

以经验为后盾 以实用为目标 以实例为导向 以实践为指导

Struts 2

权威指南

——基于WebWork核心的MVC开发

- 通过实例演示Struts 2框架的用法
- 覆盖到Struts 2近80%的API
- 可以作为Struts 2框架的权威手册

李 刚 编著

光盘包含：

本书所有实例代码



CHINA-PUB.COM

本章要点

- Web 应用的发展
- Model 1 和 Model 2
- MVC 思想
- MVC 模式的优势
- 常用 MVC 框架及其特征
- Struts 1 的基本结构及其存在的问题
- WebWork 的基本结构
- Struts 2 的起源
- Struts 2 的框架架构
- Struts 2 的标签库
- Struts 2 的控制器组件
- Struts 1 和 Struts 2 的对比

Struts 1 是全世界第一个发布的 MVC 框架，它由 Craig McClanahan 在 2001 年发布，该框架一经推出，就得到了世界上 Java Web 开发者的拥护，经过长达 6 年时间的锤炼，Struts 1 框架更加成熟、稳定，性能也有了很好的保证。因此，到目前为止，Struts 1 依然是世界上使用最广泛的 MVC 框架。

目前，基于 Web 的 MVC 框架非常多，发展也很快，每隔一段时间就有一个新的 MVC 框架发布，例如像 JSF、Tapestry 和 Spring MVC 等。除了这些有名的 MVC 框架外，还有一些边缘团队的 MVC 框架也很有借鉴意义。

对于企业实际使用 MVC 框架而言，框架的稳定性则应该是最值得考虑的问题。一个刚刚起步的框架，可能本身就存在一些隐藏的问题，会将自身的 BUG 引入自己的应用。这也是笔者不推荐开发者自己实现框架的原因。

虽然 Struts 2 号称是一个全新的框架，但这仅仅是相对 Struts 1 而言。Struts 2 与 Struts 1 相比，确实有很多革命性的改进，但它并不是新发布的新框架，而是在另一个赫赫有名的框架：WebWork 基础上发展起来的。从某种程度上来讲，Strut2 没有继承 Struts 1 的血统，而是继承了 WebWork 的血统。或者说，WebWork 衍生出了 Struts 2，而不是 Struts 1 衍生了 Struts 2。因为 Struts 2 是 WebWork 的升级，而不是一个全新的框架，因此稳定性、

性能等各方面都有很好的保证；而且吸收了 Struts 1 和 WebWork 两者的优势，因此，是一个非常值得期待的框架。

1.1 MVC 思想概述

今天，我们见到的绝大部分应用，都是基于 B/S（浏览器/服务器）架构的，其中的服务器就是 Web 服务器。可见，Web 应用是目前广泛使用的应用模式，而 Struts 2 是一个具有很好的实用价值的 Web MVC 框架。介绍 Struts MVC 框架之前，我们首先介绍 Web 应用的发展历史和 MVC 思想。

1.1.1 Web 技术的发展

随着 Internet 技术的广泛使用，Web 技术已经广泛应用于 Internet 上，但早期的 Web 应用全部是静态的 HTML 页面，用于将一些文本信息呈现给浏览者，但这些信息是固定写在 HTML 页面里的，该页面不具备与用户交互的能力，没有动态显示的功能。

很自然地，人们希望 Web 应用里应该包含一些能动态执行的页面，最早的 CGI（通用网关接口）技术满足了该要求，CGI 技术使得 Web 应用可以与客户端浏览器交互，不再需要使用静态的 HTML 页面。CGI 技术可以从数据库读取信息，将这些信息呈现给用户；还可以获取用户的请求参数，并将这些参数保存到数据库里。

CGI 技术开启了动态 Web 应用的时代，给了这种技术无限的可能性。但 CGI 技术存在很多缺点，其中最大的缺点就是开发动态 Web 应用难度非常大，而且在性能等各方面也存在限制。到 1997 年时，随着 Java 语言的广泛使用，Servlet 技术迅速成为动态 Web 应用的主要开发技术。相比传统的 CGI 应用而言，Servlet 具有大量的优势：

- Servlet 是基于 Java 语言创建的，而 Java 语言则内建了多线程支持，这一点大大提高了动态 Web 应用的性能。
- Servlet 应用可以充分利用 Java 语言的优势，例如 JDBC（Java DataBase Connection）等。同时，Java 语言提供了丰富的类库，这些都简化了 Servlet 的开发。
- 除此之外，Servlet 运行在 Web 服务器中，由 Web 服务器去负责管理 Servlet 的实例化，并对客户端提供多线程、网络通信等功能，这都保证 Servlet 有更好的稳定性和性能。

Servlet 在 Web 应用中被映射成一个 URL（统一资源定位），该 URL 可以被客户端浏览器请求，当用户向指定 URL 对应的 Servlet 发送请求时，该请求被 Web 服务器接收到，该 Web 服务器负责处理多线程、网络通信等功能，而 Servlet 的内容则决定了服务器对客户端的响应内容。

图 1.1 显示了 Servlet 的响应流程。

正如图 1.1 所显示的，浏览器向 Web 服务器内指定的 Servlet 发送请求，Web 服务器根据 Servlet 生成对客户端的响应。

实际上，这是后来所有的动态 Web 编程技术所使用的模型，这种模型都需要一个动态的程序，或者一个动态页面，当客户端向该动态程序或动态页面发送请求时，Web 服务器根据该动态程序来生成对客户端的响应。

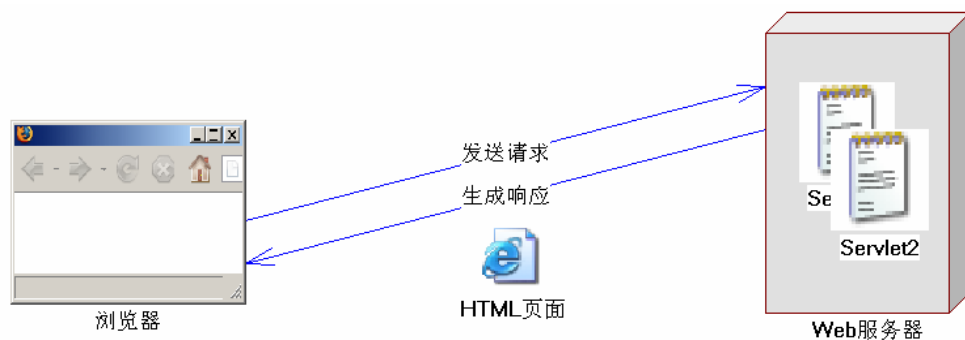


图 1.1 Servlet 的响应流程

到了 1998 年，微软发布了 ASP 2.0。它是 Windows NT 4 Option Pack 的一部分，作为 IIS 4.0 的外接式附件。它与 ASP 1.0 的主要区别在于它的外部组件是可以初始化的，这样，在 ASP 程序内部的所有组件都有了独立的内存空间，并可以进行事务处理。标志着 ASP 技术开始真正作为动态 Web 编程技术。

当 ASP 技术在世界上广泛流行时，人们很快感受到这种简单的技术的魅力：ASP 使用 VBScript 作为脚本语言，它的语法简单、开发效率非常高。而且，世界上已经有了非常多的 VB 程序员，这些 VB 程序员可以很轻易地过渡成 ASP 程序员——因此，ASP 技术马上成为应用最广泛的动态 Web 开发技术。

随后，由 Sun 带领的 Java 阵营，立即发布了 JSP 标准，从某种程度上来看，JSP 是 Java 阵营为了对抗 ASP 推出的一种动态 Web 编程技术。

ASP 和 JSP 从名称上如此相似，但它们的运行机制存在一些差别，这主要是因为 VBScript 是一种脚本语言，无需编译，而 JSP 使用 Java 作为脚本语句——但 Java 从来就不是解释型的脚本语言，因此 JSP 页面并不能立即执行。因此，JSP 必须编译成 Servlet，这就是说：JSP 的实质还是 Servlet。不过，书写 JSP 比书写 Servlet 简单得多。

JSP 的运行机理如图 1.2 所示。



图 1.2 JSP 的运行机理

对比图 1.1 和图 1.2，发现不论是 Servlet 动态 Web 技术，还是 JSP 动态 Web 技术，它们的实质完全一样。可以这样理解：JSP 是一种更简单的 Servlet 技术，这也是 JSP 技术出现的意义——作为一个和 ASP 对抗的技术，简单就是 JSP 的最大优势。

随着实际 Web 应用的使用越来越广泛，Web 应用的规模也越来越大，开发人员发现动态 Web 应用的维护成本越来越大，即使只需要修改该页面的一个简单按钮文本，或者一段静态的文本内容，也不得不打开混杂的动态脚本的页面源文件进行修改——这是一种很大的风险，完全有可能引入新的错误。

这个时候，人们意识到：使用单纯的 ASP，或者 JSP 页面充当过多角色是相当失败的选择，这对于后期的维护相当不利。慢慢地开发人员开始在 Web 开发中使用 MVC 模式。

随后就是 Java 阵营发布了一套完整的企业开发规范：J2EE（现在已经更名为 Java EE），紧跟着微软也发布了 ASP.NET 技术，它们都采用一种优秀的分层思想，力图解决 Web 应用维护困难的问题。

动态 Web 编程技术大致有如图 1.3 所示的路线。

1.1.2 Model 1 和 Model 2

对于 Java 阵营的动态 Web 编程技术而言，则经历了所谓的 Model 1 和 Model 2 时代。

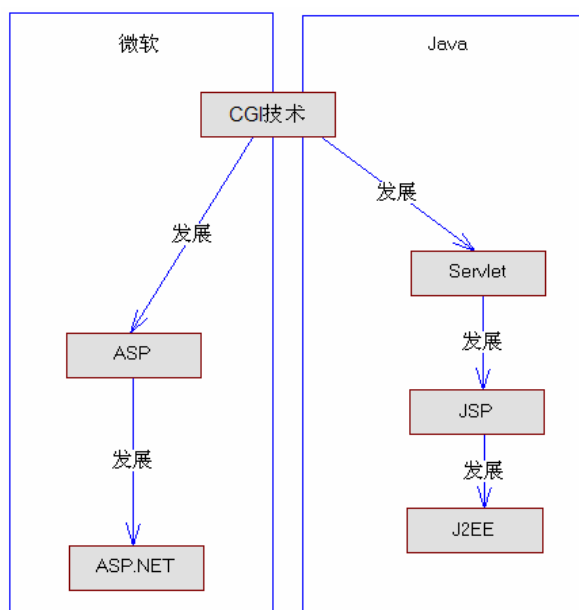


图 1.3 动态 Web 编程技术的发展历史

所谓 Model 1 就是 JSP 大行其道的时代，在 Model 1 模式下，整个 Web 应用几乎全部由 JSP 页面组成，JSP 页面接收处理客户端请求，对请求处理后直接做出响应。用少量的 JavaBean 来处理数据库连接、数据库访问等操作。

图 1.4 显示了 Model 1 的程序流程。

Model 1 模式的实现比较简单，适用于快速开发小规模项目。但从工程化的角度看，它的局限性非常明显：JSP 页面身兼 View 和 Controller 两种角色，将控制逻辑和表现逻辑混杂在一起，从而导致代码的重用性非常低，增加了应用的扩展性和维护的难度。

早期有大量 ASP 和 JSP 技术开发出来的 Web 应用，这些 Web 应用都采用了 Model 1 架构。

Model 2 已经是基于 MVC 架构的设计模式。在 Model 2 架构中，Servlet 作为前端控制器，负责接收客户端发送的请求，在 Servlet 中只包含控制逻辑和简单的前端处理；然后，调用后端 JavaBean 来完成实际的逻辑处理；最后，转发到相应的 JSP 页面处理显示逻辑。其具体的实现方式如图 1.5 所示。

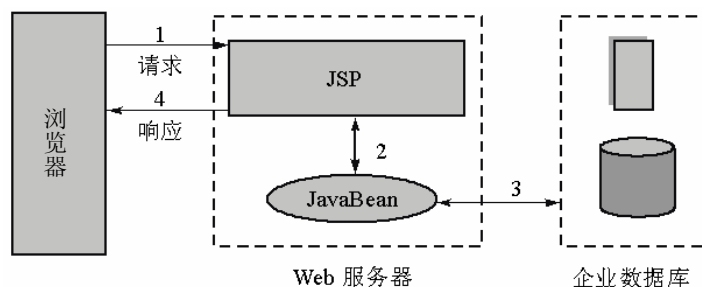


图 1.4 Model 1 的程序流程

图 1.5 显示了 Model 2 的程序流程。

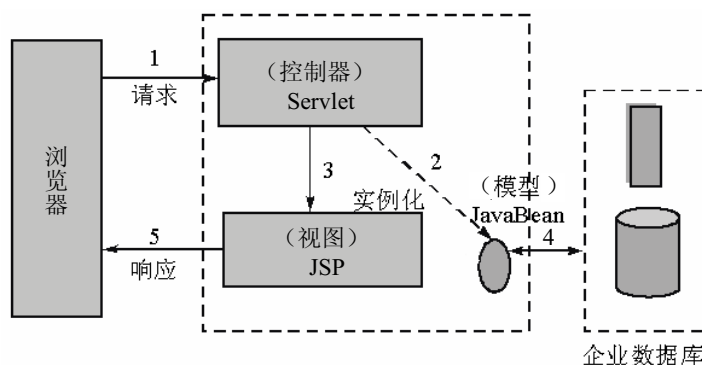


图 1.5 Model 2 的程序流程

正如图 1.5 中看到的，Model 2 下 JSP 不再承担控制器的责任，它仅仅是表现层角色，仅仅用于将结果呈现给用户，JSP 页面的请求与 Servlet（控制器）交互，而 Servlet 负责与后台的 JavaBean 通信。在 Model 2 模式下，模型（Model）由 JavaBean 充当，视图（View）由 JSP 页面充当，而控制器（Controller）则由 Servlet 充当。

由于引入了 MVC 模式，使 Model 2 具有组件化的特点，更适用于大规模应用的开发，但也增加了应用开发的复杂程度。原本需要一个简单的 JSP 页面就能实现的应用，在 Model 2 中被分解成多个协同工作的部分，需花更多时间才能真正掌握其设计和实现过程。

Model 2 已经是 MVC 设计思想下的架构，下面简要介绍 MVC 设计思想的优势。

★ 注意 对于非常小型的 Web 站点，如果后期的更新、维护工作不是特别大，可以使用 Model 1 的模式来开发应用，而不是使用 Model 2 的模式。虽然 Model 2 提供了更好的可扩展性及可维护性，但增加了前期开发成本。从某种程度上讲，Model 2 为了降低系统后期维护的复杂度，却导致前期开发的更高复杂度。

1.1.3 MVC 思想及其优势

MVC 并不是 Java 语言所特有的设计思想，也并不是 Web 应用所特有的思想，它是所有面向对象程序设计语言都应该遵守的规范。

MVC 思想将一个应用分成三个基本部分：Model（模型）、View（视图）和 Controller（控制器），这三个部分以最少的耦合协同工作，从而提高应用的可扩展性及可维护性。

起初，MVC 模式是针对相同的数据需要不同显示的应用而设计的，其整体的效果如

图 1.6 所示。

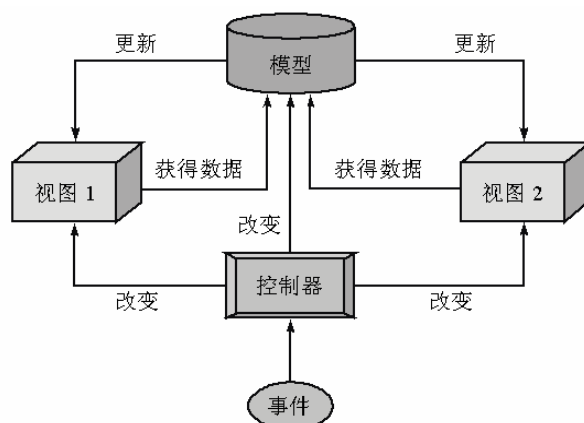


图 1.6 MVC 结构

在经典的 MVC 模式中，事件由控制器处理，控制器根据事件的类型改变模型或视图，反之亦然。具体地说，每个模型对应一系列的视图列表，这种对应关系通常采用注册来完成，即：把多个视图注册到同一个模型，当模型发生改变时，模型向所有注册过的视图发送通知，接下来，视图从对应的模型中获得信息，然后完成视图显示的更新。

从设计模式的角度来看，MVC 思想非常类似于一个观察者模式，但与观察者模式存在少许差别：观察者模式下观察者和被观察者可以是两个互相对等的对象，但对于 MVC 思想而言，被观察者往往只是单纯的数据体，而观察者则是单纯的视图页面。

概括起来，MVC 有如下特点。

- 多个视图可以对应一个模型。按 MVC 设计模式，一个模型对应多个视图，可以减少代码的复制及代码的维护量，一旦模型发生改变，也易于维护。
- 模型返回的数据与显示逻辑分离。模型数据可以应用任何的显示技术，例如，使用 JSP 页面、Velocity 模板或者直接产生 Excel 文档等。
- 应用被分隔为三层，降低了各层之间的耦合，提供了应用的可扩展性。
- 控制层的概念也很有效，由于它把不同的模型和不同的视图组合在一起，完成不同的请求。因此，控制层可以说是包含了用户请求权限的概念。
- MVC 更符合软件工程化管理的精神。不同的层各司其职，每一层的组件具有相同的特征，有利于通过工程化和工具化产生管理程序代码。

相对于早期的 MVC 思想，Web 模式下的 MVC 思想则又存在一些变化，因为对于一个应用程序而言，我们可以将视图注册给模型，当模型数据发生改变时，即时通知视图页面发生改变；而对于 Web 应用而言，即使将多个 JSP 页面注册给一个模型，当模型发生变化时，模型无法主动发送消息给 JSP 页面（因为 Web 应用都是基于请求/响应模式的），只有当用户请求浏览该页面时，控制器才负责调用模型数据来更新 JSP 页面。

✦ 注意 MVC 思想与观察者模式有一定的相似之处，但并不完全相同。经典的 MVC 思想与 Web 应用的 MVC 思想也存在一定的差别，引起差别的主要原因是因为 Web 应用是一种请求/响应模式下应用，对于请求/响应应用，如果用户不对应用发出请求，视图无法主动更新自己。

1.1.4 常用的 MVC 框架

目前常用的 MVC 框架,除了 Struts 2 的两个前身外,还有一些非常流行的 MVC 框架,这些框架都提供了较好的层次分隔能力。在实现良好的 MVC 分隔的基础上,还提供一些辅助类库,帮助应用的开发。

目前常用的 MVC 框架还有如下一些。

1. JSF

准确地说,JSF 是一个标准,而不是一个产品。目前,JSF 已经有两个实现产品可供选择,包含 Sun 的参考实现和 Apache 的 MyFaces。大部分的时候,我们所说的 JSF 都是指 Sun 的参考实现。目前,JSF 是作为 JEE 5.0 的一个组成部分,与 JEE 5.0 一起发布。

JSF 的行为方法在 POJO 中实现,JSF 的 Managed Bean 无需继承任何特别的类。因此,无需在表单和模型对象之间实现多余的控制器层。JSF 中没有控制器对象,控制器行为通过模型对象实现。

当然,JSF 也允许生成独立的控制器对象。在 Struts 1 中,Form Bean 包含数据,Action Bean 包含业务逻辑,二者无法融合在一起。在 JSF 中,既可以将二者分开,也可以合并在一个对象中,提供更多灵活的选择。

JSF 的事件框架可以细化到表单中每个字段。JSF 依然是基于 JSP/Servlet 的,仍然是 JSP/Servlet 架构,因而学习曲线相对简单。在实际使用过程中,JSF 也会存在一些不足:

- 作为新兴的 MVC 框架,用户相对较少,相关资源也不是非常丰富。
- JSF 并不是一个完全组件化的框架,它依然是基于 JSP/Servlet 架构的。
- JSF 的成熟度还有待进一步提高。

2. Tapestry

Tapestry 并不是一种单纯的 MVC 框架,它更像 MVC 框架和模板技术的结合,它不仅包含了前端的 MVC 框架,还包含了一种视图层的模板技术,使用 Tapestry 完全可以与 Servlet/JSP API 分离,是一种非常优秀的设计。

通过使用 Tapestry,开发者完全不需要使用 JSP 技术,用户只需要使用 Tapestry 提供的模板技术即可,Tapestry 实现了视图逻辑和业务逻辑的彻底分离。

Tapestry 使用组件库替代了标签库,没有标签库概念,从而避免了标签库和组件结合的问题。Tapestry 是完全组件化的框架。Tapestry 只有组件或页面两个概念,因此,链接跳转目标要么是组件,要么是页面,没有多余的 path 概念。组件名,也就是对象名称,组件名称和 path 名称合二为一。

Tapestry 具有很高的代码复用性,在 Tapestry 中,任何对象都可看作可复用的组件。JSP 开发者是真正面向对象,而不是 URL 解析。对于对页面要求灵活度相当高的系统,Tapestry 是第一选择。精确地错误报告,可以将错误定位到源程序中的行,取代了 JSP 中那种编译后的提示。

因此,笔者一直对 Tapestry 情有独钟:如果技术允许,使用 Tapestry 会带给整个应用更加优雅的架构,更好的开发效率。

但是,在实际开发过程中,采用 Tapestry 也面临着一些问题必须考虑:

- Tapestry 的学习曲线相对陡峭，国内开发群体不是非常活跃，文档不是十分丰富。官方的文档太过学院派，缺乏实际的示例程序。
- Tapestry 的组件逻辑比较复杂，再加上 OGNL 表达式和属性指定机制，因而难以添加注释。

3. Spring MVC

Spring 提供了一个细致完整的 MVC 框架。该框架为模型、视图、控制器之间提供了一个非常清晰的划分，各部分耦合极低。Spring 的 MVC 是非常灵活的，它完全基于接口编程，真正实现了视图无关。视图不再强制要求使用 JSP，可以使用 Velocity、XSLT 或其他视图技术。甚至可以使用自定义的视图机制——只需要简单地实现 View 接口，并且把对应视图技术集成进来。Spring 的 Controllers 由 IoC 容器管理。因此，单元测试更加方便。

Spring MVC 框架以 DispatcherServlet 为核心控制器，该控制器负责拦截用户的所有请求，将请求分发到对应的业务控制器。

Spring MVC 还包括处理器映射、视图解析、信息国际化、主题解析、文件上传等。所有控制器都必须实现 Controller 接口，该接口仅定义 ModelAndView handleRequest (request, response) 方法。通过实现该接口来实现用户的业务逻辑控制器。

Spring MVC 框架有一个极好的优势，就是它的视图解析策略：它的控制器返回一个 ModelAndView 对象，该对象包含视图名字和 Model，Model 提供了 Bean 的名字及其对象的对应关系。视图名解析的配置非常灵活，抽象的 Model 完全独立于表现层技术，不会与任何表现层耦合：JSP、Velocity 或者其他的技术——都可以和 Spring 整合。

但相对于 Tapestry 框架而言，Spring MVC 依然是基于 JSP/Servlet API 的。

总体上来看，Spring MVC 框架致力于一种完美的解决方案，并与 Web 应用紧紧耦合在一起。这都导致了 Spring MVC 框架的一些缺点：

- Spring 的 MVC 与 Servlet API 耦合，难以脱离 Servlet 容器独立运行，降低了 Spring MVC 框架的可扩展性。
- 太过细化的角色划分，太过烦琐，降低了应用的开发效率。
- 过分追求架构的完美，有过度设计的危险。

1.2 Struts 2 的起源和背景

Struts 2 以 WebWork 优秀的设计思想为核心，吸收了 Struts 1 的部分优点，建立了一个兼容 WebWork 和 Struts 1 的 MVC 框架，Struts 2 的目标是希望可以让原来使用 Struts 1、WebWork 的开发人员，都可以平稳过渡到使用 Struts 2 框架。

1.2.1 Struts 1 简介及存在的问题

从过去的岁月来看，Struts 1 是所有 MVC 框架中不容辩驳的胜利者，不管是市场占有率，还是所拥有的开发人群，Struts 1 都拥有其他 MVC 框架不可比拟的优势。Struts 1 的成功得益于它丰富的文档、活跃的开发群体。当然，Struts 1 是世界上第一个发布的 MVC 框架：Struts 1.0 在 2001 年 6 月发布，这一点可能是使它得到如此广泛拥戴的主要原因。

为了使读者可以明白 Struts 1 的运行机制，下面将简要介绍 Struts 1 的基本框架。

Struts 1 框架以 `ActionServlet` 作为核心控制器，整个应用由客户端请求驱动。当客户端向 Web 应用发送请求时，请求将被 Struts 1 的核心控制器 `ActionServlet` 拦截，`ActionServlet` 根据请求决定是否需要调用业务逻辑控制器处理用户请求（实际上，业务逻辑控制器还是控制器，它只是负责调用模型来处理用户请求），当用户请求处理完成后，其处理结果通过 JSP 呈现给用户。

对于整个 Struts 1 框架而言，控制器就是它的核心，Struts 1 的控制器由两个部分组成：核心控制器和业务逻辑控制器。其中核心控制器就是 `ActionServlet`，由 Struts 1 框架提供；业务逻辑控制就是用户自定义的 `Action`，由应用开发者提供。

对于大部分用户请求而言，都需要得到服务器的处理。当用户发送一个需要得到服务器处理的请求时，该请求被 `ActionServlet` 拦截到，`ActionServlet` 将该请求转发给对应的业务逻辑控制器，业务逻辑控制器调用模型来处理用户请求；如果用户请求只是希望得到某个 URL 资源，则由 `ActionServlet` 将被请求的资源转发给用户。

Struts 1 的程序运行流程如图 1.7 所示。

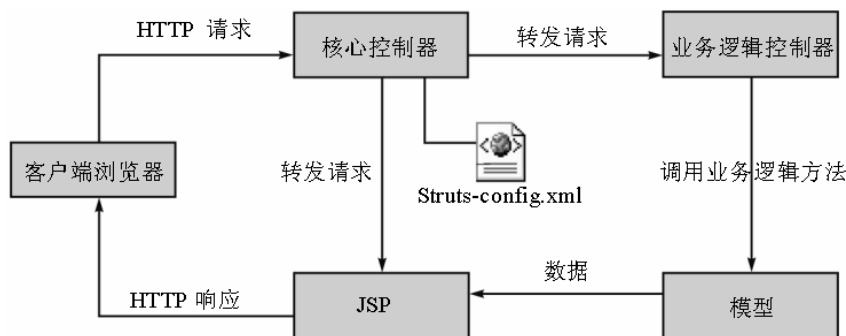


图 1.7 Struts 1 的程序运行流程

下面就 Struts 1 程序流程具体分析 MVC 中的三个角色。

（1）Model 部分

Struts 1 的 Model 部分主要由底层的业务逻辑组件充当，这些业务逻辑组件封装了底层数据库访问、业务逻辑方法实现。实际上，对于一个成熟的企业应用而言，Model 部分也不是一个简单的 `JavaBean` 所能完成的，它可能是一个或多个 `EJB` 组件，可能是一个 `WebService` 服务。总之，Model 部分封装了整个应用的所有业务逻辑，但整个部分并不是由 Struts 1 提供的，Struts 1 也没有为实现 Model 组件提供任何支持。

