 的 JSP 应用中经常看到的一样。

如我们先前讨论过的，一个更好的方法是继续使用经典的 MVC 模式，并且清晰地将在 web 层和应用层(View 和 Controller)中发生的东西从在业务层 (Model)发生的东西分离出来。关于 MVC 的详细信息，参见本书中的介绍和 Struts 用户指南 [ASF, Struts]。

14.2.6. 一个简单例子

我们来看一下 Artimus 示例应用的业务 API 。本章余下部分会描述将数据库连接到这个 API 然后解释如何添加 Lucene 搜索引擎到这个系统中。

Artimus 是一个新闻文章的张贴程序。提交的文章可以通过各种方式搜索并且进行辛迪加汇聚（syndicated）。它是一个简单但有用的应用系统。我们先来看初始业务需求。

Artimus—初始业务需求

简单说起来， Artimus 的业务需求有：

- 保存文章的标题，作者和内容
- 按照降序（最新到最旧）列出最近文章
- 按照标题、作者、内容和其他属性过滤和列出文章

随后我们会添加更多的需求，但是我们首先来看看实现这个需求集的业务 API。

Artimus—初始 API

Artimus 的初始 API 示于表格 14.2。

表格 14.2 Artimus 应用的简单业务 API

需求	方法
保存文章	`int insert (Integer article, String title, String creator, String content)`
列出最新文章	`Collection searchLast (int count)`
过滤文章	`Collection searchTitle(String value)`

在后面，我们会使用这个 API 实现 Artimus 业务需求。我们还会添加全文检索和内容汇聚的需求，然后实现它们，而不用修改业务 API。

14.3. 在 Struts 中使用 ProcessBean 和 JDBC

有好些产品能够在 JavaBean 的类层次体系和关系数据库之间提供对象-关系的映射。这些产品可以使你更加专注于设计你的对象体系，然后自动产生将你的对象体系映射到关系数据模型所需的 SQL 代码。Scott Ambler 撰写了两个经典的关于 O/R 建模和设计表现层的白皮书[Ambler]，对它们我们都强烈推荐。

ProcessBean 类广泛使用了在 Struts 用户指南[ASF, Struts] 中讨论业务逻辑 bean 的时候所描述的技术。这些技术包括：

- 封装一个应用的功能性逻辑
- 表达未决事务的状态，比如购物车
- 避免从 web 层中(比如 http 包)使用业务类
- 避免任何与 Web 应用的假设交互的代码
- 提供一个 `execute` (或者对等的) 触发器方法，以进行特定的行为

另外， ProcessBean 还

- 使用批量设置器来从另一个 JavaBean 中组装自己
- 使用批量“组装”将值拷贝到另一个 JavaBean

ProcessBeans 类***不是***一个 O/R 映射的实现。不是封装或者隐藏 SQL，ProcessBean 包提供了一个结构化的地方，你可以在其中插入你的查询，然后获得 JavaBean 集合的结果。

它也可以自动从另一个 bean(比如, ActionForm)中组装你的模型 bean，并且将一个 SQL ResultSet 尽可能快地移到你的 JavaBean 的一个集合中。最后一步类似于 O/R 映射，但是总体策略太简单，还算不上 O/R。

14.3.1. 介绍 ProcessBean

开始时，最好将 ProcessBean 想象成 Struts ActionForm bean 的强类型版本。ActionForm bean 只需要使用 `String` 和 `boolean` 属性，这主要是为了保持对 HTTP 的兼容。ProcessBean 没有这些限制，并且可以使用任何最能表现你的数据的类型。

所以，如果你有一个这样的 ActionForm：

```
private String account = null;

public String getArticle() {
 return (this.account);
}
public void setArticle(String account) {
 this.account = account;
}

private String amount = null;

public String getAmount() {
 return (this.amount);
}
public void setAmount(String amount) {
 this.amount = amount;
}
```

那么你就可以有这样一个 ProcessBean：

```
private Integer account = null;
public Integer getArticle() {
 return (this.account);
}
public void setArticle(Integer account) {
 this.account = account;
}
private Double amount = null;

public Double getAmount() {
 return (this.amount);
}

public void set Amount(Double amount) {
 this.amount = amount;
}
```

和 ActionForm bean 相似，ProcessBean ***不需要是***数据库中的表的直接表现。它们表现的是业务 API 实现的业务逻辑。当然，数据库也表达的是业务模型，而且因此这些组件通常会发生相交。有时候，属性集甚至可能是相同的。但是这只是一种相同巧合而不是设计目标。

在与数据库的关系中，ProcessBean 的属性将和 SQL 查询中所需的值以及结果集中返回的列相交。这是必要的，因为查询结果要用来组装 ProcessBean (反之亦然)。有时候，这些属性碰巧是一个单一表格中的所有列。更常见的是，属性是多个表联合而得的列的集合，或者*逻辑视图*。

这是一个重要的差别。分层的目的是让业务对象成其为业务对象，让数据库就是数据库。这些组件之间太过紧密的耦合，将导致脆弱得难以维护的应用系统。随着时间的流逝，数据库保存值的方式可能会发生变化。位于一个表格中的列可能会转移到另一个表格中。或者数据库可能会被其它组件所代替，比如搜索引擎。基于这些原因，以及其它一些原因， 对于业务层和资源层来说，重要的是要保证松散的耦合，不允许它们混成一谈。

所以再次说明， ProcessBean 并不是表达数据库中的表格；它们表达的是数据库的逻辑视图。那些属性将属于这个逻辑视图将由业务 API 来决定。

那么，逻辑视图是否就是 MVC 模式所描述的表现视图呢？ MVC 模式将会考虑这些对象的分离。但是，从根本上说，答案是肯定的，我们谈论的是同一套属性。在这里我们没有使用短语***逻辑视图 (logical view)***， 因为这个词汇在说到数据库时有其自己的含义。相同的值集合只是简单的在对象之间传输。

14.3.2. ProcessBean 作为传输对象

因为它们旨在从一个层传输数据到另一层，ProcessBean 可以视为是传输对象，即

ActionForm 较为被人忽视的另一面。在实际的模型中，Processbean 的每一个属性可能都是可以单独访问的。但是为了避免单独发送每个属性，我们把它们归集到一个单一的可以通过一个操作进行传输的对象中。

取决于具体环境，一些传输对象可能是只读的，或者说***不可修改的（immutable）***。 这时，值在对象被构造的时候被设置，随后便不允许进行修改。(没有提供 public setter，或者 *mutator*。) 最常见的是发生在使用 EJB 的环境中，因为数据库是远程的，更新的代价将十分昂贵。不可修改 bean 可以专门用来显示所需的数据，而使用其它的 bean 来进行各种插入和修改。

因为 ProcessBean 通常设计来使用本地资源，所以它们通常也是可读写的，或者说***可修改的（mutable）***。 和许多传输对象实现类似， ProcessBean 使用一个批量修改器来一次性设置其值。但是和大多数传输对象不同的是，ProcessBean 通常会很好的在其批量设置器中利用反射机制。

14.3.3. 组装 ProcessBean

Struts ActionServlet 会自动从进入的 HTTP 请求中组装 ActionForm bean。Struts 所用的这个工具也可以单独使用，并且可以从另一个 bean 中组装任何 bean 。

ProcessBean 实际上是一个接口，它定义了两个方法签名：

```
public Object execute() throws Exception;
public Object execute(Object parameters) throws Exception;
```

ProcessBeanBase 提供了一个 `execute(Object)` 方法的实现，它使用反射来组装子类提供的所有属性。为了达到这个效果， ProcessBeanBase 使用了来自 BeanUtils 的方法。

1.0 vs 1.1 在 Struts 1.0 中，BeanUtil 类是在 Struts util 包中提供的。从 1.0 之后，它移到了 Jakarta Commons 项目中，并以一个单独的包提供 (`jakarta.commons.BeanUtils`)。 [ASF, Commons]

如果你的 ProcessBean 继承自 ProcessBeanBase，可以直接使用 `execute(Object)` 的默认实现，不需要进行覆写。反射的威力可以找到你所添加的公开属性。

反射是有效率的吗？ 是的。每一个版本的 JVM 都改进了反射的效率，在 1.4 中更是取得有很大的提升。反射的使用消除了许多仅仅是调用 getter 和 setter 的之类的定制代码。相反，可以在整个应用系统中使用一个单一的过程来完成这些事情。在实践中，这个工具代码倾向于驻留在 JVM 的可执行缓存中，并且可以被 Java HotSpot 编译器优化，以取得比其它替代方法更好的性能。

14.3.4. 执行 ProcessBean

ProcessBean 的另一个方法 `execute` 使 ProcessBean 能够进行其设计要做的事情。这可能是插入数据，返回结果，或者任何需要的操作。

通常的策略是以你的业务模型需要的属性创建一个基础 bean。然后这个基础对象会根据你

的模型的要求来分别进行扩展。为了组装 bean，你可以传递一个所需属性的 Map(java.util. Map) 给 execute(Object)方法。默认实现将组装任何匹配的属性并且触发 execute 方法以完成所需的工作。

execute 方法返回一个对象。通常，这将是一个 ProcessResult 对象。ProcessResult 设计用来传递业务操作的结果到另一个层(使其为另一个传输对象)。ProcessResult 也可以包含消息，数据，或者两者都有，并且它还可以包含可以自动处理结果的方法。

如果业务操作返回了数据，数据通常包含在一个 ResultList 对象中(org.apache.commons.scaffold.util.ResultList)。这是一个具有一些能够使其容易在表现层页面中使用的方法的 Collection (java.util.Collection)。ResultList 设计来是为了替代 ResultSet (java.sql.ResultSet) 的，以便你能够使用离线 JavaBean。

如果你是从 ProcessBeanBase 子类化你的模型对象，你需要：

- 添加属性。推荐使用对象包装器，比如对 Integer。
- 覆盖 execute() 以执行适当的操作。

因为结果签名接受和返回 Object，一定要记下实际的类型，以及期望的参数。

14.3.5. 访问数据服务

Artimus 所用的 ProcessBean 是一个静态类，名为 Access (org.apache.artimus.article.Access)，用来将业务层连接到资源层。这不是一个需求，只是一个有用的惯例。Access 类表示实际的业务 API。ProcessBean 表示使用 API 的客户对象。

在外表之下， Access 类将 SQL 命令通过 SQL 语句和准备语句整合在一起，从而扮演了数据访问对象（DAO）的角色。如果你需要不止一个实现， Access 可以基于一个接口，同时 ProcessBean 可连接到一个 singleton 而不是静态类。

Access 类的实现倾向于十分简单。Scaffold 提供了一些便捷工具来使用 SQL 语句和 prepared 语句。Access 类通过适当的 SQL 命令和运行时传入的数据将这些整合在一起。. 清单 14.1 就展示了 findByArticle 访问方法 (org.apache.artimus.article.Access.findByArticle)。

清单 14.1 findByArticle 访问方法

```
public static final void findByArticle( Object target,
                                        Integer key)
                 throws ResourceException {
    findElement(target,key,ARTICLE_SELECT_KEY);
} // end findByArticle
```

静态 findElement 方法是由 Access 基类提供的。然后又调用一个 Scaffold SQL 工具 (org.apache.scaffold.StatementUtils)。其实现示于清单 14.2。

Listing 14.2 The findElement access method

清单 14.2 findElement 方法

```
public static final void findElement( Object target,
                                      Object key,
                                      String command)
                           throws ResourceException {
   try {
         StatementUtils.getElement(null,target,
                           getCommand(command),key);
      }
   catch (SQLException e) {
     throw new ResourceException(e);
    }
} // end findElement
```

getElement 方法按照表格 14.3 中所列的参数组装目标 JavaBean。

表格 14.3 StatementUtils.getElement 参数

参数	目的	清单14.2中的参数值
resource	数据源的名称； null 表示默认数据源	null
target	程序应该从结果集中的第一行按照匹配列组装的bean。	target
command	程序应该执行的SQL 语句。语句产生的结果集用于组装目标 JavaBean。	getCommand(ARTICLE_SELECT_KEY)

Scaffold 中的大部分其它基础数据的访问方法都是基于相似的手段，即 getCollection (org.apache.commons.scaffold.sql.StatementUtils.getCollection)。在这个方法中，没有组装目标 bean，target 是作为一个工厂来根据结果集中的每一行实例化一个 bean 的。

Scaffold StatementUtils 包提供了多个方便的方法签名来传递一些通用的替换参数，比如 String 或者一个 Integer。 在其外表之下，这些方法都导致执行相同的 executeUpdate 或者 executeQuery 方法。这些方法从数组中提取出替换参数，并且将它们与 SQL 查询中的参数进行匹配。方便方法只是为你创建这些数组。

清单 14.2 中的 getCommand 方法从标准属性文件中返回一个普通的 SQL 语句。这就要求 SQL 查询独立于 Java 代码之外，并且将它们放到数据库管理员也可以接触的地方。在 sql_article. properties 文件中，我们的查询只是简单地列出为：

```
article.select.key = SELECT
```

```
article,marked,contributed,contributor,
creator,cost,title,content FROM
artimus_article WHERE article=?;
```

其它还有一些 article 查询。

14.3.6. 循着典型流程

使用 ProcessBean 的典型流程的要点是：

- Struts ActionServlet 传递一个组装好的 ActionForm 给 Action 的 perform 或 execute 方法。
- Action 创建、组装和执行适当的 ProcessBean。
- 如果发生错误， Action 捕捉异常并且将控制路由到一个错误或者输入页面。
- 如果操作成功， Action 通常会从 bean 中刷新 ActionForm 然后将 ResultList 对象提交到请求上下文。

我们在讨论助手 action 的时候还会回到这个主题。不过现在，我们先来看看 Artimus 的一个运行实例。

14.3.7. 编码业务活动

我们来看看中提供按 ID 查看文章（view-article-by-ID）action 所需的代码。

Struts ActionServlet 传递一个组装好的 ActionForm 到 Action 的 perform 或 execute 方法。ActionForm 和 Action 是在 Struts 配置中定义的，使用了如下的设置：

```
<!-- Article Form bean -->
<form-bean
  name="articleForm"
  type="org.apache.artimus.article.struts.Form"/>
  <!-- View Article action mapping -->
<action
  path="/article/View"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.article.FindByArticle"
  name="articleForm"
  scope="request"
  validate="false">
  <forward
    name="success"
    path="/article/pages/View.jsp"/>
</action>
```

NOTE Artimus 的当前实现使用了 Tiles。为了清楚起见，这里展示的代码是应用使用 Tiles 之前的版本。

表 14.4 描述了这些配置设定。

表格 14.4 Struts 配置元素

元素或者属性	目的	示例代码清单中的参数值
form-bean element	创建一个 ActionForm 对象	
name property	关联一个 ActionForm 的逻辑名称	ArticleForm
type property	指定用于 ActionForm 对象的类。	org.apache.artimus.artimus.http. Form
action element	创建一个 ActionMapping 对象。	
path property	对此 mapping 关联一个 URI	/search/Article

name property	从请求中组装的 ActionForm 的名称(如果有的话)。	ArticleForm
type property	关联到这个 mapping 的 Action 对象的类	org.apache.commons.scaffold .util. ProcessBean
parameter property	一个通用的属性。在这里，主要用来为 ProcessAction 指定需要实例化的业务对象。这个用法是对核心 Struts 框架的扩展。	org.apache.artimus.article. FindByArticle

遵循与 Struts `ActionServlet` 相同的模式，Scaffold `ProcessAction` 使用反射来创建和组装助手对象。助手类将被指定为 `ActionMapping` 的 `parameter` 属性。这个过程是自动的，组装好的助手的 `execute` 访法将被 `ProcessAction` 调用而不需要开发人员的干预。清单 14.3 展示了用作 `FindByArticle` 操作的 `ProcessBean`。

清单 14.3 FindByArticle ProcessBean

```
package org.apache.artimus.article;

import org.apache.commons.scaffold.lang.ParameterException;
import org.apache.commons.scaffold.util.ProcessResult;
import org.apache.commons.scaffold.util.ProcessResultBase;
public class FindByArticle extends Bean {
public Object execute() throws Exception {
// ①
if (null==getArticle()) {
  throw new ParameterException();
}
// ②
 Access.findByArticle(this,getArticle());
// ④
 ProcessResult result = new ProcessResultBase(this);
 result.setSingleForm(true);
 return result;
} // end execute
} // end FindByArticle
```

我们的 FindByArticle bean，示于清单 14.3，扩展了一个 article bean 类(org.apache.artimus.article.Bean)。该 Bean 类（它又扩展了 org.apache.commons.scaffold.util.ProcessBeanBase）提供了所有对 ProcessBeans 来说公共的属性。其 execute(Object) 方法会自动从传递为方法参数的 Map 中组装这些属性，然后调用 execute() 方法，而后者是 FindByArticle 类中定义的唯一方法。

在 ①处，我们查找基础 article 属性并进行最终验证。如果 article 参数丢失，将抛出一个 ParameterException 异常 。

在②处，我们的 ProcessBean 将自身作为目标类型并且传递匹配的 title 来调用 Access.findByArticle 。findByArticle 数据访问方法将匹配 title 的记录返回为一个 FindByArticle bean。如果我们想要使用另一个 bean 类型，我们可以传递一个该类型的实例而不是 this。

Artimus ProcessBean 的 API 契约是它们要返回一个 ProcessResult 对象。所以，在③处，我们在返回之前将 FindByArticle bean 包装到一个 ProcessResult 中。

如你所见， ProcessBean 代码主要处理管理细节并且确保符合业务 API 契约。具体的数据访问是留给 Access.findByArticle 方法，如清单 14.1 所示。

在清单 14.1 中，我们看到 findByArticle 方法只是 Access.findElement 方法的一个包装器。好多个 ProcessBean 都要使用这个特殊的方法，所以最好将它提供为一个工具方法。在其它情况下，访问方法可能是在 ProcessBean 自身中提供的。Scaffold Access 和 StatementUtils 类提供的工具可以使之很容易做到。

这是栈的末尾了。结果行现在返回给 ProcessBean 类，并且包装在一个 ProcessResult 对象中。然后，ProcessBean 依次处理 ProcessResult，再返回给 ProcessAction。

ProcessAction 类提供了一个可重用的基础架构来处理异常和在应用中路由控制。异常处理和流程控制是在 Struts 中使用业务类和数据系统的重要部分。我们还会在本章后面回来讨论 Scaffold helper action。

STRUTS TIP

如果 SQL 列不匹配 bean 属性的约定怎么办？

SQL 允许你能够动态重新命名列名。如果你被绑定到具体的列名，比如 FIRST_NAME 和 LAST_NAME，你可以在 SQL 语句中使用别名（alias）：

```
Select FIRST_NAME firstName, LAST_NAME lastName
From CUSTOMER_TABLE Where ...
```

当语句返回时， setFirstName 和 setLastName 现在将对应于 FIRST_NAME 和 LAST_NAME 列

14.3.8. ProcessBean 作为持久层

综合在一起， ProcessBean 和这里描述的其它 Scaffold 对象构成了一个简单的持久层。表 14.5 展示了 Scaffold/Artimus 中的类是如何对应到 Scott Ambler [Ambler]所描述的持久层类的。

表格 14.5 比较 Scaffold 类和 Ambler 的持久层

Scaffold/Artimus 类	Ambler 类	说明
ProcessBean	PersistenceObject	提供业务领域所需的行为
Access	PersistenceCriteria	提供获取、更新或者删除对象群集的行为
StatementUtils, Statements	PersistenceMechanism	提供对数据服务的访问 (database, search engine, 等等)
getCommand(), commands	SqlStatement	提供 SQL 语句
ConnectionAdaptor	PersistenceBroker	提供到持久性机制的连接 (StatementUtils)

Ambler 中的类是为了提供比这里所述的持久层提供更多的功能。对于更大型的项目，你应该考虑使用综合性的数据持久性包，比如 Osage [Osage] 或者 ObjectRelationalBridge [ASF, OJB]。

14.3.9. 使用其它持久层技术

当然， ProcessBean 并不是你在 Struts 唯一可用的持久层。因为 Struts 是模型中立的，你应该能够使用你选择的持久层。我们这里展示的模式应该可以适用于所有持久性产品，从 Simper [Simper] 到 EJB [Sun, J2EE]。

如果你还没有使用一个持久层，一个可能的选择是 Jakarta ObjectRelationalBridge (OJB) [ASF, OJB]。ObjectRelationalBridge 是一个相对较新的项目并且很快地吸引了一个较高的用户群。也可以考虑其他开源产品，比如 Castor [Castor]，Simpler [Simper]，和 Osage [Osage]。

14.4. 使用结果对象

在获得了数据之后，你仍然需要将它们传输到表现层。虽然，你也可以使用标准类，比如 `Vector` 和 `ArrayList` 什么的，但通常创建一个特殊的包装器类要方便一些。在内部，包装器可以使用标准的集合来保存数据，但是在外部，它可以通过暴露方法来简化对数据试图的处理。

14.4.1. ResultList 方法

这种包装器的一个很好的例子是 `ResultList` 类 (`org.apache.commons.scaffold.util.ResultList`)。`ResultList` 类提供了多个可用在创建列出搜索结果页面的方法，包括在表 14.6 中。

表格 14.6 一些用来创建页面列表的 ResultList 方法

方法	说明
Int getSize()	返回结果列表中的元素的数目
String getLegend()	返回结果列表的描述
Iterator getIterator()	返回一个针对结果列表的迭代器

因为这些方法遵循标准的 JavaBean 命名约定，它们便可以容易用于 Struts 标签中。下面是从一个结果列表中输出一个简单 HTML 表格的一些 JSP 代码。你可以在 Artimus 示例应用的 pages/article/Result.jsp 下找到完整的代码：

```
<TABLE>
  <TR>
    <TD align="center" colspan="3">
      <bean:write name="RESULT" property="size"/>
        matches for
      <bean:write name="RESULT" property="legend"/>
    </TD>
  </TR>
  <logic:notEqual name="RESULT" property="size" value="0" >
    <TR bgcolor="FFFFEE">
      <TH>id</TH>
      <TH>article</TH>
      <TH>contributed</TH>
    </TR>
    <TR>
      <logic:iterate name="RESULT" property="iterator" id="row">
        <TD>
          <bean:write name="row" property="article"/>
        </TD>
        <TD>
          <html:link forward="article" paramName="row"
                     paramProperty="key"
                     paramId="key">
            <bean:write name="row" property="title"/>
          </html:link>
        </TD>
        <TD><bean:write name="row" property="contributed"/></TD>
      </TR>
    </logic:iterate>
    </TR>
  </logic:notEqual>
</TABLE>
```

这个页面可以用于多种不同的搜索：比如按关键字，按作者，按标题。每一种搜索都为搜索设置一个不同的说明，或者 legend，以便你可以针对每一个搜索类型定制页面。

和 ProcessBean 类似，ResultList 是基于一个你可以用在你自己的应用中的基类(org.apache.commons.scaffold.util.ResultList) 中实现的接口。

ResultList 扩展了 Collection 并且可以用在 Collection 能用的任何地方。表 14.7 就包含了一个 ResultList 在 Collection 成员之外实现的方法列表。

表格 14.7 ResultList 接口(没有 Collection 方法)

方法	说明
Iterator getIterator()	返回针对结果列表的迭代器（iterator ）
int getSize()	返回结果列表中的成员数目
String getLegend()	返回结果列表的描述
public void setLegend(String description)	设置结果列表的描述
public void setLegend(String value, String property)	使用 value 和 property 将 description 设置为${property} = ${value}
boolean addMessage(String message)	对内部列表添加一个确认信息
public boolean populate(Object o, int index) throws Exception	从列表中的索引处的元素组装一个给定的对象——o 参数的值可以改变
Map getDisplayName	返回一个 displayName map (一个 HashMap)
setDisplayName(Map displayName);	指定一个新的 displayName 列表

14.5. 使用助手 Action

Struts 开发人员中一个较通用的策略是使用一个单一的 Action 来处理多个相关操作。这将有助于重用代码和管线化流程，并且减少应用中类的数量。Scaffold 包中的 Action 就采用了这个理念，并且实现了一组你几乎可以用于应用中的所有操作的框架 Action 。

助手 Action 所用的策略就是将一组业务对象与一个 ActionMapping 相关联。业务对象被实例化、组装，并且被调用。而 Action 则集中于异常处理和流程控制。多态的业务对象可以在 Action 并不知道它们实际做什么的情况下被适时调用。这允许你能够从 Struts 配置文件中“连接”Web 应用中的更好的部分，只需要指定哪个业务对象被哪个 ActionMapping 调用即可。

我们一直在说 *helper Actions* ，是因为 ProcessAction 构建在一个基类之上，即 BaseHelperAction 。 ProcessAction 专门用于 ProcessBean ， 而 BaseHelperAction 则可以用于任何类型的业务对象。如果你想要使用 ProcessBean，你可以使用 BaseHelperAction 作为你的类体系的基础。表 14.8 就列出了我们的 Action

体系的包引用。

表格 14.8 helper Action

类名	全限定类名
BaseAction	org.apache.struts.scaffold.BaseAction
BaseHelperAction	org.apache.struts.scaffold.BaseHelperAction
ProcessAction	org.apache.struts.scaffold.ProcessAction

即便你不使用助手 Action 策略,这些 Action 所用的各种技术也可以用在你自己的类中。关于可重用 Action 技术的内容，参见第 8 章。

14.6. 使用 Lucene

数据库是不可或缺的，但是它们的确有些限制——特别是它们在处理文本字段的搜索的时候。表 14.9 包括了数据的通常行为和它们的后果。

表格 14.9 数据库行为和后果

行为	结果
ANSI SQL 文本搜索是上下文敏感的	Text 并不匹配 TEXT
SQL 通配符查询会将参数是为一个单一字符串	Wildcards 并不匹配 Wildcard
SQL 查询只能根据字段的内容进行排序	最可能（或者最相关）的匹配不会首先列出来
大多数数据库系统并不针对文本检索进行索引	针对文本的查询可能具有性能问题
许多数据库并为针对复杂的 boolean 查询进行优化	使用 boolean 操作的文本搜索可能是一个瓶颈。
数据库系统只能搜索保存在它的表格中的内容。	数据库表格外部的信息可能不能被搜索到。

一个好的搜索引擎包，比如 Jakarta's Lucene [ASF, Lucene]，就可以通过提供以下特征来解决以上这些问题，但不限于此：

- 基于单词的全文检索
- 富有效率的基于术语的逻辑检索
- 日期范围检索

■ 适当次序的匹配

假定有一个良好的分层设计，将 Lucene 连接到你的应用是十分简单的。我们来看 Artimus 是怎么做的。

14.6.1. 再看 searchProperties

先前，在清单 14.1 中，我们看到了一个来自 Artimus 中的 `findByArticle` 方法。Artimus 的产品化版本实际上是使用 Lucene 来搜索属性的：

```
public static final Collection findByProperty(
                      Object target,
                      String property,
                      String value)
       throws ParameterException,
       PopulateException,
       ResourceException
{
  return SearchUtils.getCollection(target,
                Engine.getHits(Engine.getQuery(value,property)));

} // end findByProperty
```