（2）View 部分

Struts 1 的 View 部分采用 JSP 实现。Struts 1 提供了丰富的标签库，通过这些标签库可以最大限度地减少脚本的使用。这些自定义的标签库可以输出控制器的处理结果。

虽然 Struts 1 提供了与 `Ties` 框架的整合，但 Struts 1 所支持的表现层技术非常单一：既不支持 `FreeMarker`、`Velocity` 等模板技术，也不支持 `JasperReports` 等报表技术。

（3）Controller 部分


Struts 1 的 Controller 由两个部分组成。

- 系统核心控制器：由 Struts 1 框架提供，就是系统中的 `ActionServlet`。
- 业务逻辑控制器：由 Struts 1 框架提供，就是用户自己实现的 `Action` 实例。

Struts 1 的核心控制器对应图 1.7 中的核心控制器（ActionServlet）。该控制器由 Struts 1 框架提供，继承 `HttpServlet` 类，因此可以配置成一个标准的 `Servlet`，该控制器负责拦截所有 HTTP 请求，然后根据用户请求决定是否调用业务逻辑控制器，如果需要调用业务逻辑控制器，则将请求转发给 `Action` 处理，否则直接转向请求的 JSP 页面。

业务逻辑控制器负责处理用户请求，但业务逻辑控制器本身并不具有处理能力，而是调用 `Model` 来完成处理。

Struts 1 提供了系统所需要的核心控制器，也为实现业务逻辑控制器提供了许多支持。因此，控制器部分就是 Struts 1 框架的核心。有时候，我们直接将 MVC 层称为控制器层。

 提示 对于任何的 MVC 框架而言，其实只实现了 C（控制器）部分，但它负责用控制器调用业务逻辑组件，并负责控制器与视图技术（JSP、FreeMarker 和 Velocity 等）的整合。


对于 Struts 1 框架而言，因为它与 JSP/Servlet 耦合非常紧密，因而导致了許多不可避免的缺陷，随着 Web 应用的逐渐扩大，这些缺陷逐渐变成制约 Struts 1 发展的重要因素——这也是 Struts 2 出现的原因。下面具体分析 Struts 1 中存在的种种缺陷。

（1）支持的表现层技术单一

Struts 1 只支持 JSP 作为表现层技术，不提供与其他表现层技术，例如 Velocity、FreeMarker 等技术的整合。这一点严重制约了 Struts 1 框架的使用，对于目前的很多 Java EE 应用而言，并不一定使用 JSP 作为表现层技术。

虽然 Struts 1 处理完用户请求后，并没有直接转到特定的视图资源，而是返回一个 `ActionForward` 对象（可以理解 `ActionForward` 是一个逻辑视图名），在 `struts-config.xml` 文件中定义了逻辑视图名和视图资源之间的对应关系，当 `ActionServlet` 得到处理器返回的 `ActionForward` 对象后，可以根据逻辑视图名和视图资源之间的对应关系，将视图资源呈现给用户。

从上面的设计来看，不得不佩服 Struts 1 的设计者高度解耦的设计：控制器并没有直接执行转发请求，而仅仅返回一个逻辑视图名——实际的转发放在配置文件中进行管理。但因为 Struts 1 框架出现的年代太早了，那时候还没有 FreeMarker、Velocity 等技术，因而没有考虑与这些 FreeMarker、Velocity 等视图技术的整合。

 提示 Struts 1 已经通过配置文件管理逻辑视图名和实际视图之间的对应关系，只是没有做到让逻辑视图名可以支持更多的视图技术。

虽然 Struts 1 有非常优秀的设计，但由于历史原因，它没有提供与更多视图技术的整合，这严重限制了 Struts 1 的使用。

（2）与 Servlet API 严重耦合，难于测试

因为 Struts 1 框架是在 Model 2 的基础上发展起来的，因此它完全是基于 Servlet API 的，所以在 Struts 1 的业务逻辑控制器内，充满了大量的 Servlet API。

看下面的 Action 代码片段：

```
//业务逻辑控制器必须继承 Struts 1 提供的 Action 类
public class LoginAction extends Action
{
    //处理用户请求的 execute 方法
    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) throws
        AuctionException
    {
        //获取封装用户请求参数的 ActionForm 对象
        //将其强制类型转换为登录用的 ActionForm
        LoginForm loginForm = (LoginForm) form;
        //当用户名为 scott, 密码为 tiger 时返回成功
        if ("scott".equals(loginForm.getUsername())
            && "tiger".equals(loginForm.getPassword()))
        {
            //处理成功, 返回一个 ActionForward 对象
            return mapping.findForward("success");
        }
        else
        {
            //处理失败, 返回一个 ActionForward 对象
            return mapping.findForward("success");
        }
    }
}
```

当我们需要测试上面 Action 类的 execute 方法时, 该方法有 4 个参数: ActionMapping、ActionForm、HttpServletRequest 和 HttpServletResponse, 初始化这 4 个参数比较困难, 尤其是 HttpServletRequest 和 HttpServletResponse 两个参数, 通常由 Web 容器负责实例化。

因为 HttpServletRequest 和 HttpServletResponse 两个参数是 Servlet API, 严重依赖于 Web 服务器。因此, 一旦脱离了 Web 服务器, Action 的测试非常困难。

(3) 代码严重依赖于 Struts 1 API, 属于侵入式设计

正如从上面代码片段中所看到的, Struts 1 的 Action 类必须继承 Struts 1 的 Action 基类, 实现处理方法时, 又包含了大量 Struts 1 API: 如 ActionMapping、ActionForm 和 ActionForward 类。这种侵入式设计的最大弱点在于, 一旦系统需要重构时, 这些 Action 类将完全没有利用价值, 成为一堆废品。

可见, Struts 1 的 Action 类这种侵入式设计导致了较低的代码复用。

1.2.2 WebWork 简介

WebWork 虽然没有 Struts 1 那样赫赫有名, 但也是出身名门, WebWork 来自另外一个优秀的开源组织: opensymphony, 这个优秀的开源组织同样开发了大量优秀的开源项目, 如 Qutarz、OSWorkFlow 等。实际上, WebWork 的创始人则是另一个 Java 领域的名人: Rickard Oberg (他就是 JBoss 和 XDoclet 的作者)。

相对于 Struts 1 存在的那些先天性不足而言, WebWork 则更加优秀, 它采用了一种更加松耦合的设计, 让系统的 Action 不再与 Servlet API 耦合。使单元测试更加方便, 允许

系统从 B/S 结构向 C/S 结构转换。

相对于 Struts 1 仅支持 JSP 表现层技术的缺陷而言, WebWork 支持更多的表现层技术, 如 Velocity、FreeMarker 和 XSLT 等。

WebWork 可以脱离 Web 应用使用, 这一点似乎并没有太多优势, 因为, 一个应用通常开始已经确定在怎样的环境下使用。WebWork 有自己的控制反转 (Inversion of Control) 容器, 通过控制反转, 可以让测试变得更简单, 测试中设置实现服务接口的 Mock 对象完成测试, 而不需要设置服务注册。

WebWork 2 使用 OGNL 这个强大的表达式语言, 可以访问值栈。OGNL 对集合和索引属性的支持非常强大。

WebWork 建立在 XWork 之上, 使用 ServletDispatcher 作为该框架的核心控制器, 处理 HTTP 的响应和请求。

从处理流程上来看, WebWork 与 Struts 1 非常类似, 它们的核心都由控制器组成, 其中控制器都由两个部分组成:

- 核心控制器 ServletDispatcher, 该控制器框架提供。
- 业务逻辑控制器 Action, 该控制器由程序员提供。

相对 Struts 1 的 Action 与 Servlet API 紧紧耦合的弱点来说, WebWork 的 Action 则完全与 Servlet API 分离, 因而该 Action 更容易测试。

WebWork 的 Action 可以与 Servlet API 分离, 得益于它灵巧的设计, 它使用一个拦截器链, 负责将用户请求数据转发到 Action, 并负责将 Action 的处理结果转换成对用户的响应。

当用户向 Web 应用发送请求时, 该请求经过 ActionContextCleanUp、SiteMesh 等过滤器过滤, 由 WebWork 的核心控制器拦截, 如果用户请求需要 WebWork 的业务逻辑控制器处理, 该控制器则调用 Action 映射器, 该映射器将用户请求转发到对应的业务逻辑控制器。值得注意的是, 此时的业务逻辑控制器并不是开发者实现的控制器, 而是 WebWork 创建的控制器代理。

创建控制器代理时, WebWork 需要得到开发者定义的 xwork.xml 配置文件, 控制器代理以用户实现的控制器作为目标, 以拦截器链中的拦截器作为处理 (Advice)。



提示 WebWork 中创建控制器代理的方式, 就是一种 AOP (面向切面编程) 编程方式, 只是这种 AOP 中的拦截器由系统提供, 因此无需用户参与。如果读者需要获取更多关于 AOP 编程的知识, 请参阅 AOP 相关资料, 或笔者所著的《Spring 2.0 宝典》一书的第 6 章。

开发者自己实现的业务逻辑控制器只是 WebWork 业务控制器的目标——这就是为什么开发者自己实现的 Action 可以与 Servlet API 分离的原因。当开发者自己的 Action 处理完 HTTP 请求后, 该结果只是一个普通字符串, 该字符串将对应到指定的视图资源。

指定的视图资源经过拦截器链的处理后, 生成对客户端的响应输出。

上面整个过程的数据流图如图 1.8 所示。

与前面的 Struts 1 框架对比, 不难发现 WebWork 在很多地方确实更优秀。相对 Struts 1 的种种缺点而言, WebWork 存在如下优点:

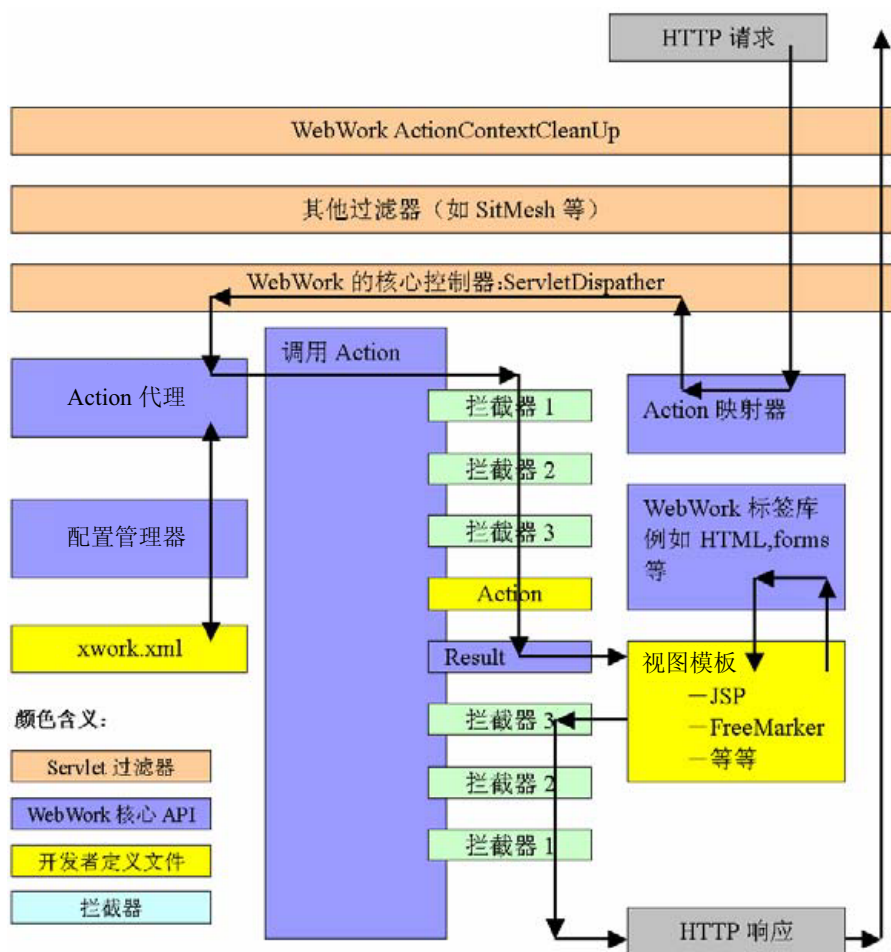


图 1.8 WebWork 的数据流图

(1) Action 无需与 Servlet API 耦合，更容易测试

相对于 Struts 1 框架中的 Action 出现了大量 Servlet API 而言，WebWork 的 Action 更像一个普通 Java 对象，该控制器代码中没有耦合任何 Servlet API。看下面的 WebWork 的 Action 示例：

```
public class LoginAction implements Action
{
    //该字符串常量将作为 Action 的返回值
    private final static String LOGINFAIL="loginfail";
    //该 Action 封装的两个请求参数
    private String password;
    private String username;
    //password 请求参数对应的 getter 方法
    public String getPassword()
    {
        return password;
    }
    //password 请求参数对应的 setter 方法
    public void setPassword(String password)
    {
        this.password = password;
    }
    //username 请求参数对应的 getter 方法
```



```
public String getUsername()
{
    return username;
}
//username 请求参数对应的 setter 方法
public void setUsername(String username)
{
    this.username = username;
}
//处理用户请求的 execute 方法
public String execute() throws Exception
{
    if ("yeeku".equalsIgnoreCase(getUsername())
        && "password".equals(getPassword()))
    {
        ActionContext ctx = ActionContext.getContext();
        //将当前登录的用户名保存到 Session
        Map session = ctx.getSession();
        session.put("username", getUsername());
        return SUCCESS;
    }
    else
    {
        return LOGINFAIL;
    }
}
}
```

在上面的 Action 代码中，我们看不到任何的 Servlet API，当系统需要处理两个请求参数：username 和 password 时，Action 并未通过 HttpServletRequest 对象来获得请求参数，而是直接调用访问该 Action 的 username 和 password 成员属性——这两个属性由 Action 拦截器负责初始化，以用户请求参数为其赋值。

即使 Action 中需要访问 HTTP Session 对象，依然没有在代码中直接出现 HttpSession API，而是以一个 Map 对象代表了 HTTP Session 对象。

当我们将 WebWork 的 Action 和 Struts 1 的 Action 进行对比时，不难发现 Struts 1 的 Action 确实太臃肿了，确实不如 WebWork 的 Action 那么优雅。

如果需要测试上面的 Action 代码，测试用例的书写将非常容易，因为 execute 方法中没有包含任何 Servlet API，甚至没有 WebWork 的 API。

(2) Action 无需与 WebWork 耦合，代码重用率高

在上面的 Action 代码中，不难发现 WebWork 中的 Action 其实就是一个 POJO，该 Action 仅仅实现了 WebWork 的 Action 接口，包含了一个 execute 方法。

Struts 1 中的 Action 类需要继承 Struts 1 的 Action 类。我们知道，实现一个接口和继承一个类是完全不同的概念：实现一个接口对类的污染要小得多，该类也可以实现其他任意接口，还可以继承一个父类；但一旦已经继承一个父类，则意味着该类不能再继承其他父类。

除此之外，Struts 1 中 Action 也包含了一个 execute 方法，但该方法需要 4 个参数，类型分别是 ActionMapping、ActionForm、HttpServletRequest 和 HttpServletResponse，一个包

含了这 4 个参数的方法，除了在 Struts 1 框架下有用外，笔者难以想象出该代码还有任何复用价值。但 WebWork 的 `execute` 方法则完全不同，该方法中没有出现任何 Servlet API，也没有出现任何 WebWork API，这个方法在任何环境下都有重用的价值。

得益于 WebWork 灵巧的设计，WebWork 中的 Action 无需与任何 Servlet API、WebWork API 耦合，从而具有更好的代码重用率。

(3) 支持更多的表现层技术，有更好的适应性

正如从图 1.8 所见到的，WebWork 对多种表现层技术：JSP、Velocity 和 FreeMarker 等都有很好的支持，从而给开发更多的选择，提供了更好的适应性。

1.2.3 Struts 2 起源

经过五年多的发展，Struts 1 已经成为一个高度成熟的框架，不管是稳定性还是可靠性，都得到了广泛的证明。但由于它太“老”了，一些设计上的缺陷成为它的硬伤。面对大量新的 MVC 框架蓬勃兴起，Struts 1 也开始了血液的更新。

目前，Struts 已经分化成两个框架：第一个框架就是传统 Struts 1 和 WebWork 结合后的 Struts 2 框架。Struts 2 虽然是在 Struts 1 的基础上发展起来的，但实质上是以 WebWork 为核心，Struts 2 为传统 Struts 1 注入了 WebWork 的设计理念，统一了 Struts 1 和 WebWork 两个框架，允许 Struts 1 和 WebWork 开发者同时使用 Struts 2 框架。

Struts 分化出来的另外一个框架是 Shale，这个框架远远超出了 Struts 1 原有的设计思想，它与原有的 Struts 1 的关联很少，它使用全新的设计思想。Shale 更像一个新的框架，而不是 Struts 的升级。Shale 在很多方面与 Struts 存在不同之处，其中有两点最为突出：

- Struts 与 JSF 集成，而 Shale 则是建立在 JSF 之上。
- Struts 实质上是一个巨大的、复杂的请求处理器；而 Shale 则是一组能以任何方式进行组合的服务，简单地说，Shale 是一种 SOA（面向服务架构）架构。

在后面的介绍中，我们会发现，Struts 2 非常类似于 WebWork 框架，而不像 Struts 1 框架，因为 Struts 2 是以 WebWork 为核心，而不是以 Struts 1 为核心的。正因为此，许多 WebWork 开发者会发现，从 WebWork 过渡到 Struts 2 是一件非常简单的事情。

当然，对于传统的 Struts 1 开发者，Struts 2 也提供了很好的向后兼容性，Struts 2 可与 Struts 1 有机整合，从而保证 Struts 1 开发者能平稳过渡到 Struts 2。

1.3 Struts 2 体系介绍

Struts 2 的体系与 Struts 1 体系的差别非常大，因为 Struts 2 使用了 WebWork 的设计核心，而不是使用 Struts 1 的设计核心。Struts 2 大量使用拦截器来处理用户请求，从而允许用户的业务逻辑控制器与 Servlet API 分离。

1.3.1 Struts 2 框架架构

从数据流图上来看，Struts 2 与 WebWork 相差不大，Struts 2 同样使用拦截器作为处理 (Advice)，以用户的业务逻辑控制器为目标，创建一个控制器代理。

控制器代理负责处理用户请求，处理用户请求时回调业务控制器的 `execute` 方法，该

方法的返回值将决定了 Struts 2 将怎样的视图资源呈现给用户。

图 1.9 显示了 Struts 2 的体系概图。

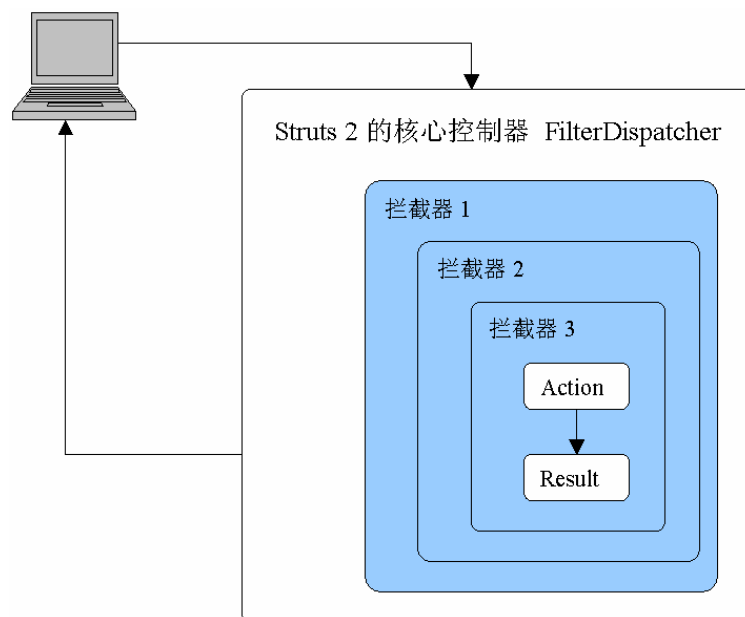


图 1.9 Struts 2 的体系概图

Struts 2 框架的大致处理流程如下：

- ① 浏览器发送请求，例如请求/mypage.action、/reports/myreport.pdf 等。
- ② 核心控制器 FilterDispatcher 根据请求决定调用合适的 Action。
- ③ WebWork 的拦截器链自动对请求应用通用功能，例如 workflow、validation 或文件上传等功能。
- ④ 回调 Action 的 execute 方法，该 execute 方法先获取用户请求参数，然后执行某种数据库操作，既可以是将数据保存到数据库，也可以从数据库中检索信息。实际上，因为 Action 只是一个控制器，它会调用业务逻辑组件来处理用户的请求。
- ⑤ Action 的 execute 方法处理结果信息将被输出到浏览器中，可以是 HTML 页面、图像，也可以是 PDF 文档或者其他文档。此时支持的视图技术非常多，既支持 JSP，也支持 Velocity、FreeMarker 等模板技术。

1.3.2 Struts 2 的配置文件

当 Struts 2 创建系统的 Action 代理时，需要使用 Struts 2 的配置文件。

Struts 2 的配置文件有两份：

- 配置 Action 的 struts.xml 文件。
- 配置 Struts 2 全局属性的 struts.properties 文件。

struts.xml 文件内定义了 Struts 2 的系列 Action，定义 Action 时，指定该 Action 的实现类，并定义该 Action 处理结果与视图资源之间的映射关系。

下面是 struts.xml 配置文件的示例：

```
<struts>
  <!-- Struts 2 的 Action 都必须配置在 package 里 -->
```

```
<package name="default" extends="struts-default">
  <!-- 定义一个 Logon 的 Action, 实现类为 lee.Logon -->
  <action name="Logon" class="lee.Logon">
    <!-- 配置 Action 返回 input 时转入/pages/Logon.jsp 页面 -->
    <result name="input">/pages/Logon.jsp</result>
    <!-- 配置 Action 返回 cancel 时重定向到 Welcome 的 Action-->
    <result name="cancel" type="redirect-action">Welcome</result>
    <!-- 配置 Action 返回 success 时重定向到 MainMenu 的 Action -->
    <result type="redirect-action">MainMenu</result>
    <!-- 配置 Action 返回 expired 时进入 ChangePassword 的 Action 链 -->
    <result name="expired" type="chain">ChangePassword</result>
  </action>
  <!-- 定义 Logoff 的 Action, 实现类为 lee.Logoff -->
  <action name="Logoff" class=" lee.Logoff">
    <!-- 配置 Action 返回 success 时重定向到 MainMenu 的 Action -->
    <result type="redirect-action">Welcome</result>
  </action>
</package>
</struts>
```

在上面的 struts.xml 文件中, 定义了两个 Action。定义 Action 时, 不仅定义了 Action 的实现类, 而且在定义 Action 的处理结果时, 指定了多个 result, result 元素指定 execute 方法返回值和视图资源之间的映射关系。对于如下配置片段:

```
<result name="cancel" type="redirect-action">Welcome</result>
```

表示当 execute 方法返回 cancel 的字符串时, 跳转到 Welcome 的 Action。定义 result 元素时, 可以指定两个属性: type 和 name。其中 name 指定了 execute 方法返回的字符串, 而 type 指定转向的资源类型, 此处转向的资源可以是 JSP, 也可以是 FreeMarker 等, 甚至是另一个 Action——这也是 Struts 2 可以支持多种视图技术的原因。

除此之外, Struts 2 还有一个配置 Struts 2 全局属性的 Properties 文件: struts.properties。该文件的示例如下:

```
#指定 Struts 2 处于开发状态
struts.devMode = false
//指定当 Struts 2 配置文件改变后, Web 框架是否重新加载 Struts 2 配置文件
struts.configuration.xml.reload=true
```

正如上面见到的, struts.properties 文件的形式是系列的 key、value 对, 它指定了 Struts 2 应用的全局属性。

1.3.3 Struts 2 的标签库

Struts 2 的标签库也是 Struts 2 的重要组成部分, Struts 2 的标签库提供了非常丰富的功能, 这些标签库不仅提供了表现层数据处理, 而且提供了基本的流程控制功能, 还提供了国际化、Ajax 支持等功能。

通过使用 Struts 2 的标签, 开发者可以最大限度地减少页面代码的书写。

看下面的 JSP 页面的表单定义片段:

```
<!-- 定义一个 Action -->
<form method="post" action="basicvalid.action">
  <!-- 下面定义三个表单域 -->
  名字: <input type="text" name="name"/><br>
  年纪: <input type="text" name="age"/><br>
  喜欢的颜色: <input type="text" name="favorite"/><br>
  <!-- 定义一个输出按钮 -->
  <input type="submit" value="提交"/>
</form>
```

上面页面使用了传统的 HTML 标签定义表单元素，还不具备输出校验信息的功能，但如果换成如下使用 Struts 2 标签的定义方式：

```
<!-- 使用 Struts 2 标签定义一个表单 -->
<s:form method="post" action="basicvalid.action">
  <!-- 下面使用 Struts 2 标签定义三个表单域 -->
  <s:textfield label="名字" name="name"/>
  <s:textfield label="年纪" name="age"/>
  <s:textfield label="喜欢的颜色" name="answer"/>
  <!-- 定义一个提交按钮 -->
  <s:submit/>
</s:form>
```

则页面代码更加简洁，而且有更简单的错误输出。图 1.10 是上面使用 Struts 2 标签执行数据校验后的输出。

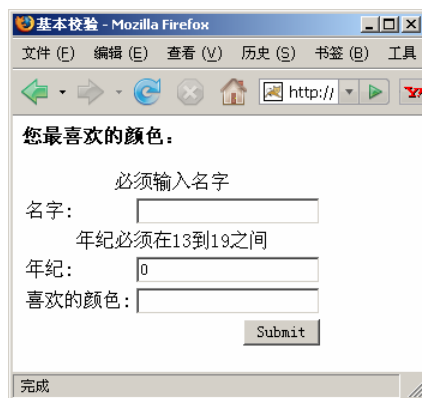


图 1.10 使用 Struts 2 标签的效果

提示 Struts 2 的标签库的功能非常复杂，该标签库几乎可以完全替代 JSTL 的标签库。而且 Struts 2 的标签支持表达式语言，这种表达式语言支持一个强大和灵活的表达式语言：OGNL（Object Graph Notation Language），因此功能非常强大。