在对搜索查询的响应中， Lucene 返回一个匹配文章的命中列表，非常类似于一个 SQL 查询返回的行的 `ResultSet`。Scaffold 包包括一个 `getCollection` 方法 (`org.apache.scaffold.search.LuceneUtils`)， 它可以将一个 Lucene 命中列表转换为一个任意类型 JavaBean 的列表。其功能和 SQL 版本的 `getCollection` (org.apache.scaffold.sql.ResultSetUtils)是一样的，这使得用 Lucene 搜索来代替一个 SQL 查询非常容易。

NOTE 我们所做的唯一修改就是在数据访问方法中。返回的 Collection 如今是从 Lucene 索引中取得的而不是一个 SQL 查询，但是应用的其它部分并不那么聪明。如果你细读 Artimus 的数据访问类 (`org.apache.artimus.articles.Access`)，你将看到方法要么访问 Artimus 数据库，或者访问 Artimus 搜索索引， 非常符合方法的意图。

Lucene 是一个良好设计的工具包，并且不可思议的易用。关于 Lucene 的综合应用超出了本章的范围，但是我们会提供一个有助于入门的总体介绍。和 Struts 一样， Lucene 也是 Jakarta 的一个开源项目[ASF, Lucene]。

介绍Lucene

Lucene 是一个搜索引擎，同时也是一个构建搜索引擎的工具包。它并不存储数据而只是对其进行索引。数据可以是数据库中的记录或者网站上的页面。唯一的需求是你的应用应该能够使用某些标识符再次获得这些数据。这可能是一个主关键字，一个 URL，或者你的应用可用）的任何东西。

关键对象！ Lucene 是通过 5 个关键对象发挥它的魔力的：Document，Field，Index，Query，和 Hits，如表 14.10 所示。

表格 14.10 Lucene 的关键类

对象	说明
Document	一个 逻辑构造，可以使数据库中的一个记录，Web 站点中的一个页面，或者其它可以再次获得信息片段。Lucene Document 被维护为一组字段，每一个都包含原始数据源的地址。
Field	Field 是 Lucene Document 的一个组成项。 每个 Lucene Field 都两个部分：一个 name ，一个 value。value 可以被索引，以便被查询并且用来获取原始数据源。Field 可以随 Lucene Document 一起保存，在这种情况下，它在一个 Query 匹配 Document 时会被返回。Document 通常至少包含一个存储的字段以唯一标识它。
Index	当 Lucene 分析 Document 中的 Fields 时，它会创建一个 Index ，以便你可以使用一个对 Field 的 Query，并且获得 Document。我们的应用就可以使用 Document 中的某个保存的 Field 来获取原始数据。Document 在一个 Hits 列表中返回以响应一个 Query。Index 是由 IndexWriter 创建并维护的。
Query	Lucene 支持多种不同的 Query 类型来进行更有效的搜索。灵活的 Query 对象可以简单地进行一个搜索字符串，或者创建一个全文的 Query 对象，就像经常被 WWW 上的目录或者门户所做的一样。
Hits	一个分级的集合，包含匹配一个 Query 的 Document。

*典型工作流。*从头开始，一个使用 Lucene 的应用将：

- 通过添加字段创建 Document
- 创建一个 `IndexWriter` 并通过 `addDocument` 添加文章到其中
- 调用 `QueryParser.parse` 来从字符串构造一个 Query
- 创建一个 IndexSearcher 并且将 Query 传递给其搜索方法
- 将命中结果处理为一个显示给用户的列表

Artimus 如何使用Lucene

为了驱动 SQL 表格和 Lucene 索引， Artimus 提供了一个 `CreateResources` ProcessBean (`org.apache.artimus.CreateResources`)。在调用一个 `Access` 方法来初始化索引之后，CreateResources 就会索引所有现有的记录。一旦索引被创建，Artimus 就会在纪录被添加、更新和删除的时候维护它。如我们所见， Lucene 搜索查询可以在你喜欢的时候和一个数据库管理系统(DBMS)切换。

创建索引。和 Lucene 中的大多数东西类似，总体的索引过程非常简单：

1. 使用 InexWrier 创建一个索引

2. 创建一个 Document 对象
3. 获得你的数据源
4. 基于数据源，添加一个或者多个 Field 到 Document 对象中
5. 添加 Document 到 Index 中
6. 对每一个数据源重复步骤 2
7. 优化并关闭 Index

最灵活的部分是为 Lucene Documents 创建 Field 对象。Lucene 提供了 3 个 Field 开关，可以根据需要酌情使用。如表格 14.11 所示。

表格 14.11 Lucene Field 开关

开关	指定…
Store	是否保持 Document 对象中的 Field 的一个副本
Index	是否分析一个 Field 一边它可以通过一个查询来搜索
Tokenize	是否将一个 Field 分解为 Token（或者单词）

你需要保存（*store*）你想要显示给用户的命中列表中的字段，以及其他一些可能需要逐字查询的字段。这些可能包括象关键词和日期之类的东西。但是，大多数字段，特别是很大的字段，可以被索引但是不需要保存。

你需要索引（*index*）那些可以作为搜索查询一部分的字段，并且用来在随后再次找到文档。

你可能需要切分（*tokenize*） 那些包含只一个单词的字段，并且每一个单词都应该作为一个单独的实体进行索引。

试图想象关于哪种情形该使用哪个开关可能会将你搞昏头。实践中，有四种通用的字段类型可以满足大多数人的需要： Keyword，Text，Unindexed，和 Stored。 为了方便， Lucene 提供了一个静态工厂方法来使得可以很容易的产生你所需的对象。

表 14.12 将静态字段和 boolean 属性进行关联

表格 14.12 字段工厂方法

工厂方法	store	index	tokenize	说明
static Field UnIndexed(String name, String value)	***			构造一个没有被 tokenized 或者被索引的，但是保存以供命中列表返回的字符串值字段
static Field UnStored(String name, String value)		***	***	构造一个被切分和索引但是不保存在索引中的字符串值字段
static Field Keyword(String name, String value)	***	***		构造一个没有被切分但是进行

				了索引和保存的字符串值字段
static Field Text(String name, String value) static Field Text(String name, Reader value)	***		***	构造一个被切分（tokenized）和索引的，并且保存以供命中列表返回的字符串值字段

这是 Lucene 中的一个通用模式。通用用法是在一个高阶 API 或者基类中提供，但是包的全部威力仍然可以针对那些需要它的人可用。

Artimus 将 Lucene 索引操作封装在一个位于 Access 类(org.apache.artimus.Access)中的一个单独方法中。注意是如何使用 Field.*factory* 方法来指示字段应该如何被保存和索引的：

```
public static final void index (
            String article,
            String contributor,
            String creator,
            String title,
            String content,
            IndexWriter index
) throws Exception
{
  try {
    Document document = new Document();
    document.add(Field.Keyword("article",article));
    document.add(Field.Text("title",title));
    document.add(Field.UnStored("contributor",contributor));
    document.add(Field.UnStored("creator",creator));
    document.add(Field.UnStored("content",content));
    index.addDocument(document);
}
catch (IOException e)
  {
    throw new ResourceException(e);
  }
}
```

Lucene 对 null 处理得并不好。如果你添加字段，你便有责任确保传递的值不为 null。一个更容易的方法是将字段包装在一个简单的工具类中，如果属性为 null，则返回一个空的 String ：

```
document.add(Field.Keyword("article",Engine.blankNull(article)));
```

这使得实现 CreateIndex Action 是一个十分简单的事情。它所做的所有工作就是从数据库中选择 Article，然后逐个传递到 Access.index 方法中：

```
ArrayList list = null;
Form article = null;
try {
  IndexWriter index =
  Engine.getIndexWriter(true);
```

```
  list = (ArrayList)
  Access.select(new Form());
  for (int i=0; i<list.size(); i++) {
    article = (Form) list.get(i);
    Access.index(article.getArticle(),
                 article.getContributor(),
                 article.getCreator(),
                 article.getTitle(),
                 article.getContent(),
                 index);
  }
  index.optimize();
  index.close();
}
catch (Exception e) {
  e.printStackTrace();
}
```

如先前在第 14.6.1 节所示，一旦索引被创建，获得一个命中列表只需要调用一个方法。

虽然表面看起来， Lucene 好像和数据访问有很大的不同，但是我们的应用架构却使得它能够非常轻易的插入到其中。

14.7. 使用内容联合

内容联合如今对应用来说已经成为一个向更多人提供更多内容的流行方法了。其策略却是极其简单，并且是标准化的，安全的，以及可伸缩的。

下面描述它如何工作：

- 内容被总结在一个 XML 文件中。
- XML 可以通过 HTTP 访问，就如同其他 web 页面一样。

然后其它站点就可以：

- 获取并缓存该文件。
- 向其访问者渲染结果概要。

如果用户看到喜欢的内容，它们便可以点击该概要，然后从你的站点就可以获得详细信息。

赖于 Struts Digester 的帮助，创建和联合内容简单得不能再简单了。你可以猜到， Artimus 应用就扮演了内容提供者和内容联合客户的角色。

我们来看如何进行。

14.7.1. 摘要 RSS

Struts 使用 Digester 来从 Struts 配置文件 (struts-config.xml)中创建对象。这是一个通用的工具，可用来从 XML 描述中创建 JavaBean，以及输入一个表现 JavaBean 的 XML。

Digester 附带了一个设计来读取 RSS 文件的特殊的 RSS 版本。RSS 是内容联合的流行格式。要使用其他格式，你可以为 Digister 创建一个新的规则集。

1.0 vs 1.1

在 Struts 1.0 中，Digester 是包装在 Struts JAR 文件中的。如今它已经移到 Jakarta Commons 项目中成为一个单独的产品了。Struts 的后续版本都会使用这个 Commons 版本的 Digester。关于如何在你的应用中使用 Digester，参见 Jakarta Commons 的分发包。[ASF, Commons]

RSS 文件其实是一个称为是 *channel* 的 *item* 的集合。每个条目都有好些属性，包括到其自身的链接，这可以应在你的站点中的某个地方。channel 通常将相关的条目组合在一起，而这些通常就是你发布的内容。

Digester 包提供了表示 RSS 条目和频道的类。因为它们都是 JavaBean，在 Java 程序中使用它们将是十分直接的事情。

14.7.2. 获取和渲染

为了从现有的 XML 文件中创建一个 RSS 频道，你只需要创建一个 Channel 对象并且传递给它一个路径，然后剩下的就由 RSSDigester 来完成了：

```
RSSDigester digester = new RSSDigester();
channel = (Channel) digester.parse(path);
request.setAttribute("CHANNEL",channel);
```

Channel 对象具有一个 `getItems` 方法，你可以用来迭代遍历所有的条目，在你的 Action 或者表现中都行。

```
<logic:iterate name="CHANNEL" property="items" id="ITEM">
  <TABLE cellspacing="2"
         cellpadding="4"
         border="1"
         width="90%"
         align="center">
    <TR>
      <TD>DESCRIPTION</TD>
      <TD><bean:write name="ITEM" property="description"/></TD>
    </TR>
    <TR>
      <TD>LINK</TD>
      <TD><bean:write name="ITEM" property="link"/></TD>
    </TR>
    <TR>
      <TD>TITLE</TD>
      <TD><bean:write name="ITEM" property="title"/></TD>
    </TR>
  </TABLE>
</logic:iterate>
```

创建你自己的 RSS 也是十分简单的事情。只需要创建一个新的 Channel 对象，添加你的内容条目，然后使用一个 Writer 来处理结果：

```
Channel channel = new Channel();
Iterator rows = modelResult.getIterator();
while (rows.hasNext()) {
```

```
  ArticleForm article = (ArticleForm) rows.next();
  Item item = new Item();
  item.setTitle(article.getTitle());
  item.setLink(article.getLink());
  channel.addItem(item
  response.setContentType("text/plain");
  channel.render(response.getWriter());
  eturn(null);
}
```

我们在结尾返回 null，以便控制器知道响应已经完成。

14.7.3. 联合 RSS

一旦 RSSDigester 将 XML 转换为一个 JavaBean，我们便可以象处理其他任何 Bean 一样处理它。但是我们如何结束循环，并且将我们自己的内容提供为一个 RSS 频道呢？

Artimus 应用已经提供了一个发送到 JSP 的最近文章的列表。如果它被渲染为一个 XML 而不是一个 HTML，这个列表就是一个很好的频道。幸福的是， Digester 就提供了这么一个双向的过滤器。如果我们将我们的列表变换为一个 ChannelBean，`RSSDigester.render` 方法将很乐于输出 XML。

在一个典型的 Struts 请求响应循环中， Action 类将创建一个 JavaBean 并且通过请求上下文传递给视图。这可以是一个 ActionForm 或者一些什么 JavaBean，当它离开 Action 类的时候，它只是一个普通的 Java 对象。典型地， Action 类会将请求转发到一个表现组件，比如一个 JSP，由它来使用请求中的对象将响应渲染为 HTML。

我们没有将列表对象发送到一个 JSP 来进行渲染，而是将它发送给了一个 RSS Action 来进行渲染。Action 被传入一个响应，并且可以在适当时候直接渲染响应。这基本上就象发送一个东西到表现组件那么简单。

清单 14.4 列出了 Artimus RenderRss Action (`org.apache.artimus.struts.RenderRss`) 的代码，它使用了一个引入的 ArticleHelper 并用它来创建了一个 Channel 对象。Channel 对象直接渲染 XML 给用户，所以方法在结尾返回 null。

清单 14.4 org.apache.artimus.struts.RenderRss

```
package org.apache.artimus.struts;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;
import java.util.Iterator;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.digester.rss.Channel;
import org.apache.commons.digester.rss.Item;
import org.apache.scaffold.sql.AccessBean;
import org.apache.artimus.http.ArticleForm;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
```

```
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

public final class RenderRss extends Action {

  public ActionForward perform(ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest request,
                               HttpServletResponse response)
        throws IOException, ServletException
  {
    ActionErrors errors = new ActionErrors();
    Channel channel = new Channel();
    ArticleHelper helper = (ArticleHelper)
    request.getAttribute(AccessBean.HELPER_KEY);
    if (helper==null) {
      errors.add(ActionErrors.GLOBAL_ERROR,
      new ActionError("access.missing.parameter"));
    }
    if (errors.empty()) {
      try {
        channel.setTitle("Articles");
        channel.setDescription(DESCRIPTION_TEXT);
        channel.setLink(CHANNEL_LINK);
        Iterator rows = helper.getRows();
        while (rows.hasNext()) {
          ArticleForm article = (ArticleForm) rows.next();
          Item item = new Item();
          item.setTitle(article.getTitle());
          item.setLink(ARTICLE_BASE +
          article.getArticle());
          channel.addItem(item);
        }
      }
      catch (Exception e) {
        errors.add(ActionErrors.GLOBAL_ERROR,
        new ActionError("rss.access.error"));
        servlet.log(e.toString());
      }
    }
    if (!errors.empty()) {
      saveErrors(request, errors);
      return (mapping.findForward("error"));
    }
    response.setContentType("text/plain");
    channel.render(response.getWriter());
    return(null);
  } // ---- End perform ----
} // ---- End RenderRss ----
```

下面是如何在 Struts 配置中设置 RenderRss Action：

```
<!-- Find recent articles -->
<action
  path="/channel/Recent"
  type="org.apache.struts.scaffold.ProcessAction"
```

```
  name="articleForm"
  scope="request"
  validate="false"
  parameter="org.apache.artimus.article.FindByLast">
  <forward
    name="success"
    path="/do/channel/Render"/>
</action>
<!-- Render result as RSS channel -->
<action
  path="/channel/Render"
type="org.apache.artimus.struts.RenderRss"/>
```

这展示了两个重要的理念：

- 当视图和数据检索（模型）之间是解耦的时候，同一数据可以进行不同的处理以提供不同的结果。原始处理是针对 JSP 设计的，但是你可以扩展它来针对 RSS，而不用对原来的处理过程作任何修改。使用相同的技术，我们可以通过 RSS 提供任何发送到 JSP 的信息。同时，我们可以使用任何能够使用标准 Servlet 上下文中的对象的任何表现设备来提供这些内容。
- 如果选择了它，一个 Action 也可以处理响应。

14.8. Struts 中使用 EJB

企业 JavaBean (EJB) 是设计来表达应用的模型层的。开发人员在构建需要在多个服务器之间分布的应用时通常都要选择 EJB。许多开发人员还因为 EJB 处理事务的透明方式而选择它。如果正确的使用， EJB 也能够很好的配合 Struts。

对于某个应用是否要使用 EJB 是一个复杂的问题。EJB 可以给一个应用提供大量在别的方式下开发人员必须自己编写的服务。但是天下没有免费的午餐。大多数开发人员都知道 EJB 是开发分布式的企业级的应用的很好选择。但是大多数应用并不在此列。在你决定在你的应用中使用 EJB 之前要仔细考虑清楚。然而，如果你的应用良好设计的，你应该能够切换到 EJB，或者说切换到任何模型层技术，而不会影响到应用的其他部分。

最灵活的方式是使用***外观（Façade）***模式来在 Struts Action 和 EJB 之间创建一个缓冲区。对 Struts 来说，facade 就好像扮演了实际的模型。实践中，façade 又和其它执行实际工作的组件进行交互。

DEFINITION 外观（ façade）通过一个单一类的接口代替了一组类的接口。外观将实现类隐藏在一个借口之后。 [Go4]

Scaffold ProcessBean (第 14.3.1 节) 就是外观模式的一个例子。为了将应用从普通的 JDBC 切换到诸如 EJB 之类的其他方式，你可以在一个 ProcessBean 中实现业务逻辑。Action 可以继续调用 ProcessBean 而不用知道到底是 EJB，JDBC， 还是其他什么东西。(当然，关于 ProcessBean 并无特别之处。任何你自己创建的类似的对象也可以工作的很好。)

使用外观来封装对你的业务模型的调用称为是业务代表模式（ Business Delegate pattern)。

DEFINITION 业务代表隐藏了下层的业务服务的实现细节，比如 EJB 架构的查找和访问细节。 [Go3]

EJB 模式中经常使用的业务代表模式称为是***会话外观（Session Façade）。***

14.8.1. 会话外观

经典的外观模式也是流行的 *Session Facade* 模式的基础 [Go3]。这里，一个会话 bean 的 EJB 组件被用于实现外观。如果你的应用需要和 EJB 通信，你可以使你的 Action 来直接调用 Session Facade。这会将你的 Action 绑定到 EJB Session Bean，但是消除了在 Struts Action 和 Session Facade 之间构造通用外观的需要。和许多实现决策一样，最好的选择取决于具体的环境。

14.8.2. 数据传输对象

为了显式操作结果，将一个 EJB 传递到表现层从技术上是可行的。Struts 标签和 Velocity 视图工具都可以显示 EJB 中的属性，这和任何 JavaBean 都是一样的。但是，每一次都调用 EJB 却是代价高昂的。因此，大多数开发人员使用另一种对象来在各层间传输数据。这种载体称为是***数据传输对象（*** *data transfer object (DTO)*）。Scaffold `ResultList` 类就是数据传输对象的一个例子。

Struts `ActionForm` 也是一种 DTO。在表现层显示数据的时候，你可以选择是从一个 EJB DTO 中组装 ActionForm 还是直接使用 DTO 。决策点在于你对 DTO 能够施加多少控制。如果你可以控制 DTO 属性，那么为了显示只读属性那可能还需要 DTO 回传。Struts 标签和 Velocity View 工具都是通过反射来工作的。只要属性名称匹配，任何类型的对象都可以使用。

当然，为了输入，你应该使用 Struts `ActionForm`。一旦通过校验， ActionForm 就可以用来组装 EJB DTO。DTO 然后就可以从外观传递到 EJB。

使用 EJB 的一个流行工具是 XDoclet [XDoclet]。这个组件已开始只是一个标准 JavaDoc 工具的增强，如今已经是一个聪明的代码生成器。你可以用它来创建和维护大多数 EJB 应用中所需的繁重的编码工作，包括 DTO。

另一个值得跟踪的 EJB 工具是 Struts-Expresso [Expresso] 框架。Expresso 支持使用或者不同 EJB 来创建应用，轻易将这两种赌注分隔开来。

14.8.3. 实现模式

为了实现最好的可伸缩性，许多 Struts/EJB 开发人员都遵循以下的模式：

- 需要时重新创建对远程接口的引用
- 使用无状态会话 EJB 来引用有状态 EJB
- 避免保留对无状态 EJB 的句柄

- 避免直接和实体交互
- 使用无状态外观来一个传输对象给 Action。

对 EJB (无状态和有状态，等等)技术的讨论超出了本书的范围。关于企业 JavaBean 技术的详细信息，我们推荐你阅读 *Mastering Enterprise JavaBeans* [Roman]。 一个很好的在线文是“Enterprise Bean Best Practices” [Dragan]。

14.9. 小结

在这一张我们涉及了大量的基础技术，但是希望以一种容易理解的方式和步骤来讲解他们。虽然主要集中于技术，你也可以看到在你闲暇时研究 Artimus 示例应用所需的完整的策略。

我们讨论的主要主题是：

- 分层数据访问
- 定义业务层
- 将资源层连接到业务层
- 在层间传输数据
- 在层间沟通用户指令

这其中最重要的是，我们说过，分层设计可以使我们能够将不同的服务，以及不同的服务实现，连接到相同的业务上，而不用修改应用的其他部分。

在本章中，我们也涉及了如何在 Struts 中使用一些流行工具的详细细节，包括：

- ProcessBean
- Lucene
- Digester (用作内容联合)

我们将 Artimus 示例应用作为我们的研究案例。其业务 API 可以使用我们实现的不同服务得到支持：

- 在 SQL 表中保存数据
- 使用搜索引擎索引数据
- 通过 XML 提供联合体内容

本章最后，我们还讨论了在 Struts 中使用 EJB 时需遵循的一些最佳实践。

在本书的第 4 部分， “通过实例学习 Struts”，我们将实际使用我们所讨论过的许多技术。

15. Artimus:全力以赴实际应用

本章内容

- 介绍Artimus 示例应用
- 在同一个应用中使用Tiles, Validator, 和Scaffold
- 理解针对实际应用的推荐做法

Just do it.

—Nike

15.1. 框架之框架

Artimus 示例应用展示了部署一个企业级的，使用 Struts 核心和其它可选组件：Scaffold, Tiles, 和 Validator 开发的的最佳实践的应用。

.

因为许多以Struts 1.0 写成的实际应用都将迁移到Struts 1.1, 我们在本章将会展示1.0 版本的 Artimus，然后在下一章向你展示如何将其代码基升级到 Struts 1.1。

WARNING	Artimus 的设计是用来展示你可能在实际应用开发中可能使用的技术的。所以本案例的材料可能会比普通的变成书籍通常表现得要超前一些。 (所以，在开始之前你可能需要将你那杯 cappuccino 一饮而尽。)

本书的其它章节关注于 Tiles 和 Validator，同时我们也介绍了一些 Scaffold 包的类。在开始我们的 Artimus 之旅之前，先介绍一些 Scaffold 工具包的相关背景是有帮助的。

NOTE	整个这一章我们都回引用 Struts *ActionMapping* 对象和 Struts *Action* 对象。在 Struts 配置文件中，创建一个 ActionMapping 的元素称为是一个 *action*。当我们引用 *action* (小写)，我们谈的是 ActionMapping 元素。而我们引用到一个 *Action* (首字母大写)，我们则说的是 Action 对象。(`<action>` 元素可能比`<mapping>` 元素形式上要好些，但 Struts 1.0 或 1.1 都不是这样。)

15.2. Scaffold—工具包的诞生

虽然其原本设计是作为一个现有各种技术的集成器，Struts 却很快地成为了一个新技术的试验场。

Struts的初始版本后来产生了多个工具程序包。Jakarta Commons BeanUtils, Collections, Digester, 和Validator 包实际上都衍生自Struts 框架。这些包中每一个都可以作为一个很好的组件在框架之外使用，所以，Struts开发人员重新将它们包装为Jakarta Commons。现在其它框架和应用也可以导入这些组件以避免重新发明轮子。

和Tiles 和Validator类似，Scaffold 也是一个诞生于创建Struts Web应用的工具包。在经过多个Struts 应用开发的经验之后，大量的针对通用需要的通用解决方案被标识出来并组合到一个名为*Scaffold*的一组可重用类之中。核心的Scaffold 包可以从Jakarta Commons [ASF, Commons]取得。 而一个可选的Struts特定的Scaffold 包则可以从Struts 站点[ASF, Struts]取得。

Struts 提供了一些Web应用的非可视支撑，Scaffold则提供了一套更加高阶的对象来帮助从组件组装应用程序。和Struts类似，Scaffold 鼓励分层设计，并且帮助将业务层代码和web层代码分离。我们在全书中各处都引入了多个Scaffold 类。这里，我们将这些类都放入一个上下文环境中，并看看如何使用它们来创建Artimus 示例应用。

有许多Scaffold 类是设计来使得从Struts 1.0 迁移到Struts 1.1更容易些的。它们可以使你可以立即开始使用许多Struts 1.1 的技术特征而将完全的迁移放到随后进行。在向前兼容不是透明的情况下，迁移通常就是修改某些import 和extends 子句而已。

这样会简化我们在下一章进行的从Struts 1.0 到Struts 1.1 的迁移。

1.0 vs 1.1	当本书出版的时候， Struts 1.1 beta 2 正在流行。可能在 Struts 1.1 最终发布版中会出现一些修改。请访问本书的站点[Husted] 查询勘误表。

15.3. 关于Artimus

Artimus 是一个基于web的新闻张贴程序。授权用户可以添加、编辑或者删除文章。

站点的访问者可以使用各种查询特征，包括全文检索，按标题、作者或者时间跨度在线查看文章。

Artimus 也可以将其文章发布为Rich Site Summary (RSS) 频道。这样可以将Artimus 和企业门户Jakarta Jetspeed [ASF, Jetspeed] 整合在一起。

同时，从开发人员的角度来看，Artimus 是一个成熟的，可以随时交付的Struts 应用。下面是它的一些技术特征：

- 应用设置可以通过部署描述符(web.xml) 或者一个外部属性文件进行配置。
- 在Struts 1.0 中，将使用一个助手 servlet 来装载定制资源以便可以不需要子类化ActionServlet。而在Struts 1.1 中，则是使用了一个PlugIn Action。
- 有一个连接池适配器可以允许从业务层使用Struts连接池，或者其它连接池。连接池可以从部署描述符(或者 PlugIn 配置)中进行修改。
- SQL 命令保存在外部属性文件中，并且可以在不需要重新编译的情况下进行修改。
- 业务层是由一组ProcessBean表示的，并使用经典的命令模式来进行分派。业务代码并不嵌入到任何绑定到 Struts 或 HTTP的类中。
- 一个标准的Action 被用来分派ProcessBean, 大大减少了应用中定制Action的数量。
- 应用可以容易进行本地化。
- 使用Jakarta Tiles 框架来进行显示页面的布局和组织。
- 使用Jakarta Struts Validator 来进行客户端和服务器端的输入验证。
- 使用Jakarta Lucene 搜索引擎来在适当时候提供全文检索。