1.3.4 Struts 2 的控制器组件

Struts 2 的控制器组件是 Struts 2 框架的核心，事实上，所有 MVC 框架都是以控制器组件为核心的。正如前面提到的，Struts 2 的控制器由两个部分组成：FilterDispatcher 和业务控制器 Action。

实际上, Struts 2 应用中起作用的业务控制器不是用户定义的 Action, 而是系统生成的 Action 代理, 但该 Action 代理以用户定义的 Action 为目标。

下面是 Struts 2 的 Action 代码示例:

```
public class LoginAction
{
    //封装用户请求参数的 username 属性
    private String username;
    //封装用户请求参数的 password 属性
    private String password;
    //username 属性的 getter 方法
    public String getUsername()
    {
        return username;
    }
    //username 属性的 setter 方法
    public void setUsername(String username)
    {
        this.username = username;
    }
    //password 属性的 getter 方法
    public String getPassword()
    {
        return password;
    }
    //password 属性的 setter 方法
    public void setPassword(String password)
    {
        this.password = password;
    }
    //处理用户请求的 execute 方法
    public String execute() throws Exception
    {
        //如果用户名为 scott, 密码为 tiger, 则登录成功
        if (getUsername().equals("scott")
            && getPassword().equals("tiger") )
        {
            return "success";
        }
        else
        {
            return "error";
        }
    }
}
```

通过查看上面的 Action 代码, 发现该 Action 比 WebWork 中的 Action 更彻底, 该 Action 无需实现任何父接口, 无需继承任何 Struts 2 基类, 该 Action 类完全是一个 POJO (普通、传统的 Java 对象), 因此具有很好的复用性。

归纳起来, 该 Action 类有如下优势:

- Action 类完全是一个 POJO，因此具有很好的代码复用性。
- Action 类无需与 Servlet API 耦合，因此进行单元测试非常简单。
- Action 类的 `execute` 方法仅返回一个字符串作为处理结果，该处理结果可映射到任何的视图，甚至是另一个 Action。

1.4 Struts 2 与 Struts 1 的对比

经过上面简要介绍，不难发现，Struts 2 确实在 Struts 1 上做出了巨大的改进，的确是一个非常具有实用价值的 MVC 框架。下面是 Struts 1 和 Struts 2 在各方面的简要对比。

- 在 Action 实现类方面的对比：Struts 1 要求 Action 类继承一个抽象基类；Struts 1 的一个具体问题是使用抽象类编程而不是接口。Struts 2 Action 类可以实现一个 Action 接口，也可以实现其他接口，使可选和定制的服务成为可能。Struts 2 提供一个 `ActionSupport` 基类去实现常用的接口。即使 Action 接口不是必须实现的，只有一个包含 `execute` 方法的 POJO 类都可以用作 Struts 2 的 Action。
- 线程模式方面的对比：Struts 1 Action 是单例模式并且必须是线程安全的，因为仅有 Action 的一个实例来处理所有的请求。单例策略限制了 Struts 1 Action 能做的事，并且要在开发时特别小心。Action 资源必须是线程安全的或同步的；Struts 2 Action 对象为每一个请求产生一个实例，因此没有线程安全问题。
- Servlet 依赖方面的对比：Struts 1 Action 依赖于 Servlet API，因为 Struts 1 Action 的 `execute` 方法中有 `HttpServletRequest` 和 `HttpServletResponse` 方法。Struts 2 Action 不再依赖于 Servlet API，从而允许 Action 脱离 Web 容器运行，从而降低了测试 Action 的难度。当然，如果 Action 需要直接访问 `HttpServletRequest` 和 `HttpServletResponse` 参数，Struts 2 Action 仍然可以访问它们。但是，大部分时候，Action 都无需直接访问 `HttpServletRequest` 和 `HttpServletResponse`，从而给开发者更多灵活的选择。
- 可测性方面的对比：测试 Struts 1 Action 的一个主要问题是 `execute` 方法依赖于 Servlet API，这使得 Action 的测试要依赖于 Web 容器。为了脱离 Web 容器测试 Struts 1 的 Action，必须借助于第三方扩展：Struts TestCase，该扩展下包含了系列的 Mock 对象（模拟了 `HttpServletRequest` 和 `HttpServletResponse` 对象），从而可以脱离 Web 容器测试 Struts 1 的 Action 类。Struts 2 Action 可以通过初始化、设置属性、调用方法来测试。
- 封装请求参数的对比：Struts 1 使用 `ActionForm` 对象封装用户的请求参数，所有的 `ActionForm` 必须继承一个基类：`ActionForm`。普通的 `JavaBean` 不能用作 `ActionForm`，因此，开发者必须创建大量的 `ActionForm` 类封装用户请求参数。虽然 Struts 1 提供了动态 `ActionForm` 来简化 `ActionForm` 的开发，但依然需要在配置文件中定义 `ActionForm`；Struts 2 直接使用 Action 属性来封装用户请求属性，避免了开发者需要大量开发 `ActionForm` 类的烦琐，实际上，这些属性还可以是包含子属性的 `Rich` 对象类型。如果开发者依然怀念 Struts 1 `ActionForm` 的模式，Struts 2 提供了 `ModelDriven` 模式，可以让开发者使用单独的 `Model` 对象来封装用户请求参数，但该 `Model` 对象无需继承任何 Struts 2 基类，是一个 POJO，从而降低了代码污染。
- 表达式语言方面的对比：Struts 1 整合了 JSTL，因此可以使用 JSTL 表达式语言。

这种表达式语言有基本对象图遍历，但在对集合和索引属性的支持上则功能不强；Struts 2 可以使用 JSTL，但它整合了一种更强大和灵活的表达式语言：OGNL(Object Graph Notation Language)，因此，Struts 2 下的表达式语言功能更加强大。

- 绑定值到视图的对比：Struts 1 使用标准 JSP 机制把对象绑定到视图页面；Struts 2 使用“ValueStack”技术，使标签库能够访问值，而不需要把对象和视图页面绑定在一起。
- 类型转换的对比：Struts 1 ActionForm 属性通常都是 String 类型。Struts 1 使用 Commons-Beanutils 进行类型转换，每个类一个转换器，转换器是不可配置的；Struts 2 使用 OGNL 进行类型转换，支持基本数据类型和常用对象之间的转换。
- 数据校验的对比：Struts 1 支持在 ActionForm 重写 validate 方法中手动校验，或者通过整合 Commons alidator 框架来完成数据校验。Struts 2 支持通过重写 validate 方法进行校验，也支持整合 XWork 校验框架进行校验。
- Action 执行控制的对比：Struts 1 支持每一个模块对应一个请求处理（即生命周期的概念），但是模块中的所有 Action 必须共享相同的生命周期。Struts 2 支持通过拦截器堆栈（Interceptor Stacks）为每一个 Action 创建不同的生命周期。开发者可以根据需要创建相应堆栈，从而和不同的 Action 一起使用。

1.5 WebWork 和 Struts 2 对比

从某种程度上来看，Struts 2 是 WebWork 的升级，而不是 Struts 1 的升级，甚至在 Apache 的 Struts 2 的官方文档都提到：WebWork 到 Struts 2 是一次平滑的过渡。实际上，Struts 2.0 其实是 WebWork 2.3 而已，从 WebWork 2.2 迁移到 Struts 2.0 不会比从 WebWork 2.1 到 2.2 更麻烦。

在很多方面，Struts 2 仅仅是改变了 WebWork 下的名称，因此，如果开发者具有 WebWork 的开发经验，将更加迅速地进入 Struts 2 的开发领域。

下面是 Struts 2 与 WebWork 命名上存在改变（见表 1.1）：

表 1.1 Struts 2 和 WebWork 成员名称的对应

Struts 2 成员	WebWork 成员
com.opensymphony.xwork2.*	com.opensymphony.xwork.*
org.apache.Struts2.*	com.opensymphony.webwork.*
struts.xml	xwork.xml
struts.properties	webwork.properties
Dispatcher	DispatcherUtil
org.apache.Struts2.config.Settings	com.opensymphony.webwork.config.Configuration

除此之外，Struts 2 也删除了 WebWork 中少量特性：

- AroundInterceptor: Struts 2 不再支持 WebWork 中的 AroundInterceptor。如果应用程序中需要使用 AroundInterceptor，则应该自己手动导入 WebWork 中的 AroundInterceptor 类。
- 富文本编辑器标签：Struts 2 不再支持 WebWork 的富文本编辑器，如果应用中需要

使用富文本编辑器，则应该使用 Dojo 的富文本编辑器。

- **IoC 容器支持：**Struts 2 不再支持内建的 IoC 容器，而改为全面支持 Spring 的 IoC 容器，以 Spring 的 IoC 容器作为默认的 Object 工厂。

1.6 本章小结

本章大致介绍了 Web 应用的开发历史，从历史的角度介绍了 Model 1 和 Model 2 的简要模型和特征，进而介绍了 MVC 模式的主要策略和主要优势。接着介绍了常用的 MVC 框架，包括 JSF、Tapestry 和 Spring MVC，以及这些框架的基本知识和相关特征。本章重点介绍了 Struts 2 的两个前身：Struts 1 和 WebWork，以及这两个框架的架构和主要特征，从而引申出对 Struts 2 起源的介绍。最后大致介绍了 Struts 2 框架的体系，包括 Struts 2 框架的架构、标签库、控制器组件等，并就 Struts 1 和 Struts 2 的相关方面进行了比较。

下一章将以一个简单的 HelloWorld 应用为示例，来介绍基于 Struts 2 的应用，让读者感受基于 Struts 2 的 MVC 应用。

Struts 2 下的 HelloWorld

本章要点

- Struts 2 的下载和安装
- 纯手工创建一个 Web 应用
- 纯手工创建一个 Struts 2 应用
- 实现 Struts 2 的 Action
- 配置 Struts 2 的 Action
- 在 Action 中访问 HttpSession
- 在 JSP 中输出 Action 的返回值
- 使用 Struts 2 的表单标签
- 程序国际化初步
- 数据校验初步

前面已经简要介绍了 Struts 2 的起源，以及 Struts 2 的两个前身：Struts 1 和 WebWork，并详细对比了 Struts 2 和 Struts 1 的差异，对比了 Struts 2 和 WebWork 的差异，而且指出：Struts 2 是 WebWork 的升级，而不是 Struts 1 的升级。

虽然 Struts 2 提供了与 Struts 1 的兼容，但已经不是 Struts 1 的升级。对于已有 Struts 1 开发经验的开发者而言，Struts 1 的开发经验对于 Struts 2 并没有太大的帮助；相反，对于已经有 WebWork 开发经验的开发者而言，WebWork 的开发经验对 Struts 2 的开发将有很好的借鉴意义。

下面将以一个 Struts 2 的 HelloWorld 应用为例，介绍 Struts 2 MVC 框架如何拦截用户请求，如何调用业务控制器处理用户请求，并介绍 Action 处理结果和资源之间的映射关系。

本 HelloWorld 应用是一个简单的应用：用户进入一个登录页面，允许用户输入用户名、密码，如果用户输入的用户名和密码符合要求，则进入一个欢迎页面；如果用户输入错误，则进入一个提示页面。当用户提交表单时，本应用会有基本的数据校验。

2.1 下载和安装 Struts 2 框架

下面我们从下载、安装 Struts 2 开始，慢慢开始体验 Struts 2 MVC 框架的魅力。

笔者写本书的时候，Struts 2 已经发布了其产品化 GA（General Availability）版，其实最新的产品化 GA 版是 Struts 2.06，故本书的所有应用都是基于该版本的 Struts 2。建议读

者下载 Struts 2.06 版，而不是下载最新的 Beta 版，如果 Struts 2 有最新的 GA 版，读者也可以下载更新的 GA 版，相信不会有太大差异。

下载和安装 DWR 请按如下步骤进行。

① 登录 <http://struts.apache.org/download.cgi#Struts206> 站点，下载 Struts 2 的最新 GA 版。在 Struts 2.06 下有如下几个选项：

- **Full Distribution:** 下载 Struts 2 的完整版。通常建议下载该选项。
- **Example Applications:** 下载 Struts 2 的示例应用，这些示例应用对于学习 Struts 2 有很大的帮助，下载 Struts 2 的完整版时已经包含了该选项下全部应用。
- **Blank Application only:** 仅下载 Struts 2 的空示例应用，这个空应用已经包含在 Example Applications 选项下。
- **Essential Dependencies:** 仅仅下载 Struts 2 的核心库，下载 Struts 2 的完整版时将包括该选项下的全部内容。
- **Documentation:** 仅仅下载 Struts 2 的相关文档，包含 Struts 2 的使用文档、参考手册和 API 文档等。下载 Struts 2 的完整版时将包括该选项下的全部内容。
- **Source:** 下载 Struts 2 的全部源代码，下载 Struts 2 的完整版时将包括该选项下的全部内容。
- **Alternative Java 4 JARs:** 下载可选的 JDK 1.4 的支持 JAR。下载 Struts 2 的完整版时将包括该选项下的全部内容。

通常建议读者下载第一个选项：下载 Struts 2 的完整版，将下载到的 Zip 文件解压缩，该文件就是一个典型的 Web 结构，该文件夹包含如下文件结构：

- **apps:** 该文件夹下包含了基于 Struts 2 的示例应用，这些示例应用对于学习者是非常有用的资料。
- **docs:** 该文件夹下包含了 Struts 2 的相关文档，包括 Struts 2 的快速入门、Struts 2 的文档，以及 API 文档等内容。
- **j4:** 该文件夹下包含了让 Struts 2 支持 JDK 1.4 的 JAR 文件。
- **lib:** 该文件夹下包含了 Struts 2 框架的核心类库，以及 Struts 2 的第三方插件类库。
- **src:** 该文件夹下包含了 Struts 2 框架的全部源代码。

② 将 lib 文件夹下的 Struts2-core-2.0.6.jar、xwork-2.0.1.jar 和 ognl-2.6.11.jar 等必需类库复制到 Web 应用的 WEB-INF/lib 路径下。当然，如果你的 Web 应用需要使用 Struts 2 的更多特性，则需要将更多的 JAR 文件复制到 Web 应用的 WEB-INF/lib 路径下。如果需要在 DOS 或者 Shell 窗口下手动编译 Struts 2 相关的程序，则还应该将 Struts2-core-2.0.6.jar 和 xwork-2.0.1.jar 添加到系统的 CLASSPATH 环境变量里。



提示 大部分时候，使用 Struts 2 的 Web 应用并不需要利用到 Struts 2 的全部特性，因此没有必要一次将该 lib 路径下 JAR 文件全部复制到 Web 应用的 WEB-INF/lib 路径下。



编辑 Web 应用的 web.xml 配置文件，配置 Struts 2 的核心 Filter。下面是增加了 Struts 2 的核心 Filter 配置的 web.xml 配置文件的代码：

```
<?xml version="1.0" encoding="GBK"?>
<!-- web-app 是 Web 应用配置文件的根元素，指定 Web 应用的 Schema 信息 -->
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.
    com/xml/ns/j2ee/web-app_2_4.xsd">
  <!-- 定义 Struts 2 的 FilterDispatcher 的 Filter -->
  <filter>
    <!-- 定义核心 Filter 的名字 -->
    <filter-name>struts2</filter-name>
    <!-- 定义核心 Filter 的实现类 -->
    <filter-class>org.apache.Struts2.dispatcher.FilterDispatcher
      </ filter-class>
  </filter>
  <!-- FilterDispatcher 用来初始化 Struts 2 并且处理所有的 Web 请求 -->
  <filter-mapping>
    <filter-name>Struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

经过上面 3 个步骤，我们已经可以在一个 Web 应用中使用 Struts 2 的基本功能了，下面将带领读者进入 Struts 2 MVC 框架的世界。

2.2 从用户请求开始

Struts 2 支持大部分视图技术，当然也支持最传统的 JSP 视图技术，本应用将使用最基本的视图技术：JSP 技术。当用户需要登录本系统时，用户需要一个简单的表单提交页面，这个表单提交页面包含了两个表单域：用户名和密码。

下面是一个最简单的表单提交页面，该页面的表单内仅包含两个表单域，甚至没有任何动态内容，实际上，整个页面完全可以是一个静态 HTML 页面。但考虑到需要在该页面后面增加动态内容，因此依然将该页面以 jsp 为后缀保存。下面是用户请求登录的 JSP 页面代码：

```
<%@ page language="java" contentType="text/html; charset=GBK"%>
<html>
<head>
<title>登录页面</title>
</head>
<body>
<!-- 提交请求参数的表单 -->
<form action="Login.action" method="post">
  <table align="center">
    <caption><h3>用户登录</h3></caption>
    <tr>
      <!-- 用户名的表单域 -->
      <td>用户名: <input type="text" name="username"/></td>
    </tr>
    <tr>
```


[illegible]

正如前面介绍的，该页面没有包含任何的动态内容，完全是一个静态的 HTML 页面。但我们注意到该表单的 `action` 属性：`login.action`，这个 `action` 属性比较特殊，它不是一个普通的 `Servlet`，也不是一个动态 JSP 页面。可能读者已经猜到了，当表单提交给 `login.action` 时，Struts 2 的 `FilterDispatcher` 将自动起作用，将用户请求转发到对应的 Struts 2 Action。

✱ 注意 Struts 2 Action 默认拦截所有后缀为.action 的请求。因此，如果我们需要将某个表单提交给 Struts 2 Action 处理，则应该将该表单的 action 属性设置为*.action 的格式。



图 2.1 用户登录的页面

该页面就是一个基本的 HTML 页面, 在浏览器中浏览该页面, 看到如图 2.1 所示的界面。

整个页面就是一个标准的 HTML 页面,整个单独的页面还没有任何与用户交互的能力。下面我们开始动手创建一个 Struts 2 的 Web 应用。

2.3 创建 Struts 2 的 Web 应用

Struts 2 的 Web 应用就是一个普通的 Web 应用，然后增加 Struts 2 功能，该应用就可以充分利用 Struts 2 的 MVC 框架了。

2.3.1 创建 Web 应用

笔者一直相信：要想成为一个优秀的程序员，应该从基本功练起，所有的代码都应该用简单的文本编辑器（包括 EditPlus、UltraEdit 等工具）完成。笔者经常见到一些有两三年开发经验的程序员，一旦离开了熟悉的 IDE（集成开发环境，如 Eclipse、JBuilder 等），完全不能动手写任何代码。而他们往往还振振有词：谁会不用任何工具来开发？

实际上，真正优秀的程序员当然可以使用 IDE 工具，但即使使用 VI (UNIX 下无格式编辑器)、记事本也一样可以完成非常优秀的项目。笔者对于 IDE 工具的态度是：可以使用 IDE 工具，但绝不可依赖于 IDE 工具。学习阶段，千万不要使用 IDE 工具；开发阶段，才去使用 IDE 工具。

❗ 提醒 对于 IDE 工具，业内有一个说法：IDE 工具会加快高手的开发效率，但会使初学者更白痴。

为了让读者更加清楚 Struts 2 应用的核心，笔者下面将“徒手”建立一个 Struts 2 应用。建立一个 Web 应用请按如下步骤进行。

- ❶ 在任意目录新建一个文件夹，笔者将以该文件夹建立一个 Web 应用。
- ❷ 在第 1 步所建的文件夹内建一个 WEB-INF 文件夹。
- ❸ 进入 Tomcat，或任何 Web 容器内，找到任何一个 Web 应用，将 Web 应用的 WEB-INF 下的 web.xml 文件复制到第 2 步所建的 WEB-INF 文件夹下。
- ❹ 修改复制的 web.xml 文件，将该文件修改成只有一个根元素的 XML 文件，修改后的 web.xml 文件代码如下：

```
<?xml version="1.0" encoding="GBK"?>
<!-- web-app 是 Web 应用配置文件的根元素，指定 Web 应用的 Schema 信息 -->
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.
        com/xml/ns/j2ee/web-app_2_4.xsd">
</web-app>
```

- ❺ 在第 2 步所建的 WEB-INF 路径下，新建两个文件夹：classes 和 lib，它们分别用于保存单个 *.class 文件和 JAR 文件。

经过上面步骤，已经建立了一个空 Web 应用。将该 Web 应用复制到 Tomcat 的 webapps 路径下，该 Web 应用将可以自动部署在 Tomcat 中。

将 2.2 节所定义的 JSP 页面文件复制到第 1 步所建的文件夹下，该 JSP 页面将成为该 Web 应用的一个页面。该 Web 将有如下文件结构：

```
Struts2qs
|—WEB-INF
|   |—classes
|   |—lib
|   |—web.xml
|—login.jsp
```

上面的 Struts2qs 是 Web 应用所对应文件夹的名字，可以更改；login.jsp 是该 Web 应用下 JSP 页面的名字，也可以修改。其他文件夹、配置文件都不可以修改。

启动 Tomcat，在浏览器中浏览 2.2 节定义的 JSP 页面，将看到如图 2.1 所示的页面。

2.3.2 增加 Struts 2 功能

为了给 Web 应用增加 Struts 2 功能，只需要将 Struts 2 安装到 Web 应用中即可。在 Web 应用中安装 Struts 2 框架核心只需要经过如下三个步骤。

- ❶ 修改 web.xml 文件，在 web.xml 文件中配置 Struts 2 的核心 Filter。
- ❷ 将 Struts 2 框架的类库复制到 Web 应用的 WEB-INF/lib 路径下。
- ❸ 在 WEB-INF/classes 下增加 struts.xml 配置文件。

下面是增加了 Struts 2 功能后 Web 应用的文件结构：

Struts2qs

```
|—WEB-INF
|   |—classes (struts.xml)
|   |—lib (commons-logging.jar, freemarker.jar, ognl.jar, struts2-core.jar, xwork.jar)
|   |—web.xml
|—login.jsp
```

在上面的文件结构中，lib 下 Struts 2 框架的类库可能有版本后缀。例如 commons-logging.jar，可能是 commons-logging-1.1.jar；struts2-core.jar 可能是 struts2-core-2.0.6.jar。

修改后的 web.xml 文件在 2.1 节已经给出了，故此处不再赘述。

此处需要给读者指出的是，Struts 2 的 Web 应用默认需要 Java 5 运行环境，需要 Web 容器支持 Servlet API 2.4 和 JSP API 2.0。如果读者需要使用更低版本的 Java 运行时环境，则需要使用 Struts 2 框架的 JDK 1.4 支持。为了简单起见，笔者建议读者使用 Java 5 运行时环境，使用 Tomcat 5.5 或者更高版本。

✱ 注意 Struts 2 应用默认需要 Java 5 运行时环境，需要支持 Servlet API 2.4 和 JSP API 2.0 的 Web 容器。

2.4 实现控制器

前面介绍 MVC 框架时，已经指出：MVC 框架的核心就是控制器。当用户通过 2.2 节的页面提交用户请求时，该请求需要提交给 Struts 2 的控制器处理。Struts 2 的控制器根据处理结果，决定将哪个页面呈现给客户端。

2.4.1 实现控制器类


Struts 2 下的控制器不再像 Struts 1 下的控制器，需要继承一个 Action 父类，甚至可以无需实现任何接口，Struts 2 的控制器就是一个普通的 POJO。

实际上，Struts 2 的 Action 就是一个包含 execute 方法的普通 Java 类，该类里包含的多个属性用于封装用户的请求参数。下面是处理用户请求的 Action 类的代码：

```
//Struts 2 的 Action 类就是一个普通的 Java 类
public class LoginAction
{
    //下面是 Action 内用于封装用户请求参数的两个属性
    private String username;
    private String password;
    //username 属性对应的 getter 方法
    public String getUsername()
    {
        return username;
    }
    //username 属性对应的 setter 方法
    public void setUsername(String username)
```

```
{
    this.username = username;
}
//password 属性对应的 getter 方法
public String getPassword()
{
    return password;
}
//password 属性对应的 setter 方法
public void setPassword(String password)
{
    this.password = password;
}
//处理用户请求的 execute 方法
public String execute() throws Exception
{
    //当用户请求参数的 username 等于 scott,密码请求参数为 tiger 时,返回 success
    字符串
    //否则返回 error 字符串
    if (getUsername().equals("scott")
        && getPassword().equals("tiger") )
    {
        return "success";
    }
    else
    {
        return "error";
    }
}
}
```

上面的 Action 类是一个再普通不过的 Java 类，该类里定义了两个属性：username 和 password，并为这两个属性提供了对应的 setter 和 getter 方法。除此之外，该 Action 类里还包含了一个无参数的 execute 方法——这大概也是 Action 类与 POJO 唯一的差别。实际上，这个 execute 方法依然是一个很普通的方法，既没有与 Servlet API 耦合，也没有与 Struts 2 API 耦合。

 提示 表面上看起来，该 Action 的两个属性只提供了对应的 setter 和 getter 方法，很难理解请求参数在什么时候赋值给该 Action 的属性，事实上，因为 Struts 2 的拦截器机制，它们负责解析用户的请求参数，并将请求参数赋值给 Action 对应的属性。

2.4.2 配置 Action

上面定义了 Struts 2 的 Action，但该 Action 还未配置在 Web 应用中，还不能处理用户请求。为了让该 Action 能处理用户请求，还需要将该 Action 配置在 struts.xml 文件中。

前面已经介绍过了，struts.xml 文件应该放在 classes 路径下，该文件主要放置 Struts 2 的 Action 定义。定义 Struts 2 Action 时，除了需要指定该 Action 的实现类外，还需要定义

Action 处理结果和资源之间的映射关系。下面是该应用的 struts.xml 文件的代码：

```
<?xml version="1.0" encoding="GBK"?>
<!-- 指定 Struts 2 配置文件的 DTD 信息 -->
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<!-- struts 是 Struts 2 配置文件的根元素 -->
<struts>
    <!-- Struts 2 的 Action 必须放在指定的包空间下定义 -->
    <package name="strutsqs" extends="struts-default">
        <!-- 定义 login 的 Action, 该 Action 的实现类为 lee.Action 类 -->
        <action name="Login" class="lee.LoginAction">
            <!-- 定义处理结果和资源之间映射关系。 -->
            <result name="error">/error.jsp</result>
            <result name="success">/welcome.jsp</result>
        </action>
    </package>
</struts>
```

上面映射文件定义了 name 为 login 的 Action，即：该 Action 将负责处理向 login.action URL 请求的客户端请求。该 Action 将调用自身的 execute 方法处理用户请求，如果 execute 方法返回 success 字符串，请求将被转发到/welcome.jsp 页面；如果 execute 方法返回 error 字符串，则请求被转发到/error.jsp 页面。

2.4.3 增加视图资源完成应用

经过上面步骤，这个最简单的 Struts 2 应用几乎可以运行了，但还需要为该 Web 应用增加两个 JSP 文件，两个 JSP 文件分别是 error.jsp 页面和 welcome.jsp 页面，将这两个 JSP 页面文件放在 Web 应用的根路径下（与 WEB-INF 在同一个文件夹下）。

这两个 JSP 页面文件是更简单的页面，它们只是包含了简单的提示信息。其中 welcome.jsp 页面的代码如下：