Struts 1.0 版本的Artimus 定义了3个扩展Struts的类：两个ActionForms 和一个用作RSS频道的适配器Action。而 Artimus的其余部分则完全是纯粹的业务类和从Scaffold 包中导入的类。

该系统的架构可以在不需要任何其它额外的Struts 或Scaffold 类的情况下加入任意数量的新的业务操作。你只需要加入更多的业务bean就行。所有东西的设计都是可重用的。

而Struts 1.1 版本的Artimus (见第16章) 则根本没创建任何新的Struts类。所有东西都是直接来自Struts 或者Scaffold JAR。

Artimus和本章所用的Scaffold的完整的源代码和二进制Web 文档(WAR)都可以在本书的站点[Husted]获取。因为Artimus 和Scaffold 都是一直在持续维护的，新的发布版本也可以从Jakarta Struts 站点 [ASF, Struts]取得。

15.3.1. 构建Artimus

Artimus 所用的所有配置和源代码都保存在一个中心目录中。Ant 构建文件用于编译Java 源代码文件并将其它更新后的源文件放置到规定的位置。

在Artimus WAR中，包含所有源文件，包括JSP和配置文件，的中心目录是/WEB-INF/src。位于/WEBINF/src/build.xml 的构建文件包含了将使适当的文件从/WEB-INF/src 拷贝（部署）到应用根目录或者WEBINF 文件夹的适当位置的ANT目标。

表15.1列出了Artimus 源文件的位置，和它们最终部署时的位置。

表格 15.1 Artimus 源文件的部署位置

源代码位置	部署到
/WEB-INF/src/conf	/WEB-INF
/WEB-INF/src/java/**/*.java	/WEB-INF/classes/**/*.class
/WEB-INF/src/pages/**/*.jsp	/**/*.jsp
/WEB-INF/src/resources	/WEB-INF/classes/resources

这种方法有多种优点：

- 和源代码存储库，比如CVS，保持兼容。WEB-INF/src 下的文件可以在签入到，并在CVS中维护。系统中的其它文件，比如WEB-INF/lib 下的JAR，则不需要签入。
- 它将应用所用的全部文件整合在一起，便于查找和编辑。在web 应用文件夹，这些文件中有一些可能是多层目录。
- 有些文件，比如消息资源文件，没有默认的位置，只需要放置到CLASSPATH 的某个地方即可。这些文件可以将它们保存在源代码树中的自己的文件夹中，然后在部署时拷贝到/WEB-INF/classes 文件夹中。
- WEB-INF 是大量各种配置文件的默认位置。如果你是从你自己的源代码文件夹中部署这些文件。你便有机会按你自己喜欢的方式组织它们，而不是在WEB-INF 下创建非标准的文件夹。
- 一个完善的源代码树可以给你一个地方来保存各种可选的配置文件，以及关于组成一个Web应用的各种文件的其它信息。这些文件可以在源代码树中进行维护，在这里它们不会被丢失，但是却没有被部署，而在部署时它们可能会混淆。

换句话说，虽然很多开发人员使用这种方法，但是它绝不是那么简单的，所以还是需要解释一下(如这里我们所讲)。也可能有潜在的错误，比如你编辑了文件的错误的拷贝。虽然Ant 自

身会减轻这种危险，因为它不会用一个更旧的文件来覆盖一个更新的文件。你还是需要在修改了任何源文件之后重新构建，就象修改了一个Java 源文件时所要做的工作一样。在本章我们说到源文件，我们指的是文件的*部署位置*。 如果你基于你自己对Artimus的部署，只需要知道这些文件的原始文件是在目录树中的某个地方维护的。当应用被编译时，任何修改过的源文件都会被部署到web 应用树中的正确的位置。

15.4. 部署描述符(web.xml)

大多数Struts 应用的部署描述符都好像是一个定式。

通常，就是指定一个Struts ActionServlet，以及如果在你使用Struts 1.0的时候可能还有标准的Validator 或者Tiles servlet之类。我们在第4章讨论过部署描述符的通用格式。Artimus 1.0的web.xml 则稍微要有趣一些。除了Struts标准的东西之外，它还包括对其自己的ArtimusServlet以及声明性的安全设置的配置。ArtimusServlet 并不是一个处理请求的运行时servlet，而是一个资源装载器，以载入和初始化我们的业务层类。声明性的安全设置保护了编辑命令不受非授权用户的访问。

1.0 vs 1.1	在 Struts 1.1 版本的 Artimus，我们使用了一个 PlugIn Action 而不是一个单独的 servlet 来做这个事情。

在这一节，我们将展示着两个特别的配置块：即 ArtimusServlet 和声明性安全。标准Struts组件的配置参见第4章。

清单15.1 展示了 ArtimusServlet 和声明性安全的初始化块。每一块代码都会在代码清单后的各自小节中讨论。代码说明的编号指向对应的小节编号。

清单 15.1 /WEB-INF/web.xml (ArtimusServlet 和安全声明)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Applicaiton 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
<!-- [15.4.1] Configuring Artimus -->
<servlet>
  <servlet-name>artimus</servlet-name>
  <servlet-class>org.apache.artimus.http.ArtimusServlet</servlet-class>
  <!-- [15.4.2] Our Applicaiton properties -->
  <init-param>
    <param-name>default</param-name>
    <param-value>resources/artimus.properties</param-value>
```

```
  </init-param>
  <!-- [15.4.3] Our connection adaptor -->
  <init-param>
    <param-name>adaptor</param-name>
    <param-value>org.apache.commons.scaffold.sql.ServletAdaptor</paramvalue>
  </init-param>
  <init-param>
    <param-name>adaptor.key</param-name>
    <param-value>org.apache.struts.action.DATA_SOURCE</param-value>
  </init-param>
  <!-- [15.4.4] Our startup priority -->
  <load-on-startup>1</load-on-startup>
</servlet>
<!-- [15.4.5] other configuration blocks ... -->
<!-- [15.4.6] Our security settings-->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Administrative</web-resource-name>
    <!-- [15.4.7] The URLs we protect -->
    <url-pattern>/do/admin/*</url-pattern>
  </web-resource-collection>
  <!-- [15.4.8] The authorized users -->
  <auth-constraint>
    <role-name>manager</role-name>
    <role-name>editor</role-name>
    <role-name>contributor</role-name>
  </auth-constraint>
</security-constraint>
<!-- [15.4.9] Our authentication strategy -->
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>EBasic Authentication Area</realm-name>
</login-config>
</web-app>
```

15.4.1. 配置Artimus

在清单15.1顶部中的法定的XML的官样文件之后，我们首先给了我们的Artimus servlet 一个引用名称并指定了一个容器装载的类。注意，它不是一个Struts ActionServlet，后者是在描述符中的其它地方配置的。

15.4.2. 应用属性

Artimus servlet 的目的是为了装载一个默认的属性文件，非常象Struts装载默认消息资源束的方式。在清单15.1中， `<default>` 参数指定文件的路径。属性文件是一个标准的Java 组件 [Sun, Properties]。在这里是一个使用和资源束（参见第13章）一样的格式的基于文本的属性文件。我们用它来保存系统路径和不因场所而改变的那些设定。

一个Java Web应用中的大多数系统设置都可以使用容器提供的服务进行自动调整。比如，超链接，就可以设置为相对的，或者重写为包含正确的路径。其它系统设置则不可用这种方式决定。许多intranet 应用都象这样硬编码设置，如果它们发生改变，就需要对应用重新进行编译。

Artimus 通过从一个外部属性文件获取设置来避免对它们进行硬编码。比如，某些包，象Lucene，就需要在服务器中的某个地方存储文件。属性文件中的一个设置就可以使我们能够指定Lucene所用的文件路径，可以随时修改而不用重新编译源代码。

属性文件可以在任何文本编辑器中修改。甚至你的应用也可以提供一个简单的Swing程序来读取和编辑应用系统中的定制属性。

15.4.3. 连接适配器

为了达到最广泛的受众，这里所述的Artimus 示例应用使用 JDBC 作为默认的存储系统。关于Struts 如何使用各种数据服务，参见第14章。

在15.1，我们使用了`<adaptor>` 和 `<adaptor.key>` 元素来配置我们的连接适配器以使用Struts 通用连接池。Struts 将在应用上下文中保存一个对其连接池的引用。因为这是一个通用方法， Scaffold 提供了一个标准的ServletAdaptor。该适配器使业务层组件可以使用连接池而不用直接绑定到一个web 应用。

用于在应用上下文中保存引用的属性关键字可以不同。这里我们指定使用Struts所用的默认关键字。我们也在第15.4.2节中描述的属性文件中指定了这个关键字。数据库管理员可以修改属性文件中的属性关键字，而不用关系更复杂的部署描述符。

Scaffold 还为Poolman 和基于JNDI的连接池，比如Resin，提供了其它标准的连接适配器。你也可以通过继承基础适配器并且覆写一两个方法来创建你自己的定制适配器。详细细节参见Scaffold JavaDoc 和源代码。

15.4.4. 启动优先级

因为Artimus servlet 将要载入可能在其他servlets中所用的属性，我们在清单15.1中使用 `<load-on-startup>` 元素，并且指定一个较低的启动顺序值。这意味着容器将在具有较高启动顺序值的servlets 之前装载它。

15.4.5. 其它配置设置

关于标准Struts和标签库组件的配置参见第4章。

Artimus 对此并无不同。

15.4.6. 安全设置

Artimus 使用标准的声明性安全方案。容器将管理这种类型的安全，所以在清单15.1中，我们提供了<security-constraint> 元素所需的必要的配置信息。

15.4.7. 我们所保护的URL

定义声明性安全的第一步是指定需要安全保护的URL 模式。在清单15.1中，我们的<url-pattern> 就指示了对应用中位于/do/admin 之下的任何位置的访问的限制。如果请求了一个类似/do/admin/article/Edit 的URI，容器将检查用户是否登录到了应用系统之中。如果没有，容器将发出一个挑战对话框要求用户登录到应用中。如果用户的凭证通过，容器将授权对该资源的访问。

一个应用可以定义任意数量的资源，每一个资源都有其自己的URL 模式和约束设置。

1.0 vs 1.1	在 Struts 1.1 版本的 Artimus 中(参见第 16 章)，我们使用了新的构建在 Struts ActionMappings 中的安全特征。这两个版本中都使用相同的安全角色。这里我们使用它是为了保持对 Struts 1.1 的向前兼容。

15.4.8. 授权角色

一旦用户登入，清单15.1中的 <auth-constraint> 意思是说将访问授权给manager, editor, 和 contributor；manager 也可能是一个contributor。可以为每一个角色设置单独的资源。我们随后会看到，Artimus 通过基于角色的安全提供不同的菜单，从而达到精细的安全控制。

15.4.9. 认证策略

清单15.1中的 <auth-method> 元素告诉容器使用基本认证策略。当然还有其它更安全的认证策略，但是它们并不都被所有的浏览器所支持。因为Artimus 并不是一个需要高安全的应用，所以我们选择了最简单最兼容的方案。

我们这就完成了对部署描述符的讨论。接下来，我们详细看看通过部署描述符载入的ArtimusServlet 以及它为我们的应用提供的资源。

15.5. ArtimusServlet

ArtimusServlet被用于初始化我们的应用所用的几个定制资源。它可以通过直接继承ActionServlet 而来，但是通常最好使用另一个servlet来装载你的资源。使用你自己的资源装载其的优点有：

- 可以利用标准的子类来完成大部分工作；
- 限制绑定到Struts的定制代码的数量；
- 保护你免于受到可能在ActionServlet 或者PlugIn 接口中发生的变更的影响；

总体而言，Artimus 需要装载3个属性文件并且初始化两个内部服务：

- Artimus 系统属性文件
- 两个SQL 属性文件
- 连接适配器
- Lucene 搜索引擎

我们的资源装载器扩展了Scaffold 包中的标准基类，并且使用了极少的定制代码就完成了工作。 Scaffold 包还有一个类似的针对PlugIn的基类，我们将在本章后面使用它以将which we will use ArtimusServlet 迁移至PlugIn。

清单15.2 列出了ArtimusServlet的全部源代码。

清单 15.2 ArtimusServlet

```
package org.apache.artimus.http;

import java.io.IOException;
import java.util.Properties;
import javax.servlet.ServletException;
import org.apache.commons.scaffold.lucene.Engine;
import org.apache.commons.scaffold.sql.ConnectionAdaptor;
import org.apache.commons.scaffold.http.ConnectionServlet;

<!-- [15.5.1] Our subclass -->
public class ArtimusServlet extends ConnectionServlet {

<!-- [15.5.2] Our String tokens -->
private static String KEYS_PARAMETER = "sql_keys";
private static String KEYS_PATH ="resources/sql_keys.properties";
private static String ARTICLE_ PARAMETER = "sql_article";
private static String ARTICLE_PATH =
          "resources/sql_article.properties";
private static String INDEX_ PARAMETER = "index.path";
private static String INDEX_PATH = "/var/lucene/artimus";

<!-- [15.5.3] Our extension point -->
protected void initCustom()
          throws IOException, ServletException {

<!-- Fetch the SQL commands -->
```

```
Properties keysCommands =
          loadProperties(KEYS_ PARAMETER,KEYS_PATH,null);
org.apache.artimus.keys.Access.init(keysCommands);
Properties articleCommands =
          loadProperties(ARTICLE_ PARAMETER,ARTICLE_PATH,null);
org.apache.artimus.article.Access.init(articleCommands);

<!-- Initialize the Lucene index path -->
String indexPath =getProperties().getProperty(INDEX_ PARAMETER);
if (null==indexPath) {
  indexPath = getInitString(INDEX_ PARAMETER,INDEX_PATH);
}
Engine.init(indexPath);
} // end initCustom
} // end ArtimusServlet
```

15.5.1. 我们的子类

清单15.2中的ArtimusServlet扩展了ConnectionServlet，后者是一个标准的Scaffold类。ConnectionServlet 将自动初始化连接适配器 (第15.4.3节)，所以不需要提供任何代码来处理那些工作。

而ConnectionServlet 其次又扩展了Scaffold ResourceServlet。同样，这个类也会自动装载默认的属性文件 (第15.4.2节)，所以我们也几乎不用担心什么。

ResourceServlet 还提供了一个工具方法来装载其他属性文件。我们将使用这个方法来载入我们的应用的SQL 命令。

15.5.2. 我们的字符串常数

清单15.2 中的静态 String 块可以在需要时在线提供。但是基于编码风格考虑，因为 Artimus 和Scaffold 包都对所有字面String提供了静态常数。这就提供了一个机会来记录这些字面字符串，并且避免了难以跟踪的缺陷。这里我们没有展示JavaDoc，但是在源代码中，每一个静态常量都包括了一个记录默认值的说明。

15.5.3. 我们的扩展点

ResourceServlet 超类(第15.5.1节) 提供了一个initCustom 方法来作为子类可以（就像我们的）可以添加自己的初始化代码的扩展点。在清单15.2中，ArtimusServlet 覆写了initCustom 以载入我们的SQL 命令以及设置Lucene 索引文件的路径。

取得SQL 命令

每个SQL 属性文件都是使用ResourceServlet 超类提供的loadProperties 方法载入的。这个方法首先检查部署描述符(web.xml) 获取定制设置，如果没有找到则使用默认设置。

为了使用loadProperties，我们将向其传入：

- 初始化参数的名称(t web.xml 中的<param-name> 元素)
- 如果在web.xml中没有给出，则需要使用的默认设置
- 可选的，应用上下文的一个默认属性名

如果传入了一个属性名称， loadProperties 将会把对属性文件的引用保存在应用上下文中。 Artimus 属性文件不需要这么做，所以传入null。

ResourceServlet 超类实际使用这个特征来将对默认应用属性的引用。Scaffold 包中的其它类将会查找这个引用以获得系统属性。核心Struts 框架使用这个相同的策略来暴露应用资源给应用中的其他组件。

在SQL 属性文件被从外部文件中载入后，每个对象都被传递到使用它的(Access.init(Properties commands))数据访问组件。

组件保存对其的引用并且在运行时将它提供给数据访问程序。

初始化Lucene 索引路径

Lucene 搜索引擎需要创建一系列必须保存在服务器文件系统中的索引文件。Scaffold 针对Lucene提供了大量的可以知道在何处保存索引文件的方便工具方法。Artimus 使你可以在默认的属性文件中或者部署描述符中指定索引文件位置。

如果这两处都没找到，则使用默认的位置。

在清单15.2中，这是通过两行调用了ResourceServlet 超类中的工具方法的代码来完成的。getProperties 方法检查默认的属性文件 (自动由超类创建)。getInitString 方法则检查部署描述符中的参数，如果没有找到的话就返回一个默认值。结果然后在运行时传递到Lucene 工具类使用。

15.6. 应用和SQL属性文件

前面说过， Artimus 属性文件使用了与标准的资源束（ResourceBundle）中所用的一样的名值对格式。(实际上，这几本上是另一种方式。资源文件是属性文件的扩展，但是谁在意呢？)

虽然这些文件非常简单，我们也要来看一看。

这些文件的工作拷贝可以在/artimus/WEB-INF/classes/resource下面找到。Artimus 属性文件示于清单15.3。

清单 15.3 /WEB-INF/classes/resources/artimus.properties

```
index.path = /var/lucene/artimus
rss.link.base=http://localhost/artimus/
rss.link.view=http://localhost/artimus/article/View.do?article=
```

清单15.3中的第一行是Lucene 搜索引擎(第15.5.3节)的索引文件路径。其他的则是用于配置默认的RSS 频道。我们将这些都放入属性文件中是为了便于系统管理员能够方便修改它们。

也可以使用XML 文件，但是那样的话有些新手可能就不那么容易处理它了。

Artimus 用来保存SQL命令的一个属性文件示于清单15.4。

清单 15.4 /WEB/-INF/classes/resources/sql_keys.properties

```
keys.next = SELECT next FROM artimus_keys WHERE name=?;
keys.inc = UPDATE artimus_keys SET next=next+1 WHERE name=?;
```

同样，清单15.4中的 SQL 命令行是任何数据库管理员都可以容易审查和编辑的内容。这里所示的两个命令是用于产生序列号。`keys.next` 命令选择当前的关键字，而 `keys.inc` 命令则对其递增以供下一次调用。

实际上该文件中还有很多命令，包括表格创建的命令，但是它们似乎太长了，所以没有在此列出。另外还有一个SQL命令文件， sql_articles.properties, 但其基本上是一样的。(两行SQL是大多数Java 无论如何都要使用的。)

在运行时，数据访问对象从属性文件获得其需要的命令，然后用它们来检索和更新数据库记录。Artimus中访问JDBC和其他数据服务在第14中讨论过。

请记住，这些文件仅在启动时载入一次。为了激活任何修改，必须重新启动应用。

15.7. index.jsp

现在我们的资源已经被初始化，我们转到 web.xml 看看Artimus 在运行时是如何呈现出来的。我们的部署描述符中的一个标准的附属物是对一个标准欢迎页面的引用，即index.jsp。

当一个访问者没有指定具体的页面时，容器便查找欢迎页面，然后返回给用户。

不幸的是，这个请求并没有直接通过通常servlet的火力范围，所以我们不能对我们的欢迎页面使用类似于**`index.do`** 之类的东西。它应该是系统中的一个物理文件，比如HTML 页面或者JSP。

同时，大多数动态应用，比如Artimus，都是设计来避免对物理文件的直接引用。我们希望控制是通过一个servlet 然后才是输出到一个页面。

作为折衷，Artimus 使用了一个普通的index.jsp，由它来转发到一个Struts Action。这是一个可以原样用在任何Struts 应用中的页面。整个JSP 示于清单15.5中；它只有两行：

清单 15.5 /index.jsp

```
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<logic:redirect forward="welcome"/>
```

index.jsp 使用了一个Struts 逻辑标签来将控制重定向到`welcome`， 后者是一个在Struts 中配置的全局转发。我们将在多个小节中展示struts-config.xml，就从下一节的全局转发 (global

forwards）开始。

15.8. 全局转发

一个Struts 配置文件可以有多个部分。首先，是一个规定的XML 头部，其后是分别针对数据源（data source），form bean，全局转发（global forward），和action mapping 的配置部分。

1.0 vs 1.1	在 Struts 1.1 中，配置文件可能还包括全局异常（global exception），控制器（controller），消息资源（message resource），以及 plug-in 元素。关于 Struts 1.1 配置的详细信息参见第 4 章和第 16 章。

差不多有400多行， struts-config.xml是Artimus 应用中最长的源文件。因此，我们将逐部分的展示它。每一个部分都是该文件的一个逻辑片段，但是要获得关于struts-config.xml 的总体印象，请参见第4章，那里有深入讨论。

清单15.6 展示的是Struts 配置文件的`<global-forwards>` 部分。

清单 15.6 /WEB-INF/struts-config.xml (全局转发)

```
<global-forwards>
<!-- default forwards -->
  <forward
    name="welcome"
    path="/do/find/Recent"/>
  <forward
    name="cancel"
    path="/do/Menu"
    redirect="true"/>
  <forward
    name="done"
    path="/do/Menu"/>
  <forward
    name="exit"
    redirect="true"
    path="http://jakarta.apache.org/"/>
  <forward
    name="failure"
    path="/do/Menu"/>
```

```
  <forward
    name="baseStyle"
    path="/assets/styles/base.css"/>
    <!-- MENU forwards -->
  <forward
    name="logon"
    path="/do/admin/Menu"/>

  <!-- ARTICLE forwards -->
  <forward
    name="article"
    path="/do/article/View"/>
</global-forwards>
```

我们已经看到第一个全局转发，welcome，起作用了。第15.7节的index.jsp 用它来将控制重定向到我们的应用中的Struts欢迎页面。而清单15.6中的welcome forward 将控制发送到上下文相对的URI /do/find/Recent上。如果我们随后想要修改欢迎页面，我们只需要修改ActionForward中的引用路径即可。

在 web.xml 中对Struts ActionServlet的设置中，我们要求容器将所有以/do/* 开头的URL都转发到servlet，而不是查找具有相同名称的文件。它们实际上是我们将通过Struts 配置中定义的ActionMappings 对象进行路由的Struts 动作。我们将在下一节讨论/find/Recent 动作（action）。

全局的cancel, done, 和 failure ActionForward 对许多应用来说都是标准的普通action。如果一个特定的ActionMapping针对这些通用事件没有提供其自己的转发，全局版本的转发便会起作用。cancel forward 的redirect 属性设置为true。这样以确保浏览器重新发送一个干净的请求，以确保请求是被确实取消了。如果某个单独的action 需要与此不同地处理cancel，它可以定义自己的转发。

Artimus 菜单页面设计来显示未决的错误或者消息，并使其成为一个路由未处理的done，Failure，和success 事件的方便之地。但是如果它发生了改变，路经可以在这里修改，而这些默认的事件将会被路由到别处。exit 转发则仅仅是给完成应用操作的用户一个某某出处的菜单。这里，我们来到了我们喜欢的站点，Jakarta Project。

NOTE	你可能注意到在清单 15.6 中并没有全局的 success 转发。在大多数情况下，每一个 action 都有其自己的成功事件的去处—通常是一个显示页面。因为未处理的成功似乎是一个意外而不是一个规则，所以 Artimus 就没有定义一个。

logon 和 article 转发比其他更加特殊一些。如果这是一个模块化的Struts 1.1 应用，它们可以其他menu 和article 元素放入它们自己的Struts配置文件中连同。

logon forward 指向admin之下的一个URI。这样就会触发我们的安全约束(第15.4.6节)。一旦用户登录进系统，menu 页面将显示一套针对映射到该用户的角色的菜单。

article forward 用于创建超链接。在运行时，使用Struts <html:link> 标签来将这个路径与对一篇文章的引用进行合并。JSP 源代码就可能看起来是这样：

```
<html:link forward="article"
           paramProperty="article"
           paramId="article">
```

但是在JSP 标签渲染之后，一个适当的超文本引用就会发送给浏览器，比如：

```
<a href="/artimus/do/article/view?article=101">Article 101</a>
```

baseStyle forward 的用法与其他不同。它不是用来产生用户可以导航的超链接的。而是用来引用存储在我们的Web应用中的样式表文件。在页面中，我们可以使用这样的标签来载入样式表：

```
<LINK rel="stylesheet"
      type="text/css" href="<html:rewrite forward='baseStyle'/>">
```

可以使用另外一种方式：

```
<LINK rel="stylesheet" type="text/css"
      href="<html:rewrite page='/assets/styles/base.css/'>">
```

但是这样就将链接绑定到具体的文件系统路径了。如果不修改所有的JSP文件，便不能重新组织你的文件系统，而这正是Struts 试图避免的事情。

Artimus 样式表非常简单，只是定义了一些基本的字体和表格大小。这里我们不会展示样式表的源代码文件，但是如果你对其感兴趣，现在你应该知道到哪里去找到它——请查看struts-config.xml中的全局转发!

15.9. /find/Recent

默认情况下， Artimus 的welcome Action 映射到/do/find/Recent。这个action 将显示最近20个提交的文章列表。这是多个查询操作的结果。Artimus 也可以按照作者，标题，内容，或者事件跨度来查找文章。

清单15.7 展示的是Struts 配置(struts-config.xml) 中描述的Artimus Find action。

清单 15.7 /WEB-INF/struts-config.xml (**Find** action)

```
<action
  path="/find/Recent"
  forward="/do/find/Last?articles=20"/>
<action
  path="/find/Last"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.article.FindByLast"
  name="articleForm"
  validate="false">
  <forward
    name="success"
    path=".article.Result"/>
</action>
<action
  path="/find/Author"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.article.FindByCreator"
  name="articleForm"
  validate="false">
  <forward
    name="success"
    path=".article.Result"/>
</action>
<action
  path="/find/Title"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.article.FindByTitle"
  name="articleForm"
  validate="false">
  <forward
    name="success"
    path=".article.Result"/>
</action>
<action
```

```
  path="/find/Content"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.article.FindByContent"
  name="articleForm"
  validate="false">
  <forward
    name="success"
    path=".article.Result"/>
</action>
<action
  path="/find/Hours"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.article.FindByHours"
  name="articleForm"
  validate="false">
  <forward
    name="success"
    path=".article.Result"/>
</action>
```

你可以从清单 15.7中看到，我们的 /find/Recent action 实际上是一个别名。它只是简单地将请求转发到do/find/Last，同时在URI上面添加了一个查询字符串`?articles=20`。这个参数限制了/find/Last action 只返回20篇文章。为了显示更多或者更少的文章，你只需要修改Struts配置中的设个设置。

你还会注意到Find action 都十分相似。实际的不同只是每一个都具有一个不同的parameter属性。参数实际上是能够执行搜索操作的业务逻辑对象的全限定类名。根据Struts 用户指南推荐，Artimus 将业务逻辑对象实现为普通的，没有绑定到Struts和Web层的 Java 类（POJOs）。这使得业务逻辑对象非常容易编写，并且允许它们用于其他环境之中。

你可能会注意到局部转发中的路径不象到JSP的文件系统路径。实际上它们是对Tiles 定义（Definition）的引用。我们在第11章讨论了Tiles，并且还会在第15.10节讨论在Artimus 中如何使用Tiles。作为约定，我们将使用点号开头命名Tiles定义，而URI 通常是以斜杠开始的。这使为了将Tiles定义和实际的文件系统路径区别开来。点号对Tiles来说并无特别的意义。

清单15.8 展示了我们的FindByLast 类的源代码。我们将在下一节讨论其热点。所有的查询类都是相似的方式实现。在讨论一个之后，你应该能够没有困难地看懂其他类的源代码。

应该小心记住的是，这是一个将由Struts Action 对象调用的业务逻辑类。而不是一个Struts类。它执行业务逻辑然后*返回结果*给Struts Action。

清单 15.8 org.apache.artimus.article.FindByLast

```
package org.apache.artimus.article;
import org.apache.commons.scaffold.lang.ParameterException;
import org.apache.commons.scaffold.util.ProcessResultBase;
import org.apache.commons.scaffold.util.ResultList;
import org.apache.commons.scaffold.util.ResultListBase;
import org.apache.artimus.lang.Tokens;
// [15.9.1] extends bean
public class FindByLast extends Bean {
public static final String PROPERTY = Tokens.LAST;
public Object execute(Object parameters) throws Exception {
 // [15.9.2] super.execute
   super.execute(parameters);
 // [15.9.3] getArticles
  Integer articles = getArticles();
  if (null==articles) {
    throw new ParameterException();
  }
  // [15.9.4] findByLast and ResultList
  ResultList list = new ResultListBase(
  Access.findByLast(this,articles)
  );
  list.setLegend(PROPERTY,articles.toString());
  // [15.9.5] ProcessResult
return new ProcessResultBase(list);
} // 执行
} // 完成 FindByLast
```

15.9.1. 扩展bean

Artimus 对其ActionForm和业务逻辑Bean都使用一个“粗粒度” bean 的方式 (参见第5章) 。Bean 的超类包含了被Artimus `article` 包中的其它业务逻辑bean使用的所有属性。在一个大型应用中，每一个包都可以由其自己的基类bean，并且也许还由不同的开发团队所管理。

Bean 超类然后扩展了Scaffold `ProcessBeanBase`类 (`org.apache.commons.scaffold.util.ProcessBeanBase`)。这是一个提供了两个

方便属性，并且更重要的是，还有一个调用Bean的入口点的方法，的轻量类。

标准的入口方法使得另一个进程，比如一个Struts Action，可以按相同的方法调用每一个对象，而不用管其具体子类。Struts 使用这种同样的控制反转技术[Johnson] 来调用你的应用定义的Action的。

Scaffold 定义了一个可执行的接口 (`org.apache.commons.scaffold.util.Executable`)，该接口提供了对`ProcessBean`的进入方法，因而也提供了对Artimus 业务bean的进入方法。

Artimus 针对每一个请求实例化一个新的业务bean，比如清单15.8所示的`FindByLast`类，这和Struts 实例化一个新的ActionForm几乎是一样的方式。还有一个方法是，你可以将业务逻辑bean编写为singleton，就像Struts Action一样，以便它们在每一个应用中仅仅被实例化一次。这可以在某些情况下提供更好的性能，但是代价是业务bean 必须是线程安全的。

15.9.2. super.execute

我们的业务bean中的FindByLast 类是通过其`execute` 方法进入的。如清单15.8所示，它是从调用超类的`execute` 方法开始的。继承自ProcessBeanBase的默认行为是将参数对象转型为一个Map，并且使用它来通过反射组装自身。标准的`BeanUtils.populate` 方法处理了数据传输。这也是Struts 用来组装ActionForm bean的同一个工具类。

另外还设计了一个接口以便任何可以创建Map的应用都可以在不绑定到Struts或者HTTP的情形下调用`FindByLast.execute` 方法。

15.9.3. getArticles

清单 15.8中的FindByLast 类需要一个参数 ;返回的文章数。它必须被作为参数对象中的Map的一个条目传入。如果它被适当的传入，超类的 `execute` 方法应该能够找到它并且用它来组装我们的bean的 `articles` 属性。如果`getArticles()` 返回null，FindByLast 将抛出一个一个`ParameterException` (`org.apache.commons.scaffold.lang.ParameterException`)。

因为缺失参数在一个多层应用中是经常发生的事情，Scaffold 为此定义了一个标准的异常，基于一个链式异常类 (`org.apache.commons.scaffold.lang.ChainedException`)。如果需要，其它层可以添加其自己的异常到链中。

15.9.4. 访问.findByLast 和ResultList

清单 15.8中的`FindByLast` 类是属于业务层的。其职责是为资源层收集参数，获得结果，并且将结果回传给应用层。Artimus 将一个包所需的数据访问方法组合到一个访问类中。该类中的方法是用普通的Java 参数，比如 `Integer` 和 `String`，并且返回一个对象或者对象的集合。

Scaffold 提供了一套有用的JDBC 工具集来帮助这个处理，但是任何数据访问方法，或者个各种方法的综合，也可以用在访问类中。例如，Artimus就使用了JDBC 和Lucene [ASF, Lucene]来执行不同的操作。关于Artimus 中如何使用各种数据服务，参见第14章。

Artimus 是用静态方法来实现访问类，但是它也可以实现为singleton 以提供更大的灵活性。也可以实现一个完全不同的类，并且从业务层bean调用。你也可以在业务bean中直接实现数据访问过程，同样的方式你可以在Struts Action中实现任何东西。但是这样却不能提供许多应用架构师所推荐的最佳的分层组件数量。许多现代设计包括一套针对应用，业务，数据访

问，以及资源层的离散的组件。表15.2 展示了一些通用的架构分层，以及对应的Artimus 组件。

ResultList

Java 语言提供了一个十分有用的Collection 接口(java.util.Collection)。Collection 经常用来将数据访问操作的结果传输到表现层。在实践中，表现层还需要知道比结果更多一些的东西。通常有描述Collection 的说明图例以及一些列的表头，这两者可能都需要本地化。

表现页面也需要知道Collection中有多少成员，如果有的话。最常见的是，页面需要一个迭代器来遍历Collection的所有成员。

表格 15.2 通用分层和 Artimus 的对应组件

架构分层	Artimus 组件
表现层	JavaServer Pages, JSP tags
控制层 I	ActionServlet, ActionForms, ActionForwards, ActionMappings, Actions
业务层	ProcessBeans, ResultList, ProcessResult, MessageResources, Properties
数据访问层	Access class, StatementUtils, LuceneUtils, ConnectionAdaptor, JDBC driver
资源层	DBMS, Lucene

Scaffold 包中的ResultList 接口(org.apache.commons.scaffold.util.ResultList) 就将这些需求包装进一个简洁的bundle之中。

ResultList 扩展了Collection，所以可以用于Collection 能用的任何地方。但是它也为方便表现层提供了一些额外的属性。关于ResultList 对象的更多细节参见第14章(第14.4节)。

清单 15.8中的FindByLast 类包装了数据访问类返回的Collection 以创建一个ResultList 对象。它也设置了ResultList 的legend 属性。在表现页面中，legend 将被在顶部渲染为last=20 以及标准页面描述的一部分，20 matches for last=20。这使得可以很容易的针对所有查询方法定制相同的表现页面。

15.9.5. ProcessResult

业务操作可以有各种各样不同的结果。一些操作可能会返回一个单一对象。而其它操作则可能返回一个对象的群集。有些可能只是返回一个消息，或者甚至一个对工作流改变的建议。通常，一个操作返回一个需要保存在上下文中以供其它组件访问的对象。这可以是servlet 上下文， JNDI 上下文，某些定制类型的上下文，或者仅仅是另一个对象的一个属性。

Scaffold 包的ProcessResult 接口(org.apache.commons.scaffold.

util.ProcessResult) 定义了一个可以描述各种可能结果的传输对象。ProcessResultBase 是一个实现了ProcessResult 接口的标准bean 类。任何操作都可以返回一个数据对象或者群集，一个消息列表，分派建议，或者任何上述东西的结合。它也可以请求将对象保存在某个特定的属性名称之下，以供另一个组件使用。

表15.3 列出了ProcessResult 的属性(由ProcessResultBase实现)它们可以用来将结果关联到操作。业务bean 设置属性，而ProcessAction 则按其行动。

表格 15.3 ProcessResult 属性

属性	说明
Name	包含结果对象的属性名称（attribute name），或者，如果应该是用默认的属性名称，则为 null。
Scope	指定应用范围或者其它上下文来将结果对象保存为一个属性（attribute）。
singleForm	指定结果是一个单一对象还是一个对象 Collection。
exposed	指定结果是否应该暴露为一个 attribute [true]。
Data	指定包含操作结果的对象。
aggregate	指定该 ProcessResult 是否是一个群集或者其它 ProcessResults[false]。
messages	包含一个应用层消息的列表，以应用的消息资源为关键字。
dispatch	包含一个对控制器的特定的路由建议。可以是一个路径或者一个逻辑名称(比如， ActionForward 名称)。很少使用。
dispatchPath	指定分派建议是一个逻辑名称（首选）还是一个 URI。

我们的FindByLast 操作只是简单地返回一个对象集合，它可以包装为一个标准的ProcessResultBase 对象。所以，它在即将灭亡的时候可以只调用默认构造器。(简单的事情*可以*仍旧简单。) 在第15.15节中我们创建Artimus 的主菜单项的时候，会展示ProcessResult 对象的更高级的用法。

15.9.6. ProcessAction

所有示于清单 15.7中的Find action 都调用了Scaffold 包的ProcessAction (org.apache.struts.scaffold.ProcessAction)。该Action 对象然后调用给做parameter属性的业务bean 类，比如我们在第15.9.1节到第15.9.5节中所描述的FindByLast 类。ProcessAction 是一个经过多个Struts 应用的开发经验之后提炼而开发出来的综合对象。它扩展了Scaffold BaseHelperAction类，后者在第8章讨论过。BaseHelperAction 负责处理从parameter属性实例化业务bean (或beans) 的具体细节。

ProcessAction 只是必须调用bean 并且处理结果。为了调用我们的业务bean，

ProcessAction 从引入的ActionForm (如果有) 提取出一个Map (java.util.Map) 然后调用我们的业务bean的execute 方法， 将Map传递给它。 这和ActionServlet 在调用一个Action时所用的是相同的控制反转技术。不同点在于：

- 我们的业务bean 没有绑定到任何web层的类。
- 我们的业务bean 之间可以达成不能在Web层面达成的API 契约需求。

后一个差别意味着可以做一些事情，比如希望将数据表达为任何正确的类型—或者如果它们不正确则触发一系列异常。在web层，无效的输入通常是一个我们应该在一般情况下客气地纠正的用户错误。在这个层次上，坏（错误）数据通常表现为需要引起开发人员和管理员注意的编程缺陷。

对于一个ProcessBean，比如我们的FindByLast bean (第15.9.1 到第 15.9.5节)，的API 契约，是从一个Map中提取它需要的参数，并且在一个ProcessResult中返回结果。action 则使用ProcessResult 对象来决定操作的结果。这和ActionServlet 在检查ActionForward 来决定下一个步骤时所用的是同样的模式。

如果ProcessResult 包括有消息，ProcessAction 将会把它们包装为 ActionErrors 或(从Struts 1.1以后) ActionMessage 对象。如果结果中包含数据，action 将会把对象或者对象群集保存在适当的范围中。

如果结果包括特定的分派建议， action 将查找或者创建适当的ActionForward。action 是通过分析示于表15.3中的ProcessResult 的属性来完成这个工作的。

在我们的FindByLast 操作中，如清单15.8所示，bean 只是返回包装在一个默认 ProcessResult 对象中的一个ResultList 群集。ProcessAction 然后将 ResultList 集合保存在request 范围中，并使用默认的 RESULT 属性名称，然后将控制转发到success。

如果因为某些原因 articles 参数丢失(第15.9.3节)，ProcessAction 将自动捕获并记录异常，并将消息包装在一个ActionError中，然后将控制转发到failure。

在这两种情况下，我们都将控制路由到了表现层，在那里，我们的ResultList或者异常消息将会显示给用户。

而它会带我们到Tiles。

15.10. tiles.xml 和Article.jsp

我们注意到示于清单 15.6 中的局部转发的路径都是对Tiles Definition (我们在第11章讨论了Tiles) 的引用。它们是从一个tiles.xml 配置文件中装载的，该配置文件非常象 struts-config.xml 或者web.xml 文件。清单 15.9 展示了我们的Artimus的Tiles 配置文件。

清单 15.9 /WEB-INF/tiles.xml (Artimus 1.0)

```
<tiles-definitions>
  <definition
        name=".article.Base"
        path="/article/common/layouts/Article.jsp">
```

```
    <put name="title" value ="${title}"/>
    <put name="header" value="/article/common/header.jsp"/>
    <put name="message" value="/article/common/message.jsp"/>
    <put name="content" value="${content}"/>
    <put name="navbar" value="/article/common/navbar.jsp"/>
  </definition>
  <definition
         name=".article.Result" extends=".article.Base">
    <put name="title" value="article.Result.title"/>
    <put name="content" value="/article/content/result.jsp"/>
  </definition>
  <definition name=".article.View" extends=".article.Base">
    <put name="title" value="article.View.title"/>
    <put name="content" value="/article/content/view.jsp"/>
  </definition>
  <definition name=".article.Form" extends=".article.Base">
    <put name="title" value="article.Form.title"/>
    <put name="header" value="/article/common/headerForm.jsp"/>
    <put name="content" value="/article/content/form.jsp"/>
  </definition>
  <definition name=".article.Menu" extends=".article.Base">
    <put name="title" value="article.Menu.title"/>
    <put name="content" value="/article/content/menu.jsp"/>
    <put name="navbar" value="/article/common/navbarMenu.jsp"/>
  </definition>
</tiles-definitions>
```

我们的Tiles 配置文件定义了一个基础布局定义（layout definition）和四个显示页面：Result、View、Form、 以及Menu。显示页面从基础定义继承了大部分的标记。我们只需要提供一个标题（title）字符串和内容文件即可。标题字符串实际上是来自于我们的应用的消息资源中的一个关键字。这样可以避免将字面文本掩埋在程序员的配置文件中并且使Artimus更容易在随后进行本地化。

作为约定，我们将我们的Tiles定义用点号开始进行命名，而这里URI通常以斜杠开始。这是为了将定义从实际的文件系统中区分开来。

点号对Tiles来说并无任何特殊意义。

通过使用Tiles，当ActionServlet 处理一个ActionForward的时候，Tiles 定义首先被

检查。如果ActionForward 路径匹配一个Tiles Definition 的名称，ActionServlet 将为该定义创建一个Tiles 上下文，然后将控制转发到Definition的路径指定的URI。该路径通常会导引到一个使用Tiles 标签和来自于Definition的详细内容来渲染结果相应的JSP页面。清单 15.10 就展示了我们的Article 页面的基础Tile。

清单 15.10 /pages/article/tiles/layouts/Base.jsp

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<!-- [15.10.1] useAttribute-->
<tiles:useAttribute name="title" scope="request"/>
<html:html>
<HEAD>
  <html:base/>
  <!-- [15.10.2] baseStyle-->
  <LINK rel="stylesheet" type="text/css" href="<html:rewrite
        forward='baseStyle'/>">
  <!-- [15.10.3] title-->
  <TITLE>
    <bean:message key="app.title"/>
    <bean:message name="title"/>
  </TITLE>
</HEAD>
<!-- [15.10.4 tiles -->
<tiles:get name="header"/>
<tiles:get name="message"/>
<tiles:get name="content"/>
<tiles:get name="navbar"/>
</BODY>
</html:html>
```

15.10.1. useAttribute

Tiles 使用其自己的上下文，类似于标准的请求或者会话上下文。当Tiles Definition，比如清单 15.9中所示的那个，对一个值上使用 `put`， 就是将它放入Tiles 上下文。`<useAttribute>` 标签，示于清单 15.10，则使改值能够被某个标准上下文所访问。默认情况下是page 范围的上下文。

我们的 `<useAttribute>` 标签指定了request 范围，以便其它tiles 也可以访问`title` 属

性（attribute）。 (每个tile有其自己的page上下文)。

`title` 属性是我们的应用的消息资源中的一个关键字，我们将在页面中随后使用它。

15.10.2. baseStyle

为了保持一致性，我们的所有 tiles 都使用同一组样式。为了使其能够很容易的切换到另一个样式，在清单 15.10 中，我们将到样式表的路径维护为一个ActionForward。`rewrite` 标签负责从Struts 配置中返回样式表的路径。

15.10.3. title

因为传递给我们的页面的`title` 属性实际上是应用消息资源中的一个关键字，在清单15.10中我们使用了`<bean:message>` 标签来渲染实际的文本。我们还查找了另一个消息，app.title。这样最终产生了一个诸如 “Artimus—Article Search Result”的页面标题。

15.10.4. Tiles

清单15.10多渲染的页面大部分来自于组件tiles。布局页面使用逻辑名称引用tile。到JSP 的路径在示于清单 15.9中的Definition 中给出。当ActionServlet 处理一个使用了Tiles Definition 的ActionForward 时，servlet 会将Definition 放入布局页面能够找到的Tiles 上下文中。比如我们的结果页面的情形中，布局将包含标准的header.jsp，message_1_0.jsp，和navbar.jsp，还有就是result.jsp 内容标题。

因为Artimus 页面十分简单，这些 tiles 中的标记数量并没有什么大不了的。在使用了更加复杂的标记的页面中，其节省就比较可观了。

清单15.11，15.12，和15.13 分别展示了header，message，和navbar tiles的JSP源代码。

清单 15.11 清单 15.11 /pages/article/tiles/heading.jsp

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<BODY>
<!-- OUTER TABLE -->
<TABLE class="outer">
  <TR>
    <TD align="center">
    <!-- INNER TABLE -->
      <TABLE class="inner">
    <TR>
      <TD class="heading"
           colspan="3">
        <bean:message name="title"/>
      </TD>
    </TR>
```

清单 15.12 /pages/article/tiles/message.jsp (1.0)

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-validator" prefix="validator" %>
<validator:errorsExist>
<TR>
  <TD class="error" colspan="3">
    <UL>
      <validator:errors id="error">
        <LI><bean:write name="error"/></LI>
      </validator:errors>
    </UL>
  </TD>
</TR>
</validator:errorsExist>
<validator:messagesExist>
  <TR>
    <TD class="message" colspan="3">
      <UL>
        <validator:messages id="message">
          <LI><bean:write name="message"/></LI>
        </validator:messages>
      </UL>
    </TD>
  </TR>
</validator:messagesExist>
```

清单 15.13 /pages/article/tiles/navbar.jsp

```
    <%@ taglib uri="/tags/struts-html" prefix="html" %>
  </TABLE>
  </TD>
</TR>
```

```
<TR>
  <TD class="navbar">
    <html:link forward="done">DONE</html:link>
  </TD>
</TR>
</TABLE>
```

清单 15.12 使用了Struts Validator 1.0 标签来显示错误和确认消息。这个标签库已经集成到Struts 1.1 的标签库中了。在下一章我们将Artimus 迁移到Struts 1.1的时候，这个tile将把这些标签使用更新的标签替换掉。

message tile 用来输出在请求中发现的错误和/或消息。Artimus 没有使用`<html:error>`标签来渲染错误消息。这个标签的设计是你可以很容易的渲染错误消息，但是鼓励你在消息资源文件中放入HTML 标签。我们使用的是Struts Validator 1.0 的标签集。Validator的错误和消息标签可以象迭代器（iterator）一样的工作。

在每个迭代器中，文本消息将在你所指定的ID下面进行暴露。

然后`<bean:write>`标签就可以象使用其它字符串一样来渲染这些消息。

和Struts 1.1类似，validator 标签支持两种消息队列：一个针对实际错误，另一个则针对其它消息。我们的message 标题首先显示错误然后再是消息。Scaffold 的BaseAction (`org.apache.commons.scaffold.BaseAction`) 在Struts 1.0 和 Struts 1.1下都仅支持消息类型的队列。

为了能够将复杂的代码，比如清单 15.12 所示，放入一个tile ，可以避免我们在每一个需要显示消息的页面之间进行大量的拷贝和粘贴工作。

前面说过，我们在迁移到Struts 1.1时将需要修改清单 15.12 中所用的标签，但是布局和输出策略会保持不变。

因为它是一个tile，我们将只需要载一个地方进行修改。如果没有tiles，我们便必须修改显示消息的每一个页面(在Artimus 是所有页面)。

清单15.11 和15.13 都是Tiles/JSP 标记的简洁明了的例子。将这些片断表示为tiles 意味着我们可以不必在所有页面中拷贝和粘贴相同的标记代码。我们的result, view, form, 和menu tiles 都只包含与其它每一个页面相比独特的标记。

results page中的内容tile 在第15.11中讨论。

15.11. result.jsp

因为result.jsp 是一个tile 而不是一个完整的JSP，清单 15.14 中的标记只涉及到具体的业务部分。所有通常的HTML 装饰都是由其他tiles提供的，所以所有的result所需做的就是渲染该页面的独特内容。

当ProcessAction (第15.9.6) 将我们的ResultList 放入到request 上下文时，它使用默认的属性

名称RESULT。这就是说我们可以使用各种Struts标签并且指定name="RESULT"来访问结果列表中的各种属性。

我们从业务层接收到的ResultList并不是一个ActionForm bean的群集。我们所拥有的只是一个我们的业务bean (org.apache.artimus.article.Bean)的群集。但是因为我们设计我们的ActionForm时，其属性名称匹配业务bean的属性名称，我们就可以最方便的使用任何一个bean。

Struts 框架需要一个ActionForm 来自动捕捉和校验HTTP 参数。否则，它并不在意你使用何种形式的JavaBean 来传输你的数据。标签绑定到属性名称，而不是bean 类型。

当然，从技术上讲，直接将业务bean 传递到页面是很糟糕的(kluge)。在实现时，我们将安排我们的业务层和表现层的属性名进行匹配。然而，因为这些层并不是邻近的，业务层应该能够修改其属性名称而不会影响到表现页面 [POSA]。

幸亏在实践中，业务层将只是做那些事情。我们的bean 并不是直接来自于业务层，但是是由控制器*中继*到表现层的。在属性名称改变的情况下，控制器将会将业务bean 再次包装到匹配属性名称的适配器对象。现在担心这些没有实际的价值，所以我们做了简化而直接传递以业务bean，如清单 15.14所示。

清单 15.14 /pages/article/content/result.jsp

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<!-- [15.11.1] legend -->
  <TR class="message">
    <TD align="center" colspan="3">
      <bean:write name="RESULT" property="size"/>
        matches for
      <bean:write name="RESULT" property="legend"/>
    </TD>
  </TR>
  <!-- [15.11.2] isResult? -->
  <logic:notEqual name="RESULT" property="size" value="0">
  <TR class="subhead">
    <TH>id</TH>
    <TH>article</TH>
    <TH>contributed</TH>
  </TR>
  <TR>
  <!-- [15.11.3] RESULT -->
    <logic:iterate id="row" name="RESULT" property="iterator">
```

```
    <TR>
      <TD nowrap>
        <bean:write name="row" property="article"/>
      </TD>
      <TD width="100%">
        <html:link forward="article" paramName="row"
                paramProperty="article" paramId="article">
          <bean:write name="row" property="title"/>
        </html:link>
      </TD>
      <TD nowrap>
        <bean:write name="row" property="contributedDisplay"/>
      </TD>
    </TR>
  </logic:iterate>
</TR>
</logic:notEqual>
```

15.11.1. legend

在清单 15.14中的 result tile的顶部，我们将显示页面的说明。

说明是从ResultList 对象(第15.9.4节)所提供的一些属性组合起来的。`size` 属性返回Collection中的条目的数量。`legend` 属性是在业务层创建的。对/find/Recent action (第15.9)，legend 通常会读作`20 matches for last=20`。

同样的result tile 也会被所有搜索操作所使用。legend 负责定制页面并且保持关于用户处于工作流的何种位置的相关信息。

15.11.2. isResult?

许多搜索操作并没有返回匹配结果。清单 15.14中的 result tile 将检查ResultLis 的`size` 属性一看是否有匹配结果返回。如果有，将显示表格的列头，然后继续迭代整个Collection 条目。

如果没有匹配结果，将跳过剩下的标记。这时，这个tile唯一显示的内容是`0 matches for...`.

15.11.3. RESULT

当有匹配结果返回时， Struts `<logic:iterate>` 标签被用于遍历Collection。在迭代过程中， 每一个条目都会被暴露为一个名为`row` 的脚本变量。ResultList 对象提供了一个能够和Struts一起使用的标准`getIterator()` 方法。 针对群集中的每一个条目都会产生一行代码块 (`<TR> ... </TR>`)。那么发送给浏览器的标记可能像这样：

```
<TR>
  <TD nowrap>101</TD>
  <TD width="100%">
    <a href="/artimus/do/view/Article?article=101">
      Welcome to Artimus
    </a>
  </TD>
  <TD nowrap>2002-12-25 18:30:03</TD>
</TR>
```

<html:link>

<html:link> 标签首先得到article 全局转发 (第15.8节)的 path 属性，这里是/do/view/Article。 paramProperty 和 paramId 属性指定将article 用作查询字符串参数，以及 其值的row.getArticle() 的结果的名称 (或 id)。<html:link> 标签添加了servlet 上下文，并将所有综合在一起产生这样的链接：

```
<a href="/artimus/do/view/Article?article=101">
    Welcome to Artimus
</a>
```

这个链接也写成为：

```
<html:link
       page="/do/view/Article"
       paramName="row" paramProperty="article"
       paramId="article">
```

但是这样嵌入了太多的我们的API细节到页面中。使用ActionForward 证明我们需要一个到article 的超链接，并且使随后我们修改article 超链接更加容易。

一个例子是，在一个intranet环境中，每一个使用应用的用户都是一个Contributor。在繁重的编辑期间，用户可能会跳过查看（view ）阶段而直接跳到编辑页面。如果你将article 的ActionForward 路径从/do/View/article 修改为/do/Edit/article，其连接就会在应用中出现的所有地方自动修改。后来，如果他们又改变了主意 (如用户习惯做的)，一处修改就将一切还原。

contributedDisplay

一个应用中总有很多属性需要一特殊的方式进行渲染。数据库可能将一个电话号码保存唯一个简单的数字字符串，但是你的业务需求可能说电话号码必须以空格或者连字符进行格式化

以便容易阅读。

处理这些需求的一个好方法的使用助手方法。助手方法从原始数据库值开始，然后将其修改为符合业务需求的格式。一个需求可能是需要人名全部都是大写字母。而在其他地方，人的名字应该首先显示姓（last name）。使用助手模式，我们可以在我们在传输对象中一次性保存院士值，然后根据需要进行修改。`personAllCap` 方法可以连接name 属性并已全部大写字母返回结果。

`personDirectoryStyle` 方法则可返回name 属性的另一个连接形式，以last name 开始，并且不改变大小写。这两个方法都可以使用相同的基础属性集，只是以不同的方法处理它们。

STRUTS TIP	使用 Display 属性来转换和传输数据。

日期和其它一些地方也需要使用助手方法。在内部，数据库将投稿日期保存为一个二进制的时间片（Timestamp）。但是当我们在业务层和应用层之间传输该属性时，我们宁愿将其表示为用户可以查看和编辑的字符串。

Artimus 使用一个`contributedDisplay` 助手属性来将`Timestamp` 处理为`String`，然后由将其转换回`Timestamp`。ActionForm 仅使用 `String` 属性，`contributedDisplay`。在业务层，有两个属性：一个`Timestamp` 类型的 `contributed` 属性，用来保持数据库值；以及另一个`String` 类型的`contributedDisplay` 助手属性：