```
<%@ page language="java" contentType="text/html; charset=GBK"%>
<html>
  <head>
    <title>成功页面</title>
  </head>
  <body>
    您已经登录!
  </body>
</html>
```

上面的页面就是一个普通的 HTML 页面，登录失败后进入的 error.jsp 页面也与此完全类似。

在如图 2.1 所示页面的“用户名”输入框中输入 scott，在“密码”输入框中输入 tiger，页面将进入 welcome.jsp 页面，将看到如图 2.2 所示的页面。

对于上面的处理流程，可以简化为如下的流程：用户输入两个参数，即 `username` 和 `password`，然后向 `login.action` 发送请求，该请求被 `FilterDispatcher` 转发给 `LoginAction` 处理，如果 `LoginAction` 处理用户请求返回 `success` 字符串，则返回给用户 `welcome.jsp` 页面；如果返回 `error` 字符串，则返回给用户 `error.jsp` 页面。

图 2.3 显示了上面应用的处理流程。

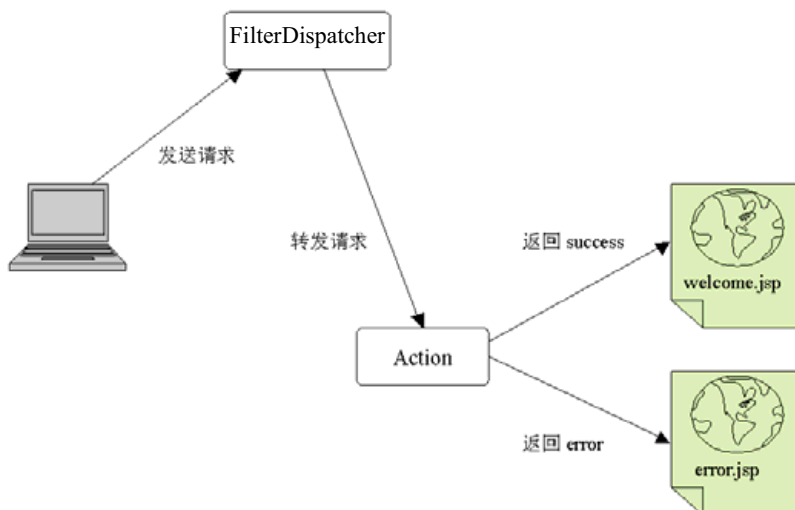


图 2.3 HelloWorld 应用的处理流程



图 2.2 登录成功页面

2.5 改进控制器

通过前面介绍，读者已经可以完成简单的 Struts 2 的基本应用了，但还可以进一步改进应用的 `Action` 类，例如该 `Action` 类可以通过实现 `Action` 接口，利用该接口的优势。前面应用的 `Action` 类没有与 `JavaBean` 交互，没有将业务逻辑操作的结果显示给客户端。

2.5.1 实现 `Action` 接口

表面上看起来，实现 Struts 2 的 `Action` 接口没有太大的好处，仅会污染该 `Action` 的实现类。事实上，实现 `Action` 接口可以帮助开发者更好地实现 `Action` 类。下面首先看 `Action` 接口的定义：

```
public interface Action
{
    //下面定义了 5 个字符串常量
    public static final String SUCCESS = "success";
    public static final String NONE = "none";
    public static final String ERROR = "error";
    public static final String INPUT = "input";
    public static final String LOGIN = "login";
    //定义处理用户请求的 execute 抽象方法
    public String execute() throws Exception;
}
```


在上面的 Action 代码中，我们发现该 Action 接口里已经定义了 5 个标准字符串常量：SUCCESS、NONE、ERROR、INPUT 和 LOGIN，它们可以简化 execute 方法的返回值，并可以使用 execute 方法的返回值标准化。例如对于处理成功，则返回 SUCCESS 常量，避免直接返回一个 success 字符串（程序中应该尽量避免直接返回数字常量、字符串常量等）。

因此，借助于上面的 Action 接口，我们可以将原来的 Action 类代码修改为如下：

```
//实现 Action 接口来实现 Struts 2 的 Action 类
public class LoginAction implements Action
{
    //下面是 Action 内用于封装用户请求参数的两个属性
    private String username;
    private String password;
    //username 属性对应的 getter 方法
    public String getUsername()
    {
        return username;
    }
    //username 属性对应的 setter 方法
    public void setUsername(String username)
    {
        this.username = username;
    }
    //password 属性对应的 getter 方法
    public String getPassword()
    {
        return password;
    }
    //password 属性对应的 setter 方法
    public void setPassword(String password)
    {
        this.password = password;
    }
    //处理用户请求的 execute 方法
    public String execute() throws Exception
    {
        //当用户请求参数的 username 等于 scott, 密码请求参数为 tiger 时, 返回 success
        //字符串
        //否则返回 error 的字符串
        if (getUsername().equals("scott")
            && getPassword().equals("tiger") )
        {
            return SUCCESS;
        }
        else
        {
            return ERROR;
        }
    }
}
```

对比前面 Action 和此处的 Action 实现类，我们发现两个 Action 类的代码基本相似，除了后面的 Action 类实现了 Action 接口。因为实现了 Action 接口，故 Action 类的 execute 方法可以返回 Action 接口里的字符串常量。

2.5.2 跟踪用户状态

前面的 Action 处理完用户登录后，仅仅执行了简单的页面转发，并未跟踪用户状态信息——通常，当一个用户登录成功后，需要将用户的用户名添加为 Session 状态信息。

为了访问 HttpSession 实例，Struts 2 提供了一个 ActionContext 类，该类提供了一个 getSession 的方法，但该方法的返回值类型并不是 HttpSession，而是 Map。这又是怎么回事呢？实际上，这与 Struts 2 的设计哲学有关，Struts 2 为了简化 Action 类的测试，将 Action 类与 Servlet API 完全分离，因此 getSession 方法的返回值类型是 Map，而不是 HttpSession。

虽然 ActionContext 的 getSession 返回的不是 HttpSession 对象，但 Struts 2 的系列拦截器会负责该 Session 和 HttpSession 之间的转换。

为了可以跟踪用户信息，我们修改 Action 类的 execute 方法，在 execute 方法中通过 ActionContext 访问 Web 应用的 Session。修改后的 execute 方法代码如下：

```
//处理用户请求的 execute 方法
public String execute() throws Exception
{
    //当用户请求参数的 username 等于 scott，密码请求参数为 tiger 时，返回 success 字符串
    //否则返回 error 的字符串
    if (getUsername().equals("scott")
        && getPassword().equals("tiger") )
    {
        //通过 ActionContext 对象访问 Web 应用的 Session
        ActionContext.getContext().getSession().put("user" , getUsername());
        return SUCCESS;
    }
    else
    {
        return ERROR;
    }
}
```

上面的代码仅提供了 Action 类的 execute 方法，该 Action 类的其他部分与前面的 Action 类代码完全一样。在上面的 Action 类通过 ActionContext 设置了一个 Session 属性：user。为了检验我们设置的 Session 属性是否成功，我们修改 welcome.jsp 页面，在 welcome.jsp 页面中使用 JSP 2.0 表达式语法输出 Session 中的 user 属性。下面是修改后的 welcome.jsp 页面代码：

```
<%@ page language="java" contentType="text/html; charset=GBK"%>
<html>
  <head>
    <title>成功页面</title>
```

```
</head>
<body>
    欢迎, ${sessionScope.user}, 您已经登录!
</body>
</html>
```

上面的 JSP 页面与前面的 JSP 页面没有太大改变, 除了使用了 JSP 2.0 语法来输出 Session 中的 user 属性。关于 JSP 2.0 表达式的知识, 请参看笔者所著的《轻量级 J2EE 企业应用实战》一书的第 2 章。

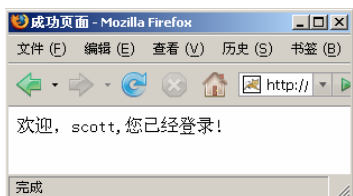


图 2.4 登录成功页面

在如图 2.1 所示页面的“用户名”输入框中输入 scott, 在“密码”输入框中输入 tiger, 然后单击“登录”按钮, 将看到如图 2.4 所示的页面。

在上面登录成功的页面中, 已经输出登录所用的用户名: scott, 可见在 Action 通过 ActionContext 设置 Session 是成功的。

2.5.3 添加处理信息

到目前为止, Action 仅仅控制转发用户请求, JSP 页面并未获得 Action 的处理结果。对于大部分 Web 应用而言, 用户需要获得请求 Action 的处理结果, 例如, 在线购物系统需要查询某个种类下的商品, 则 Action 调用业务逻辑组件的业务逻辑方法得到该种类下的全部商品, 而 JSP 页面则获取该 Action 的处理结果, 并将全部结果迭代输出。

下面将为应用增加一个 Action, 该 Action 负责获取某个系列的全部书籍。为了让该 Action 可以获取这系列的书籍, 我们增加一个业务逻辑组件, 它包含一个业务逻辑方法, 该方法可以获取某个系列的全部书籍。

下面是系统所用的业务逻辑组件的代码:

```
public class BookService
{
    //以一个常量数组模拟了从持久存储设备（数据库）中取出的数据
    private String[] books =
        new String[]{
            "Spring2.0 宝典",
            "轻量级 J2EE 企业应用实战",
            "基于 J2EE 的 Ajax 宝典",
            "Struts, Spring, Hibernate 整合开发"
        };
    //业务逻辑方法, 该方法返回全部图书
    public String[] getLeeBooks()
    {
        return books;
    }
}
```

上面的业务逻辑组件实际上就是 MVC 模式中的 Model, 它负责实现系统业务逻辑方法。理论上, 业务逻辑组件实现业务逻辑方法时, 必须依赖于底层的持久层组件, 但此处的业务逻辑组件则只是返回一个静态的字符串数组——因为这只是一种模拟。

✱ 注意 此处的业务逻辑组件只是模拟实现业务逻辑方法，并未真正调用持久层组件来获取数据库信息。

在系统中增加如下 Action 类，该 Action 类先判断 Session 中 user 属性是否存在，并且等于 scott 字符串——这要求查看图书之前，用户必须已经登录本系统。如果用户已经登录本系统，则获取系统中全部书籍，否则返回登录页面。

新增的 Action 类的代码如下：

```
public class GetBooksAction implements Action
{
    //该属性并不用于封装用户请求参数，而用于封装 Action 需要输出到 JSP 页面信息
    private String[] books;
    //books 属性的 setter 方法
    public void setBooks(String[] books)
    {
        this.books = books;
    }
    //books 属性的 getter 方法
    public String[] getBooks()
    {
        return books;
    }
    //处理用户请求的 execute 方法
    public String execute() throws Exception
    {
        //获取 Session 中的 user 属性
        String user = (String)ActionContext.getContext().getSession().get("user");
        //如果 user 属性不为空，且该属性值为 scott
        if (user != null && user.equals("scott"))
        {
            //创建 BookService 实例
            BookService bs = new BookService();
            //将业务逻辑组件的返回值设置成该 Action 的属性
            setBooks(bs.getLeeBooks());
            return SUCCESS;
        }
        else
        {
            return LOGIN;
        }
    }
}
```

通过上面的 Action 类，我们发现 Action 类中的成员属性，并不一定用于封装用户的请求参数，也可能是封装了 Action 需要传入下一个 JSP 页面中显示的属性。

❗ 提示 Action 中的成员属性，并不一定用于封装用户的请求参数，也可能是封装了 Action 需要传入下一个页面显示的值。实际上，这些值将被封装在 ValueStack 对象中。

当我们的控制器需要调用业务逻辑方法时，我们直接创建了一个业务逻辑组件的实例，这并不是一种好的做法，因为控制器不应该关心业务逻辑组件的实例化过程。比较成熟的做法可以利用工厂模式来管理业务逻辑组件；当然，目前最流行的方式是利用依赖注入——这将在后面章节里介绍。

✴ 注意 实际项目中不会在控制器中直接创建业务逻辑组件的实例，而是通过工厂模式管理业务逻辑组件实例，或者通过依赖注入将业务逻辑组件实例注入控制器组件。

该 Action 处理用户请求时，无需获得用户的任何请求参数。将该 Action 配置在 struts.xml 文件中，配置该 Action 的配置片段如下：

```
<!-- 定义获取系统中图书的 Action，对应实现类为 lee.GetBooksAction -->
<action name="GetBooks" class="lee.GetBooksAction">
  <!-- 如果处理结果返回 login，进入 login.jsp 页面 -->
  <result name="login">/login.jsp</result>
  <!-- 如果处理结果返回 success，进入 showBook.jsp 页面 -->
  <result name="success">/showBook.jsp</result>
</action>
```

当用户向 getBooks.action 发送请求时，该请求将被转发给 lee.GetBooksAction 处理。

2.5.4 输出处理信息

如果用户没有登录，直接向 getBooks.action 发送请求，该请求将被转发到 login.jsp 页面。如果用户已经登录，getBooks.action 将从系统中加载到系统中的所有图书，并将请求转发给 showBook.jsp 页面，因此 showBook.jsp 页面必须负责输出全部图书。

下面笔者将以最原始的方式：JSP 脚本来输出全部图书。

✴ 注意 在实际应用中，几乎绝对不会使用笔者这种方式来输出 Action 转发给 JSP 输出的信息，但笔者为了让读者更清楚 Struts 2 标签库在底层所完成的动作，故此处使用 JSP 脚本来输出全部图书信息。

当 Action 设置了某个属性值后，Struts 2 将这些属性值全部封装在一个叫做 struts.valueStack 的请求属性里。

❗ 提示 读者可能感到奇怪：笔者是如何知道 Struts 2 将这些属性值封装在 struts.valueStack 请求属性里的？这一方面与编程经验有关，另一方面可以通过查看 Struts 2 的各种文档，最重要的一点是可以在 showBook.jsp 页面中通过 getAttributeNames 方法分析请求中的全部属性。

为了在 JSP 页面中输出需要输出的图书信息，我们可以通过如下代码来获取包含全部输出信息的 ValueStack 对象。

```
//获取封装输出信息的 ValueStack 对象
request.getAttribute("struts.valueStack");
```

上面代码返回一个 `ValueStack` 对象，该对象封装了全部的输出信息。该对象是 Struts 2 使用的一个 `ValueStack` 对象，可以通过 OGNL 表达式非常方便地访问该对象封装的信息。

从数据结构上来看，`ValueStack` 有点类似于 `Map` 结构，但它比 `Map` 结构更加强大（因为它可以根据表达式来查询值）。Action 所有的属性都被封装到了 `ValueStack` 对象中，Action 中的属性名可以理解为 `ValueStack` 中 `value` 的名字。

大致理解了 `ValueStack` 对象的结构后，我们可以通过如下代码来获取 Action 中设置的全部图书信息。

```
//调用 ValueStack 的 findValue 方法查看某个表达式的值
vs.findValue("books");
```

理解了上面关键的两步，整个 JSP 页面的代码就比较容易了解了。下面是 `showBook.jsp` 页面的代码：

```
<%@ page language="java" contentType="text/html; charset=GBK"%>
<% @page import="java.util.*,com.opensymphony.xwork2.util.*"%>
<html>
  <head>
    <title>作者李刚的图书</title>
  </head>
  <body>
    <table border="1" width="360">
      <caption>作者李刚的图书</caption>
      <%
        //获取封装输出信息的 ValueStack 对象
        ValueStack vs = (ValueStack)request.getAttribute("struts.valueStack");
        //调用 ValueStack 的 findValue 方法获取 Action 中的 books 属性值
        String[] books = (String[])vs.findValue("books");
        //迭代输出全部图书信息
        for (String book : books)
        {
          %>
          <tr>
            <td>书名: </td>
            <td><%=book%></td>
          </tr>
          <%}%>
        </table>
      </body>
    </html>
```

不可否认，上面 JSP 页面的代码是丑陋的，而且难以维护，因为里面镶嵌了大量的 Java 脚本。但它对于读者理解 Struts 2 如何处理封装在 Action 的 `ValueStack` 却很有帮助。

在浏览器中向 `getBooks.action` 发送请求，将看到如图 2.5 所示的页面。

通过上面页面，我们看到 JSP 页面已经输出了 Struts 2 控制器的返回信息。上面整个过程，已经完全包括了 Struts 2 框架的 3 个部分：视图、控制器和模型。

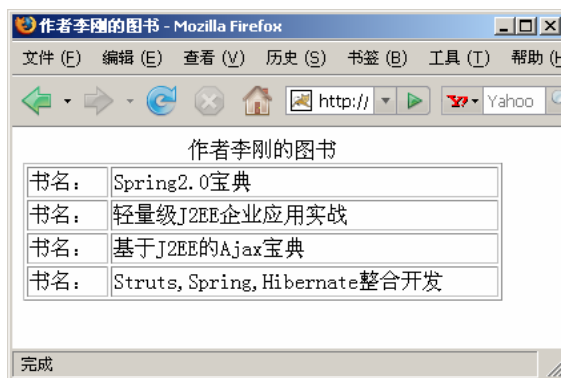


图 2.5 在 JSP 页面中输出 Action 的返回信息

2.6 改进视图组件

通过前面的几节的介绍，我们已经明白了 Struts 2 MVC 框架的基本数据流，已经完成了 Struts 2 应用中模型、控制器、视图 3 个组件的开发。但应用中的视图组件：JSP 页面非常丑陋，特别是输出 Action 返回信息的 JSP 页面，使用了大量的 Java 脚本来控制输出，下面将会使用 Struts 2 的标签来改善整个应用视图组件。

2.6.1 改善输出页面

为了控制输出 Struts 2 的 ValueStack 中封装的值，Struts 2 提供了大量的标签。其中比较常用的标签有：

- **if**: 该标签支持标签体，如果 if 标签里判断的表达式返回真，则输出标签体内容。
- **else**: 该标签不能独立使用，它需要与 if 标签结合使用，如果 if 标签内判断的表达式返回假，则输出该标签里的标签体。
- **iterator**: 主要用于迭代输出某个集合属性的各个集合元素。
- **property**: 该标签用于输出指定属性值。

关于 Struts 2 标签库更深入的使用，第 10 章还会深入介绍，故此处不再详细讲解。通过使用上面的几个标签，替换 showBook.jsp 页面中的 Java 脚本，修改后的 showBook.jsp 页面代码如下：

```
<%@ page language="java" contentType="text/html; charset=GBK" %>
<!-- 导入 Struts 2 的标签库 -->
<%@taglib prefix="s" uri="/struts-tags"%>
<html>
  <head>
    <title>作者李刚的图书</title>
  </head>
  <body>
    <table border="1" width="360">
      <caption>作者李刚的图书</caption>
      <!-- 迭代输出 ValueStack 中的 books 对象，其中 status 是迭代的序号 -->
      <s:iterator value="books" status="index">
        <!-- 判断序号是否为奇数 -->
        <s:if test="#index.odd == true">
          <tr style="background-color:#cccccc">
```

```
</s:if>
<!-- 判断迭代元素的序号是否不为偶数 -->
<s:else>
    <tr>
</s:else>
    <td>书名: </td>
    <td><s:property/></td>
</tr>
</s:iterator>
</table>
</body>
</html>
```

上面的 JSP 页面使用了 Struts 2 的标签库,因此必须在 JSP 页面的首部添加 `taglib` 指令,该 `taglib` 指令用于导入标签库。

❗ 提示 如果需要使用某个标签库中的标签,则必须在页面的开始导入该标签库。

页面中使用 Struts 2 的 `iterator` 标签迭代输出 `ValueStack` 中的 `books` 数组,并为每个数组元素定义了一个序号: `index`。通过判断序号是否为奇数,如果行序号为奇数,则输出一个有背景色的表格行;否则输出一个无背景色的表格行。

在浏览器中再次向 `getBooks.action` 发送请求(发送请求之前,必须先登录本系统),将看到如图 2.6 所示的界面。

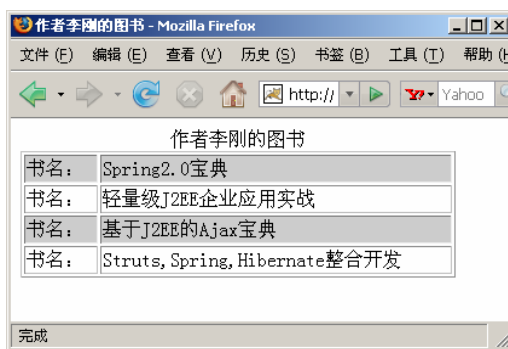


图 2.6 使用 Struts 2 标签改善后的输出界面

上面页面的输出效果与图 2.5 并没有太大的不同,只是使用不同颜色来分隔了记录行。这也得益于 Struts 2 标签库的简洁。

关键在于 2.5.4 节中的 JSP 页面代码与本节页面代码的差异:前面 JSP 页面使用了大量的 Java 脚本,让整个页面的代码看起来非常凌乱,降低了可阅读性、可维护性。但本页面中仅使用 Struts 2 脚本控制输出,完全消除了页面中的 Java 脚本,降低了该页面的后期维护成本。

2.6.2 使用 UI 标签简化表单页面

前面已经提到过,Struts 2 的一个重要组件就是标签库。Struts 2 标签库中不仅提供了前面所示的基本控制、数据输出等功能,还提供了非常丰富的 UI 组件,除了提供系列的

主题相关标签外，还提供了一系列的表单相关的标签。

Struts 2 为常用表单域都提供了对应的标签，下面是常用的表单域标签。

- **form**: 对应一个表单元素。
- **checkbox**: 对应一个复选框元素。
- **password**: 对应一个密码输入框。
- **radio**: 对应一个单选框元素。
- **reset**: 对应一个重设按钮。
- **select**: 对应一个下拉列表框。
- **submit**: 对应一个提交按钮。
- **textarea**: 对应一个多行文本域。
- **textfield**: 对应一个单行文本框。

关于这些界面相关的标签，同样将在第 10 章详细介绍。下面将使用 Struts 2 的表单相关标签简化用户登录的 login.jsp 页面，修改的 login.jsp 页面的代码如下：

```
<%@ page language="java" contentType="text/html; charset=GBK"%>
<%@taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
<title>登录页面</title>
</head>
<body>
<!-- 使用 form 标签生成表单元素 -->
<s:form action="Login">
  <!-- 生成一个用户名文本输入框 -->
  <s:textfield name="username" label="用户名"/>
  <!-- 生成一个密码文本输入框 -->
  <s:textfield name="password" label="密 码"/>
  <!-- 生成一个提交按钮 -->
  <s:submit value="登录"/>
</s:form>
</body>
</html>
```

将该页面与前面的表单页面进行对比，我们发现该页面的代码简洁多了。因为使用了 Struts 2 的表单标签，定义表单页面也更加迅速。在浏览器中浏览该页面，看到如图 2.7 所示的界面。



图 2.7 使用 Struts 2 表单标签后的表单页

当然，Struts 2 的标签还有许多功能，此处先不详述，本书的第 10 章将会详细介绍

Struts 2 标签的用法。

2.7 完成程序国际化

因为一个企业应用经常需要面对多区域的用户，因此，程序国际化是一个企业应用必须实现的功能。Struts 2 提供了很好的程序国际化支持。

2.7.1 定义国际化资源文件

Struts 2 的程序国际化支持建立在 Java 程序国际化的基础之上，关于 Java 程序的国际化笔者将在第 9 章简要介绍。此处不会详细介绍，但我们要明白一个概念：程序国际化的设计思想是非常简单的，其主要思想是：程序界面中需要输出国际化信息的地方，我们不要在页面中直接输出信息，而是输出一个 **key** 值，该 **key** 值在不同语言环境下对应不同的字符串。当程序需要显示时，程序将根据不同的语言环境，加载该 **key** 对应该语言环境下的字符串——这样就可以完成程序的国际化。

图 2.8 显示了程序国际化的示意图。

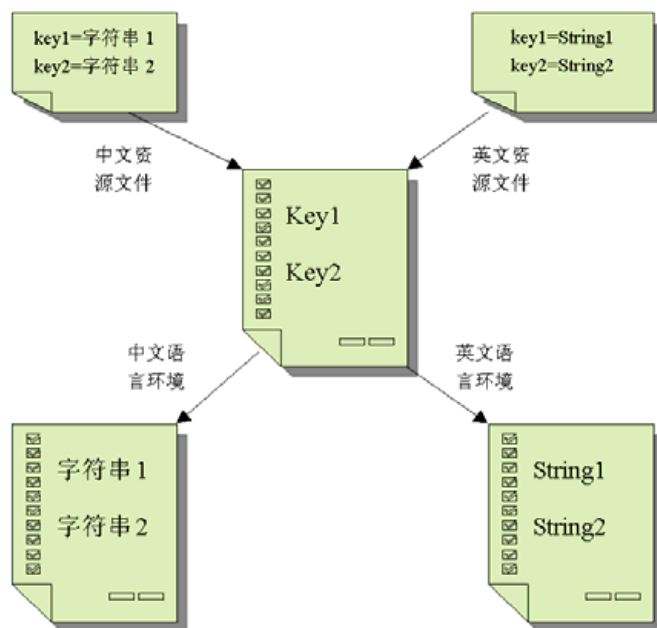


图 2.8 程序国际化示意图

从图 2.8 可以看出，如果需要程序支持更多的语言环境，只需要增加更多语言资源文件即可。

为了给本应用增加程序国际化支持（支持英文和中文），则应该提供两份语言资源文件。下面是本应用所使用的中文语言环境下资源文件的代码。

```
loginPage=登录页面
errorPage=错误页面
succPage=成功页面
failTip=对不起，您不能登录！
succTip=欢迎，${0}，您已经登录！
viewLink=查看作者李刚已出版的图书
```

```
bookPageTitle=作者李刚已出版的图书
bookName=书名:
user=用户名
pass=密 码
login=登录
```

因为该资源文件中包含了非西欧字符，因此必须使用 `native2ascii` 命令来处理该文件。将上面文件保存在 `WEB-INF/classes` 路径下，文件名为“`messageResouce.properties`”。保存该文件后，必须使用 `native2ascii` 命令来处理该文件，处理该文件的命令格式为：

```
native2ascii messageResouce.properties messageResouce_zh_CN.properties
```

上面命令将包含非西欧字符的资源文件处理成标准的 ASCII 格式，处理完成后生成了一份新文件：`messageResouce_zh_CN.properties` 文件。这个文件的文件名符合资源文件的命名格式，资源文件的文件名命名格式为：

```
basename_语言代码_国家代码.properties
```

当请求来自简体中文的语言环境时，系统将自动使用这种资源文件中的内容输出。

✱ 注意 对于包含非西欧字符的资源文件，一定要使用 `native2assii` 命令来处理该文件，否则将看到一堆乱码。

除此之外，还应该提供如下英文语言环境的资源文件。