```
private Timestamp contributed = ConvertUtils.getTimestamp();
public Timestamp getContributed() {
  return (this.contributed);
}

public void setContributed(Timestamp contributed) {
  this.contributed = contributed;
}

public void setContributedDisplay(String contributedDisplay) {
if (null==contributedDisplay)
  this.contributed = null;
else
  this.contributed = Timestamp.valueOf(contributedDisplay);
}

public String getContributedDisplay() {
```

```
Timestamp contributed = getContributed();
if (null==contributed)
  return null;
else
  return contributed.toString();
}
```

注意contributedDisplay 属性没有其自己的字段。它是来自于contributed 属性，转换值，然后返回所需的结果。

再说一下，它们是article 业务bean (org.apache.artimus.article.Bean)中的方法。我们用于编辑的ActionForm，示于清单 15.15，只是将contributedDisplay 保存为一个简单的String 属性。由业务bean来处理转换问题。

目前应用中的日期处理是很初级的，可能会在应用的将来发布版本中进行改善。

清单 15.15 org.apache.artimus.struts.Form

```
public class Form extends BaseForm {
private String primaryKey = null;

public String getPrimaryKey() {
  return (this.primaryKey);
}

public void setPrimaryKey(String primaryKey) {
  this.primaryKey = primaryKey;
}

private String marked = ConvertUtils.STRING_ZERO;

public String getMarked() {
  return (this.marked);
}

public void setMarked(String marked) {
  this.marked = marked;
}
```

```
private String hours = null;

public String getHours() {
  return (this.hours);
}
public void setHours(String hours) {
  this.hours = hours;
}
private String articles = null;

public String getArticles() {
  return (this.articles);
}

public void setArticles(String articles) {
  this.articles = articles;
}

private String article = null;

public String getArticle() {
  return (this.article);
}

public void setArticle(String article) {
  this.article = article;
}

private String contributor = null;

public String getContributor() {
  return (this.contributor);
}

public void setContributor(String contributor) {
  this.contributor = contributor;
```

```
}

private String contributedDisplay =
        ConvertUtils.getTimestamp().toString();

public String getContributedDisplay() {
  return this.contributedDisplay;
}

public void setContributedDisplay(String contributedDisplay) {
  this.contributedDisplay = contributedDisplay;
}

private String creator = null;

public String getCreator() {
  return (this.creator);
}

public void setCreator(String creator) {
  this.creator = creator;
}

private String title = null;

public String getTitle() {
  return (this.title);
}

public void setTitle(String title) {
  this.title = title;
}

private String contentDisplayHtml = null;

public String getContentDisplayHtml() {
```

```
  return (this.contentDisplayHtml);
}

public void setContentDisplayHtml(String contentDisplayHtml) {
  this.contentDisplayHtml = contentDisplayHtml;
}
} // end Form
```

15.12. Article actions

当用户从结果页面（第15.11节）选择了一个文章链接，其结果就是调用应用的/article/View action。清单 15.16 展示了/article /View 和其它Article action 是在Struts 配置中是如何定义的。

清单 15.16 /WEB-INF/struts-confg.xml (Article action)

```
<action
  path="/article/View"
  type="org.apache.struts.scaffold.ProcessAction"
  paramenter="org.apache.artimus.article.FindByArticle"
  name="articleForm"
  scope="request"
  validate="false">
  <forward
    name="success"
    path=".article.View"/>
</action>
<action
  path="/admin/article/Edit"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.article.FindByArticle"
  name="articleForm"
  scope="request"
  validate="false">
  <forward
```

```
    name="success"
    path=".article.Form"/>
</action>
<action
  path="/admin/article/Input"
  forward=".article.Form"
  name="articleForm"
  scope="request"
  validate="false"/>
<action
  path="/admin/article/Store"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.article.Store"
  name="articleForm"
  scope="request"
  validate="true"
  input=".article.Form">
  <forward
    name="success"
    path=".article.View"/>
</action>
<action
  path="/admin/article/Delete"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.article.Delete"
  name="articleForm"
  scope="request"
  validate="false">
  <forward
    name="success"
    path=".article.Menu"/>
</action>
<action
  path="/admin/article/Restore"
  type="org.apache.struts.scaffold.ProcessAction"
```

```
  parameter="org.apache.artimus.article.Restore"
  name="articleForm"
  scope="request"
  validate="false">
  <forward
    name="success"
    path=".article.Menu"/>
</action>
```

和先前所述的Find action (清单 15.7)类似，示于清单 15.16中的大多数Article action 都是使用标准的ProcessAction 来启动业务bean的。

业务 bean 是通过`parametery`属性指定的。Action 也使用了所有共享的 `articleForm` form bean，它定义了被这些所有action所需的属性。它们中大多数都不需要校验，除了Store action之外。

我们的View action 使用了FindByArticle 业务类来通过其ID编号来定位文章。如果操作成功，请求将被转发到文章查看页面：

```
<forward
  name="success"
  path=".article.View"/>
```

到查看页面的系统路径是通过一个Tiles 定义来处理的(第15.10节)：

```
<definition name=".article.View" extends=".article.Base">
  <put name="title" value="article.View.title"/>
  <put name="content" value="/pages/article/view.jsp"/>
</definition>
```

换句话说，如果操作没有成功，我们的全局`failure` ActionForward (第15.8节)便会发挥作用：

```
<forward
  name="failure"
  path="/do/Menu"/>
```

它将请求发送到Menu action 和最终的菜单页面。(我们将在第15.15节讨论Artimus菜单。)

同样的模式在其它action中重复使用。ProcessAction 从ActionForm 属性中创建一个Map ，并将它传递给业务bean。业务bean执行具体的业务操作并且返回结果。ProcessAction 分析结果并且将控制路由到`success` 或者 `failure`。

QUERY	成功和失败：这就是全部吗？当然，架构并不限于是成功和失败。在实践中，只是这一经足够处理任务的绝大多数情况了。在第 15.15 节，我们更好地使用了 ActionForward 所提供的灵活性，即我们使用它们来作为菜单任务的路由器。

当后台action 该一段落的时候，现在是该来看看表现层了。我们首先来看view.jsp tile 并且看它是如何根据用户的安全角色来调整其表现的。

15.13. view.jsp

最常见的情况是，用户通过点击一个文章的标题来从结果页面 (第15.11节)来到查看页面。这也可以通过菜单页面使用文章ID直接来到查看页面。 (我们将在第15.15节讨论菜单actions。)

清单 15.17 展示了view.jsp tile。记住，这只是一个内容tile。

其它tiles，比如通过Tiles Definition 和布局页面指定的(第15.10节)。则提供了余下的标记。

这段代码清单引入了一个我们还没用过的标签：`<req:isUserInRole>`。这是来自于Jakarta Taglibs Request 标签库。它测试用户的安全角色，即在我们的部署描述符中声明的(第15.4.6节)。 查看页面用它来决定是否显示Edit 和Delete 按钮。

和结果页面 (第15.11节)一样，这个表单背后的bean 并不是一个ActionForm 而是一个我们的业务bean的实例。

清单 15.17 /pages/article/content/view.jsp

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<%@ taglib uri="/tags/tiles" prefix="tiles" %>
<%@ taglib uri="/tags/request" prefix="req" %>
<TR>
  <!-- [15.13.1] headline-->
  <TD class="headline" colspan="3">
    <H2><bean:write name="articleForm" property="title"/></H2>
  </TD>
</TR>
<TR>
  <TD class="author" colspan="3">
    by <bean:write name="articleForm" property="creator"/>
  </TD>
```

```
</TR>
<TR>
  <!-- [15.13.2] content -->
  <TD class="article" colspan="3">
    <bean:write name="articleForm" property="contentDisplayHtml"
filter="false"/></TD>
</TR>
  <!-- [15.13.3] contributor -->
<req:isUserInRole role="contributor">
<TR>
  <TD colspan="3"><HR /></TD>
</TR>
<TR>
  <%-- DELETE --%>
  <logic:equal name="articleForm" property="marked" value="0">
    <html:form action="/admin/article/Delete">
      <TD class="button.left">
        <html:submit>DELETE</html:submit>
      </TD>
      <html:hidden property="article"/>
    </html:form>
  </logic:equal>
  <%-- RESTORE --%>
  <logic:equal name="articleForm" property="marked" value="1">
    <html:form action="/admin/article/Restore">
      <TD class="button.left">
        <html:submit>RESTORE</html:submit>
      </TD>
      <html:hidden property="article"/>
    </html:form>
  </logic:equal>
  <html:form action="/admin/article/Edit">
    <TD class="button" colspan="2">
      <html:hidden property="article"/>
      <html:submit>EDIT</html:submit>
      <html:cancel>CANCEL</html:cancel>
```

```
    </TD>
  </html:form>
</TR>
</req:isUserInRole>
```

15.13.1. 大标题

通过很好的利用样式表，清单15.17 中的页面在tile的顶部显示了title 和creator (或者author) 属性。样式表的引用是在布局页面中定义的 (第15.10节)。

15.13.2. 内容

我们的文章内容是作为一个字符串插入到页面中的。注意在清单 15.17中，contentDisplayHtml 属性的 <bean:write> 标签的filter 属性是设置为false的。这可以让HTML 能够直通到页面。<bean:write> 的默认设置是 filter=true。那么过滤器将对HTML 标签进行转义，以确保它们不会扰乱周围的标记或者产生一些安全问题。

contentDisplayHtml 属性的API 契约是，方法负责处理任何标记问题。如果需要调用原本未经处理的content 属性，那么过滤器应该保证打开。

contentDisplayHtml 属性其实是另一个助手模式的例子。

助手模式也用在 contributedDisplay 属性中 (第15.11.3节)。

QUERY	为什么我们那么需要一个 contentDisplayHtml 助手方法？拷贝 HTML 标记之类的不是 JSP 应该做的吗？处理从其它层返回的数据并且正确显示给用户是表现层的职责。有时这些数据是需要用标记包围的纯文本。而另一些时候，它们可能是必须位于页面中的其它 HTML 标记之中的 HTML 片段。

在contentDisplayHtml的情况中，内容可以存储为其它某些格式，并且必须在显示之前将之转换为HTML 。因为JSP 不能期望用来复制任意的格式，那么这就成了业务层来管理这个转换的职能。内容也可能被其它除JSP之外的，能够理解HTML的组件所访问。

15.13.3. contributor

如前所述， <req:isUserInRole> 用于决定是否显示contributor 控件。可能还有其它方式来处理它，但是在实践中，这是最通用的方式。

一些 MVC 行家可能会抱怨，安全角色并不应该是表现层所关系的东西，并且不应该嵌入到标记中。但是实际应用中，许多表现设计都会围绕用户在应用中的角色来设计。如果他们登录进系统，则故事开始展开，看到的页面可能是这个样子。如果没有，看到的页面可能有时另一个样子。

所以，我们可以开发两个内容tile，一个用作contributor，而另一个针对非contributor。

我们也可以使用Tiles 来排列它，以便使contributor tile 仅针对contributor才包括进来。但是它们之间仅有语义上的差别。

重要之处在于安全角色是通过一个我们用来控制我们的应用的定制标记来决定的。页面只知道标签所告诉它的东西。如果我们想要用一个完全不同的方式来决定用户角色，我们可以替换掉该标签的实现而不用修改页面中的任何标记。标签自身就提供了一个Model 2/MVC 设计所需的分层机制。

但是关键在于，标签并不是安全的最终裁决者。当操作被请求时，用户的凭证将在其被执行之前自动被检查。能够请求操作还不够；你还必须能够访问action的URI。而访问许可则超出了表现层的控制。

delete / restore

清单 15.17中的另一些逻辑是测试我们的`marked` 属性。`marked` 属性用来指示一篇文章是否被选择。如果`marked` 被设置为0，则表示没有被标记选择。所以我们会显示一个Delete 按钮，并将它指向 delete action。/admin/article/Delete action 将`marked` 设置为1。所有标准的FindBy操作都设计来忽略`marked` 的文章，所以显示的记录都可以被删除。

其中的异常是FindByArticle 操作。如果我们知道文章的ID，我们仍然可以提取出它，即便它已经被删除了。在这种情况下，将不会显示Delete 按钮，而代之以Restore 按钮 (`marked` 等于1)。/admin/article/Restore action 会将`marked` 设置回 0，文章便可以有效地进行反删除。

那么为什么我们围着Robin Hood的谷仓来来去去就只是为了删除一个记录？在实际应用中，从数据库表中移除记录通常是留给数据库管理员最好的作业。一个记录的空间消耗如今已不是什么问题了，特别是删除可以会使系统的性能退化。Artimus 起先只是标记需要删除的文章，但是实际的删除会留给另一个操作。安全第一。小心始得万年船。.

edit

除了删除和恢复操作之外，一个contributor 也可以编辑文章。

/admin/article/Edit action，示于清单 15.17，只是转发到编辑页面。因为URI 位于/admin/*之下，容器将只允许那些属于contributor 或者manager角色的用户进行(第15.4.8节)。

15.14. edit.jsp

和view.jsp (第15.13节)相似，form.jsp 只是一个更大页面的一个标题。

编辑页面的全部源代码示于清单清单 15.18。因为用于客户端校验的标签在Struts 1.0 和Struts 1.1之间发生了一些轻微的改变，我们将在下一章展示本页面更新到1.1 的版本。清单15.18 展示了(几乎一样的) 1.0 版本。

对于记录而言，这实际上是一个“添加/编辑”页面。我们的业务bean 会足够聪明，可以区分插入还是修改操作，所以我们可以在这两种情况下都使用同一个页面：

清单 15.18 /pages/article/content/form.jsp

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<html:form action="/admin/article/Store"
           onsubmit="return validateArticleForm(this);">
```

```
  <TR>
    <TD class="label" nowrap>Title:</TD>
    <TD class="input" colspan="2">
    <html:text property="title" size="50" maxlength="255"/></TD>
  </TR>
  <TR>
    <TD class="label" nowrap>Author:</TD>
    <TD class="input">
      <html:text property="creator" size="30" maxlength="75"/></TD>
    <TD class="hint">
      Full name of person who originated the article.
    </TD>
  </TR>
    <!-- [15.14.1] article content -->
  <TR>
    <TD class="label" nowrap>Article:</TD>
    <TD class="input" colspan="2">
    <!-- The Struts html:textarea tag does not support wrapping -->
    <!-- so we use this trick instead -->
      <textarea name="contentDisplayHtml"
            rows="12" cols="50" tabindex="2"
            wrap="soft">
        <bean:write name="articleForm"
                    property="contentDisplayHtml"/>
      </textarea>
    </TD>
  </TR>
  <!-- [15.14.2] contributed / contributor -->
  <html:hidden property="contributedDisplay"/>
  <html:hidden property="contributor"/>
  <!-- [15.14.3] article id -->
</TR>
<TD class="label" nowrap>Article ID:</TD>
<logic:notPresent name="articleForm" property="article">
  <TD class="dimmed">
    <i><html:hidden property="article"/>not assigned</i>
```

```
  </TD>
</logic:notPresent>
<logic:present name="articleForm" property="article">
  <TD class="input"><html:hidden property="article"/>
    <bean:write name="articleForm" property="article"/></TD>
</logic:present>
<TD> </TD>
</TR>
<TR>
<TD class="button" colspan="3">
<html:submit>SAVE</html:submit>
<html:cancel>CANCEL</html:cancel>
</TD>
</TR>
</html:form>
<!-- [15.14.4] client-side validation (1.0)-->
<validator:javascript formName="articleForm"/>
```

唔，这个页面背后的bean 是我们的业务bean的一个实例吗？或者说我们最终会将其保存到一个ActionForm吗?

答案？此表单背后的bean 的确是一个业务 bean 实例 或者一个 ActionForm。但我们获取一篇文章来编辑的时候，本页面便被一个我们接收自业务层的业务bean组装。如果我们是创建一篇新文章，或者我们的文章没有通过Web层的校验，那么页面将使用一个ActionForm来进行渲染。

改变bean 的类型对标签来说是透明的。标签只看属性名称； bean 的类型不是实质性的。只要属性名称批匹配，任意数量的bean都可以共享同一页面。

15.14.1. 文章内容

清单 15.18 是用了一个`<textarea>` 元素来进行文章内容的编辑。Struts html 标签库的确提供了一个`<html:textarea>` 标签来创建这个元素，但是我们没有选择使用它。在我们的`<textarea>` 元素中，我们想要使用`wrap=soft` 属性。这个属性不是`<textarea>`的一个官方HTML 4.01 属性，但是流行的浏览器都会将其解释为文本区域中的长行都应该被自动折叠以适合文本区域。你也可以手工完成同样的事情，但是如果没有使用这个特征用户将会有抱怨。

基于标准的原因， Struts 团队没有添加一个`wrap` 属性到`<html:textarea>` 标签中，除非并且直到它被采纳为官方规范的一部分。

所以，通过在标准和用户基础之间的选择，我们采用了创建我们自己的动态`<textarea>` 元素的方法。这很简单。只需要按照旧式的风格编写HTML 元素，然后使用`<bean:write>` 来

组装它。

在查看页面(第15.13节)中，我们添加了`filter=false` 到`<bean:write>` 标签中，以便它不会转义任何可能在内容文本中发现的HTML 标记。这可以允许撰稿人在其文章中使用HTML。而从编辑的角度来看，我们却保持过滤开关打开，以保证标签被转义。否则，我们将不能够编辑它们。这使得在HTML之间往返进行编辑更加容易。

和处理我们的投稿数据 (第15.11.3节)相同，内容被渲染的方式简单但已充分。这些特征可以在将来的版本中改善。理想情况下，各种内容类型都应该支持；如果要使用，HTML 应该在接受之前被校验并且/或者转换成XHTML，等等。

15.14.2. Contributed / contributor

未来版本的Artimus 将要支持对投稿人的跟踪以及可以选择投稿日期。相关属性已经加入，但是目前还没有完全实现。所以，在清单 15.18中，目前为止我们只是讲属性作为隐藏字段进行传递。(好像很熟悉?)

15.14.3. Article ID

如前所述，这个页面同时用于编辑和修改文章。业务 bean 通过查找文章ID来区分不同操作。如果没有article ID，那么就是一个插入操作。否则就是一个更新操作。在清单 15.18中，我们使用了`<logic:present>` 和`<logic:notPresent>` 标签来显示article ID 或者`not assigned` 的图示说明。

15.14.4. Validation

Struts Validator 旨在使你可以基于同一套规则执行客户端校验和服务器端校验。校验器和校验规则都是在XML 文件中定义的。标准ActionForm的 `validate` 方法也可以连接到Validator框架中。大多数应用，比如Artimus，可以只使用Struts Validator来进行初步校验。你还可以通过你自己的插件扩展标准校验器，或者调用Struts Validator ，然后应用你自己的硬编码校验集。关于Validator的更多信息，参见第12章。

服务器端校验

服务器端校验器实际上是通过XML对需要被检查的每一个字段进行配置的Java类。清单 15.19 展示了我们的validation.xml 文件。

清单 15.19 /WEB-INF/validation.xml

```
<form-validation>
  <global>
    <validator name="required"
        classname="com.wintecinc.struts.validation.StrutsValidator"
        method="validateRequired"
        msg="errors.required">
      <javascript><![CDATA[
        function validateRequired(form) {
          var bValid = true;
```

```
        var focusField = null;
        var i = 0;
        var fields = new Array();
        oRequired = new required();
        for (x in oRequired) {
        if ((form[oRequired[x][0]].type == 'text' ||
           form[oRequired[x][0]].type    ==    'textarea'    ||
           form[oRequired[x][0]].type ==
           'select' || form[oRequired[x][0]].type == 'radio' ||
           form[oRequired[x][0]].type    ==    'password')    &&
           form[oRequired[x][0]].value
           == '') {
          if (i == 0)
            focusField = form[oRequired[x][0]];
          fields[i++] = oRequired[x][1];
          bValid = false;
        }
        }
      if (fields.length > 0) {
        focusField.focus();
        alert(fields.join('\n'));
      }
      return bValid;
      }]]>
    </javascript>
  </validator>
</global>
<formset>
  <form name="articleForm">
    <field
       property="title"
       depends="required">
       <arg0 key="article.title.displayname"/>
    </field>
    <field
       property="creator"
```

```
        depends="required">
        <arg0 key="article.creator.displayname"/>
    </field>
    <field
        property="contentDisplayHtml"
        depends="required">
        <arg0 key="article.content.displayname"/>
    </field>
  </form>
 </formset>
</form-validation>
```

对article form 的校验只是简单的检查看是否每一个字段都被完成。Validator 集成到消息资源，所以我们可以使用相同的消息关键字来标识字段。任何失败的的服务器端校验产生的结果都将显示在我们的message tile中。

客户端校验

Struts Validator 也支持JavaScript 校验。每一个校验器都有其自己的脚本，框架会自动将其整合插入到一个单独的，无缝的脚本之中。脚本是通过ActionForm bean的名称标识的。要调用脚本，需要：

- 在form标签中添加一个 `onsubmit` JavaScript 事件处理句柄
- 在引用表单名称的页面中某处插入一个`<validator:javascript>` 标签。

参见清单 15.18 来看edit JSP 中是如何使用者两个元素的。注意脚本的名称和form的名称是一样的，但是具有一个*validate* 前缀。

15.15. /do/Menu

Artimus 中最后一组action 是菜单action。到菜单action的链接是在默认的navbar tile中提供的。 如果没有单独指定一个，菜单将是默认的失败页面。除了导致菜单页面的默认action 之外，Artimus 中还有多个其它的菜单action。菜单页面使用这些action来访问我们在第15.9节中看到的各种查询操作。清单 15.20 展示了Artimus 菜单action。

清单 15.20 /WEB-INF/struts-confg.xml (the menu actions)

```
<!-- [15.15.1] logon -->
<action
   path="/admin/Menu"
  forward="/do/Menu"/>
```

```
  <!-- [15.15.2] menu -->
<action
  path="/Menu"
  name="menuForm"
  type="org.apache.struts.scaffold.ExistsAttributeAction"
  parameter="Application;HOURS">
 <forward
    name="success"
    path=".article.Menu"/>
  <forward
    name="failure"
    path="/do/MenuCreate"/>
</action>
<!-- [15.15.2] menu init -->
<action
  path="/MenuCreate"
  name="menuForm"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.article.MenuCreate">
  <forward
    name="success"
    path="/do/Menu"/>
  <forward
    name="failure"
    path=".article.Menu"/>
</action>
<!-- [15.16.2] find -->
<action
  path="/menu/Find"
  type="org.apache.struts.scaffold.ParameterAction"
  name="menuForm"
  validate="false"
  parameter="keyValue">
  <forward
    name="title"
```

```
    path="/do/find/Property?keyName=title&amp;keyValue="/>
  <forward
    name="author"
    path="/do/find/Property?keyName=creator&amp;keyValue="/>
  <forward
    name="content"
    path="/do/find/Property?keyName=content&amp;keyValue="/>
  <forward
    name="article"
    path="/do/article/View?article="/>
</action>
<!-- [15.16.4] contributor -->
<action
  path="/menu/Contributor"
  type="org.apache.struts.scaffold.RelayAction"
  name="menuForm"
  validate="false">
  <forward
    name="input"
    path="/do/admin/article/Input"/>
</action>
<!-- [15.16.5] manager -->
<action
  path="/menu/Manager"
  type="org.apache.struts.scaffold.RelayAction"
  name="menuForm"
  validate="false">
  <forward
    name="reload"
    path="/do/admin/Reload"/>
  <forward
    name="createResources"
    path="/do/admin/CreateResources"/>
</action>
```

前三个所列的用来显示菜单页面。其它的则为菜单自身所用。我们将在第15.16节讨论后者，即在我们展示菜单页面JSP代码的时候。

15.15.1. logon

清单 15.20中的/admin/Menu action 对于使容器发出一个logon挑战使用了一点技巧。通过使用/admin/之下的一个URI ，容器将确保在显示菜单页面之前（再次）用户是已经登录的。然后这会使得 菜单页面基于用户角色显示选项，就如我们在view 页面中所为 (15.13)。

15.15.2. menu

我们的菜单提供了多个选择列表。我们没有将它们硬编码到JSP中，或者在每次请求时生成它们， Artimus 是将它们保存为应用属性（attribute）。只要属性存在，它们就会工作的不错，但是如果不，就会显得凌乱。

为了确保属性在页面被显示之前被正确创建，默认的菜单命令/Menu, 示于清单 15.20，使用了一个标准的Scaffold Action 来测试一个属性是否存在。ExistsAttributeAction 期望要检查的范围 (应用, session, request) 应该被给定为Parameter属性的第一个记号。要检查的属性（attribute）则是parameter属性的第2个记号。在这里，我们告诉ExistsAttributeAction 去检查application范围时候有一个名为HOURS 的attribute。如果有， Action 返回success。否则，ExistsAttributeAction返回failure。

如果 HOURS attribute没有在application范围之中， action 将会失败，并且控制将转发到/menu/Create action。这个action 使用了(令人惊奇地!) ProcessAction 来执行一个用来创建我们的菜单页面所需的选项列表的业务 bean。清单 15.21 展示了MenuCreate bean的源代码。

清单 15.21 org.apache.artimus.article.MenuCreate

```
package org.apache.artimus.article;

import java.util.List;
import java.util.ArrayList;
import org.apache.artimus.lang.Tokens;
import org.apache.commons.scaffold.util.LabelValueBean;
import org.apache.commons.scaffold.util.ProcessBeanBase;
import org.apache.commons.scaffold.util.ProcessResult;
import org.apache.commons.scaffold.util.ProcessResultBase;
public class MenuCreate extends ProcessBeanBase {
public Object execute(Object parameters) throws Exception {
   <!-- [15.15.3 Our controls] -->
  ArrayList controls = new ArrayList();
  ArrayList hours = new ArrayList();
  hours.add(new LabelValueBean("Day", "24"));
  hours.add(new LabelValueBean("Week", "168"));
```

```
  hours.add(new LabelValueBean("Month", "720"));
  saveResult(Tokens.MENU_HOURS,hours,controls);
  ArrayList find = new ArrayList();
  find.add(new LabelValueBean("-- select --", "done"));
  find.add(new LabelValueBean("Title", "title"));
  find.add(new LabelValueBean("Author", "author"));
  find.add(new LabelValueBean("Content", "content"));
  find.add(new LabelValueBean("ID", "article"));
  saveResult(Tokens.MENU_FIND,find,controls);
  ArrayList contributor = new ArrayList();
  contributor.add(new LabelValueBean("Add Article", "input"));
  saveResult(Tokens.MENU_CONTRIBUTOR,contributor,controls);
  ArrayList manager = new ArrayList();
  manager.add(new LabelValueBean("Test channel", "channel""));
  manager.add(new LabelValueBean("Delete marked", "remove""));
  manager.add(new LabelValueBean("Reload config", "reload"));
  manager.add(new LabelValueBean("Create resources", "create"));
  saveResult(Tokens.MENU_MANAGER,manager,controls);
  <!-- [15.15.5 Our results] -->
  ProcessResult results = new ProcessResultBase(controls);
  results.setAggregate(true); // I'm a list of other results
  return (Object) results;
}
<!-- [15.15.4] saveResult -->
private void saveResult(String name, List items, List controls) {
  ProcessResult result = new ProcessResultBase(items);
  result.setName(name);
  result.setScope(org.apache.commons.scaffold.lang.Tokens.APPLICATION);
  controls.add(result);
}
} // end MenuCreate
```

15.15.3. 我们的控件

很明显，我们的menu 页面包含了四个具有选项列表的控件。它们每一个都可以保存为application范围内的一个单独的attribute。这是的它们能够易于独立适用，如果必要的话。

清单 15.21中的 `MenuCreate` bean 首先在其实例化时创建一个ArrayList 来持有选项集

合。每个选项被表达为一个LabelValueBean (一个JavaBean，具有一个`label` 属性和一个`value` 属性)。

15.15.4. saveResult

因为我们位于业务层，attribute实际上不能直接保存到应用上下文之中 … 至少现在不能。ProcessAction 可以在结果从业务层返回到web 层的时候做这个事情，但是我们必须告诉它该怎么做。结果对象用来在它为我们保存attribute 的时候保存我们想要action 使用的设置。

我们想要保存的设置被封装在一个工具`saveResult` 方法之内。这样就为列表创建了一个新的ProcessResult，并给了它一个attribute 名称，并且将范围设置为application。ProcessAction 然后就可以使用这些设置来在正确的地方以正确的名称保存attribute了。

15.15.5. Our results

一旦所有的选项列表都被创建并且保存在我们的控制列表中，我们就可以向ProcessAction返回结果了。为了这样，我们将我们的列表包装到另一个ProcessResult中，但是设置这个ProcessResult为一个聚合体。这样就告诉ProcessAction 将每一个条目处理为一个单独的ProcessResult。我们的每个选项列表通常保存为applcaiton范围中的一个单独attribute，就好像没一个都有一个单独的业务操作一样。

事实上，我们可以随时将`MenuCreate` 重构到一个单独的业务 bean中。`MenuCreate` bean可以很轻易的调用一个`HoursCreate` bean, 一个 `FindCreate` bean, 诸如此类，然后返回一个这些结果的聚合。应用的其余部分不受影响。它只是调用`MenuCreate` 而不会关心它是如何被创建的。

这是Model 2/MVC分层架构的本质。只要层间的API 保持不变，我们就应该可以修改层的实现而不会影响到相邻的层。从另一面说，我们应该能够修改两层之间的API 而不会影响到他们上面或者下面的其它层。虽然在实践中这是非常难以满足的设计需求，但是随着应用增长其投资回报却是巨大的。

将我们的对象保存在应用范围内之后， ProcessAction 返回其 `success` forward。对于清单 15.20中的/menu/Create mapping ，将会把我们路由回到默认的/Menu action。但是此时，ExistsAttributeAction 成功执行，控制被转发到菜单页面。

15.16. menu.jsp

菜单页面将渲染我们在/menu/Create action (第15.15.2节)所产生的选项列表。列表被用于组装用于列出最近提交文章，以及用于按标题、作者、或者关键字查找文章的空间。并且还有针对那些属于contributor 或者manager 角色的特别菜单。

清单 15.22 展示了menu.jsp tile的源代码。

清单 15.22 /pages/article/content/menu.jsp

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<%@ taglib uri="/tags/request" prefix="req" %>
```

```
<!-- [15.16.1] /find/Hours -->
<html:form action="/find/Hours">
  <TR>
    <TD class="label">List articles posted in the last:</TD>
    <TD class="input">
      <logic:iterate id="row" name="HOURS"
          type="org.apache.commons.scaffold.util.LabelValueBean">
        <html:radio property="hours" value="<%=row.getValue()%>"/>
        <bean:write name="row" property="label"/>
      </logic:iterate>
    </TD>
    <TD class="input">
      <html:submit property="submit" value="GO"/>
    </TD>
  </TR>
</html:form>
<!-- [15.16.2] /menu/Find -->
<html:form action="/menu/Find">
  <TR>
    <TD class="label" nowrap>Find articles by: </TD>
    <TD class="input">
      <html:select property="dispatch">
        <html:options collection="FIND"
                      labelProperty="label" property="value" />
      </html:select>
      <html:text property="keyValue" size="10" maxlength="30"/>
    </TD>
    <TD><html:submit>GO</html:submit></TD>
  </TR>
</html:form>
<!-- [15.16.3] /find/Last -->
<html:form action="/find/Last">
  <TR>
    <TD> </TD><html:hidden property="articles" value="20"/>
    <TD class="input">
```

```
      <html:submit property="submit"> LATEST NEWS </html:submit>
    </TD>
    <TD> </TD>
  </TR>
</html:form>
<!-- [15.16.4] /menu/Contributor -->
<req:isUserInRole role="contributor">
  <html:form action="/menu/Contributor">
    <TR >
      <TD class="label" nowrap>Contributor options: </TD>
      <TD class="input">
        <html:select property="dispatch">
          <html:options collection="CONTRIBUTOR"
                    property="value"
                    labelProperty="label"/>
        </html:select>
      </TD>
      <TD><html:submit>GO</html:submit></TD>
    </TR>
  </html:form>
</req:isUserInRole>
<!-- [15.16.5] /menu/Manager-->
<req:isUserInRole role="manager">
  <html:form action="/menu/Manager">
    <TR>
      <TD class="label" nowrap>Manager options: </TD>
      <TD class="input">
        <html:select property="dispatch">
          <html:options collection="MANAGER"
               property="value" labelProperty="label"/>
        </html:select>
      </TD>
      <TD><html:submit>GO</html:submit></TD>
    </TR>
  </html:form>
```

```
</req:isUserInRole>
```

15.16.1. /find/Hours

Struts 1.0 中的<html:radio> 标签缺少一些其他Struts标签所有的优美。清单 15.22 中的/find/Hours 表单不得不做一些临时措施来使它可以工作于我们在menu/Create action (第15.15.2节)所创建的label和值的列表。

因为我们需要产生多个单选按钮，我们首先将<html:radio> 标签嵌套到<logic:iterate>中。在每次迭代中，我们都要从我们保存在attribute HOURS 之下的列表中暴露下一个LabelValue bean。在循环制之中，我们按通常从一个<html:radio> 开始，但是转变到一个scriptlet 来提供值。运行时表达式 (<%=**row.getValue()**%>)从当前bean中获得value 属性，然后<bean:write> 输出label。

在渲染时，我们的radio 标签被解析为

```
<input type="radio" name="hours" value="24"> Day
<input type="radio" name="hours" value="168"> Week
<input type="radio" name="hours" value="720"> Month
```

当这些选项中其中一个被提交，我们最终便得到类似于下面的查询字符串

```
/artimus/find/Hours?hours=24
```

15.16.2. /menu/Find

清单 15.22 中的/menu/Find 表单输出了一个后面紧跟一个文本输入字段的选项列表。这个列表是来自于/menu/Create action (第15.15.2节)产生的 Find 列表，并且包括表15.4种所示的label/value 条目。

表单让用户选择查询风格(Title, Author, Content, ID) 然后在文本字段中输入一个限定词。清单 15.22 展示了 /menu/Find action的Mapping设置。它将表单提交到Scaffold ParameterAction，并且同时组装一个Scaffold MenuForm bean。MenuForm 是一个维护菜单页面的方便类，就像这个。它捕获我们在/menu/Find 表单中所用的dispatch 和keyValue 属性。这样就省去了我们在应用的ActionForm中定义工具书性的工作。

ParameterAction 用于通过添加参数完成查询字符串。添加何种参数是在ActionMapping 属性中指定为parameter属性。查询字符串则从本地转发中选择一个。

在清单 15.20中， /menu/Find action 的mapping包括如下的本地转发：

```
<forward
  name="article"
  path="/do/article/View?article="/>
```

当从菜单中选择ID label 时，浏览器将提交这样的查询字符串

```
/artimus/menu/Find?dispatch=article&keyValue=101
```

PaarmeterAction 查找dispatch 参数并且用它来作为具有我们的基本查询字符串的forward的名称。基本查询字符串来自于forward的 path，并且添加了运行时参数。在我们的例子中，查询字符串的结果相当于

```
/artimus/do/article/View?article=101
```

我们也可以通过使用一个独立的表单，/article/View和一个名为article的文本输入字段来完成这个任务。但是那样，我们还得对Title, Author, 和内容查询再做同样的事情。MenuForm和ParameterAction 使我们可以添加任意多的查询类型到我们的列表中，然后使她们在输入列表框内共享，即便它们还需要具有不同名称的参数。

标注	值
Title	title
Author	author
Content	content
ID	article
-- select --	done

15.16.3. /find/Last

清单 15.22中的/find/Last 表单复制了默认的 /find/Recent action的功能。它使用一个隐藏字段来设置文章数目为20。

我们这里还是用了一个到 welcome forward 的超链接，或者甚至直接连接到/do/find/Recent action。没有经过修改，有一件事情我们不能做，那就是将/find/Recent 作为<html:form>的目标。

为什么不能？如果你检查清单 15.6中的/find/Recent mapping，你会发现我们忽略了指定一个form bean。如果你想要在<html:form>使用一个没有定义的form bean，它会抱怨说找不到该bean。因为<html:form> 标签的主要目的是从一个ActionForm中组装元素，没有制定一个Action 会产生一个错误。

当然，我们可以只为/find/Recent指定一个form bean。基于此目的Scaffold中甚至有一个NullForm 类(org.apache.struts.scaffold.NullForm) 。其它工作将是将HTML form 调整成这样：

```
<FORM action="<html:rewrite forward='welcome'/>" method="POST">
<TR>
  <TD> </TD>
  <TD class="input">
```

```
    <INPUT type="submit" value=" LATEST NEWS " name="SUBMIT">
  </TD>
  <TD> </TD>
</TR>
</FORM>
```

但是这似乎简单得就像使用隐藏字段一样。

15.16.4. /menu/Contributor

某些特征可能只针对contributors有效。在清单 15.22中， `<req:isUserInRole>` 标签被用于略去或者包括一个contributor 命令列表。

我们也在View页面中使用了这个标签 (第15.13节)来略去或者包括Edit 和Delete 按钮。

现在，仅针对contributor的命令列表中的条目就是添加文章。

虽然也可以使用一个按钮，我们前进一步，设置一个选项列表，这样使得今后添加其它命令更容易些。这也使得和下一节所讲的/menu/Manager 菜单保持一致。

因为这个命令不需要什么限定词，我们就不需要再使用ParameterAction。相反我们可以使用一个RelayAction。和ParameterAction一样，RelayAction (`org.apache.struts.scaffold.RelayAction`) 查找一个 `dispatch` 参数 并且用它来选择一个ActionForward。

我们的，被/menu/Create action 保存的contributor 命令列表渲染为一个简短的，一个条目的列表：

```
<select name="dispatch">
<option value="input">Add Article</option></select>
```

当Add Article选项被选择，并且表单被提交时，浏览器将产生这样的请求

```
/artimus/do/menu/Contributor?dispatch=input
```

控制器将这个请求传递到RelayAction。 `dispatch` 参数告诉RelayAction 查找名为`input`的forward (以mapping的本地forward开始)，就像这个

```
<forward
name="input"
path="/do/admin/article/Input"/>
```

/admin/ 部分确保请求被接受之前，用户是已经登录的。如果请求被接受，将被转发到文章输入action。如前面的清单 15.16所示，Input action 只是简单地将控制转发到edit页面。

因为我们没有获得一篇文章，我们的ActionForm 还是空的，并且一个空表单会展示给contributor。如果表单被提交，文章保存action 看到article ID 是空的，然后将文章插入到数

据库之中。

15.16.5. /menu/Manager

清单 15.22 中的/menu/Manager 表单使用了与contributor 菜单 (第15.16.4节)相同的方法，但是使用了不同的命令列表。和contributor 菜单类似，列表限于是属于manager 角色的用户。我们的默认Artimus 用户既是一个contributor又是一个manager。所以，如果你使用Artimus 用户的凭证登录，则可以看到两个菜单。

15.17. 小结

哇! 我们经历了长长的Artimus之旅。到此，我们已经讨论了所有的基本组件。还有多个业务bean 类我们没有讨论，但是它们都是我们已经讨论了的两个类的简单变体。当然，所有类的完整的源代码都可以从本书的网站取得[Husted]。

在下一章，我们将会重构Artimus 到Struts 1.1，以便它可以采用新版本的Struts所提供的最新特征和优点。

16. 回家:迁移到 Struts 1.1

本章内容

- 升级Struts 1.0应用
- 使用动态ActionForm
- 实现基于Action的安全
- 配置Tiles 和Validator 1.1

You only ever write one software application, then you spend the rest of your life going back and rewriting it.

—Craig McClanahan's computer science professor

16.1. 下一站， Struts 1.1

从2000年6月发布以来， Struts 1.0 很快就成为一个非常成功的产品。有大量的实际应用都基于Struts。随着Struts 1.1的诞生，很多团队都将想要把他们的应用进行升级以利用新的特征。在这一章，我们我们将指明把Artimus 1.0 应用翻新到Struts 1.1的路线。

首先，重要的是要注意到大部分Struts 1.0 应用都能够立即在Struts 1.1 之下运行。所有实际需要做的工作就是：

- 更换Struts 1.1 和Commons JAR文件。
- 重新构建全部内容，包括JSP。

如果你使用了一些已经移到Commons的Struts 类，在重新构建应用之前你必须修改一些 `import` 语句。但这些应该足以能够设置和运行的应用了。

有好些Struts 已经移到框架之外了。在 Struts 1.0 和Struts 1.1之间，它们被重新打包，并且移到了Jakarta Commons [ASF, Commons]项目中。你可能需要导入的新类如下：

- **Commons BeanUtils** 包 (`org.apache.commons.beanutils`) 代替了 `org.apache.struts.utils.BeanUtils`, `org.apache.struts.utils.ConvertUtils`, 和 `org.apache.struts.utils.PropertyUtils`.
- **Commons Collections**包(`org.apache.commons.collections`)代替了 `org.apache.struts.util.ArrayStack`, `org.apache.struts.util.FastArrayList`, `org.apache.struts.util.FastHashMap`, 和 `org.apache.struts.util.FastTreeMap`.
- **Commons Digester** 包(`org.apache.commons.digester`) 代替了 `org.apache.struts.digester.*`

如果你还是用了 1.0 版本的 Tiles 和 Struts Validator，你还必须：

- 修改任何引用Validator 的com.wintec.*的`import` 语句。
- 修改任何引用到Tiles 的组件版本的`import` 语句(尽管可能很少见)。
- 从web.xml 删除对Tiles 或者Validator的引用。
- 在struts-config.xml 中添加新的引用以装载Tiles 和Validator集成版本。

我们将一步步通过我们的Artimus应用经历这个过程以便你能够看到如何进行。但是在开始之前，我们先来看看Struts 1.1 所提供的东西。

NOTE	Commons BeanUtils 提供了比原来的 Struts 1.0 工具更好的转换能力。他还修改了一个基本假设。当从 `Strings` 转换到数值类型时，Struts 1.0 BeanUtil 会将 null 转换为 null 的数值对象类型(比如 `Integer`)，但是将 null 转换为原生类型的 0 (比如 `int`)。而 Commons 版本则将 null 数值都转换为 0 。如果你的代码希望将 null `Strings` 转换为 null 数值对象，你

	讲必须调整你的代码。ActionServlet 中有一个开关(convertNull) 可以告诉 Struts 保留 1.0 行为。但是如果你在代码中使用了 BeanUtils，你还必须调整你自己的类。

16.1.1. Struts 1.1 特征摘要

Struts 1.1 提供的新特征涵盖了框架的方方面面，从对模块化应用的高级支持到许多标签的低层调整。表16.1 总结了Struts 1.1 提供的新特征，并且指出你应该从何处找到新信息。

表格 16.1 Struts 1.1 (beta 1)的新特征

特征	类或包以及说明
多消息资源	(org.apache.struts.config.MessageResourcesConfig) 如今你可以从Struts 配置中载入多个消息资源束，使得组织应用的资源更加容易。
模块化应用	(org.apache.struts.config.*) 这是一个对将应用组织为模块（module），或者子应用，的新的支持。每个模块都有其自己的配置文件集并且可以当作一个单独的应用进行开发。大多数Struts 1.0 应用仅需要对配置文件作一些小修改就可以用作应用模块。关于应用模块，参见第4章。
动态ActionForm	(org.apache.struts.action.DynaActionForm) ActionForm 的一个特殊子类，允许你从Struts配置文件中定义你所需的JavaBean 属性。因为是完全动态的，自由配置的ActionForm，你还可以使用基于Map的ActionForm。关于ActionForm 和动态ActionForm，参见第5章。
Action的Roles 属性	(org.apache.struts.action.ActionConfig.roles) 如今你可以注册一个JAAS 安全角色列表到一个ActionMapping中。如果用户不是列表中的角色，访问会被拒绝。其运行类似于标准的基于容器的安全，但是允许你对特定的ActionMapping 进行授权而不是针对一个URL 模式。它可以被用于基于容器的安全，也可以代替它。通过提供你自己的RequestProcessor，你将可以使用这个属性采用其他安全方案。
分离错误和消息队列	(org.apache.struts.action.ActionMessage) (org.apache.struts.action.ActionMessages) (org.apache.struts.taglib.logic.MessagesPresentTag) (org.apache.struts.taglib.logic.MessagesNotPresentTag) (org.apache.struts.taglib.html.MessagesTag) 有一个新的通用消息类可用来将其他消息从错误消息中分离出来。消息队列是通过Action 类的saveMessages 方法（第8章）和html标签库的消息标签（第10章）暴露的。
LookupDispatchAction	(org.apache.struts.actions.LookupDispatchAction) 一个新版本的标准Struts LookupAction使得将多个操作整合到一个Action 中，并且从表现页面中的本地化控件选择操作更加容易。
SwitchAction	(org.apache.struts.actions.SwitchAction) 一个新的Action，使得从一个应用模块转发请求到另一个应用模块更加容易。
声明性异常处理	(org.apache.struts.action.ExceptionHandler) 可以不在你的Action之中捕获异常，你可以将它们回传给控制器，然后通过Struts ExceptionHandler来捕捉。你可以根据需要来可多可少地注册异

	常处理器。默认的异常处理器将异常消息包装在一个ActionErrors实例中，并且将控制转发到一个指定的URI。你也可以创建你自己的ExceptionHandler 子类来提供其它行为。
RequestProcessor 组件	(org.apache.struts.action.RequestProcessor) 你可以通过提供一个RequestProcessor 子类来很容易的定制一个模块如何处理每一个请求。RequestProcessor 对象提供了多个扩展点，以使得可以容易地仅仅针对需要修改的地方进行修改。
PlugIn Actions 管理资源	(org.apache.struts.PlugIn) 为了载入你自己的资源，你可以注册一个PlugIn Action 到控制器中。这给你一个机会来在启动时创建一个资源，并且在停止时销毁它，而不需要创建你自己的特殊servlet 或者ActionServet 子类
Commons Logging接口	(org.apache.commons.logging) Struts ActionServlet 和相关的组件如今都实现了Commons Logging接口，可以更易将Struts 与高级logging包，比如Log4j 和Java 1.4 logging包进行集成。
LabelValueBean	(org.apache.struts.utils.LabelValueBean) 许多JSP 标签需要显示具有标注和值的控件。这个方便对象就可以很好地和期望一个具有label 和value方法的bean的标签一起工作。关于Struts 标签库的更多信息，参见第10章。
Nested taglibs	(org.apache.struts.taglib.nested.*) nested 标签扩展了基本 Struts 标签，以允许它们能够在嵌套的状态下彼此相关。关于Struts 标签库的更多信息，参见第10章。
New empty/ notEmpty 标签	(org.apache.struts.taglib.logic.EmptyTag) 新的逻辑标签，使得更易测试一个字段是否为null或者包含一个空字符串。
新的frame 标签	(org.apache.struts.taglib.html.Frame) 一个新的HTML 帧（frame）标签，使得更易创建帧之间的超链接，并具有Struts HTML 链接标签的全部能力。
新的optionsCollection 标签	(org.apche.struts.taglib.html.OptionsCollection) 新的HTML optionsCollection 标签使得更易保存一个用来在ActionForm 之上和其他属性一起组装一个控件的集合。
Radio标签的新的idName 属性	(org.apache.struts.taglib.html.RadioTag) 为了更易编写一系列动态的单选按钮，你可以通过radio标签的value和新的 idName属性指示一个bean。
新的indexed 标签属性	(button, checkbox, file, hidden, image, link, password, radio, select, submit, text, textarea) 许多页面都使用一个迭代器创建一系列的控件。多个HTML 标签如今都支持indexed 属性，它给每一个迭代的控件一个唯一的名称。
New Java- ScriptValidator tags	(org.apache.struts.taglib.html.JavaScriptValidatorTag) Struts Validator 使得你能够针对表单中的一个字段创建一个单独的JavaScript validators，并且用一个标签将它们全部插入。
Tiles framework	(org.apache.struts.Tiles) (org.apache.struts.taglib.Tiles) Tiles 框架如今集成到了核心Struts 分发包中。这个JSP的高级文档装配包使得通过整合可重用的页面片断创建页面更加容易。
Struts Validator	(org.apache.struts.Validator) Struts Validator 使得更易从相同的配置产生客户端和服务器端校验。

当然，大多数应用都没有用到每一个Struts 1.0特征，也将不会用到Struts 1.1的每一个新特征。但是列表中一定有某些东西能够给大多数应用带来益处。

16.1.2. 我们可使用的特征

开始迁移我们的Artimus 时，我们首先转移到集成版本的Tiles 和Validator 并且实现两个新的特征：DynaActionForms 和基于action的安全。

在Artimus 1.0中，我们必须创建一个ActionForm 子类来处理我们的书如属性的集合。在Struts 1.1中，我们可以使用*DynaActionForm* (`org.apache.struts.action.DynaActionForm`) ，从而避免了创建一个新的Java类。

你仍然需要声明DynaActionForm的属性，但是因为这些声明可以位于一个XML中，它们非常容易维护。

新的基于Action的安全（*action-based security*） 特征意味着我们可以对每一个单独的ActionMapping应用声明性安全而不是依赖于注册到容器中的URL 模式。

Struts 1.1 也绑定了Tiles 和Validator 作为标准分发包的一部分。它们的工作方式和Artimus 1.0中的一样但是初始化方式不同。我们还需要修改Artimus 1.0的 message tile 中的标签名来使用Validator 1.0 标签的内建版本。

我们在第15章研究了Artimus 的特征集和实现。在这一章，我们仅集中于Artimus 1.0 中将要迁移到Struts 1.1的部分。我们将不再详细的讨论Artimus 应用的设计和功能。要发挥本章的最大效果，你需要首先阅读第15章。

DynaActionForms 和action-based 安全都是Struts 1.1的新特征。

16.2. 基线化变更

我们的升级活动中最好的部分将发生在Struts 配置文件(struts-config.xml)中。两个基线变更涉及到Tiles 和Validator。在Struts 1.1中，这些组件集成到了主分发包中了，并且需要不同方式的装载。我们还需要删除一个对废除的标准Action的action mapping。我们将首先进行这3个修改，然后再进行我们的自由清单上的第二关的特征。

但是最最首要的是设置新的Struts 1.1 JAR。对Tiles 和Struts Validator可能稍微有些不同 (因为它们已经移到了核心代码基之中)，但是其他所有东西都是即插即用的。

Struts 1.1 分发包中有一些包含我们需要的大多数新文件的便捷库。为了开始升级，你只需要下在库文件，然后将它们放置到你的应用的WEB-INF/lib 文件夹中。

如果你想要跟着我们升级Artimus的过程，首先需要从本书网站下载Artimus 1.0 分发包[Husted]。 还需要从站点下载Scaffold 1.1 库。然后，将Struts1.1中的所有文件拷贝到Artimus WEB-INF/lib 文件夹下，以及两个Scaffold JAR。

你可以用1.1版本的文件覆盖所有同名文件。如果你的应用，比如Artimus，采用了使用`org.apache.struts`包的 `Tiles`版本，你还应该删除tiles.jar 和tiles.tld 。否则，将会出现与Struts JAR 中的1.1版本的类装载器的冲突。

NOTE	Java JAR 文件使用了流行的 ZIP 归档格式。你可以使用任何 ZIP 文件浏览器打开一个 JAR 来看它的具体内容。文件的路径表示它们被打包的方式。

如果你的应用导入了一些已经移到Commons中的Struts 工具包 (见第16.1节)，你必须修改相应的`import`语句以指向新的 版本：

```
// import org.apache.struts.util.BeanUtils; // Struts 1.0.x
Import org.apache.commons.beanutils.BeanUtils; // Struts 1.1
```

Artimus 没有直接导入任何Struts 1.0 工具类，所以我们不需要对此修改任何Java 源代码。

这时，你应该能够运行Artimus 的clean build，不会产生任何编译错误。如果Ant 构建工具[ASF, Ant] 在你的系统路径之中，你可以进入WEB-INF/src 目录，输入

```
> ant clean.build
```

来运行clean build。

然而，Artimus 自身还没有彻底完成升级。配置文件仍然设置来载入1.0 版本的Tiles 和Validator。我们来看看如何设置和运行新版本的Tiles和Valiadtor。

16.2.1. Struts 1.1 的 Tiles

Tiles 时你能够使用一个称为*Tiles Definition* 的描述符来从页面组件构造页面。和其它框架对象类似， Definition 对象可以通过一个XML 配置文件来创建。XML 元素声明一个layout页面并且向它传递组件（或tiles）的名称，以在运行时插入。第11张详细讨论了Tiles。

Tiles 时通过观察引用到一个它的Definitions 的ActionForwards 来发挥其威力的。当它看到一个匹配其某个Definitions 的路径时，Tiles 就会装配页面并且将页面传递给JSP 服务进行渲染。

虽然它是一个漂亮的Struts组件，但是从可选的Tiles 1.0升级至集成的Tiles 1.1包只是一个简单的操作。在 Struts 1.0中，Tiles 必须子类化ActionServlet 和数个方法。在Struts 1.1中，Tiles则是用新的RequestProcessor 对象(`org.apache.struts.action.RequestProcessor`)。ActionServlet 将会把请求委托给新的new RequestProcessor来处理。