```
loginPage=Login Page
errorPage=Error Page
succPage=Welcome Page
failTip=Sorry,You can't log in!
succTip=welcome,{0},you has logged in!
viewLink=View LiGang\'s Books
bookPageTitle=LiGang\'s Books
bookName=BookName:
user=User Name
pass=User Pass
login=Login
```

将上面资源文件保存在 `WEB-INF/classes` 路径下，文件名为“`messageResouce_en_US.properties`”。当请求来自美国时，系统自动使用这份资源文件的内容输出。

2.7.2 加载资源文件

Struts 2 支持在 JSP 页面中临时加载资源文件，也支持通过全局属性来加载资源文件。通过全局属性加载资源文件更简单，本应用使用全局属性加载 Struts 2 国际化资源文件。

加载资源文件可以通过 `struts.properties` 文件来定义，本应用的 `struts.properties` 文件仅有如下行代码：

```
//定义 Struts 2 的资源文件的 baseName 是 messageResource
struts.custom.i18n.resources=messageResource
```

在 `struts.properties` 文件中增加上面的代码定义后，表明该应用使用的资源文件的

baseName 为 “messageResource” ——这与我们前面保存资源文件的 baseName 是一致的。

Struts 2 默认加载 WEB-INF/classes 下的资源文件，在上一节中，我们就是将资源文件保存在该路径下的。如果将该资源文件保存在 WEB-INF/classes 的子目录下，例如保存在 WEB-INF/classes/lee 路径下，则需要修改 struts.properties 中的定义如下：

```
//定义 Struts 2 的资源文件的 baseName 是 messageResource，且文件放在 WEB-INF/
classes/lee 路径下
struts.custom.i18n.resources=lee.messageResource
```

2.7.3 输出国际化信息

为了让程序可以显示国际化信息，则需要在 JSP 页面中输出 key，而不是直接输出字符串常量。

Struts 2 提供了如下两种方式来输出国际化信息：

- `<s:text name="messageKey"/>`：使用 `s:text` 标签来输出国际化信息。
- `<s:property value="%{getText("messageKey")}"/>`：使用表达式方式输出国际化信息。

因此，我们再次修改表现层的 JSP 页面，使用国际化标签输出国际化信息。修改后的 showBook.jsp 页面代码如下：

```
<%@ page language="java" contentType="text/html; charset=GBK"%>
<%@taglib prefix="s" uri="/struts-tags"%>
<html>
  <head>
    <!-- 使用 s:text 输出国际化信息 -->
    <title><s:text name="bookPageTitle"/></title>
  </head>
  <body>
    <table border="1" width="360">
      <!-- 使用 s:text 输出国际化信息 -->
      <caption><s:text name="bookPageTitle"/></caption>
      <s:iterator value="books" status="index">
        <s:if test="#index.odd == true">
          <tr style="background-color:#cccccc">
            </s:if>
            <s:else>
              <tr>
                </s:else>
                <td><s:text name="bookName"/></td>
                <td><s:property/></td>
              </tr>
            </s:iterator>
          </table>
        </body>
      </html>
```

我们发现，上面的 JSP 页面不再包含任何直接字符串，而是全部通过 `<s:text name="..."/>` 来输出国际化提示。

再次在浏览器浏览该页面，将看到与图 2.7 相同的界面。

重新设置浏览者所在的语言/区域选项，设置语言/区域选项请先进入“控制面板”，在

控制面板中单击“区域和语言选项”，进入如图 2.9 所示的对话框。

如果我们选择“英语（美国）”选项，然后单击“确定”按钮，将设置本地的语言环境为美国英语。

再次向服务器请求 login.jsp 页面，将看到如图 2.10 所示的页面。

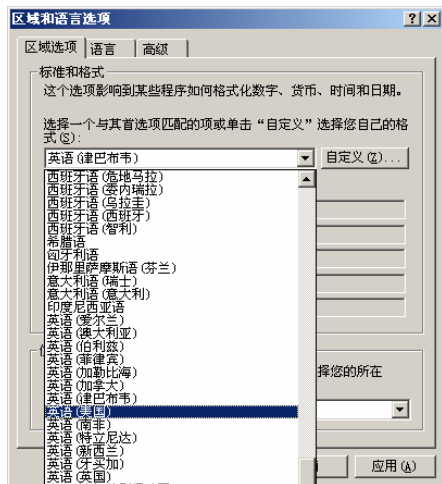


图 2.9 设置语言/区域选项



图 2.10 程序国际化的效果

如果我们使用 Firefox 浏览器来浏览该页面时，发现依然显示中文界面——这是因为 Firefox 的语言环境并不受 Windows 系统的控制。为了让 Firefox 也使用美国英语环境，单击 Firefox 浏览器菜单栏中的“工具”菜单，选择“选项”菜单项，将出现“选项”对话框，单击“高级”按钮，将看到如图 2.11 所示的界面。

在如图 2.11 所示的对话框中单击“选择”按钮，将出现“语言和字符编码”对话框，在该对话框下面的下拉列表框中选择“英语/美国 [en-us]”选项，如图 2.12 所示，然后单击下拉框右边的“添加”按钮，将添加了“英语/美国”的语言环境。并在该对话框上面的列表框中选择“英语/美国 [en-us]”，然后单击“上移”按钮，将该选项移至最上面，让整个页面优先使用英语/美国的环境。

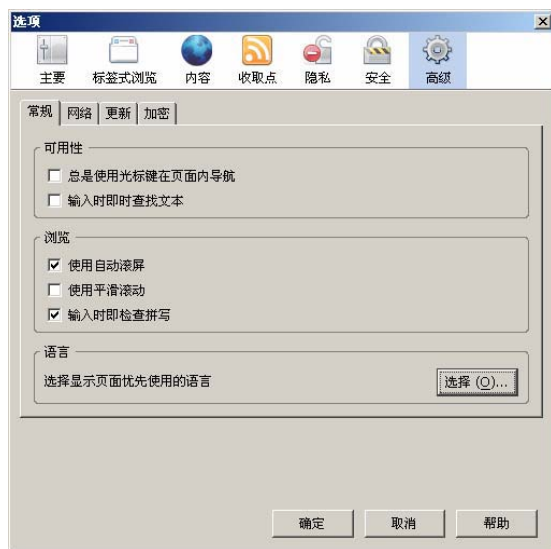


图 2.11 设置 Firefox 的语言环境

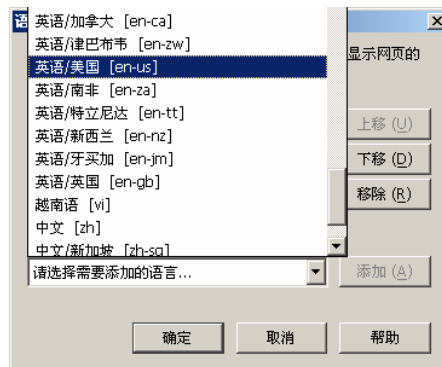


图 2.12 设置 Firefox 的语言环境为英语/美国

再次使用 Firefox 浏览器浏览 login.jsp 页面，将看到该页面变成了英文界面。

2.8 增加数据校验

在上面应用中，即使浏览者输入任何用户名、密码，系统也会处理用户请求。在我们整个 HelloWorld 应用中，这种空用户名、空密码的情况不会引起太大的问题。但如果数据需要保存到数据库，或者需要根据用户输入的用户名、密码查询数据，这些空输入可能引起异常。

为了避免用户的输入引起底层异常，通常我们会在进行业务逻辑操作之前，先执行基本的数据校验。

2.8.1 继承 ActionSupport

ActionSupport 类是一个工具类，它已经实现了 Action 接口。除此之外，它还实现了 Validateable 接口，提供了数据校验功能。通过继承该 ActionSupport 类，可以简化 Struts 2 的 Action 开发。

在 Validateable 接口中定义了一个 validate() 方法，重写该方法，如果校验表单输入域出现错误，则将错误添加到 ActionSupport 类的 fieldErrors 域中，然后通过 OGNL 表达式负责输出。

为了让 Struts 2 增加输入数据校验的功能，改写程序中的 LoginAction，增加重写 validate 方法。修改后的 LoginAction 类代码如下：

```
//Struts 2 的 Action 类就是一个普通的 Java 类
public class LoginAction
{
    //下面是 Action 内用于封装用户请求参数的两个属性
    private String username;
    private String password;
    //username 属性对应的 getter 方法
    public String getUsername()
    {
        return username;
    }
    //username 属性对应的 setter 方法
    public void setUsername(String username)
    {
        this.username = username;
    }
    //password 属性对应的 getter 方法
    public String getPassword()
    {
        return password;
    }
    //password 属性对应的 setter 方法
    public void setPassword(String password)
    {
        this.password = password;
    }
}
```

```
}
//处理用户请求的 execute 方法
public String execute() throws Exception
{
    //当用户请求参数的 username 等于 scott,密码请求参数为 tiger 时,返回 success
    字符串
    //否则返回 error 的字符串
    if (getUsername().equals("scott")
        && getPassword().equals("tiger") )
    {
        return "success";
    }
    else
    {
        return "error";
    }
}
//完成输入校验需要重写的 validate 方法
public void validate()
{
    //如果用户名为空, 或者用户名为空字符串
    if (getUsername() == null || getUsername().trim().equals(""))
    {
        //添加表单校验错误
        addFieldError("username", "user.required");
    }
    //当密码为空, 或者密码为空字符串时, 添加表单校验错误
    if (getPassword() == null || getPassword().trim().equals(""))
    {
        addFieldError("password", "pass.required");
    }
}
}
```

上面的 Action 类重写了 validate 方法, 该方法会在执行系统的 execute 方法之前执行, 如果执行该方法之后, Action 类的 fieldErrors 中已经包含了数据校验错误, 请求将被转发到 input 逻辑视图处。

为了在校验失败后, 系统能将视图转入 input 处, 必须在配置该 Action 时配置 input 属性。下面是修改后 login Action 的配置片段:

```
<!-- 定义 login 的 Action -->
<action name="Login" class="lee.LoginAction">
    <!-- 定义 input 的逻辑视图名, 对应 login.jsp 页面 -->
    <result name="input">/login.jsp</result>
    <!-- 定义 error 的逻辑视图名, 对应 error.jsp 页面 -->
    <result name=" success ">/error.jsp</result>
    <!-- 定义 welcome 的逻辑视图名, 对应 welcome.jsp 页面 -->
    <result name="success">/welcome.jsp</result>
</action>
```

对比上面的 Action 配置与前面的 Action 配置，我们发现该 Action 配置片段中增加了 input 逻辑视图的配置，该逻辑视图映射到 login.jsp 页面。

前面已经提到：当用户提交请求时，请求得到 execute 方法处理之前，先会被 validate 方法处理，如果该方法处理结束后，Action 的 fieldErrors 里的校验错误不为空，请求将被转发给 input 逻辑视图。如果我们不输入用户名、密码而直接提交表单，将看到如图 2.13 所示的界面。



图 2.13 输入校验的界面

看到这里也许读者觉得非常神奇：我们仅仅在 Action 添加了数据校验错误，并未在输入页面输出这些校验错误信息，但图 2.13 所示的页面，却可以看到页面已经输出了这些校验信息——这是因为 Struts 2 的标签，上面的 JSP 页面中表单使用的并不是 HTML 表单，而是用了 `<s:form .../>` 标签，Struts 2 的 `<s:form .../>` 标签已经具备了输出校验错误的能力。

提示 Struts 2 的 `<s:form .../>` 默认已经提供了输出校验错误的能力。

但上面的程序还存在一个问题：校验信息的国际化。查看上面的 Action 类代码发现：重写 validate 方法时，如果发生校验失败的问题，校验错误的提示信息是以硬编码方式写死了——这就失去了国际化的能力。

实际上，ActionSupport 类已经提供了国际化信息的能力，它提供了一个 `getText(String key)` 方法，该方法用于从资源文件中获取国际化信息。为了让校验信息支持国际化，再次改写 Action 里的 validate 方法，改写后的 validate 方法代码如下：

```
//执行数据校验的 validate 方法
public void validate()
{
    //如果用户名为空，或者为空字符串
    if (getUsername() == null || getUsername().trim().equals(""))
    {
        //添加校验错误提示，使用 getText 方法来使提示信息国际化
        addFieldError("username", getText("user.required"));
    }
    if (getPassword() == null || getPassword().trim().equals(""))
    {
        addFieldError("password", getText("pass.required"));
    }
}
```


在上面的 `validate` 方法中，添加校验错误提示时，并不是直接给出了错误提示的字符串，而是调用了 `getText` 方法来获取错误提示。因为在 `Action` 中，使用 `getText` 方法来获取了两个国际化提示：`user.required` 和 `pass.required`，因此应该在国际化资源文件中添加这两条提示信息。

提示 `ActionSupport` 增加了让提示信息国际化的能力，`ActionSupport` 提供的 `getText` 方法可以根据资源文件加载获得国际化提示信息。

此时，如果没有任何输出，直接提交登录表单，将看到如图 2.14 所示的界面。

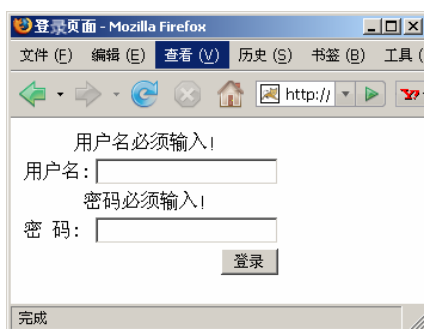


图 2.14 国际化数据校验的错误提示

2.8.2 使用 Struts 2 的校验框架

上面的输入校验是通过重写 `ActionSupport` 类的 `validate` 方法实现的，这种方法虽然不错，但需要大量重写的 `validate` 方法——毕竟，重复书写相同的代码不是一件吸引人的事情。

类似于 `Struts 1`，`Struts 2` 也允许通过定义配置文件来完成数据校验。`Struts 2` 的校验框架实际上是基于 `XWork` 的 `validator` 框架。

下面还是使用原来的 `Action` 类（即不重写 `validate` 方法），却增加一个校验配置文件，校验配置文件通过使用 `Struts 2` 已有的校验器，完成对表单域的校验。`Struts 2` 提供了大量的数据校验器，包括表单域校验器和非表单域校验器两种。

本应用主要使用了 `requiredstring` 校验器，该校验器是一个必填校验器——指定某个表单域必须输入。

下面是校验规则的定义文件：

```
<?xml version="1.0" encoding="GBK"?>
<!-- 指定校验规则文件的 DTD 信息 -->
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<!-- 校验规则定义文件的根元素 -->
<validators>
  <!-- 校验第一个表单域: username -->
  <field name="username">
    <!-- 该表单域必须填写 -->
    <field-validator type="requiredstring">
      <!-- 如果校验失败，显示 user.required 对应的信息 -->
      <message key="user.required"/>
    </field-validator>
  </field>
</validators>
```

```
</field-validator>
</field>
<!-- 校验第二个表单域: password -->
<field name="password">
    <field-validator type="requiredstring">
        <!-- 如果校验失败, 显示 pass.required 对应的信息 -->
        <message key="pass.required"/>
    </field-validator>
</field>
</validators>
```

定义完该校验规则文件后, 该文件的命名应该遵守如下规则:

ActionName-validation.xml: 其中 ActionName 就是需要校验的 Action 的类名。

因此上面的校验规则文件应该命名为 “LoginAction-validation.xml”, 且该文件应该与 Action 类的 class 文件位于同一个路径下。因此, 将上面的校验规则文件放在 WEB-INF/classes/lee 路径下即可。

当然, 在 struts.xml 文件的 Action 定义中, 一样需要定义 input 的逻辑视图名, 将 input 逻辑视图映射到 login.jsp 页面。

如果不输入用户名、密码而提交表单, 将再次看到如图 2.14 所示的界面。在这种校验方式下, 无需书写校验代码, 只需要通过配置文件指定校验规则即可, 因此提供了更好的可维护性。

2.9 本章小结

本章以一个 HelloWorld 应用为例, 简要介绍了 Struts 2 MVC 框架的基本流程, 从 Action 类基本流程控制讲起, 详细介绍了如何开发一个 Struts 2 应用。本章的后面部分在基本 Struts 2 应用基础上, 介绍了一些 Struts 2 的深入应用, 包括在 Action 中访问 HttpSession 状态, 将 Action 处理结果传回 JSP 页面显示, 本应用也综合应用了 Struts 2 的标签库、数据校验、程序国际化等常用功能。通过阅读本章的内容, 读者应该对 Struts 2 框架有一个大致的掌握。

本章要点

- Struts 1 框架的基本知识
- 使用 Struts 1 框架开发 Web 应用
- WebWork 框架的基本知识
- 使用 WebWork 框架开发 Web 应用
- 在 Eclipse 中整合 Tomcat
- 使用 Eclipse 开发 Web 应用
- 为 Web 应用增加 Struts 2 支持
- Struts 2 框架的 MVC 组件
- Struts 2 框架的流程
- 通过 web.xml 文件加载 Struts 2 框架
- 通过 struts.properties 文件配置 Struts 2 属性
- struts.xml 文件的结构
- 前面一章已经大致介绍了 Struts 2 应用的基本情况，通过前一章的学习，读者应该对 Struts 2 框架有了一个大致的掌握。但对于各知识点的细节，还需要进一步学习，本章将详细介绍 Struts 2 框架的基础部分。

因为 Struts 2 是在 WebWork 和 Struts 1 的基础上发展起来的，因此掌握一定的 WebWork 和 Struts 1 框架的知识，将对于掌握 Struts 2 框架大有裨益。特别是 WebWork 方面的开发经验，绝大部分都可以适用于 Struts 2 应用的开发。本章将会简要介绍 Struts 1 和 WebWork 框架的使用。

本章还将介绍在 Eclipse IDE 工具中开发 Struts 2 应用，并且详细介绍 Struts 2 配置文件的各个细节。

3.1 Struts 1 的 MVC 实现

Struts 1 是世界上最“古老”的 MVC 框架，它出现于 2001 年。Struts 1 以 ActionServlet 作为核心控制器，由 ActionServlet 负责拦截用户的所有请求。Struts 1 框架有 3 个重要组成部分：Action、ActionForm 和 ActionForward 对象。

3.1.1 下载和安装 Struts 1 框架

本节的示例程序基于 Struts 1 的 1.2.9 版，请读者也下载该版本的 Struts 1 框架，下载和安装 Struts 1 框架请按如下步骤进行。

① 登录 struts.apache.org 站点，下载 Struts 的合适版本。解压缩下载到的压缩包，解压缩后，发现有如下文件结构：

- contrib: 该路径下存放了 Struts 1 表达式语言支持的类库和标签库文件。
- lib: 该路径下存放了 Struts 1 框架的核心类库。
- webapps: 该路径下存放了 Struts 1 框架的示例应用及文档应用。
- 另外还一些关于 Struts 的说明和 LICENSE 等文档

② 将 lib 下的 JAR 文件全部复制到 WEB-INF/lib 下，并将 TLD 文件全部复制到 WEB-INF 下，再将 validator-rules.xml 文件也复制到 WEB-INF 下。如果需要用到 Struts-EL（表达式）语言，还应将 contrib 下的 JAR 文件复制到 WEB-INF/lib 下。

③ 为了在编译程序时可以导入 Struts 1 类库，还应将 struts.jar 文件增加到环境变量中。

3.1.2 实现 ActionForm

Struts 1 使用单独的 ActionForm 封装请求参数，当用户提交一个请求时，Struts 1 将使用单独的 ActionForm 来封装所有的请求参数。从结构上看，ActionForm 就是一个 POJO：它需要为用户的每个请求参数都提供一个对应的属性，并为该属性设置相应的 setter 和 getter 方法——但 ActionForm 的设计不是 POJO，它必须继承 Struts 1 的 ActionForm 基类。

下面是一个简单的登录页面，包含两个表单域，分别代表用户名和密码。该登录页面的代码如下：

```
<%@ page language="java" contentType="text/html; charset=gb2312" errorPage="" %>
<!-- 导入 Struts 的三个标签库-->
<%@include file="taglibs.jsp"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<!-- 使用国际化资源文件的 key 输出标题-->
<title><bean:message key="login"/></title>
</head>
<body>
<!-- 输出登录页面标题 -->
<h3><bean:message key="loginTitle"/></h3>
<!-- 输出 Action 处理完后的错误信息。 -->
<font color="red">${requestScope.err}</font>
<!-- 通过 messagePresent 判断输入校验后的出错提示是否存在 -->
<logic:messagesPresent>
  <!-- 输出出错提示 -->
  <bean:message key="errors.header"/>
  <ul>
    <html:messages id="error">
      <li><bean:write name="error"/></li>
    </html:messages>
  </ul><hr />
```

```
</logic:messagesPresent>
<!-- 下面是登录表单 -->
<html:form action="login.do">
    <!-- 使用国际化信息来输出标签 -->
    <bean:message key="username"/><html:text property="username"/><br>
    <bean:message key="pass"/><html:password property="pass"/><br>
    <html:submit><bean:message key="login"/></html:submit><br>
</html:form>
</body>
</html>
```



图 3.1 系统登录的页面

在浏览器中浏览该页面时，看到该页面包含了一个简单的登录表单。该页面的界面如图 3.1 所示。

该页面中有两个表单域，这两个表单域封装了需要向服务器发送的请求参数，对于 Struts 应用而言，请求参数是通过 `ActionForm` 来封装的，`ActionForm` 非常类似一个 POJO（定义几个属性，为每个属性提供 `setter` 和 `getter` 方法），但该 `ActionForm` 需要继承 `ActionForm`，或者是它的子类。因为本应用使用了 Struts 的校验框架，因此本应用中的 `ActionForm` 继承了 `ValidatorForm` 类。下面是本应用中 `ActionForm` 类的代码：

```
public class LoginForm extends ValidatorForm
{
    //下面两个属性用于封装两个表单域的请求参数
    private String username;
    private String pass;
    //以下是属性字段的系列 get、set 方法
    public String getUsername()
    {
        return username;
    }
    public void setUsername(String username)
    {
        this.username = username;
    }
    public String getPass()
    {
        return pass;
    }
    public void setPass(String pass)
    {
        this.pass = pass;
    }
}
```

上面 `ActionForm` 类的代码非常简单，除了该类需要继承 `ValidatorForm` 之外。也正是因为该 `ActionForm` 继承了 `ValidatorForm`，从而导致了该类的污染，降低了该类的代码复用。

对于 Struts 1 而言，`ActionForm` 类是一个既烦琐又没有技术含量的类，写起来难免让

程序员觉得意兴索然。虽然有很多 IDE（集成开发环境）可以自动生成该 `ActionForm` 类，但 Struts 1 框架也感觉到大量重复书写该类是一个负担，后来提供了动态 `ActionForm`，让人可以避免书写 `ActionForm` 类。

✱ 注意 Struts 1 中的 `ActionForm` 类是一个非常简单的类，它实质上是一个普通的 `JavaBean`，但必须继承 `ActionForm` 类。除此之外，`ActionForm` 还可以使用动态 `FormBean`，从而允许通过配置文件来定义 `ActionForm`。

3.1.3 实现 Action

`Action` 就是用于处理用户请求的业务控制器，当用户请求发送到 `ActionServlet` 后，`ActionServlet` 拦截到用户请求，将请求转发到系统的业务控制器处理。`ActionServlet` 在转发用户请求时，会将请求参数封装成 `ActionForm` 实例，并将该 `ActionForm` 实例转发给 `Action` 实例。

`Action` 实例从 `ActionForm` 中取出用户请求参数，然后调用业务逻辑组件处理用户请求，并根据处理结果，调用不同的视图页面来呈现处理结果。

下面是系统的 `Action` 处理类代码。

```
public class LoginAction extends Action
{
    //必须重写该 execute 方法，该方法用于处理用户请求
    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) throws
        Exception
    {
        //获取封装用户请求参数的 ActionForm 实例
        LoginForm loginForm = (LoginForm) form;
        //从 ActionForm 中取出用户请求参数
        String username = loginForm.getUsername();
        String pass = loginForm.getPass();
        //处理用户请求
        if (username != null && username.equals("scott"))
        {
            return mapping.findForward("welcome");
        }
        else
        {
            return mapping.findForward("error");
        }
    }
}
```

上面的 `Action` 代码非常简单，甚至没有调用任何业务逻辑组件，只是直接判断用户请求参数的用户名和密码是否为 `scott` 和 `tiger`，如果用户名和密码正确，则返回 `welcome` 的 `ActionForward`，否则返回 `error` 的 `ActionForward`。

★ 注意 ActionForward 就是一个逻辑视图，通过在配置文件中定义 ActionForward 的映射，完成逻辑视图名和实际视图资源之间的映射。

Struts 1 的 Action 类与 Struts 2 的 Action 类有一定的类似性，都通过调用 execute 方法来处理用户请求。但最大的区别在于 Struts 1 Action 的 execute 方法与 Servlet API 耦合，但 Struts 2 Action 类的 execute 方法无需与 Servlet API 耦合。

3.1.4 配置 Struts 1 的 Action

实现了 Struts 1 的 Action 后，还需要在 struts-config.xml 文件中配置 Action，配置 Action 需要指定 Action 的实现类，以及 Action 处理请求的 URL。

配置 Action 时，还应该配置该 Action 对应的 ActionForm，每个 ActionForm 使用一个 <form-bean.../> 元素定义。定义 Action 和 ActionForm 之间的关联关系时，在定义 Action 的 <action.../> 元素中通过 name 属性指定与此 Action 关联的 ActionForm。

因为本应用还使用了 Struts 1 的数据校验框架，因此配置 <action.../> 元素时，还应该增加 validate 属性，并将该属性值设置成 true；还需要增加 input 属性，该属性指向输入校验失败后转入的视图资源。

下面是本应用的 struts-config.xml 文件代码。

```
<?xml version="1.0" encoding="GBK"?>
<!-- 指定 Struts 1 配置文件的 DTD 信息 -->
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
    "http://struts.apache.org/dtds/struts-config_1_2.dtd">
<!-- 指定 Struts 1 配置文件的根元素 -->
<struts-config>
    <!-- 所有的 ActionForm 都应该在 form-beans 元素中定义 -->
    <form-beans>
        <!-- 每个 form-bean 定义一个 ActionForm -->
        <form-bean name="loginForm" type="lee.LoginForm"/>
    </form-beans>
    <!-- 所有的 Action 都放在 action-mappings 元素中定义 -->
    <action-mappings>
        <!-- 定义 Action，其处理类为 lee.LoginAction，对应的 ActionForm 为 loginForm -->
        <action path="/login" type="lee.LoginAction" name="loginForm"
            validate="true" scope="request" input="/login.jsp">
            <!-- 当返回 welcome 的 ActionForward 时，转入 welcome.jsp 页 -->
            <forward name="welcome" path="/welcome.jsp" />
            <!-- 当返回 error 的 ActionForward 时，转入 error.jsp 页 -->
            <forward name="error" path="/error.jsp" />
        </action>
    </action-mappings>
    <!-- 加载国际化资源文件 -->
    <message-resources parameter="MyResource"/>
    <!-- 加载 Struts 1 校验框架的插件 -->
    <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
```

```
<!-- 指定数据校验的两个校验规则文件 -->
<set-property property="pathnames" value="/WEB-INF/validator-rules.xml,
    /WEB-INF/validation.xml" />
<set-property property="stopOnFirstError" value="true"/>
</plug-in>
</struts-config>
```

正如前面看到的，页面中使用大量国际化标签来输出国际化消息，因此本应用必须加载相应的国际化资源文件，本应用的国际化资源文件的 `baseName` 是 `MyResource`。指定应用的国际化资源文件通过 `<message-resources .../>` 元素指定。

本应用使用了 Struts 1 的校验框架来完成输入校验，因此在本应用中加载了两个校验规则文件，其中 `validator-rules.xml` 文件是一个系统规则文件，通常无需修改；`validation.xml` 文件是开发者定义的规则文件，该文件定义了需要校验的表单应该满足的规则。

下面是校验规则文件的代码。

```
<?xml version="1.0" encoding="GBK"?>
<!-- 校验规则定义文件 -->
<!DOCTYPE form-validation PUBLIC "-//Apache Software Foundation
    //DTD Commons Validator Rules Configuration 1.1.3//EN"
    "http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">
<!-- 校验规则文件的根元素 -->
<form-validation>
    <formset>
        <!-- 定义需要校验的 ActionForm，定义该 ActionForm 为 loginForm -->
        <form name="loginForm">
            <!-- 定义 username 表单域必须满足必填、模式匹配两个规则 -->
            <field property="username" depends="required,mask">
                <arg key="username" position="0"/>
                <!-- 指定模式匹配在正则表达式 -->
                <var>
                    <var-name>mask</var-name>
                    <var-value>^\w+$</var-value>
                </var>
            </field>
            <!-- 定义 pass 表单域必须满足必填、最小长度两个规则 -->
            <field property="pass" depends="required,minlength">
                <arg key="pass" position="0"/>
                <!-- 定义最小长度的规则 -->
                <arg name="minlength" key="{var:minlength}" resource="false"
                    position="1"/>
                <var>
                    <var-name>minlength</var-name>
                    <var-value>4</var-value>
                </var>
            </field>
        </formset>
    </form-validation>
```

上面的规则文件定义了表单域的两个表单必须满足的规则：`username` 是必填的，而且

必须匹配`^w+$`的正则表达式；`pass` 是必填的，并且至少有 4 位。

定义了上面的校验规则后，如果用户的输入不能通过该输入校验，则系统自动转入配置`<action .../>`元素时指定的 `input` 属性所对应的视图资源。

输入校验的提示信息都保存在国际化资源文件中，因此使用输入校验时，通常都建议使用国际化资源文件来保存校验提示信息。

❗ 提示 使用输入校验时，通常建议使用国际化资源文件来保存校验失败的提示信息。

3.1.5 完成应用流程

经过上面步骤，可以非常清楚地看出上面示例应用的流程，用户进入系统的 `login.jsp` 页面，用户输入请求信息，发送请求。如果用户的请求参数不能通过输入校验，则系统将请求转发到 `login.jsp` 页面，否则请求将被转发到业务逻辑控制器 `Action` 处。

`Action` 处理用户请求参数，如果处理结果为 `success` 的 `ActionForward`，则进入 `welcome.jsp` 页面；如果处理结果为 `error` 的 `ActionForward`，则进入 `error.jsp` 页面。

图 3.2 显示了该应用的顺序图。

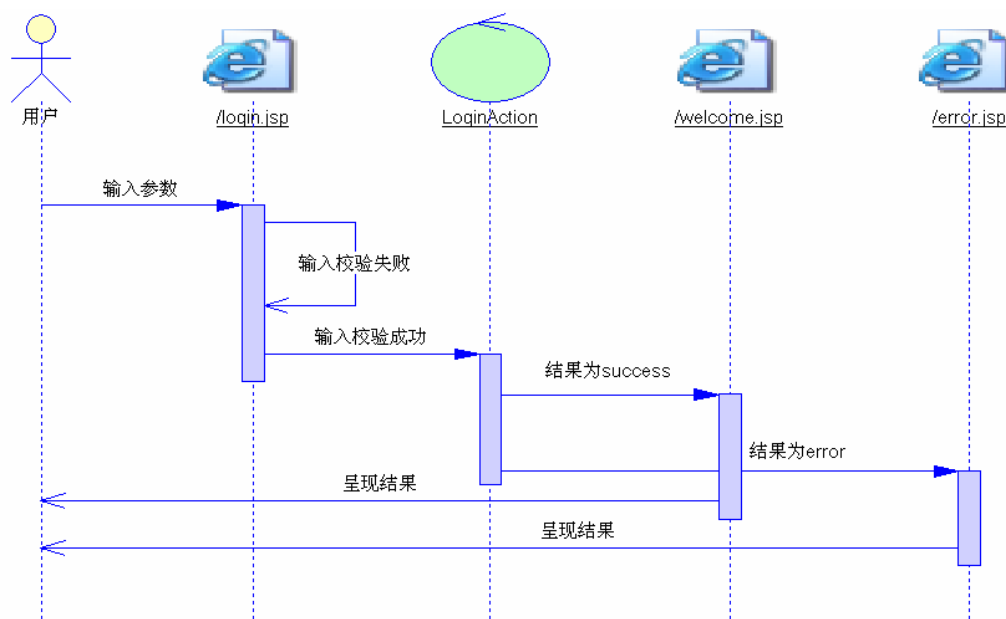


图 3.2 系统顺序图

如果用户输入的用户名正确、密码正确，用户登录成功，可以在 `welcome.jsp` 页面见到国际化的欢迎信息，并且通过 `Struts 1` 的标签输出登录的用户名。下面是 `welcome.jsp` 页面的代码。

```
<%@ page contentType="text/html; charset=gb2312" errorPage="error.jsp"%>
<!-- 导入 Struts 的三个标签库-->
<%@include file="taglibs.jsp"%>
<html>
<head>
<!-- 通过 Struts 1 标签输出国际化的页面标题 -->
<title><bean:message key="loginSuccess"/></title>
```

```
</head>
<body>
<h3><bean:message key="loginSuccess"/></h3>
<hr>
<!-- 通过 Struts 1 标签输出国际化的欢迎信息，并把登录用的用户名作为参数传入 -->
<bean:message key="welcome" arg0="{loginForm.username}"/>
</body>
</html>
```

如果用户登录成功，将看到如图 3.3 所示的页面。



图 3.3 登录成功页面

上面的 Struts 1 应用已经包含了 Struts 1 框架的核心部分，如 ActionForm、Action 实现，数据校验完成，应用的国际化支持等。

通过上面的示例应用，我们看到 Struts 1 框架的最大缺点是：Struts 1 应用的类与 Servlet API 耦合严重，而且大都需要继承 Struts 1 框架提供的基类，或实现 Struts 1 框架提供的接口，造成了代码污染。

3.2 WebWork 的 MVC 实现

WebWork 由 OpenSymphony 组织负责开发、维护，WebWork 2 前身是 Rickard Oberg 开发的 WebWork。现在我们所说的 WebWork，通常指的是 WebWork 2，它是由 Xwork 1 和 WebWork 2 两个项目组成的。

WebWork 紧紧以 ServletDispatcher 为核心，使用 ServletDispatcher 来处理所有的用户请求，它是整个 WebWork 框架的核心控制器。WebWork 支持多视图表示，视图部分可以使用 JSP、Velocity、FreeMarker、JasperReports、XML 等。WebWork 当前的最新版本是 2.2.5，本节所使用的范例是基于该版本的 WebWork 完成的。由于 WebWork 已经全面整合到 Struts 2 框架中，故 WebWork 不会再有新版本发布，即不会有 WebWork 2.3.X 发布。

3.2.1 WebWork 的下载和安装

WebWork 当前的最新稳定版本是 2.2.5，笔者建议读者也下载该版本。WebWork 的下载和安装按如下步骤进行。

① 登录 <http://www.opensymphony.com/webwork/> 站点，下载 WebWork 的最新稳定版，解压缩后发现如下的文件结构：

- dist: 该路径下存放了 WebWork 的两个编译版本。
- docs: 该路径下存放了 WebWork 的各种文档，包括 WebWork 的 Wiki 文档、API

文档等。

- **lib**: 该路径下存放了 WebWork 编译和运行所需的第三方类库, 其中 **default** 路径下的所有 JAR 文件是 WebWork 应用必需的类库文件。
- **src**: 该路径下存放了 WebWork 的源文件, 如果使用 WebWork 过程中遇到无法解决的问题, 可以参考该源文件解决。
- **webapps**: 该路径下存放了 WebWork 的示例应用, 这些示例应用是学习 WebWork 极好的资料。
- **webwork-2.2.5.jar**: 该文件是 WebWork 框架的核心类库文件。
- **webwork-src-2.2.5.jar**: 该压缩文件里包含了 WebWork 框架所有源代码。
- 此外还有一些 WebWork 框架的编译脚本文件及各种说明文件。

WebWork 通常都作为 Web 框架使用, 在 Web 应用中使用该框架。为 Web 应用增加 WebWork 框架的方法, 与在 Web 应用中增加其他框架的方法完全一样。下面介绍如何安装 WebWork 框架。

② 将 **webwork-2.2.5.jar** 文件复制到 Web 应用的 **WEB-INF/lib** 路径下, 再将 **lib/default** 路径下的 ***.jar** 文件全部复制到 **WEB-INF/lib** 下。

✱ 注意 使用 WebWork 框架必须有 **webwork-2.2.5.jar** 和 **lib/default** 路径下的全部 JAR 文件。至于 **lib** 路径下的其他 JAR 文件, 则是可选的——只有当应用中需要使用这些第三方框架时, 才需要这些类库文件。

☞ 为了可以编译 WebWork 应用程序, 建议将 **webwork-2.2.5.jar** 增加到 **CLASSPATH** 的环境变量中。当然也可以使用 **Ant** 工具, 则无需增加环境变量。

3.2.2 实现 Action

用户需要实现的 WebWork 核心组件依然是业务控制器, 业务控制器就是一个 **Action**, 该 **Action** 也包含一个 **execute** 方法, 但该方法不再包含任何 **Servlet API**, 也不包含任何 **WebWork API**。这个 **Action** 与 **Struts 1** 的 **Action** 不同, 它非常类似于一个普通的 **POJO**。

本应用的 **Action** 代码如下:

```
public class LoginAction extends ActionSupport
{
    //下面两个属性用于封装用户请求参数
    private String user;
    private String pass;
    //如下请求参数用于封装 Action 处理请求后的反馈
    private String tip;
    //user 属性对应的 setter 和 getter 方法
    public String getUser()
    {
        return user;
    }
    public void setUser(String user)
    {
        this.user = user;
    }
}
```

```
}
//pass 属性对应的 setter 和 getter 方法
public String getPass()
{
    return pass;
}
public void setPass(String pass)
{
    this.pass = pass;
}
//tip 属性对应的 setter 和 getter 方法
public String getTip()
{
    return tip;
}
//tip 属性对应的 setter 和 getter 方法
public void setTip(String tip)
{
    this.tip = tip;
}
//处理用户请求的 execute 方法
public String execute() throws Exception
{
    if (getUser().equals("scott")
        && getPass().equals("tiger"))
    {
        setTip(getText("loginSuccess"));
        return SUCCESS;
    }
    else
    {
        setTip(getText("loginFail"));
        return ERROR;
    }
}
}
```

上面 Action 类的代码与 Struts 1 的 Action 类存在如下几个区别：

- WebWork 的 Action 类无需与 Servlet API 耦合，从而可以更方便地进行单元测试。
- WebWork 的 Action 类可无需 ActionForm 的支持，它本身包含了两个属性用于封装用户的请求参数。
- WebWork 的 Action 类里包含的属性不仅可以封装用户请求参数，还可以封装 Action 的处理结果。
- WebWork 的 Action 的 execute 方法返回的逻辑视图是一个标准字符串，而不是一个 ActionForward 对象。

本应用中的 Action 继承了 WebWork 的 ActionSupport 基类，继承该基类导致了 Action 的代码污染，WebWork 不强制要求 Action 类继承 ActionSupport 基类，但继承该基类可以更便捷地实现 Action 类，例如程序国际化。

上面的 Action 实现类中包含了两个国际化信息，这两个国际化信息对应的 key 分别为 loginSuccess 和 loginFail。为了在 Action 类中访问这两个国际化信息，需要在 Action 类中使用 getText 方法，该方法可以输出对应 key 的国际化信息。该 getText 方法正是来自于 ActionSupport 基类的方法。

✱ 注意 WebWork 的 Action 类不强制要求使用 ActionForm 类封装用户请求参数。但可使用模型驱动模式，在模型驱动模式下，Action 不再负责封装请求参数，也不负责封装 Action 的处理结果。而是将请求参数和处理结果放在模型中封装——模型就是一个类似于 ActionForm 的对象，但该对象是一个 POJO，无需继承任何 WebWork 基类。相对于直接使用 Action 封装请求参数的模式，模型驱动模式的编程更加复杂，因为需要额外增加一个模型类；但模型驱动模式更易理解：Action 专门负责处理用户请求，而模型则专门负责封装数据。

因为上面的 Action 封装了 user 和 pass 两个请求参数，故对应的 JSP 页面也应该包含这样两个表单域。下面是对应 JSP 页面的代码。

```
<%@ page contentType="text/html; charset=GBK"%>
<!-- 导入 WebWork 的标签库文件 -->
<%@ taglib prefix="ww" uri="/webwork"%>
<html>
<head>
    <!-- 使用国际化信息标签输出国际化标题 -->
    <title><ww:text name="loginTitle"/></title>
</head>
<body>
<ww:text name="loginTitle"/>
<hr>
<!-- 使用 WebWork 标签生成登录表单 -->
<ww:form method="post" action="login">
    <!-- 生成登录用的用户名文本框 -->
    <ww:textfield name="user" label="%{getText('user')}" />
    <!-- 生成登录用的密码文本框 -->
    <ww:textfield name="pass" label="%{getText('pass')}" />
    <!-- 生成登录按钮 -->
    <ww:submit value="%{getText('submit')}" />
</ww:form>
</body>
</html>
```

读者可能已经注意到，上面的表单定义中，大量使用了 WebWork 的标签库，借助于 WebWork 标签库，可以更简单的生成页面的表单元素。因此，上面页面中的表单定义中，甚至无需为表单域指定对应的 Label 信息，但可生成如下表单定义。图 3.4 显示了在浏览器中浏览该页面的效果。

当使用<ww:textfield name="user" label="%{getText('user')}" />来生成一个文本框时，该标签负责生成一个名为 user 的文本输入框，该输入框对应的 Label 通过%{getText('user')}表

达式生成输出, 该表达式负责输出一个国际化信息, 该国际化信息对应的 key 为 user。

3.2.3 配置 Action

实现上面的 Action 后, 需要在应用中配置该 Action。WebWork 使用一个 xwork.xml 文件来配置 Action。

在配置 WebWork 的 Action 之前, 需要先配置 Web 应用加载 WebWork 框架。让 Web 应用加载 WebWork 框架就是配置 WebWork 的核心控制器, WebWork 的核心控制器是 ServletDispatcher。

因为 Web 应用默认只加载 web.xml 文件, 因此为了加载 WebWork 框架, 必须在 web.xml 文件中配置 WebWork 的核心控制器: ServletDispatcher。修改后的 web.xml 文件如下:

```
<?xml version="1.0" encoding="GBK"?>
<!-- 指定 Web 应用配置文件的 Schema 信息 -->
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.
    com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
  <!-- 配置 WebWork 的核心 Servlet -->
  <servlet>
    <!-- 指定 WebWork 核心 Servlet 的名字 -->
    <servlet-name>webwork</servlet-name>
    <!-- 指定 WebWork 核心 Servlet 的实现类 -->
    <servlet-class>com.opensymphony.webwork.dispatcher.ServletDispatcher
    </servlet-class>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <!-- 配置 WebWork 核心 Servlet 的映射 -->
  <servlet-mapping>
    <!-- 配置 WebWork 核心 Servlet 映射的 URL -->
    <servlet-name>webwork</servlet-name>
    <url-pattern>*.action</url-pattern>
  </servlet-mapping>
</web-app>
```

指定了上面的 web.xml 配置文件后, Web 应用会自动加载 WebWork 框架, 并加载 Web 应用的 WEB-INF/classes 路径下的 xwork.xml 文件。xwork.xml 文件就是 WebWork 的配置文件, 该文件主要配置应用的 Action, 以及 Action 对应的 URL。

```
<?xml version="1.0" encoding="GBK"?>
<!-- 下面指定了 WebWork 配置文件的 DTD 信息 -->
<!DOCTYPE xwork PUBLIC
  "-//OpenSymphony Group//XWork 1.1//EN"
  "http://www.opensymphony.com/xwork/xwork-1.1.dtd">
<!-- xwork 是 WebWork 配置文件的根元素 -->
<xwork>
```

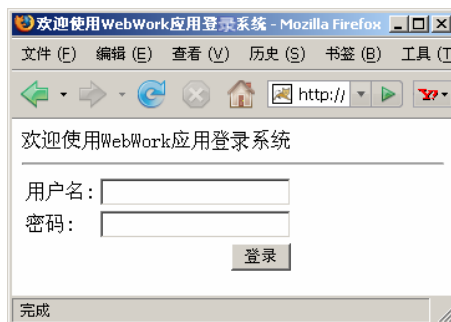


图 3.4 登录页面的效果

```
<!-- 导入 WebWork 默认配置文件 -->
<include file="webwork-default.xml" />
<!-- WebWork 默认将 Action 配置放在 package 下管理 -->
<package name="default" extends="webwork-default">
  <!-- 指定 login 的 Action, 该 Action 的实现类为 lee.LoginAction -->
  <action name="Login" class="lee.LoginAction">
    <!-- 指定 input 逻辑视图对应的物理视图资源 -->
    <result name="input">/login.jsp</result>
    <!-- 指定 error 逻辑视图对应的物理视图资源 -->
    <result name="error">/error.jsp</result>
    <!-- 指定 success 逻辑视图对应的物理视图资源 -->
    <result name="success">/success.jsp</result>
  </action>
</package>
</xwork>
```

在上面的 `xwork.xml` 配置文件中, 所有的 `Action` 放在 `package` 元素下管理, 因为 WebWork 以包来管理所有的 `Action`, 所以全部的 `action` 元素都作为 `package` 元素的子元素存在。

★ 注意 WebWork 以 `package` 来管理 WebWork 的 `Action`, 故所有的 `action` 元素都是 `package` 元素的子元素。

配置 `action` 元素时, 必须指定两个主要属性: `name` 属性和 `class` 属性。其中 `name` 属性指定了该 `Action` 负责处理的 URL, 而 `class` 属性指定了该 `Action` 对应的实现类。例如上面的 `Action` 配置, 该 `Action` 的 `name` 为 `login`, 表示该 `Action` 负责处理向 `login.action` 的 URL 发送的请求, 处理请求的 Java 类为 `lee.LoginAction`。


实际上, WebWork 可以包含两个配置文件, 还有一个 `webwork.properties` 配置文件, 这个文件是一个标准的属性文件, 由一系列的 WebWork 属性的 `key` 和 `value` 组成。下面是 `webwork.properties` 文件的代码。

```
#指定该 WebWork 应用处于开发模式下, 如果出错, 将输出更多出错调试信息
webwork.devMode = true
#指定每次请求都重载 WebWork 的配置文件
webwork.configuration.xml.reload=true
#指定 WebWork 应用的国际化资源文件的 baseName 为 mess
webwork.custom.il8n.resources=mess
```

因为本应用处于开发阶段, 故设置了 WebWork 应用的开发模式为 `true`, 并且每次请求都会重载 WebWork 的配置文件。上面配置文件的最后一行指定了 WebWork 的国际化资源文件的 `baseName` 为 `mess`。

上面配置 WebWork 应用时, 指定了 `Action` 处理后逻辑视图名和物理视图之间的对应关系: "input"字符串对应/login.jsp 页面, "error"字符串对应/error.jsp 页面, "success"字符串对应/success.jsp 页面。对比上面的 `Action` 类代码, 发现该类的 `execute` 方法分别返回了 `SUCCESS` 和 `ERROR` 两个字符串常量——它们的定义包含在 `ActionSupport` 类中, 其值分别为 "success"和 "error"字符串。

这就是本系统流程：当用户输入的用户名为 `scott`，且密码为 `tiger` 时，Action 返回一个 `success` 的逻辑视图，该逻辑视图对应为 `/success.jsp` 页面；否则将进入 `/error.jsp` 页面。那么 `input` 逻辑视图的作用呢？`input` 逻辑视图是输入校验失败后默认转入的视图页面。

 提示 与 Struts 1 类似，WebWork 采用 `input` 逻辑视图来作为输入校验失败后转入的视图。

3.2.4 完成数据校验

WebWork 可以支持手动完成输入校验，但这不是本应用将采用的方式，因为手动完成输入校验需要重写 Action 类的 `validate()` 方法，需要大量书写重复代码，相当烦琐。本应用利用了 XWork 的校验框架来完成数据校验，完成数据校验只需要在 Action class 文件所在的路径下存放对应的校验文件即可。

下面是本应用的数据校验文件代码。

```
<?xml version="1.0" encoding="gb2312"?>
<!-- WebWork 校验文件的 DTD 信息 -->
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork
    Validator 1.0.2//EN"
    "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<!-- validators 是校验文件的根元素 -->
<validators>
    <!-- 使用表单校验器校验 user 表单域 -->
    <field name="user">
        <!-- 指定 user 表单域必须满足 requiredstring（必填）规则 -->
        <field-validator type="requiredstring">
            <!-- 违反输入校验后的提示信息 -->
            <message key="user.required"/>
        </field-validator>
    </field>
    <!-- 使用表单校验器校验 pass 表单域 -->
    <field name="pass">
        <!-- 指定 pass 表单域必须满足 requiredstring（必填）规则 -->
        <field-validator type="requiredstring">
            <message key="pass.required"/>
        </field-validator>
        <!-- 指定 pass 表单域必须满足 stringlength（长度范围）规则 -->
        <field-validator type="stringlength">
            <!-- 指定最小长度 -->
            <param name="minLength">3</param>
            <!-- 指定最大长度 -->
            <param name="maxLength">6</param>
            <message key="pass.length"/>
        </field-validator>
    </field>
</validators>
```

上面的校验文件使用了两个表单校验器：`requiredstring` 和 `stringlength`，分别对应用的两个表单域进行校验，分别校验用户名必须满足必填规则，密码必须满足必填且字符串长

度必须在 3 和 6 之间的规则。

如果用户输入不满足该规则，则看到如图 3.5 所示的界面。

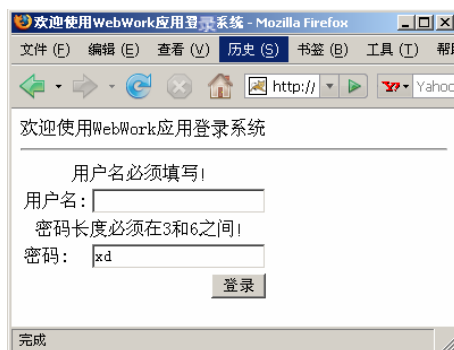


图 3.5 输入校验的效果

通过上面的示范代码，可以看出，结合 XWork 的校验框架，可以大大简化 WebWork 应用的数据校验。

对比上面的 Struts 1 框架和 WebWork 框架的使用，不难发现 WebWork 与 Struts 2 的设计更类似。

3.2.5 完成应用流程

正如我们见到的，如果我们输入了合适的用户名和密码，Action 处理完用户请求后，将会封装一个名为 `tip` 的属性在结果里，并转入 `success` 的逻辑视图，该逻辑视图对应了 `/success.jsp` 页面。该页面的代码非常简单，它只是简单地使用 WebWork 标签来输出 `tip` 的提示信息。页面代码如下：

```
<%@ page contentType="text/html; charset=GBK"%>
<!-- 导入 WebWork 的标签库 -->
<%@ taglib prefix="ww" uri="/webwork"%>
<html>
<head>
  <!-- 输出 tip 提示信息 -->
  <title><ww:property value="tip"/></title>
</head>
<body>
<h3><ww:property value="tip"/></h3>
<!-- 使用 WebWork 的国际化标签来输出国际化提示 -->
<ww:text name="welcomeMsg">
  <!-- 传入国际化提示信息的参数 -->
  <ww:param value="user"/>
</ww:text>
</body>
</html>
```

在上面的页面中，我们使用了 `<ww.text .../>` 标签来输出国际化消息，该标签输出的国际化消息如下：