通过使用RequestProcessor，我们可以使用ActionServlet 而不用创建一个正式的子类。当使用应用模块时(见第4章)，每个模块都可以有其自己的RequestProcessor。

除了RequestProcessor之外，Tiles 还需要从一个XML文当中载入其Definitions 。在Struts 1.0中， ActionServlet 子类会为我们处理这些事情。在Struts 1.1中，我们可以使用一个PlugIn Action 来做同样的事。PlugIn Action 有一个`init()`方法，就像一个servlet一样。我们是通过Struts配置文件注册PlugIns 的。ActionServlet 将对每个Plugin在启动时调用`init()` 方法，在注销时调用其`destroy()`方法。这是一个创建和释放应用资源的很好的方式，比如Tiles Definitions。

除了载入 Definitions之外，Tiles PlugIn Action (`org.apache.struts.tiles.TilesPlugInAction`) 也会设置Tiles RequestProcessor(`org.apache.struts.tiles.TilesRequestProcessor`)。通过这种方式，你根本不需要担心TilesRequestProcessor 和Tiles PlugIn的配置。

但是，如果你需要随后载入你自己的RequestProcessor，请确保它是TilesRequestProcessor的子类。 (关于RequestProcessors 和PlugIns的详细信息，参见第9章。)

清单16.1 列出了你可以添加到`Struts`配置文件中的`<plug-in>` 元素。struts-config.xml 中PlugIn 配置段出现在很靠后的位置，只在最后关闭的 `</struts-config>` 元素之前，而位于`<action-mappings>`元素和其它新配置块之后。

清单 16.1 struts-config.xml (Tiles plug-in 元素)

```
<!--- other elements -->
  </action-mappings>
  <plug-in
       className="org.apache.struts.tiles.TilesPlugin">
    <set-property
       property="definitions-config"
       value="/WEB-INF/tiles-defs.xml"/>
  </plug-in>
</struts-config>
```

NOTE	在 Struts 1.0 配置中，所有元素都必须是被包围的。比如有一个<action-mappings>元素针对 <action> 元素，以及一个<form-beans> 元素来包围<form-bean> 元素。在 Struts 1.1 中，很多元素，比如<plug-in>，就不是被包围的 ;并没有<plug-ins>元素存在。

关于配置Struts 应用的总体信息，包括使用`<set-property>` 元素，参见第4章。

Tiles 包如今已经是标准Struts 1.1 JAR的一部分。我们将需要调整部署描述符以载入新的 We will need to adjust the deployment descriptor to load the 标签库描述符 (TLD)，但是我们不需要修改任何页面或者Tiles 配置文件。

下面是部署描述符中的新的 Tiles TLD 引用：

```
<taglib>
  <taglib-uri>/tags/tiles</taglib-uri>
  <!--<taglib-location>/WEB-INF/lib/tiles.tld</taglib-location>
-->
  <taglib-location>/WEB-INF/lib/struts-tiles.tld</taglib-location>
</taglib>
```

为了向后兼容，我们利用了`<taglib-uri>` 的逻辑名称形式，而没有将引用从*tiles* 修改至

struts-tiles。因为这仍旧是一个小应用，并不是很难以修改tiles，但是因为我们现在并不是非要坐这个修改，我们将修改放到后面进行，而只是修改`taglib-location` 来匹配新的TLD名称。

我们还可以从web.xml 中删除对TilesComponent servlet 的引用，而转回去使用默认的ActionServlet：

```
<servlet>
  <servlet-name>action</servlet-name>
  <!-- org.apache.struts.tiles.ActionComponentServlet -->
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
<! -- … -->
```

我们现在可以安全的从WEB-INF/lib 中删除tiles.jar 和tiles.tld 文件了（如果还没有删除的话）。Tiles 类如今已经在主struts.jar文件中，并且一个strutstiles.tld 文件也被包含在Struts 1.1的库分发中：

在关闭struts-config.xml之前，还需要做一件事：在文件的顶部，将DTD 引用从

```
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 1.0//EN"
"http://jakarta.apache.org/struts/dtds/strutsconfig_1_0.dtd">
```

修改为

```
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 1.1//EN"
"http://jakarta.apache.org/struts/dtds/strutsconfig_1_1.dtd">
```

这告诉Digester 对Struts 1.1使用正确的DTD进行验证。在部署描述符中(WEB-INF/src/conf/web.xml)，我们现在可以将对*ActionComponentServlet* 的`<servlet-class>` 引用替换为对标准ActionServlet（`org.apache.struts.action.ActionServlet`）的引用。

这就是升级到Tiles1.1所需做的全部工作。但是在运行应用之前，我们还需对Validator做类似的工作。

NOTE	我们在第 15.3.1 节讨论过，Artimus 的所有源代码都在 WEB-INF/src 之下。包括所有的 Java, XML, JSP, 和属性文件。Ant 构建文件在应用被编译时部署源文件。

16.2.2. Struts 1.1 的 Validator

将Validator 从1.0 转移到1.1 也很直接，虽然包的变更迫使我们必须在最后调整一些页面。

Struts Validator 将检查ActionForm 的属性，以确保至少这些值*看起来*是有效的。这样可以有助于防止不正确的输入被输入被传递到业务层，就像前台接线员可以阻止一些讨厌的电话不至于打扰繁忙的经理。

Validator 分为两个截然不同的部分。首先，有一个关于应用可用的不同校验器的注册表。一个校验器可能是检查某个字段是否有输入。而另一个则可能检查输入是否有期望的长度。Validator 本身附带多个基本校验器，并且你也可以插入你自己的校验器。Validator的第二个部分是实际的校验。校验部分描述你的表单和字段以便你可以将特定的校验器连接到特定的字段。第12章详细讨论了Struts Validator。

和如今的大多数组件一样， Validator 也是通过XML配置的。

1.0 版本的Validator 使用一个servlet 来初始化其资源。在Struts 1.1，则使用一个新的PlugIn Action 。ValidatorPlugInAction 替代了我们在Artimus 1.0中使用的ValidatorServlet。清单16.2 展示了我们可以添加到struts-config.xml 中来初始化Struts Validator的`<plug-in>` 元素。

清单 16.2 /WEB-INF/src/conf/struts-config.xml (Validator PlugIn 元素)

```
<plug-in
   className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
     property="pathnames"
     value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```

Struts 1.0 校验器使用一个文件的保存校验器配置和表单校验。而在Struts 1.1中，我们可以将其分为两个单独的配置文件：一个针对校验器的validation-rules.xml 文件和一个针对表单校验配置的validation.xml。其思想是*校验器（validators）* (validator-rules.xml) 可以在应用之间进行复制。但是表单校验*（form validations）* (validation.xml) 则会因各个应用而趋于不同。通过使用分离的文件，我们可以更容易地修改标准的校验器而不用干扰我们具体的校验。

对Artimus来说，我们可以从新的validator-rules.xml 文件从WEB-INF/lib 目录(它位于库分发包中) 移动到WEB-INF/src/conf 文件夹。

因为这个文件有我们所需的全部校验器，我们可以编辑旧的validation.xml 文件(也位于conf 文件夹) 并且删除位于顶部的`<global><validator>` ...`</validator></global>` 元素之间的配置段。位于底部的`<formset>` 元素予以保留，并有一个`<form>` 元素针对我们的`articleForm`。

1.0 vs 1.1	本书完成时，为 Struts Validator 正在加入一个 DTD。对于 Struts 1.1 的最终版本，你可能需要添加一个 DTD 到验证器的配置文件中。

在不少户描述符(WEB-INF/src/conf/web.xml) 中，我们可以完全删除*validator* `<servlet>` 元素，因为新的PlugIn 会载入Validator的资源。我们也可以删除struts-validator `<taglib>` 元素， 因为我们需要的标签如今都在主Struts JAR中了。为了避免任何混淆，你还应该从WEB-INF/lib中删除struts-validator.jar 和struts-validator.tld 文件。在Struts 1.1中，Validator 包位于主JAR中，而Validator 标签如今位于html 和logic 标签库中。

16.2.3. Struts 1.1 的 ReloadAction

模块应用的复杂性和其他一些因素导致删除了Struts 1.0 中提供的一些管理性Action。这种简化包括非常便捷的ReloadAction (`org.apache.struts.actions.ReloadAction`)。

不幸的是，我们必须从Artimus 1.1的 struts-config.xml 中删除对Reload 命令的映射，因为它已经不再存在。在本章末尾，我们还将对我们的菜单action作相应的修改。

16.2.4. 其他对 web.xml 和 struts-config.xml 的基线变更

没有。

这是对的。为了将Artimus 迁移到r Struts 1.1，我们必须：

- 删除废除的`<action>` 元素
- 插入两个新的配置元素到struts-config.xml中
- 从web.xml删除两个废除的元素
- 修改标签库引用

但是这就是对struts-config.xml 和web.xml 所需要做的。大多数应用甚至都不需要做这么多。

唯一的坏消息是Validator 标签的包在集成到Struts 1.1中的时候发生了变化。因为我们在Artimus 1.0中使用了可选的Validator 组件，我们需要对页面做一些小小的改动来反应这个变化。

16.2.5. message.jsp (1.1)

当Struts Validator 被集成到Struts 1.1时，其标签被集成到了现有的标签库中。所以我们不用引用`<validator:messagesExist>`， 如今我们可以引用`<logic:messagesExist>`。标签的底层实现是一样的；只有打包方式发生了变化。

清单16.3 展示了我们的修改后的message.jsp。因为这是一个和其它页面共享的tile，我们必须仅仅在一个文件中做这些修改，即便站点中的每一个页面都要显示消息。

清单 16.3 /WEB-INF/src/pages/article/common/message.jsp

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
```

```
<%-- [1.0] Tag is part of html in 1.1
<%@ taglib uri="/tags/struts-validator" prefix="validator" %>
--%>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<logic:messagesPresent>
  <TR>
    <TD class="error" colspan="3">
      <UL>
        <html:messages id="error">
          <LI><bean:write name="error"/></LI>
        </html:messages>
      </UL>
    </TD>
  </TR>
</logic:messagesPresent>
<logic:messagesPresent message="true">
  <TR>
    <TD class="message" colspan="3">
      <UL>
        <html:messages id="message" message="true">
          <LI><bean:write name="message"/></LI>
        </html:messages>
      </UL>
    </TD>
  </TR>
</logic:messagesPresent>
```

16.2.6. form.jsp (1.1)

我们还需要调整我们的form tile中的标签，如清单16.4所示。这里还是只是需要修改标签名称。实际的JSP 代码并未改变。

清单 16.4 /WEB –INF/src/pages/article/content/form.jsp

```
<%@ taglib uri="/tags/struts-html" prefix="html"%>
```

```
<%@ taglib uri="/tags/struts-bean" prefix="bean"%>
<%@ taglib uri="/tags/struts-logic" prefix="logic"%>
<%-- [1.0] Tag is part of html in 1.1
<%@ taglib uri="/tags/struts-validator" prefix="validator" %>
--%>
<html:form action="/admin/article/Store"
           onsubmit="return
            validateArticleForm(this);">
   <TR>
      <TD class="label" nowrap>Title:</TD>
      <TD class="input" colspan="2">
         <html:text property="title" size="50"
          maxlength="255" /></TD>
   </TR>
   <TR>
      <TD class="label" nowrap>Author:</TD>
      <TD class="input">
         <html:text property="creator" size="30"
          maxlength="75" /></TD>
      <TD class="hint">
         Full name of person who originated the article.</TD>
   </TR>
   <TR>
      <TD class="label" nowrap>Article:</TD>
      <TD class="input" colspan="2">
      <!-- The Struts html:textarea tag does not support wrapping -->
      <!-- so we use this trick instead -->
       <textarea
          name="contentDisplayHtml" rows="12"
         cols="50" tabindex="2"
          wrap="soft">
         <bean:write name="articleForm"
         property="contentDisplayHtml" />
       </textarea></TD>
   </TR>
   <html:hidden property="contributedDisplay" />
   <html:hidden property="contributor" />
   <TR>
     <TD class="label" nowrap>Article ID:</TD>
     <logic:notPresent name="articleForm" property="article">
      <TD class="dimmed">
        <i><html:hidden property="article" />
            not assigned
      </i></TD>
    </logic:notPresent>
    <logic:present name="articleForm" property="article">
      <TD class="input"><html:hidden property="article" /><bean:write
      name="articleForm" property="article" /></TD>
   </logic:present>
   <TD> </TD>
   </TR>
   <TR>
      <TD class="button" colspan="3">
         <html:submit accesskey="S">SAVE</html:submit>
        <html:cancel accesskey="C">CANCEL</html:cancel></TD>
   </TR>
</html:form>
```

```
<%--
<validator:javascript formName="articleForm"/>
--%>
<html:javascript formName="articleForm"/>
```

在第16.3.3节中，我们将需要对此页面在实现一个新的特征之后再作一些修改。但是现在，我们只是调整校验器的标签名。

此时，我们升级到Artimus 1.1的第一轮应该可以启动运行了。如果你运行了一次clean build然后重新启动容易，你应该点击整个应用而正常的使用它。唯一的事情是要避免我们从Struts配置中删除的Reload action。我们还没有修改菜单，所以它还是被列出来了，即使其映射已经不再有效。我们来关心下一轮升级。

16.2.7. MenuCreate (1.1)

Struts 1.1 删除了我们在管理员菜单中提供了的Reload 管理action 。在第16.2.3节中，我们删除了对 ReloadAction的映射。现在我们还需要通过从创建我们的菜单对象的类中删除一行。在我们的`MenuCreate` 业务对象bean (`org.apache.artimus.article.MenuCreate`)中，原先的代码示于清单15.21，我们将管理员菜单块从

```
ArrayList manager = new ArrayList();
manager.add(new LabelValueBean("Reload Config", "reload"));
manager.add(new LabelValueBean("Create Resources",
"createResources"));
saveResult(Tokens.MENU_MANAGER,manager,controls);
```

修改为

```
ArrayList manager = new ArrayList();
manager.add(new LabelValueBean("Create Resources",
"createResources"));
saveResult(Tokens.MENU_MANAGER,manager,controls);
```

菜单页面不需要修改，因为它只是输出菜单列表而不知道到底哪个菜单有效。

当然，另一个可选方案是创建一个reload 页面来解释这个命令在本版本中已经无效。但是这里，最简单的就是修改业务bean。

LabelValueBean

在Struts 1.1中，一个LabelValueBean 对象被加入到Struts 工具包中，就象Scaffold 工具包中的那个一样。我们可以将LabelValueBean 的依赖性从Scaffold 类转到Struts类，但是我们决

定保持Scaffold 版本。MenuCreate 类是一个业务bean，理想情况下，不应该从Struts导入类。所以我们没有在MenuCreate中引入对Struts JAR的依赖性，而是将依赖性保持到框架独立的Scaffold Commons 包。

16.2.8. 向前

在确认我们的应用可以在Struts 1.1下启动运行后，我们就可以转而进行任意的修改。

16.3. 任意修改

为了发挥Struts 1.1的最大威力，我们决定进行四个自己决定的修改。

最为这次迭代的一部分，我们还将：

- 配置一个DynaActionForm bean.
- 添加角色到受保护的映射以实现基于Action的安全
- 加入一个新的配置元素来载入默认的消息资源

16.3.1. 修改表单为 DynaActionForm

Artimus 1.0 声明了一个定制的ActionForm ，它在Struts 配置文件中定义为`articleForm`:

```
<!-- Article Form Bean -->
<form-bean
name="articleForm"
type="org.apache.artimus.struts.Form"/>
```

和大多数ActionForm一样，我们的`articleForm` 除了一些简单的属性之外没有什么。

它只是对表单输入进行缓冲，直到属性被校验然后传输给我们的业务bean。JavaBean 属性并不难以设置，但是一会儿工夫之后还是有些痛苦。

Struts 1.1 提供了一个新的方式来定义只有简单属性的ActionForm。DynaActionForm 类能够使你将属性定义为XML文件中的`<form-bean>` 元素的组成部分。对另一个组件来说，DynaActionForm 看起来就象常规的JavaBean。它可以被用在能够使用“常规” ActionForm 类的任何地方。DynaActionForm 也可以用于任何设计来使用JavaBean的开发工具，就像你通过手工编码器所有属性一样。

如果你使用了Struts Validator，就可以使用DynaValidatorForm 类以便你能够和Validator一起使用DynaForm。这样可以消除对离散的ActionForm 类的总体需求。清单16.5 展示了来自于清单15.15的`articleForm` ，编码为一个DynaActionForm后的代码。

清单 16.5 /WEB-INF/struts-config.xml (<form-bean>元素)

```
<form-bean
  name="articleForm"
  type="org.apache.struts.action.DynaValidatorForm">
  <form-property
    name="keyName"
    type="java.lang.String"/>
  <form-property
    name="keyValue"
    type="java.lang.String"/>
  <form-property
    name="marked"
    type="java.lang.String"
    initialValue="0"/>
  <form-property
    name="hours"
    type="java.lang.String"/>
  <form-property
    name="articles"
    type="java.lang.String"/>
  <form-property
    name="article"
    type="java.lang.String"/>
  <form-property
    name="contributor"
    type="java.lang.String"/>
  <form-property
    name="contributedDisplay"
    type="java.lang.String"/>
  <form-property
    name=" "
    type="java.lang.String"/>
  <form-property
    name="creator"
    type="java.lang.String"/>
```

```
  <form-property
    name="title"
    type="java.lang.String"/>
  <form-property
    name="contentDisplayHtml"
    type="java.lang.String"/>
</form-bean>
```

我们所需做的就是从struts-config.xml 删除旧的`articleForm <form-bean>` 元素，然后在适当位置加入以上的配置代码。因为在Struts 1.1中没有reload action，我们需要在测试我们的DynaBean之前重新装载我们的应用。

设置好动态的`articleForm` 之后，现在我们就可以从Artimus 1.1代码基中删除 `org.apache.artimus.struts.Form` 类，以及其后代类 `org.apache.artimus.article.struts.ArticleForm`了。我们的Form 类被正式废除。

NOTE	如果你在你自己的应用中使用 DynaForms，一定要意识到默认的 `reset` 方法会设置所有动态属性为它们的初始值(即，null)。对常规的 ActionForm 来说，默认的 `reset` 方法什么都不做。

16.3.2. 基于 Action 的安全

我们的Artimus 1.0 应用依赖于基于容器的安全以确保只有授权的用户可以访问插入、编辑和删除文章的Action。我们告诉容易将所有对以/admin/* 开头的URI的访问保留给那些属于manager, editor, 或者contributor 角色的用户。在Artimus 1.0中，这是由web.xml 中的 `<auth-constraint>` 元素处理的。

Struts 1.1 能够使你比基于容器的安全更进一步。一个ActionMapping可以指定其自己的可以访问它的角色的列表。所以为了保护我们的文章编辑Action，我们只需要添加一个`roles` 属性到mapping中，就像这样：

```
<action
  roles="manager"
  path="/admin/CreateResources"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.CreateResources">
  <forward
    name="success"
    path="/do/find/Recent"/>
```

```
</action>
```

在调用与此mapping相关的Action 之前，控制器如今将检查看发出请求的用户是否已经被分派到列出的安全角色中。控制器在检查对匹配/admin/*模式的URI的请求的时候，使用和容器所用相同的API。所以，基于action的安全不是一个*新*的系统而是对*现有*系统的改进用法。

这种改进用法不会和我们在Artimus 1.0中设置的授权约束有冲突。事实上，ActionMapping 补充了标准。我们可以原样使用原来的认证方案。当请求通过容器的约束之后，Struts 将应用ActionMapping的约束。这可以使你能够仔细的调整安全约束而不用修改应用级的部署描述符。

同时，你还可以将它用作容器安全约束的一个替代方案。

使用普通的`url-pattern` 方式，我们必须调整我们的URI 命令结构以匹配安全约束。这可以工作，但是通常会和Struts 开发人员使用action mapping的方式相干扰。如今，我们将所有 `article` 命令放入/article之下，除了那些需要安全保护的 (它们位于/admin/article之下)。现在这还不是大问题，但是在开发具有自己的`url-pattern` 要求的模块应用时，会开始变得复杂起来。

通过基于Action的安全，我们可以指定那些mapping 必须被保护而且可以使用我们喜欢的URI。所以之前我们有一个这样的mapping：

```
<action
  path="/admin/article/Edit"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.article.FindByArticle"
  name="articleForm"
  scope="request"
  validate="false">
  <forward
    name="success"
    path=".article.Form"/>
</action>
```

现在我们可以使用这样的mapping：

```
<action
  roles="contributor,manager"
  path="/article/Edit"
  type="org.apache.struts.scaffold.ProcessAction"
  parameter="org.apache.artimus.article.FindByArticle"
  name="articleForm"
```

```
  scope="request"
  validate="false">
  <forward
    name="success"
    path=".article.Form"/>
</action>
```

用户何时，或者如何登录系统，我们如何将它们注册到系统，或者当他们被授权或者没有授权时会发生什么，都没有发生一点改变。我们只是将声明基于URL模式的安全转移到声明我们想要保护的具体的哪个操作。

关于基于Action的安全的更多信息，参见第7章。

结果

从路径中删除/admin 意味着我们将需要更新一些引用到该URI的页面。如果做这个修改有问题，我们将保持ActionMapping 路径为原样。我们也提出了一个Struts forward 从/admin/article/store 到/article/store 知道页面被更新。这就可以在过度期间两个路径都可以工作。当然，也可以在容器一级做一个重定向，但是在strut-config.xml中做这种方式的重构要简单一些。

现在做这个修改的一个原因是简化今后将Artimus 用作一个应用模块。常规的`url-pattern` 方法到声明性安全在同时也要使用`url-patterns` 来标识应用模块的时候会变得复杂起来。

几乎所有的修改都可以在struts-config.xml中进行。只需要在文件中查找/admin 。如果 /admin 被在一个`<action>` 路径中找到，删除/admin 并且添加一个`roles` 属性。如果还有其它一些路径，只需要从路径中删除/admin 组件。

例外情况是

```
<forward name="logon" path="/do/admin/Menu"/>
```

和

```
<action path="/admin/Menu" forward="/do/Menu"/>
```

这是使容器能够询问用户凭证的一个小窍门。`/admin` 路径强迫发起挑战。如果他们通过，我们只需要回到菜单(但是这时已经登入)。否则，浏览器呈现标准的Unauthorized 屏幕。

16.3.3. Action 路径修改

因为 `<html:form>` 标签使用了对ActionMapping 路径的内嵌的引用，因此修改任何这些路径都意味着要修改标签。虽然ActionMappings 是一个虚*拟*引用，但它们不是真正的*逻辑*引用。其它组件使用URI 路径并且有它们自己的关于它们应该如何工作的异常。

我们修改到基于Action的安全是一把双刃剑。它简化了我们的内部API 但是强迫我们修改外部API。 我们可以通过工作减轻这些改变，但是因为Artimus 是一个如此之小的应用，我们可以bite the bullet and do the right thing。

需要使用新的action路径的两个页面是form.jsp 和view.jsp。

/WEB-INF/src/pages/articles/content/form.jsp

在form tile中的<html:form> 标签的action路径可以从/admin/article/store 修改为/article/store。

/WEB-INF/src/pages/articles/content/view.jsp

同样，view tile 中的<html:form> 标签的action path 也可以从/admin/article/Store 修改为/article/Store。

16.3.4. Struts 1.1 中的应用资源

Struts 1.1 对应用模块的支持导致了指定默认的资源束名称的方式的改变。在Struts 1.0中，应用资源束的名称是做为ActionServlet的一个初始化参数。

并且，虽然已不赞成，它仍然是这样。但是因为每一个应用模块都必须能够指定其自己的应用资源，你如今可以在Struts配置文件中指定它们自己的应用资源。

一个新的配置元素， <message-resources>，可用来制定资源。如清单16.6所示，它必须位于<controller> 元素之后（如果有）以及<plug-in> 元素之前（如果有）。

```
<message-resources
        parameter="resources.application"/>
```

因为每一个模块都有其自己的Struts配置文件，每一个模块便都可以有其自己的默认消息资源束。如果一个应用（或者模块）想要使用多个消息资源束，你可以指定多个<message-resources> 元素。任何附加的消息资源元素都必须同时指定资源束的key属性。

正如我们的Artimus 1.0中的 <init-param> 一样，这里我们告诉Struts 在资源束（或者文件夹）中查找一个名为application.properties 的文件。我们的ANT构建文件将该资源束放在/WEB-INF/classes 文件夹之下以确保资源在CLASSPATH中，这样容器就可以找到它们。

设置好 <message-resources> 元素之后，现在我们就可以从web.xml中的ActionServlet元素中删除<init-param> 了。

为了将来

当然， Artimus 并不需要使用新的<message-resources> 元素。

从应用的角度来看，我们可以将资源束的名称通过<init-param>元素传递给Artimus，而一切都会运作如常。但是因为Artimus 可能被用作一个大型应用的一个模块，我们还是将<init-param> 转换为<message-resources> 元素，以防万一。

16.4. 小结

到此为止，这就是我们将Artimus 升级到Struts 1.1 所需做的全部工作。我们现在可以使用集成版本的Struts Validator 和Tiles，对form bean使用DynaActionForm，以及使用基于角色的安全来保护我们的Action。

当然，还有一些事情可以在第二次迭代中来处理：

- 将ArtimusServlet 转换为一个PlugIn
- 重配置Artimus 为一个默认的应用和一个article 模块，以便使Artimus 能够更加容易地插入到一个更大的多模块应用之中。
- 使用新的`frame`标签实现一个帧接口

17. Velocity: JSP 的替代选择

本章内容

- 介绍 Velocity 模板
- 在 Web 应用中使用 Velocity
- 在 Struts 应用中使用 Velocity
- 将 JavaServer Pages 转换为 Velocity 模板

Change is the constant, the signal for rebirth, the egg of the phoenix.