```
welcomeMsg=欢迎您，{0}，您已经成功登录本系统！
```

看到上面的国际化消息需要传入一个参数，我们通过为<ww:text .../>标签指定一个<ww:param .../>子标签来传入参数：第一个<ww:param .../>子元素为第一个参数指定值，第二个<ww:param .../>子元素为第二个参数指定值……依此类推。

上面代码使用了<ww:param value="user"/>来传入参数，该参数的值就是 Action 中 user 属性值。当用户输入的用户名和密码正确时，将看到如图 3.6 所示的界面。



图 3.6 登录成功的界面

3.3 在 Eclipse 中开发 Struts 2

前面已经详细介绍了如何开发一个 Struts 2 应用，但没有使用任何 IDE 工具，那种开发方式虽然能让读者更清楚开发的细节，但对于开发效率却不如使用 IDE 工具的开发效率高。本节将以 Eclipse 为例，介绍如何在 IDE 工具中开发 Struts 2 应用。

3.3.1 创建 Web 应用

笔者所使用的 Eclipse 已经安装了 MyEclipse 插件，因此可以很方便地建立一个 Web 应用，并将其部署在 Web 服务器中。

下面以 Tomcat 5520 为例，介绍如何在 Eclipse 中整合 Tomcat 服务器，新建一个 Web 应用，并将该应用部署到 Tomcat 服务器中。

现在开始在 Eclipse 中整合 Tomcat 服务器，实际上，如果我们安装了 MyEclipse 插件，则可以在 Eclipse 中整合大部分的 Java EE 服务器（包括 Web 服务器）。但如果我们没安装任何插件，Eclipse 则很难与任何 Java EE 服务器整合。

提示 Eclipse 必须安装了相应插件后才具有丰富的功能。为了开发 Java EE 应用，通常都推荐安装 MyEclipse 插件，该插件是一个商业产品，必须支付相应费用才可使用。

为了在 Eclipse 中整合 Tomcat 服务器，请按如下步骤进行。

① 单击 Eclipse 窗口里的“Window”菜单，在出现的“Window”的系列菜单项中选择“Preferences”菜单项，将弹出如图 3.7 所示的界面。

该 Preferences 设置窗口是 Eclipse 大部分参数的设置窗口，该窗口还可设置 Eclipse 插件的相关属性。MyEclipse 插件的相关属性就是在这个窗口里设置的，而整合 Java EE 服务器只是 MyEclipse 插件的相关属性。

② 单击图 3.7 所示窗口左边导航树中的“MyEclipse”节点，然后单击“MyEclipse”节点下的“Application Servers”子节点，将看到如图 3.8 所示的界面。

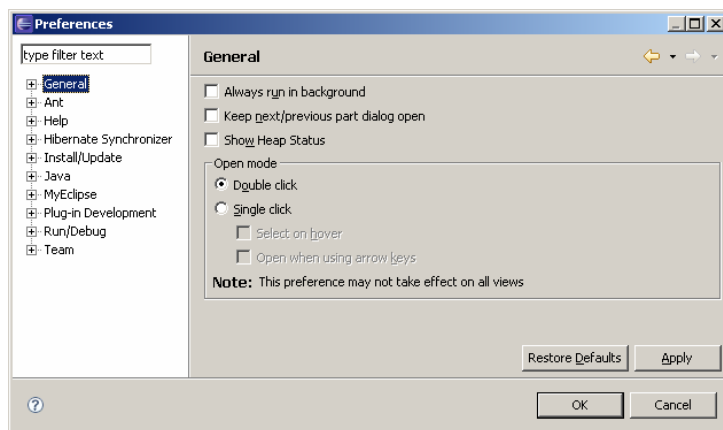


图 3.7 Eclipse 的 Preferences 设置窗口

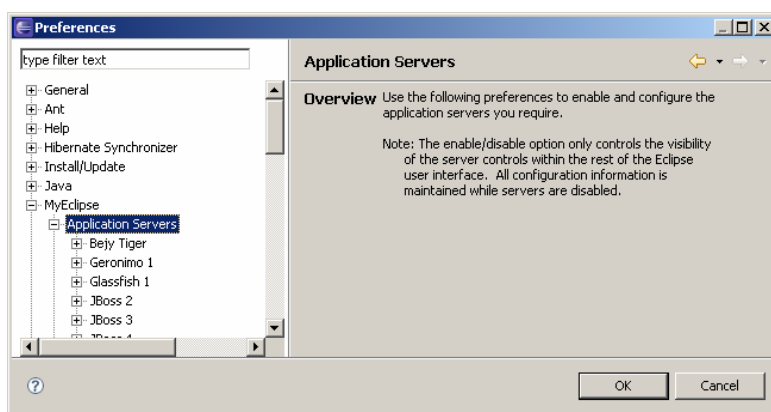


图 3.8 在 Eclipse 中整合应用服务器

③ 拖动图 3.8 所示窗口左边的垂直滚动条，我们可以发现大部分应用服务器的名字，例如 Jboss、Jetty、Resin 等，当然也可以发现我们需要使用的 Tomcat 节点，然后单击 Tomcat 5 节点，出现如图 3.9 所示的界面。

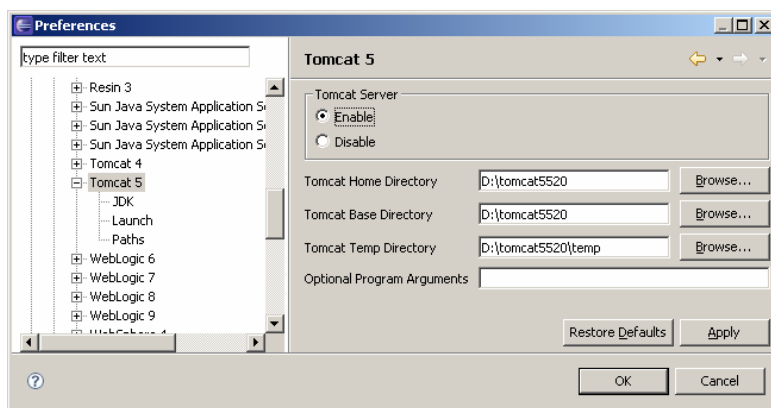


图 3.9 整合 Tomcat5

④ 在图 3.9 右上方看到两个单选框：Enable 和 Disable，为了整合 Tomcat，显然应该选中 Enable 单选框。选中该单选框后，还应该下面的三个文本输入框内输入相应的地址，例如，Tomcat Home Directory 里应该输入 Tomcat 的安装路径，笔者的 Tomcat 安装在 D 盘的 tomcat5520 路径下，故在该文本框内输入了 D:\tomcat5520 的内容。通常，一旦我们输入了正确的 Tomcat 安装路径，下面的两个文本框会自动输入，无需用户干预。

⑤ 单击图 3.9 所示窗口右下方的“OK”按钮，返回 Eclipse 主界面，Eclipse 和 Tomcat 的整合成功。

下面介绍如何在 Eclipse 中建立一个 Web 应用，在 Eclipse 中建立一个 Web 应用也可借助于 MyEclipse 的帮助。借助于 MyEclipse 建立 Web 应用按如下步骤进行。

① 单击 Eclipse 主界面中的“File”菜单，在出现的菜单中单击“New”下的“Other...”菜单项，将弹出一个如图 3.10 所示的对话框。

② 在如图 3.10 所示的对话框中，选中“Web Project”节点，表明将要创建一个 Web 应用，然后单击“Next”按钮，看到如图 3.11 所示的对话框。

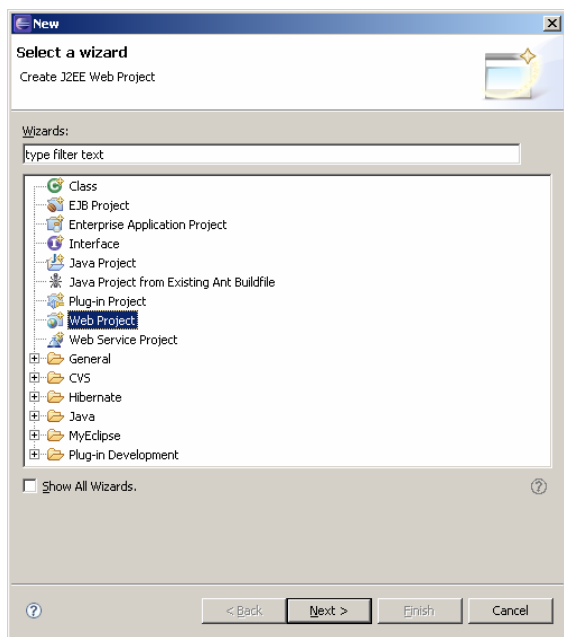


图 3.10 新建 Web 应用的对话框

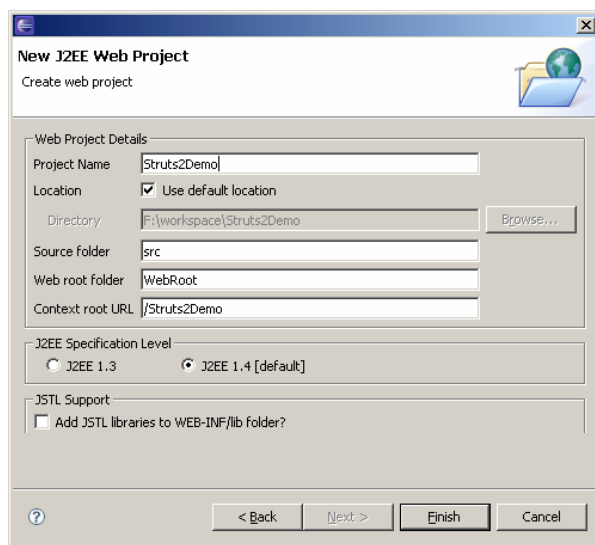


图 3.11 新建 Web 应用

③ 在如图 3.11 所示对话框的 Project Name 文本框中输入应用的名字，本应用使用 Struts2Demo 作为应用的名字，输入完成后可看到如图 3.11 所示的对话框。单击“Finish”按钮，新建 Web 应用成功。

3.3.2 增加 Struts 2 支持

为了让 Web 应用具有 Struts 2 支持功能，必须将 Struts 2 框架的核心类库增加到 Web 应用中。将 Struts 2 框架下 lib 路径下的 struts2-core-2.0.6.jar、xwork-2.0.1.jar 等 Struts 2 框架的核心类库复制到 Web 应用的 lib 路径下，也就是“%workspace%\Struts2Demo\WebRoot\WEB-INF\lib”路径下。

返回 Eclipse 的主界面，在 Eclipse 主界面的左上角资源导航中看到了 struts2Demo 节点，选中该节点，然后按 F5 键，将看到 Eclipse 主界面左上角资源导航中出现如图 3.12 所示的界面。

看到如图 3.12 所示的界面，表明该 Web 应用已经加入了 Struts 2 的核心类库，但还需要修改 web.xml 文件，让该文件负责加载 Struts 2 框架。

在如图 3.12 所示的导航树中，单击“WEB-INF”节点前的加号，展开该节点，看到该节点下包含的 web.xml 文件子节点。

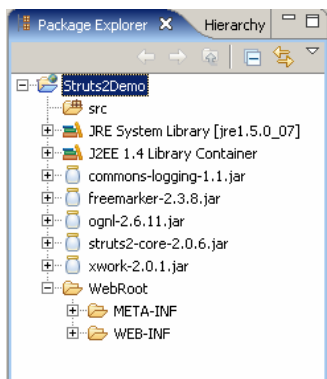


图 3.12 添加了 Struts 2 类库

单击 web.xml 文件节点，编辑该文件，编辑该文件没有丝毫特殊之处，同样是在 web.xml 文件中定义 Struts 2 的核心 Filter，并定义该 Filter 所拦截的 URL 模式。至此，该 Web 应用完全具备了 Struts 2 框架的支持。

3.3.3 部署 Struts 2 应用

本应用的功能非常类似于第 2 章所介绍的 Struts 2 应用，因此本节所使用的 JSP 页面与第 2 章所使用的 JSP 页面也非常类似，故此处不再给出这些 JSP 页面的代码。

在 Eclipse 中添加 JSP 页面也是很简单的，此处不再赘述。

建立了这些 JSP 页面后，单击 Eclipse 主界面上部署 Web 应用的工具按钮，部署 Web 应用和启动服务器的两个按钮如图 3.13 所示。



图 3.13 在 Eclipse 中部署 Web 应用和启动服务器的按钮

在 Eclipse 中部署 Web 应用请按如下步骤进行。

① 单击部署 Web 应用的按钮，弹出如图 3.14 所示的对话框。

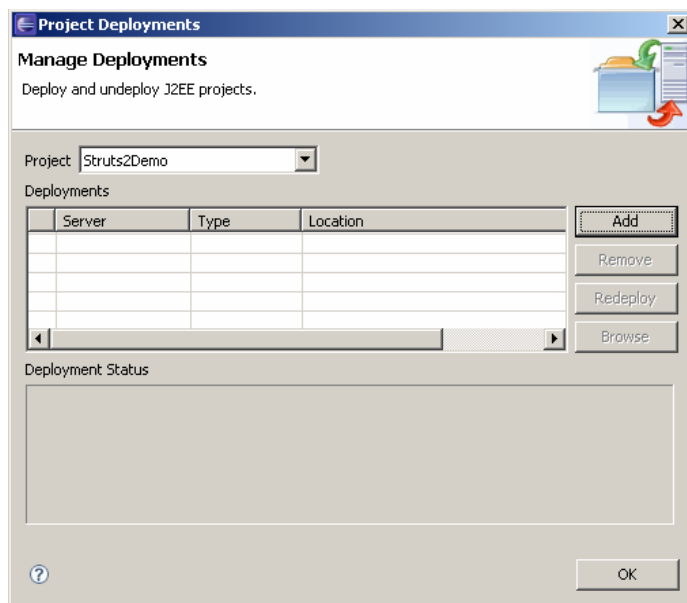


图 3.14 部署 Web 应用的对话框

② 在图 3.14 所示对话框的上面看到选择项目的下拉列表，选中需要部署的 Web 应用，例如本示例就是 Struts2Demo。单击右边的“Add”按钮，该按钮用于添加想要部署到的

Web 服务器，将看到如图 3.15 所示的对话框。

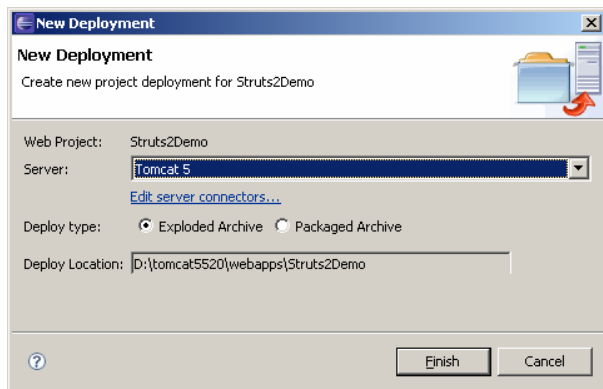


图 3.15 部署到应用服务器

③ 在图 3.15 所示对话框上面的 Server 下拉列表中选中“Tomcat 5”选项，其他选项设置参照图 3.15，然后单击“Finish”按钮，返回图 3.14 所示对话框，单击“OK”按钮，Web 应用部署成功。

④ Web 应用部署成功后，单击如图 3.13 所示工具条中的“启动服务器”按钮旁的小三角，出现下拉菜单，选择“Tomcat 5”下的“Start”菜单项，如图 3.16 所示，启动 Tomcat 服务器。

⑤ 在浏览器中访问刚才的 Struts2Demo 应用，将可看到本应用登录页面。以 Tomcat 的端口为 8888 为例，应该在浏览器中访问如下地址：<http://localhost:8888/Struts2Demo/login.jsp>，将看到如图 3.17 所示的界面。

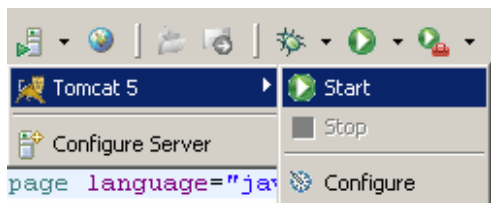


图 3.16 启动服务器

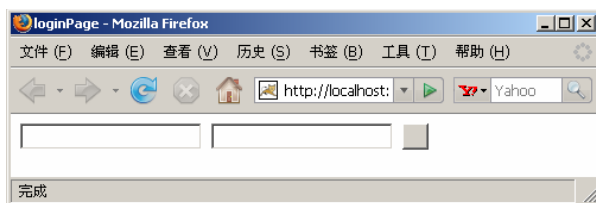


图 3.17 部署成功后的登录界面

在图 3.17 中看到该登录页面的文本框没有任何提示信息，按钮上也没有文本，这是因为本应用缺少国际化资源文件。增加该应用所需的国际化资源文件，该资源文件与前面应用的资源文件相同，此处不再给出。

为了让 Struts 2 应用加载该国际化资源文件，还应该使用 `struts.properties` 文件，该文件内指定如下一行代码：

```
#指定 Struts 2 的国际化资源文件的 baseName 为 messageResource  
struts.custom.i18n.resources=messageResource
```

至此，Struts 2 应用已经获得了国际化支持。如果在浏览器中再次浏览该页面，将看到如图 3.18 所示的界面。

3.3.4 增加应用的 Action

在 Eclipse 工具中新建一个 Java 类，该 Java 类的类名为“LoginAction”，其代码与第 2 章所使用的 LoginAction 类代码相同，将该类文件保存在 Struts2Demo 应用的 src 路径下的 lee 目录下。



图 3.18 增加了国际化资源文件后的登录页面

新建一个 Java 文件的方法非常简单，此处不再赘述。本应用的 Action 类的代码与第 2 章的 Action 类完全一样，此处也不再给出。

增加了 Struts 2 的 Action 类后，还需要增加对应的配置文件，也是通过单击 Eclipse 的“File”菜单，然后单击“New”下的“File”菜单项来操作的。

❗ 提示 因为 Eclipse 暂时还没有专门的 Struts 2 插件，因此只能用最原始的方式来建立一个 Struts 2 的配置文件。等到专门的 Struts 2 插件被开发出来后，应该可以有更简单的方法来建立 Struts 2 的配置文件。

建立该文件后，为该文件增加 Struts 2 的 Action 定义，定义该 Action 与前面介绍的基本一致，需要定义 Action 的 name 属性、class 属性等。增加 Action 定义后的 struts.xml 文件代码如下：

```
<?xml version="1.0" encoding="GBK"?>
<!-- 指定 Struts 2 配置文件的 DTD 信息 -->
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<!-- 指定 Struts 2 配置文件的根元素 -->
<struts>
    <!-- 所有的 Action 定义都应该放在 package 下 -->
    <package name="lee" extends="struts-default">
        <!-- 定义 Action -->
        <action name="Login" class="lee.LoginAction">
            <!-- 定义三个逻辑视图和物理资源之间的映射 -->
            <result name="input">/login.jsp</result>
            <result name="error">/error.jsp</result>
            <result name="success">/welcome.jsp</result>
        </action>
    </package>
</struts>
```

至此，整个 Struts 2 应用完全建立成功，如果读者还需要增加数据校验，则可增加对应的校验文件，并将该文件放在与对应 Action 源文件的同一路径下即可。

3.4 Struts 2 的基本流程

经过前面介绍，我们已经基本了解了 Struts 2 框架的 MVC 实现。大致上，Struts 2 框架由 3 个部分组成：核心控制器 `FilterDispatcher`、业务控制器和用户实现的业务逻辑组件。在这 3 个部分里，Struts 2 框架提供了核心控制器 `FilterDispatcher`，而用户需要实现业务控制器和业务逻辑组件。

3.4.1 核心控制器：FilterDispatcher

`FilterDispatcher` 是 Struts 2 框架的核心控制器，该控制器作为一个 `Filter` 运行在 Web 应用中，它负责拦截所有的用户请求，当用户请求到达时，该 `Filter` 会过滤用户请求。如果用户请求以 `action` 结尾，该请求将被转入 Struts 2 框架处理。

Struts 2 框架获得了 `*.action` 请求后，将根据 `*.action` 请求的前面部分决定调用哪个业务逻辑组件，例如，对于 `login.action` 请求，Struts 2 调用名为 `login` 的 `Action` 来处理该请求。

Struts 2 应用中的 `Action` 都被定义在 `struts.xml` 文件中，在该文件中定义 `Action` 时，定义了该 `Action` 的 `name` 属性和 `class` 属性，其中 `name` 属性决定了该 `Action` 处理哪个用户请求，而 `class` 属性决定了该 `Action` 的实现类。

Struts 2 用于处理用户请求的 `Action` 实例，并不是用户实现的业务控制器，而是 `Action` 代理——因为用户实现的业务控制器并没有与 `Servlet API` 耦合，显然无法处理用户请求。而 Struts 2 框架提供了系列拦截器，该系列拦截器负责将 `HttpServletRequest` 请求中的请求参数解析出来，传入到 `Action` 中，并回调 `Action` 的 `execute` 方法来处理用户请求。

显然，上面的处理过程是典型的 AOP（面向切面编程）处理方式。图 3.19 显示了这种处理模型。

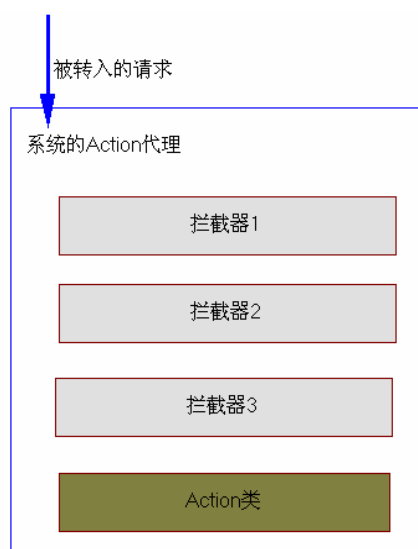


图 3.19 Struts 2 的拦截器和 Action

从图 3.19 中可以看出，用户实现的 `Action` 类仅仅是 Struts 2 的 `Action` 代理的代理目标。用户实现的业务控制器（`Action`）则包含了对用户请求的处理。用户的请求数据包含在 `HttpServletRequest` 对象里，而用户的 `Action` 类无需访问 `HttpServletRequest` 对象。拦截器

负责将 `HttpServletRequest` 里的请求数据解析出来，并传给业务逻辑组件 `Action` 实例。

3.4.2 业务控制器

正如从图 3.19 所看到的，业务控制器组件就是用户实现 `Action` 类的实例，`Action` 类里通常包含了一个 `execute` 方法，该方法返回一个字符串——该字符串就是一个逻辑视图名，当业务控制器处理完用户请求后，根据处理结果不同，`execute` 方法返回不同字符串——每个字符串对应一个视图名。

程序员开发出系统所需要的业务控制器后，还需要配置 Struts 2 的 `Action`，即需要配置 `Action` 的如下三个部分定义：

- `Action` 所处理的 URL。
- `Action` 组件所对应的实现类。
- `Action` 里包含的逻辑视图和物理资源之间的对应关系。

每个 `Action` 都要处理一个用户请求，而用户请求总是包含了指定 URL。当 `FilterDispatcher` 拦截到用户请求后，根据请求的 URL 和 `Action` 处理 URL 之间的对应关系来处理转发。

3.4.3 Struts 2 的模型组件

实际上，模型组件已经超出了 MVC 框架的覆盖范围。对于 Struts 2 框架而言，通常没有为模型组件的实现提供太多的帮助。

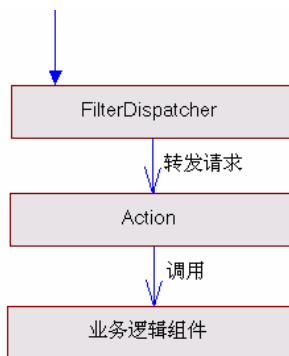


图 3.20 控制器调用模型组件

Java EE 应用里的模型组件，通常指系统的业务逻辑组件。而隐藏在系统的业务逻辑组件下面的，可能还包含了 DAO、领域对象等组件。

通常，MVC 框架里的业务控制器会调用模型组件的方法来处理用户请求。也就是说，业务逻辑控制器不会对用户请求进行任何实际处理，用户请求最终由模型组件负责处理。业务控制器只是中间负责调度的调度器，这也是称 `Action` 为控制器的原因。

图 3.20 显示了这种处理流程。



提示 在图 3.20 中看到 `Action` 调用业务逻辑组件的方法。当控制器需要获得业务逻辑组件实例时，通常并不会直接获取业务逻辑组件实例，而是通过工厂模式来获得业务逻辑组件的实例；或者利用其他 IoC 容器（如 Spring 容器）来管理业务逻辑组件的实例。

3.4.4 Struts 2 的视图组件

Struts 2 已经改变了 Struts 1 只能使用 JSP 作为视图技术的现状，Struts 2 允许使用其他的模板技术，如 FreeMarker、Velocity 作为视图技术。

当 Struts 2 的控制器返回逻辑视图名时，逻辑视图并未与任何的视图技术关联，仅仅


是返回一个字符串，该字符串作为逻辑视图名。

当我们在 `struts.xml` 文件中配置 Action 时，不仅需要指定 Action 的 `name` 属性和 `class` 属性，还要为 Action 元素指定系列 `result` 子元素，每个 `result` 子元素定义一个逻辑视图和物理视图之间的映射。前面所介绍的应用都使用了 JSP 技术作为视图，故配置 `result` 子元素时没有指定 `type` 属性，默认使用 JSP 作为视图资源。

如果需要在 Struts 2 中使用其他视图技术，则可以在配置 `result` 子元素时，指定相应的 `type` 属性即可。例如，如果需要使用 FreeMarker，则为 `result` 指定值为 `freemarker` 的 `type` 属性；如果想使用 Velocity 模板技术作为视图资源，则为 `result` 指定值为 `velocity` 的 `type` 属性……

3.4.5 Struts 2 的运行流程

经过上面介绍，我们发现 Struts 2 框架的运行流程非常类似于 WebWork 框架的流程。

 提示 在 Struts 2 的官方站点，我们可以找到如下说法：Essentially, Struts 2.0 is the technical equivalent of WebWork 2.3. Aside from the package and property renaming, it isn't much different than, say, migrating from WebWork 2.1 to 2.2——意思是说：Struts 2.0 技术等同于 WebWork 2.3 框架，除了包和属性被改名外。从 WebWork 2.2 迁移到 Struts 2 不会比从 WebWork 2.1 迁移到 WebWork 2.2 更复杂。

这里我们可以看到，Struts 2 其实就是 WebWork 2.2 的升级版，这也就不难理解：为什么 WebWork 和 Struts 2 如此相似！

因此，Struts 2 的运行流程与 WebWork 的运行流程完全相同，读者可以参看图 1.8 来了解 Struts 2 的运行流程。

3.5 Struts 2 的基本配置

前面大致了解了 Struts 2 框架的基本内容，但这些基本内容都必须建立在 Struts 2 的配置文件基础之上，这些配置文件的配置信息也是 Struts 2 应用的核心部分。

3.5.1 配置 web.xml 文件

任何 MVC 框架都需要与 Web 应用整合，这就不得不借助于 `web.xml` 文件，只有配置在 `web.xml` 文件中 Servlet 才会被应用加载。

通常，所有的 MVC 框架都需要 Web 应用加载一个核心控制器，对于 Struts 2 框架而言，需要加载 `FilterDispatcher`，只要 Web 应用负责加载 `FilterDispatcher`，`FilterDispatcher` 将会加载应用的 Struts 2 框架。

因为 Struts 2 将核心控制器设计成 Filter，而不是一个普通 Servlet。故为了让 Web 应用加载 `FilterDispatcher`，只需要在 `web.xml` 文件中配置 `FilterDispatcher` 即可。

配置 `FilterDispatcher` 的代码片段如下：

```
<!-- 配置 Struts 2 框架的核心 Filter -->
<filter>
  <!-- 配置 Struts 2 核心 Filter 的名字 -->
  <filter-name>struts</filter-name>
  <!-- 配置 Struts 2 核心 Filter 的实现类 -->
  <filter-class>org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
  <init-param>
    <!-- 配置 Struts 2 框架默认加载的 Action 包结构 -->
    <param-name>actionPackages</param-name>
    <param-value>org.apache.struts2.showcase.person</param-value>
  </init-param>
  <!-- 配置 Struts 2 框架的配置提供者类 -->
  <init-param>
    <param-name>configProviders </param-name>
    <param-value>lee.MyConfigurationProvider</param-value>
  </init-param>
</filter>
```

正如上面看到的，当配置 Struts 2 的 FilterDispatcher 类时，可以指定一系列的初始化参数，为该 Filter 配置初始化参数时，其中有 3 个初始化参数有特殊意义：

- **config**：该参数的值是一个以英文逗号(,)隔开的字符串，每个字符串都是一个 XML 配置文件的位置。Struts 2 框架将自动加载该属性指定的系列配置文件。
- **actionPackages**：该参数的值也是一个以英文逗号(,)隔开的字符串，每个字符串都是一个包空间，Struts 2 框架将扫描这些包空间下的 Action 类。
- **configProviders**：如果用户需要实现自己的 ConfigurationProvider 类，用户可以提供一个或多个实现了 ConfigurationProvider 接口的类，然后将这些类的类名设置成该属性的值，多个类名之间以英文逗号(,)隔开。

除此之外，还可在此处配置 Struts 2 常量，每个<init-param>元素配置一个 Struts 2 常量，其中<param-name>子元素指定了常量 name，而<param-value>子元素指定了常量 value。



提示 关于 Struts 2 常量的讲解，请参阅本书的 4.1.2 节。

在 web.xml 文件中配置了该 Filter，还需要配置该 Filter 拦截的 URL。通常，我们让该 Filter 拦截所有的用户请求，因此使用通配符来配置该 Filter 拦截的 URL。

下面是配置该 Filter 拦截 URL 的配置片段：