—Christina Baldwin

17.1. 转移到 Velocity 模板

变化乃是计算机科学的核心。没有可变性，本质上说计算机程序将是毫无用处的好奇心而已— 智力训练一次次产生相同的，可预知的结果。我们可以说 $z=x+y$，而提供我们自己的 x 和 y 则可以使程序变得有用。人们总是希望程序能够进行修改，包括从内部和外部。开发人员则希望应用框架能够使这种修改更加容易实现。

在这一章，我们将对我们的应用进行一个看起来很猛烈的修改— 抛开 JavaServer Page 而使用 Velocity 模板。你可能会惊讶这种转变是多么的容易—至少在使用 Struts 之类的框架是。

17.2. 改变成就框架

能够快速地构建一个应用只是打赢了一半的战争。众所周知，一个应用一半以上的成本将花费在维护上[Linday]。Struts 框架通过将最可能改变的部分进行封装来改善了维护。这些细节中最主要的部分就是Action所引用的页面。

在这一章，我们将对我们在第3章构建的logon应用的整个表现层进行修改，从JSP转移到另一个流行的表现技术，Velocity 模板。

和Struts一样，Velocity [ASF, Velocity] 也是一个Apache Jakarta 的开源项目。它很适合于应用在Model 2/MVC 架构中。Velocity 特别在页面设计员不是Java 工程师或者不是JSP爱好者的情况下特别有用。

将JSP 改变为Velocity 可能听起来是一个比本身更大的事情。对Struts中的许多应用级的改变的

关键在于其配置文件。配置文件的设计目的是为了便于大部分实现细节—比如表现页面或者模板的名称—可以保存在一个集中的地方。如果一个应用所用的所有超连接都是通过ActionForward的，那么只是修改Struts 配置文件就可以修改每一个超链接所引用的页面。当然，当我们在第3章建我们的logon 应用的时候，我们已经小心地使用了ActionForward而不是直接的超链接。关于ActionForward 和Struts 配置文件的详细信息，参见第4章。

我们先来介绍Velocity 模板，看看它是如何工作的，然后将它插入到我们的logon 应用中。

17.3. 我们为何需要 Velocity

嗯，既然我们已经有了JSP，为什么还需要Velocity 模板呢？

17.3.1. Velocity 轻巧、快速和多能

Velocity模板旨在要求比相应的JSP更少的代码和更快速的渲染。虽然 Velocity 模板可以用于多种web 应用框架，包括Struts，但是Velocity 本身却并不限于web 应用。

Velocity 模板可以用于任何需要创建运行时定制输出的Java 应用中。这在部署一个应用到多环境中是一个重要考虑。Velocity 也是一个多用途的工具，可以用来从模板中产生SQL，PostScript，e-mail， 或者XML。

17.3.2. Velocity 与其它和谐共处

Velocity 代码可以很好的显示在可视化的 HTML 编辑器中，所以页面设计人员通常会发现Velocity 模板比JavaServer Pages更容易使用。在本书编写时，在大多数HTML 编辑器中支持JSP定指标签还几乎不见。实际应用中， JSP 标签实际看起来太像HTML 标记。大多数可视HTML 编辑器都会隐藏他们不认识的标签，使得它们在JSP中更难以使用而不是更加容易使用。很痛苦，但的确如此。

同时，基于Velocity 模板的动态页面可以很容易地使用Allaire HomeBase，Macromedia Dreamweaver，或者Microsoft FrontPage之类的成熟工具进行编辑，to name a few。具有讽刺意味的是，因为 Velocity 标记并*不象*HTML，标准编辑软件会象处理文本一样处理Velocity代码。页面设计人员可以毫无困难的查看和编辑代码。

17.3.3. Velocity 简单而强大

都知道 *Velocity 模板语言（Template Language）* (VTL) 是一个就像“装上火柴盒盖子”那么简单的API 。从HTML 页面设计人员的角度来看，这是一件很好事情。多数人都可以在一天之类学会设置并运行Velocity。同时， VTL 已经被证明是一个完整解决方案；你可以用它来做你想做的任何事情。VTL 的一个概述总结在表17.1中。

在本章， 我们将展示将Velocity 模板用作我们的logon 应用的表现层的实际例子。关于Velocity的更多信息，请访问Velocity 的Jakarta站点[ASF, Velocity]。你会惊讶于它居然真能干那么多的事情。

17.4. 在 Web 应用中使用 Velocity

传统 Web 网站的页面都是静态的不改变的。它们可以作为 HTML 文件从磁盘中读取然后返回给浏览器。web 浏览器然后读取 HTML 并且显示格式化的页面。动态 Web 应用中的页面必须针对每个用户进行定制化。在实际使用中，大多数页面都是静态和不变的；只有少数页面须的确需要变化。

在第 10 章，我们描述了使用服务器页面来综合静态标记和动态数据的策略。一个模板提供服务器页面的静态部分。这个模板则混合使用了 HTML 标记和定制页面所需的特殊代码。

JSP 是通过scriptlets 和JSP 标签来创建可定制性的。JSP 模板被编译为一个特殊的servlet。JSP servlet 渲染HTML 标记并且处理JSP 代码。当一个JSP 页面被请求时，容易实际调用的是编译后的JSP servlet。它不会再次在如何处理JSP模板。

在Velocity 中(如同在多数服务器页面系统中)，模板是实际上在运行时用来创建响应的。一个中心引擎读取模板，处理代码并返回定制结果。在一个web 环境中，是Velocity servlet 将结果返回作为HTTP 请求的响应的。

表格 17.1 Velocity 模板语言

元素	说明
#set	建立一个引用的值
#if / #elseif / #else	提供基于语句为真的条件输出
#foreach	通过一个对象列表之上的循环
#include	渲染不被Velocity 解析的本地文件
#parse	渲染被Velocity 解析的本地模板
#stop	停止模板引擎
#macro	定义一个Velocimacro (VM)，一个VTL模板的可根据需要重复的段
##	标示一个单行注释
#* … *#	标示一个多行注释
${variable}	提供一个变量引用 (对上下文中的一个属性（Attribute）)
${purchase.Total}	引用一个属性（property）— 例如，返回purchase.Total 或者purchase.getTotal()的值
${purchase.setTitle("value")}	因用一个方法 —例如，调用purchase.setTitle("value")

当模板被处理时， Velocity 模板引擎也被给予一个包含变量信息的运行时上下文。这和Web应用所用的servlet 上下文很类似，但是并没有绑定到Web层。因为Velocity 提供了其自己的context 对象，所以Velocity 模板引擎用在web应用中和常规Java 应用中是一样的。

在一个web 应用中， Velocity 可以采用标准的servlet context 并将它用作Velocity context。当Velocity servlet 采用一个上下文时，它使用了和JSP标签一样的范围策略。请求范围被首先检查，然后是会话范围，最后是应用范围。但是对引擎来说，它看起来像是一个单独的上下文。

在引擎处理模板时，它要查找Velocity 语句 *（statements）和 引用(references)。*Velocity 语句是由一个英镑符号(#) 以及出现在一行开头的关键字标识的。而引用则是上下文中的变量。

引用是由一个美元符号($) 后跟一个引用名称标识的。任何Java 对象都可以放入上下文中作

为一个引用。饮用可以访问一个Java类的任何公开方法。(表17.1就包含了Velocity 语句和引用。)

17.4.1. 与其他 Servlet 资源使用 Velocity

一个Web应用中组件间的大部分通信都涉及到Servlet API提供的四个标准资源：application context，session，request，以及response。应用上下文能够让组件共享数据和服务。会话对象则提供了事关每个用户的数据和服务。请求和响应对象一起工作来完成声名狼藉的HTTP请求-响应循环 (见第一章)。

因为一个动态页面只是Web应用中的另一个组件，它应该具有对与其他组件一样的资源的访问。Velocity提供了一个标准的VelocityViewServlet，这使得将Velocity 与你的应用中的servlet集成更加容易。

此外，这个servlet 会自动使用来自于Servlet API 的通常内容创建和组装Velocity 上下文。表17.2 类除了Velocity attributes (或者工具) 以及它们表示的API 对象。

因为这个“四人帮”[3]如今仅是Velocity上下文中的对象而已，它们便可以通过Velocity 模板语言进行访问(见表17.1)。

我们来将访问这些工具和使用定制标签来做同样的事情作一比较。Jakarta Taglibs 项目 [ASF, Taglibs] 提供了一个Request 标签库，它可以将未决的HttpServletRequest 暴露为属性，就像 `$request` Velocity 工具所为一样。为了使用`<request>` 标签检查用户的安全策略，我们可以这样：

```
<request:isUserInRole role="contributor">
<%-- … -%>
</request:isUserInRole>
```

同样的检查在Velocity中这样完成：

```
#if $request.isUserInRole("contributor")
# …
#endif
```

请求、会话、应用，或者响应对象中的任何方法和属性都可以同样的方式进行访问—但是并不需要导入标签库之类的官样文章。一旦HttpServletRequest 对象被置入Velocity 上下文之后，你就可以使用*对象的原生API*来访问它了，而不是要通过一个定制标签实现来进行传递。

[3] 译注：指四个标准 Servlet 资源。不是模式经典的 GoF。

17.4.2. 通过上下文属性使用 Velocity

大部分web 应用，特别是Struts 应用，都很好的利用了标准的servlet context。全部可以共享的对象都可以提交给应用上下文。属于单独用户的对象则放到会话上下文中。如果对象只是用于创建HTTP 响应，我们则将其塞到请求上下文中。

表格 17.2 Velocity servlet 工具

上下文关键字	类	备注
$application	javax.servlet.ServletContext	servlet context
$session	javax.servlet.http.HttpSession	当前会话，如果有的话
$request	javax.servlet.http.HttpServletRequest	当前servlet请求
$response	javax.servlet.http.HttpServletResponse	当前servlet响应

这些上下文包含在我们第17.4.1节所介绍的标准servlet 资源之中。所以我们总是可以这样来获得它们：

```
<P>Username: ${session.getAttribute("username")}</P>
```

但这对Velocity 来说还是有些冗长。为了避免输入这么多东西， Velocity 支持大多数JSP 标签实现中都有的自动范围查找。当我们引用一个属性时，请求上下文会被首先检查，然后是session， 最后才是application。Request，session，和application 资源提供的每一个上下文都串联在一起，对Velocity 模板引擎来说，它们就像是同一个context。

这就是说，我们也可以这样来获得username ：

```
<P>Username: $!user.username</P>
```

在内部，Velocity 将在标准的request context 中查找user 对象。如果没找到，它将尝试session context。如果在session contex中找到`user` 属性，Velocity 将返回 `user.getUserName()` 访法的结果。如果到处都没有找到 `user` 属性，惊叹号 (!)告诉Velocity 返回一个空字符串。

Velocity 模板语言和VelocityViewServlet 完全提供了Struts bean 和logic 标签库，以及其他东西，比如Jakarta request 标签库所提供的通用功能。

但是关于Struts 框架提供的特殊资源又如何呢？ Struts 所用的属性关键字是很长的，不易被页面设计人员记住和输入。HTML 标签通常会处理所有这些问题。

Point taken.高兴的是， Velocity为Struts 提供了一个特殊的工具包，它可以将Struts API 按照处理Servlet API 一样的处理。Struts 框架对象可以暴露为Velocity “工具” ，这样就可以毫不奇怪地使用了。

17.4.3. Velocity 如何与 Struts 共处

在Struts 配置文件(见第4章)定义的对象创建了一个应用中所用的JavaBean, hyperlink, action, 和messages 的数据库。

许多Struts附带的标签扩展，比如`<html:link>`，都是从这个数据库中查找信息的。Struts 是通过标准的servlet context暴露这个配置对象数据库的。应用中的其它任何servlet，都可以访问Struts 配置对象—如果它们知道去何处查找的话。Struts附带的JSP 标签使用了一个称为RequestUtils (`org.apache.struts.util.RequestUtils`) 的类来访问配置对象。但是应用中的任何其他组件也都可以做同样的事情。事实上， VelocityStruts 工具包以及其视图工具在Struts Velocity 模板应用中所做的，实际上就是RequestUtils 和标签库在Struts JSP 应用中所做的。

17.4.4. VelocityStruts 工具包

VelocityStruts 工具包使得在Struts中使用Velocity 模板更加容易。该工具包包括VelocityViewServlet (见第17.4.1节) 和针对Struts的Velocity View工具。

VelocityViewServlet 是和Struts 的ActionServlet一起装载到应用中的。Velocity servlet 被配置来匹配某些URL 模式，最常见的是*.vm。一旦Velocity servlet 被载入之后，我们就可以在实际转发到JSP页面的地方已相同的方式转发到VM 文件。如果我们在login.jsp之外还创建了一个logon.vm 页面，我们就可以将forward从

```
<forward
name="continue"
path="/pages/Logon.jsp"/>
```

修改为：

```
<forward
name="continue"
path="/pages/Logon.vm"/>
```

在实践中， Struts 并不渲染JSP。那是容器所带的服务的事情。所以，Struts 可以向转发到JSP服务一样将控制转发到VelocityViewServlet。从控制器的角度来看，它们都只是URI而已。

在一个Struts JSP中，实际的威力是定制标签发挥的。它们知道如何访问Struts 配置并且使用这些对象来定制页面。

而在一个Struts Velocity模板中， View 工具也知道同样的魔咒，并且也可以象Struts标签库一样访问Struts 配置对象。在我们转发到JSP的地方，就可以转发到VM。哪里使用定制标签，哪里就可以使用View 工具。真的很简单。

但是没有偶然...

Sun 原本设计JSP标签是作为JavaServer Page 访问JavaBean的一个手段。那些十分仔细设计的标签库，比如Struts所带的标签库，就是基于这种设计理念的基础。应用创建JavaBeans 来

封装数据。JSP 标签访问JavaBean，将数据包装在HTML 标记中，然后输出结果。

Velocity 模板语言也是用相同的设计模式。对象被另一个组件放置在共享的上下文中。模板语言处理这些对象提供的数据并且输出结果。

咋看起来， Velocity 工具编码的页面和JSP标签编码的页面非常相似。它们都基于完全相同的原因以一种非常相似的方式来做同样的事情。实现不同，但策略保持一致。

17.4.5. Struts View 工具

表格 17.3 Velocity Struts View 工具(org.apache.velocity.tools.struts)

上下文关键字	类	备注
$msg	MessageTool	提供对作为国际化输出的Struts消息资源的访问
$errors	ErrorsTool	提供检测和输出Struts错误消息的方法
$link	LinkTool	提供使用URL的方法
$form	FormTool	提供与Struts应用的上下文中的表单bean一起工作的各种方法

我们曾说过， Struts 标签库提供的很多功能已经内置到Velocity 模板语言和VelocityViewServlet中了。Struts特定的特征是通过servlet提供给你的模板的四个JavaBeans, 或者说工具，提供的。 (类似于第17.4.1节讨论的Servlet API 资源)。表17.3 item就了列出了四个Struts特定的Velocity View 工具。

我们来看看一个 Struts JSP和一个Struts Velocity 模板的对比。

17.5. 我们的 logon 模板

因为它们都是要做同样的事情， Velocity 模板和JSP 版本看起来都非常相像，特别是当你仔细比较它们的时候。(表单Form follows function.) 你会惊讶地发现对于阅读模板代码，你已经知道的够多了。如果你还有些糊涂，请返回去看表 17.1 ，作为一个快速参考(或者将它写在顺手的什么纸片上)。Welcome.vm 展示在图 17.1a中，而Welcome.jsp则在图17.1b中。

```
<HTML>
<HEAD>
<TITLE>Welcome!</TITLE>
<BASE href="$link.baseRef">
```

```
<%@ taglib uri="/tags/struts-bean"
prefix="bean" %>
<%@ taglib uri="/tags/struts-html"
```

```
</HEAD>
<BODY>
#if( $user )
<H3>Welcome $user.username!</H3>
#else
<H3>Welcome World!</H3>
#end
$!errors.msgs()
<UL>
<LI><A
href="$link.setForward('logon')">
Sign in</A></LI>
#if( $user )
<LI><A href=
"$link.setForward('logoff')">
Sign out</A></LI>
#end
</UL>
<IMG src='struts-power.gif'
alt='Powered by Struts'>
<IMG src='velocity-power.gif'
alt='Powered by Velocity'>
</BODY>
</HTML>
```

```
prefix="html" %>
<%@                          taglib
uri="/tags/struts-logic"
prefix="logic" %>
<HTML>
<HEAD>
<TITLE>Welcome!</TITLE>
<html:base/>
</HEAD>
<BODY>
<logic:present name="user">
<H3>Welcome <bean:write name="user"
property="username"/>!</H3>
</logic:present>
<logic:notPresent scope="session"
name="user">
<H3>Welcome World!</H3>
</logic:notPresent>
<html:errors/>
<UL>
<LI><html:link forward="logon">Sign
in</html:link></LI>
<logic:present name="user">
<LI><html:link forward="logoff">Sign
out</html:link></LI>
</logic:present>
</UL>
<IMG src='struts-power.gif'
alt='Powered by Struts'>
</BODY>
</HTML>
```

如你所见，标记的相似之处多于不同点，我们在第3章（第3.3.2节）所示的注解也同样适用这里。

logon 页面的两个版本也和类似，分别列于图17.2a 和17.2b。

虽然可以看到Velocity 版本需要在页面中编写更多一些的HTML 标签但是在如今的可视化HTML编辑器中编写一个Velocity 模板也是很容易的。对HTML 编辑软件来说，Velocity 标记看起来就像控件的初始值。 (并且，共某种意义上说，也对。)然后Velocity 标记就可以被编辑软件的常规对话框中输入并且在一个运行应用之外的页面中单独进行测试。所以，在实践中， 之需要手工编写很少的HTML 代码。

```
<HTML>
<HEAD>
<TITLE>Sign in, Please!</TITLE>
</HEAD>
<BODY>
$!errors.msgs()
<FORM method="POST"
action="$link.setAction('/Logon
Submit')">
<TABLE border="0" width="100%">
<TR>
<TH align="right">Username:</TH>
<TD          align="left"><INPUT
type="text"
name="username"
value="$!logonForm.username"></TD
>
</TR>
<TR>
<TH align="right">Password:</TH>
<TD align="left"><INPUT
type="password" name="password"
value="$!logonForm.password"></TD
>
</TR>
<TR>
<TD align="right"><INPUT
type="submit" value="Submit"
name="submit"></TD>
<TD          align="left"><INPUT
type="reset"
value="Reset"name="reset"></TD>
```

```
<%@ taglib uri="/tags/struts-html"
prefix="html" %>
<HTML>
<HEAD>
<TITLE>Sign in, Please!</TITLE>
</HEAD>
<BODY>
<html:errors/>
<html:form action="/LogonSubmit"
focus="username">
<TABLE border="0" width="100%">
<TR>
<TH align="right">Username:</TH>
<TD align="left"><html:text
property="username"/></TD>
</TR>
<TR>
<TH align="right">Password:</TH>
<TD align="left"><html:password
property="password"/></TD>
</TR>
<TR>
<TD
align="right"><html:submit/></TD>
<TD align="left"><html:reset/></TD>
</TR>
</TABLE>
</html:form>
</BODY>
</HTML
```

</TR> </TABLE> </FORM> </BODY> </HTML>	

类似的功能对JSP也是有的。已经有一些针对 Dreamweaver 4 [ASF, CTLX]的插件，但是可悲的是，市场目前对提供JSP定制标签的直接支持还很缓慢。而同时，Velocity 模板欠却可以使用任何成熟的HTML 编辑器编辑。实践中，使用Velocity 模板的HTML 页面设计人员将会比它们使用JSP时输入更少的应用特定的代码。

通过象Struts之类的框架，你可以自己选择你的表现层系统，并使用最适合于你的应用和开发关团队的那个。来自于第3章的logon应用的修改版本可以从本书网站[Husted]下载，文件名为logon-velocity.war。你可以下载它，并且安装到你的应用中，来看看Velocity是如何运转的。你可以使用本书的作者们的名称来登录，如表17.4所示。使用名（ first name ）作为userId ，而姓作为口令。

表格 17.4 默认登录

userId	Password
Ted	Husted
Cedric	Dumoulin
George	Franciscus
David	Winterfeldt
Craig	McClanahan

口令是大小写敏感的，所以请确保使用第一个字母大写的格式。

下面我们来看一些如何既逆行配置，能够使我们的logon Velocity 应用运行起来。

17.6. 设置 VelocityViewServlet

在可以实际测试Velocity 模板之前，我们需要能够访问Velocity 模板引擎。通常你需要创建一个你自己的针对特定应用设计的引擎实例。 但是因为基于框架（比如Struts）的web 应用都具有十分相似的需要，Velocity团队变提供了一个标准的VelocityViewServlet (`org.apache.velocity.tools.view.servlet.VelocityViewServlet`) 来为我们照顾这一切。

17.6.1. 安装 VelocityViewServlet

VelocityViewServlet 自身位于logon 应用的WEB-INF/lib 文件夹中的velocity-tools-view.jar 文件中。logon 应用的web.xml 也包括了Velocity servlet 的配置，并且是一个可运行的，在一个文件中同时配置了ActionServlet 和Velocity servlet的例子。除了velocity-tools-view.jar之外，我们还添加了velocity-tools-library.jar文件，velocity-tools-struts.jar，以及dom4j.jar 文件到lib 文件夹中。对Struts 1.0.2来说，我们还需要添加jakarta-commons-collection.jar。总的来说，给定通常的Struts JAR就可以了。

当然，最新版本的 Velocity Struts 工具包可以从Velocity 在Jakarta 的网站[ASF, Velocity]得到。

给出所需的JAR后，我们只需要配置我们的部署描述符(web.xml) 来装载Velocity servlet 并且添加一个toolbox.xml 配置文件到WEB-INF 中就可以开始玩了。

17.6.2. 部署 Velocity servlet

和任何servlet一样， VelocityViewServlet 是通过web 应用部署描述符部署的。清单17.1就展示了我们添加到logon 应用的 web.xml 中的servlet 配置元素。

清单 17.1 /WEB-INF/web.xml (Velocity servlet 元素)

```
<!-- Define Velocity Template compiler -->
<servlet>
  <servlet-name>velocity</servlet-name>
  <servlet-class>
    org.apache.velocity.tools.view.servlet.VelocityViewServlet
  </servlet-class>
  <init-param>
    <param-name>toolbox</param-name>
    <param-value>/WEB-INF/toolbox.xml</param-value>
  </init-param>
  <load-on-startup>10</load-on-startup>
</servlet>
<!-- Map *.vm files to Velocity -->
<servlet-mapping>
  <servlet-name>velocity</servlet-name>
  <url-pattern>*.vm</url-pattern>
</servlet-mapping>
```

工具包参数

如前所述， Struts 配置文件是通过Velocity 成为工具（*tools*）*的*一些便捷预定义对象来暴露给Velocity 模板的。工具其实是能够使你对框架对象进行快捷和容易访问的助手对象。其它工具可以有助于进行数据变换、数学计算、以及其他你可能在你的表现页面中使用和遇到的问题。工具可以有助于以定制标签扩展JSP的相同方式扩展Velocity 模板。

Struts View 工具是众多工具中唯一由Velocity 团队规划的工具。`toolbox` 参数用于指定工具配置文件。即从中我们可以从中装载Struts 工具和其他我们可能会需要使用的东西的地方。概念上讲，这和开发一个JSP可用的定制标签库十分相似。我们将在第17.6.3节详细讨论Velocity 工具包。

properties 参数

Velocity 是一个全特征的组件并且提供大量的配置选项。它们都可以放入到一个Properties文件中并且可以根据你的应用进行调整。

我们的应用并不需要修改Velocity 的默认设置，所以我们略去了velocity.properties 文件。关于Velocity 可以配置的各种不同方式，请参考Velocity 站点的文档[ASF, Velocity]。

17.6.3. 工具包配置文件

我们的工具包配置文件，示于清单17.2，只是装载了Struts View工具。你或者其他人开发的工具也可以在此装载：

清单 17.2 /WEB-INF/conf/struts-config.xml

```
<?xml version="1.0"?>
<toolbox>
  <tool>
    <key>toolLoader</key>
    <class>org.apache.velocity.tools.tools.ToolLoader</class>
  </tool>
  <tool>
    <key>link</key>
    <class>org.apache.velocity.tools.struts.LinkTool</class>
  </tool>
  <tool>
    <key>msg</key>
    <class>org.apache.velocity.tools.struts.MessageTool</class>
  </tool>
  <tool>
    <key>errors</key>
```

```
    <class>org.apache.velocity.tools.struts.ErrorsTool</class>
  </tool>
  <tool>
    <key>form</key>
    <class>org.apache.velocity.tools.struts.FormTool</class>
  </tool>
</toolbox>
```

如果基于某些原因你需要修改工具的名称，你可以在此修改 `name` 参数，而它将会被暴露在新的属性名称之下。

17.7. 设置 struts 配置

在这里，改变到Velocity 模板只需要在struts-config .xml文件中将*.jsp* 替换为*.vm* 。你甚至可以通过查找替换来进行这件事情。

修改后的struts-config.xml文件示于清单17.3。

清单 17.3 修改后的 struts-config.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts
Configuration                                                    1.0//EN"
"http://jakarta.apache.org/struts/dtds/strutsconfig_
1_0.dtd">
<struts-config>
 <form-beans>
  <form-bean
   name="logonForm"
   type="app.LogonForm"/>
 </form-beans>
 <global-forwards>
  <forward
   name="logoff"
   path="/Logoff.do"/>
```

```
  <forward
   name="logon"
   path="/Logon.do"/>
  <forward
   name="welcome"
   path="/Welcome.do"/>
 </global-forwards>
 <action-mappings>
  <action
   path="/Welcome"
   type="org.apache.struts.actions.ForwardAction"
   parameter="/pages/Welcome.vm"/>
  <action
   path="/Logon"
   type="org.apache.struts.actions.ForwardAction"
   parameter="/pages/Logon.vm"/>
  <action
   path="/LogonSubmit"
   type="app.LogonAction"
   name="logonForm"
   scope="request"
   validate="true"
   input="/pages/Logon.vm">
   <forward
    name="success"
    path="/pages/Welcome.vm"/>
  </action>
  <action
   path="/Logoff"
   type="app.LogoffAction">
   <forward
    name="success"
    path="/pages/Welcome.vm"/>
  </action>
 </action-mappings>
</struts-config>
```

如果你在logon 应用，或者你自己构建的应用的配置文件中做了这样的修改，那么你的应用应该可以进行测试了。你应该可以看到和原来相同的结果，除了welcome页面中额外的“Powered by Velocity” 图像之外，如图17.3所示。

当然，你也可以在同一个应用中混合使用JavaServer Pages 和Velocity 模板。开始使用Velocity 的一个好方式是一次替换一个JavaServer Page。这不是一个或然命题。选择在于你。

17.8. 小结

Velocity 是处理MVC应用的表现层的一个优秀方式，包括基于Struts的web 应用。针对Struts的Velocity View 工具使得访问框架资源更加容易。而VelocityViewServlet 则使得访问Servlet API 资源，以及其它你可能放入上下文中的助手对象等资源更加容易。

因为Velocity 和JSP 定制标签都基于公共的设计理念，从一个转移到另一个是一件很直接的事情。定制标签和一个Velocity 声明或工具之间可以在一对一的基础上进行互换。

在一个诸如Struts之类的MVC web 环境中，Velocity 模板和定制标签在同场竞技。那一个技术最适合你的应用更多的在于，将由***谁***创建表现层而不是他们应该使用***什么***来创建它。如果Java 工程师已经习惯于使用JavaServer Pages创建表现页面，那么JSP 就是不二选择。换言之，如果你的工程师将和使用静态HTML页面编辑工具的页面设计人员一起工作，则Velocity模板可能是更好的选择。

如果基于某些原因你需要混合使用JSP 和Velocity 模板，你也可以这么做。这两种技术不是相互排斥的。事实上，它们可以和谐共处。

重要的是使用分层架构和一个象Struts之类多才多艺的框架，这样选择权就在于你了。你可以自己决定。