```
<!-- 配置 Filter 拦截的 URL -->
<filter-mapping>
  <!-- 配置 Struts 2 的核心 FilterDispatcher 拦截所有用户请求 -->
  <filter-name>struts</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

配置了 Struts 2 的核心 FilterDispatcher 后，基本完成了 Struts 2 在 web.xml 文件中的配置。

如果 Web 应用使用了 Servlet 2.3 以前的规范，因为 Web 应用不会自动加载 Struts 2 框架的标签文件，因此必须在 web.xml 文件中配置加载 Struts 2 标签库。

配置加载 Struts 2 标签库的配置片段如下：

```
<!-- 手动配置 Struts 2 的标签库 -->
<taglib>
  <!-- 配置 Struts 2 标签库的 URI -->
  <taglib-uri>/s</taglib-uri>
  <!-- 指定 Struts 2 标签库定义文件的位置 -->
  <taglib-location>/WEB-INF/struts-tags.tld</taglib-location>
</taglib>
```

在上面配置片段中，指定了 Struts 2 标签库配置文件物理位置：/WEB-INF/struts-tags.tld，因此我们必须手动复制 Struts 2 的标签库定义文件，将该文件放置在 Web 应用的 WEB-INF 路径下。

如果 Web 应用使用 Servlet 2.4 以上的规范，则无需在 web.xml 文件中配置标签库定义，因为 Servlet 2.4 规范会自动加载标签库定义文件。

 提示 Struts 2 的标签库定义文件包含在 struts2-core-2.0.6.jar 文件里，在 struts2-core-2.0.6.jar 文件的 META-INF 路径下，包含了一个 struts-tag.tld 文件，这个文件就是 Struts 2 的标签库定义文件，Servlet 2.4 规范会自动加载该标签库文件。

对于 Servlet 2.4 以上的规范，Web 应用自动加载该标签库定义文件。加载 struts-tag.tld 标签库定义文件时，该文件的开始部分包含如下代码片段：

```
<taglib>
  <!-- 定义标签库的版本 -->
  <tlib-version>2.2.3</tlib-version>
  <!-- 定义标签库所需的 JSP 版 -->
  <jsp-version>1.2</jsp-version>
  <short-name>s</short-name>
  <!-- 定义 Struts 2 标签库的 URI -->
  <uri>/struts-tags</uri>
  ...
</taglib>
```

因为该文件中已经定义了该标签库的 URI：struts-tags，这就避免了在 web.xml 文件中重新定义 Struts 2 标签库文件的 URI。

3.5.2 struts.xml 配置文件

Struts 框架的核心配置文件就是 struts.xml 配置文件，该文件主要负责管理 Struts 2 框架的业务控制器 Action。

在默认情况下，Struts 2 框架将自动加载放在 WEB-INF/classes 路径下的 struts.xml 文件。在大部分应用里，随着应用规模的增加，系统中 Action 数量也大量增加，导致 struts.xml 配置文件变得非常臃肿。

为了避免 struts.xml 文件过于庞大、臃肿，提高 struts.xml 文件的可读性，我们可以将一个 struts.xml 配置文件分解成多个配置文件，然后在 struts.xml 文件中包含其他配置文件。

下面的 struts.xml 文件中就通过 include 手动导入了一个配置文件：struts-part1.xml 文件，通过这种方式，就可以将 Struts 2 的 Action 按模块配置在多个配置文件中。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- 指定 Struts 2 配置文件的 DTD 信息 -->
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<!-- 下面是 Struts 2 配置文件的根元素 -->
<struts>
    <!-- 通过 include 元素导入其他配置文件 -->
    <include file="struts-part1.xml" />
    ...
</struts>
```

通过这种方式，Struts 2 提供了一种模块化的方式来管理 struts.xml 配置文件。

除此之外，Struts 2 还提供了一种插件式的方式来管理配置文件。用 WinRAR 等解压缩软件打开 struts2-core-2.0.6.jar 文件，看到如图 3.21 所示的文件结构，在光标选中的一行，看到有一个 struts-default.xml 文件。



图 3.21 struts2-core-2.0.6.jar 压缩文件的文件结构

查看 struts-default.xml 文件，看到该文件代码片段如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- 指定 Struts 2 配置文件的 DTD 信息 -->
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<!-- Struts 2 配置文件的根元素 -->
<struts>
    <!-- 下面定义了 Struts 2 框架的一些基础 Bean -->
    <bean class="com.opensymphony.xwork2.ObjectFactory" name="xwork" />
    <bean type="com.opensymphony.xwork2.ObjectFactory" name="struts"
        class="org.apache.struts2.impl.StrutsObjectFactory" />
    ...
    <!-- 下面是一些静态注入 Bean 定义 -->
    <bean class="com.opensymphony.xwork2.util.OgnlValueStack" static="true" />
    <bean class="org.apache.struts2.dispatcher.Dispatcher" static="true" />
```

```
...
<!-- 下面定义 Struts 2 的默认包空间 -->
<package name="struts-default">
  <!-- 定义 Struts 2 内建支持的结果类型 -->
  <result-types>
    <!-- 定义 Action 链 Result 类型 -->
    <result-type name="chain" class="com.opensymphony.xwork2.
      ActionChainResult"/>
    <!-- 定义 Dispatcher 的 Result 类型，并设置 default="true"，
      指定该结果 Result 是默认的 Result 类型 -->
    <result-type name="dispatcher"
      class="org.apache.struts2.dispatcher.Servlet
        DispatcherResult" default="true"/>
    <!-- 定义 FreeMarker 的 Result 类型 -->
    <result-type name="freemarker"
      class="org.apache.struts2.views.freemarker.
        FreemarkerResult"/>
    ...
  </result-types>
  <!-- 定义 Struts 2 内建的拦截器 -->
  <interceptors>
    <interceptor name="alias" class="com.opensymphony.xwork2.
      interceptor.AliasInterceptor"/>
    <interceptor name="autowiring"
      class="com.opensymphony.xwork2.spring.interceptor.
        ActionAutowiringInterceptor"/>
    ...
    <!-- 定义基本拦截器栈 -->
    <interceptor-stack name="basicStack">
      <interceptor-ref name="exception"/>
      <interceptor-ref name="servlet-config"/>
      <interceptor-ref name="prepare"/>
      <interceptor-ref name="checkbox"/>
      <interceptor-ref name="params"/>
      <interceptor-ref name="conversionError"/>
    </interceptor-stack>
    <!-- 还有系列拦截器栈 -->
    ...
  </interceptors>
  <!-- 定义默认的拦截器栈引用 -->
  <default-interceptor-ref name="defaultStack"/>
</package>
</struts>
```

上面的代码并未全部列出 `struts-default.xml` 文件，只是列出了每个元素的代表。上面配置文件中定义了一个名字为 `struts-default` 的包空间，该包空间里定义了 Struts 2 内建的 Result 类型，还定义了 Struts 2 内建的系列拦截器，以及由不同拦截器组成的拦截器栈，文件的最后还定义了默认的拦截器引用。

这个 `struts-default.xml` 文件是 Struts 2 框架的默认配置文件，Struts 2 框架每次都会自动加载该文件。查看我们前面使用的 `struts.xml` 文件，看到我们自己定义的 `package` 元素有

如下代码片段：

```
<!-- 指定 Struts 2 配置文件的根元素 -->
<struts>
  <!-- 配置名为 lee 的包空间，继承 struts-default 包空间 -->
  <package name="lee" extends="struts-default">
    ...
  </package>
</struts>
```


在上面配置文件中，名为 lee 的包空间，继承了名为 struts-default 的包空间，struts-default 包空间定义在 struts-default.xml 文件中。可见，Struts 2 框架默认会加载 struts-default.xml 文件。

不仅如此，Struts 2 框架提供了一种类似 Eclipse 的扩展方式，它允许以一种“可插拔”的方式来安装插件，例如后面将要介绍的 Spring 插件、JSF 插件等，它们都提供了一个类似 struts2-Xxx-plugin.jar 的文件——这个文件就是插件安装文件，只要将该文件放在 Web 应用的 WEB-INF/lib 路径下，Struts 2 框架将自动加载该框架。

使用 WinRAR 工具打开 struts2-spring-plugin2.06.jar 文件，找到一个 struts-plugin.xml 文件，打开该文件，该文件的代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- 指定 Struts 2 的 DTD 信息 -->
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <!-- 定义一个名字为 spring 的 ObjectFactory -->
  <bean type="com.opensymphony.xwork2.ObjectFactory" name="spring"
    class="org.apache.struts2.spring.StrutsSpringObjectFactory" />
  <!-- 指定名字为 spring 的 ObjectFactory 是 Struts 2 的 ObjectFactory -->
  <constant name="struts.objectFactory" value="spring" />
  <!-- 定义名为 spring-default 的包空间 -->
  <package name="spring-default">
    <!-- 定义整合 Spring 框架所必需的拦截器列表 -->
    <interceptors>
      <interceptor name="autowiring"
        class="com.opensymphony.xwork2.spring.interceptor.
          ActionAutowiringInterceptor"/>
      <interceptor name="sessionAutowiring"
        class="org.apache.struts2.spring.interceptor.
          SessionContextAutowiringInterceptor"/>
    </interceptors>
  </package>
</struts>
```

在上面配置文件中，配置了 Struts 2 与 Spring 框架整合必需的常量、拦截器等——这也是 Struts 2 框架使用 Spring 插件必需的配置。只要将 struts2-spring-plugin2.06.jar 文件放在 WEB-INF/lib 路径下，Struts 2 框架将自动加载该文件。

 提示 如果用户开发属于自己的 Struts 2 插件，只要将对应的 struts-plugin.xml 文件放在 JAR 文件中，Struts 2 框架将自动加载该文件。通过这种方式，Struts 2 框架允许使用可插拔的方式管理 Struts 2 的插件。

3.5.3 struts.properties 配置文件


Struts 2 框架有两个核心配置文件，其中 struts.xml 文件主要负责管理应用中的 Action 映射，以及该 Action 包含的 Result 定义等。除此之外，Struts 2 框架还包含一个 struts.properties 文件，该文件定义了 Struts 2 框架的大量属性，开发者可以通过改变这些属性来满足应用的需求。

struts.properties 文件是一个标准的 Properties 文件，该文件包含了系列的 key-value 对象，每个 key 就是一个 Struts 2 属性，该 key 对应的 value 就是一个 Struts 2 属性值。

struts.properties 文件通常放在 Web 应用的 WEB-INF/classes 路径下。实际上，只要将该文件放在 Web 应用的 CLASSPATH 路径下，Struts 2 框架就可以加载该文件。

现在的问题是，struts.properties 文件的哪些 key 是有效的？即 struts.properties 文件里包含哪些属性是有效的 Struts 2 属性。下面列出了可以在 struts.properties 中定义的 Struts 2 属性。

- **struts.configuration**: 该属性指定加载 Struts 2 配置文件的配置文件管理器。该属性的默认值是 org.apache.Struts2.config.DefaultConfiguration，这是 Struts 2 默认的配置文件管理器。如果需要通过自己的配置管理器，开发者则可以实现一个实现 Configuration 接口的类，该类可以自己加载 Struts 2 配置文件。
- **struts.locale**: 指定 Web 应用的默认 Locale。
- **struts.i18n.encoding**: 指定 Web 应用的默认编码集。该属性对于处理中文请求参数非常有用，对于获取中文请求参数值，应该将该属性值设置为 GBK 或者 GB2312。

 提示 当设置该参数为 GBK 时，相当于调用 HttpServletRequest 的 setCharacterEncoding 方法。

- **struts.objectFactory**: 指定 Struts 2 默认的 ObjectFactory Bean，该属性默认值是 spring。
- **struts.objectFactory.spring.autoWire**: 指定 Spring 框架的自动装配模式，该属性的默认值是 name，即默认根据 Bean 的 name 属性自动装配。
- **struts.objectFactory.spring.useClassCache**: 该属性指定整合 Spring 框架时，是否缓存 Bean 实例，该属性只允许使用 true 和 false 两个属性值，它的默认值是 true。通常不建议修改该属性值。
- **struts.objectTypeDeterminer**: 该属性指定 Struts 2 的类型检测机制，通常支持 tiger 和 notiger 两个属性值。
- **struts.multipart.parser**: 该属性指定处理 multipart/form-data 的 MIME 类型（文件上传）请求的框架，该属性支持 cos、pell 和 jakarta 等属性值，即分别对应使用 cos 的文件上传框架、pell 上传及 common-fileupload 文件上传框架。该属性的默认值为 jakarta。

✱ 注意 如果需要使用 cos 或者 pell 的文件上传方式,则应该将对应的 JAR 文件复制到 Web 应用中。例如,使用 cos 上传方式,则需要自己下载 cos 框架的 JAR 文件,并将该文件放在 WEB-INF/lib 路径下。

- **struts.multipart.saveDir:** 该属性指定上传文件的临时保存路径,该属性的默认值是 `javax.servlet.context.tempdir`。
- **struts.multipart.maxSize:** 该属性指定 Struts 2 文件上传中整个请求内容允许的最大字节数。
- **struts.custom.properties:** 该属性指定 Struts 2 应用加载用户自定义的属性文件,该自定义属性文件指定的属性不会覆盖 `struts.properties` 文件中指定的属性。如果需要加载多个自定义属性文件,多个自定义属性文件的文件名以英文逗号 (,) 隔开。
- **struts.mapper.class:** 指定将 HTTP 请求映射到指定 Action 的映射器,Struts 2 提供了默认的映射器: `org.apache.struts2.dispatcher.mapper.DefaultActionMapper`。默认映射器根据请求的前缀与 Action 的 `name` 属性完成映射。
- **struts.action.extension:** 该属性指定需要 Struts 2 处理的请求后缀,该属性的默认值是 `action`,即所有匹配 `*.action` 的请求都由 Struts 2 处理。如果用户需要指定多个请求后缀,则多个后缀之间以英文逗号 (,) 隔开。
- **struts.serve.static:** 该属性设置是否通过 JAR 文件提供静态内容服务,该属性只支持 `true` 和 `false` 属性值,该属性的默认属性值是 `true`。
- **struts.serve.static.browserCache:** 该属性设置浏览器是否缓存静态内容。当应用处于开发阶段时,我们希望每次请求都获得服务器的最新响应,则可设置该属性为 `false`。
- **struts.enable.DynamicMethodInvocation:** 该属性设置 Struts 2 是否支持动态方法调用,该属性的默认值是 `true`。如果需要关闭动态方法调用,则可设置该属性为 `false`。
- **struts.enable.SlashesInActionNames:** 该属性设置 Struts 2 是否允许在 Action 名中使用斜线,该属性的默认值是 `false`。如果开发者希望允许在 Action 名中使用斜线,则可设置该属性为 `true`。
- **struts.tag.altSyntax:** 该属性指定是否允许在 Struts 2 标签中使用表达式语法,因为通常都需要在标签中使用表达式语法,故此属性应该设置为 `true`,该属性的默认值是 `true`。
- **struts.devMode:** 该属性设置 Struts 2 应用是否使用开发模式。如果设置该属性为 `true`,则可以在应用出错时显示更多、更友好的出错提示。该属性只接受 `true` 和 `false` 两个值,该属性的默认值是 `false`。通常,应用在开发阶段,将该属性设置为 `true`,当进入产品发布阶段后,则该属性设置为 `false`。
- **struts.i18n.reload:** 该属性设置是否每次 HTTP 请求到达时,系统都重新加载资源文件。该属性默认值是 `false`。在开发阶段将该属性设置为 `true` 会更有利于开发,但在产品发布阶段应将该属性设置为 `false`。

! 提示 开发阶段将该属性设置了 `true`,将可以在每次请求时都重新加载国际化资源文件,从而可以让开发者看到实时开发效果;产品发布阶段应该将该属性设置为 `false`,是为了提供响应性能,每次请求都需要重新加载资源文件会大大降低应用的性能。

- **struts.ui.theme**: 该属性指定视图标签默认的视图主题, 该属性的默认值是 `xhtml`。
- **struts.ui.templateDir**: 该属性指定视图主题所需要模板文件的位置, 该属性的默认值是 `template`, 即默认加载 `template` 路径下的模板文件。
- **struts.ui.templateSuffix**: 该属性指定模板文件的后缀, 该属性的默认属性值是 `ftl`。该属性还允许使用 `ftl`、`vm` 或 `jsp`, 分别对应 `FreeMarker`、`Velocity` 和 `JSP` 模板。
- **struts.configuration.xml.reload**: 该属性设置当 `struts.xml` 文件改变后, 系统是否自动重新加载该文件。该属性的默认值是 `false`。
- **struts.velocity.configfile**: 该属性指定 `Velocity` 框架所需的 `velocity.properties` 文件的位置。该属性的默认值为 `velocity.properties`。
- **struts.velocity.contexts**: 该属性指定 `Velocity` 框架的 `Context` 位置, 如果该框架有多个 `Context`, 则多个 `Context` 之间以英文逗号 (,) 隔开。
- **struts.velocity.toolboxlocation**: 该属性指定 `Velocity` 框架的 `toolbox` 的位置。
- **struts.url.http.port**: 该属性指定 Web 应用所在的监听端口。该属性通常没有太大的用户, 只是当 `Struts 2` 需要生成 URL 时 (例如 `Url` 标签), 该属性才提供 Web 应用的默认端口。
- **struts.url.https.port**: 该属性类似于 `struts.url.http.port` 属性的作用, 区别是该属性指定的是 Web 应用的加密服务端口。
- **struts.url.includeParams**: 该属性指定 `Struts 2` 生成 URL 时是否包含请求参数。该属性接受 `none`、`get` 和 `all` 三个属性值, 分别对应于不包含、仅包含 `GET` 类型请求参数和包含全部请求参数。
- **struts.custom.i18n.resources**: 该属性指定 `Struts 2` 应用所需要的国际化资源文件, 如果有多份国际化资源文件, 则多个资源文件的文件名以英文逗号 (,) 隔开。
- **struts.dispatcher.parametersWorkaround**: 对于某些 Java EE 服务器, 不支持 `HttpServletRequest` 调用 `getParameterMap()` 方法, 此时可以设置该属性值为 `true` 来解决该问题。该属性的默认值是 `false`。对于 `WebLogic`、`Orion` 和 `OC4J` 服务器, 通常应该设置该属性为 `true`。
- **struts.freemarker.manager.classname**: 该属性指定 `Struts 2` 使用的 `FreeMarker` 管理器。该属性的默认值是 `org.apache.struts2.views.freemarker.FreemarkerManager`, 这是 `Struts 2` 内建的 `FreeMarker` 管理器。
- **struts.freemarker.wrapper.altMap**: 该属性只支持 `true` 和 `false` 两个属性值, 默认值是 `true`。通常无需修改该属性值。
- **struts.xslt.nocache**: 该属性指定 `XSLT Result` 是否使用样式表缓存。当应用处于开发阶段时, 该属性通常被设置为 `true`; 当应用处于产品使用阶段时, 该属性通常被设置为 `false`。
- **struts.configuration.files**: 该属性指定 `Struts 2` 框架默认加载的配置文件, 如果需要指定默认加载多个配置文件, 则多个配置文件的文件名之间以英文逗号 (,) 隔开。该属性的默认值为 `struts-default.xml, struts-plugin.xml, struts.xml`, 看到该属性值, 读者应该明白为什么 `Struts 2` 框架默认加载 `struts.xml` 文件了。

在有些时候, 开发者不喜欢使用额外的 `struts.properties` 文件, Struts 2 允许在 `struts.xml` 文件中管理 Struts 2 属性, 在 `struts.xml` 文件中通过配置 `constant` 元素, 一样可以配置这些属性。

`struts.xml` 配置片段如下:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- 指定 Struts 2 的 DTD 信息 -->
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <!-- 通过 constant 元素配置 Struts 2 的属性 -->
    <constant name=" struts.custom.i18n.resources " value="mess" />
    ...
</struts>
```

上面代码片段配置了一个常用属性: `struts.custom.i18n.resources`, 该属性指定了应用所需的国际化资源文件的 `baseName` 为 `mess`。

提示 Struts 2 提供了两种方式来管理 Struts 2 的属性: 既可以通过 `struts.properties` 文件来配置 Struts 2 属性, 也可通过在 `struts.xml` 文件中配置 `constant` 元素来配置 Struts 2 属性。

3.5.4 struts.xml 文件结构

`struts.xml` 文件是整个 Struts 2 框架的核心, 下面提供了一个最完整的 `struts.xml` 文件, 这个文件没有任何实际意义, 仅仅是一个 `struts.xml` 文件示范。

```
<?xml version="1.0" encoding="GBK"?>
<!-- 下面指定 Struts 2 配置文件的 DTD 信息 -->
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<!-- struts 是 Struts 2 配置文件的根元素 -->
<struts>
    <!-- 下面元素可以出现 0 次, 也可以无限多次 -->
    <constant name="" value="" />
    <!-- 下面元素可以出现 0 次, 也可以无限多次 -->
    <bean type="" name="" class="" scope="" static="" optional="" />
    <!-- 下面元素可以出现 0 次, 也可以无限多次 -->
    <include file="" />
    <!-- package 元素是 Struts 配置文件的核​​心, 该元素可以出现 0 次, 或者无限多次 -->
    <package name="必填的包名" extends="" namespace="" abstract=""
        externalReferenceResolver>
        <!-- 该元素可以出现, 也可以不出现, 最多出现一次 -->
        <result-types>
            <!-- 该元素必须出现, 可以出现无限多次 -->
            <result-type name="" class="" default="true|false">
```

```
<!-- 下面元素可以出现 0 次，也可以无限多次 -->
<param name="参数名">参数值</param>*
</result-type>
</result-types>
<!-- 该元素可以出现，也可以不出现，最多出现一次 -->
<interceptors>
  <!-- 该元素的 interceptor 元素和 interceptor-stack 至少出现其中之一，
  也可以二者都出现 -->
  <!-- 下面元素可以出现 0 次，也可以无限多次 -->
  <interceptor name="" class="">
    <!-- 下面元素可以出现 0 次，也可以无限多次 -->
    <param name="参数名">参数值</param>*
  </interceptor>
  <!-- 下面元素可以出现 0 次，也可以无限多次 -->
  <interceptor-stack name="">
    <!-- 该元素必须出现，可以出现无限多次-->
    <interceptor-ref name="">
      <!-- 下面元素可以出现 0 次，也可以无限多次 -->
      <param name="参数名">参数值</param>*
    </interceptor-ref>
  </interceptor-stack>
</interceptors>
<!-- 下面元素可以出现 0 次，也可以无限多次 -->
<default-interceptor-ref name="">
  <!-- 下面元素可以出现 0 次，也可以无限多次 -->
  <param name="参数名">参数值</param>
</default-interceptor-ref>
<!-- 下面元素可以出现 0 次，也可以无限多次 -->
<default-action-ref name="">
  <!-- 下面元素可以出现 0 次，也可以无限多次 -->
  <param name="参数名">参数值</param>*
</default-action-ref>?
<!-- 下面元素可以出现 0 次，也可以无限多次 -->
<global-results>
  <!-- 该元素必须出现，可以出现无限多次-->
  <result name="" type="">
    <!-- 该字符串内容可以出现 0 次或多次 -->
    映射资源
    <!-- 下面元素可以出现 0 次，也可以无限多次 -->
    <param name="参数名">参数值</param>*
  </result>
</global-results>
<!-- 下面元素可以出现 0 次，也可以无限多次 -->
<global-exception-mappings>
  <!-- 该元素必须出现，可以出现无限多次-->
  <exception-mapping name="" exception="" result="">
    异常处理资源
    <!-- 下面元素可以出现 0 次，也可以无限多次 -->
    <param name="参数名">参数值</param>*
  </exception-mapping>
</global-exception-mappings>
<action name="" class="" method="" converter="">
```

```

<!-- 下面元素可以出现 0 次，也可以无限多次 -->
<param name="参数名">参数值</param>*
<!-- 下面元素可以出现 0 次，也可以无限多次 -->
<result name="" type="">
    映射资源
    <!-- 下面元素可以出现 0 次，也可以无限多次 -->
        <param name="参数名">参数值</param>*
</result>
<!-- 下面元素可以出现 0 次，也可以无限多次 -->
<interceptor-ref name="">
    <!-- 下面元素可以出现 0 次，也可以无限多次 -->
        <param name="参数名">参数值</param>*
</interceptor-ref>
<!-- 下面元素可以出现 0 次，也可以无限多次 -->
<exception-mapping name="" exception="" result="">
    异常处理资源
    <!-- 下面元素可以出现 0 次，也可以无限多次 -->
        <param name="参数名">参数值</param>*
</exception-mapping>
</action>
</package>*
</struts>

```

上面的 `struts.xml` 配置文件是一个非常全面的配置文件，包含了 Struts 2 的全部配置元素。上面配置文件主要用于帮助对 XML DTD 不太熟悉的读者。

实际上，如果读者需要熟悉 XML 文件的 DTD（元素类型定义）语法，通过查看 Struts 2 配置文件的 DTD 信息，同样可以了解 Struts 2 配置文件的结构。下面是 Struts 2 配置文件的 DTD 文件代码。

```

<!--
    Struts configuration DTD.
    Use the following DOCTYPE

    <!DOCTYPE struts PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
        "http://struts.apache.org/dtds/struts-2.0.dtd">
-->
<!ELEMENT struts (package|include|bean|constant)*>
<!ELEMENT package (result-types?, interceptors?, default-interceptor-ref?,
default-action-ref?, global-results?, global-exception-mappings?, action*)>
<!ATTLIST package
    name CDATA #REQUIRED
    extends CDATA #IMPLIED
    namespace CDATA #IMPLIED
    abstract CDATA #IMPLIED
    externalReferenceResolver NMTOKEN #IMPLIED
>
<!ELEMENT result-types (result-type+)>
<!ELEMENT result-type (param*)>
<!ATTLIST result-type
    name CDATA #REQUIRED

```

```
class CDATA #REQUIRED
default (true|false) "false"
>
<!ELEMENT interceptors (interceptor|interceptor-stack)+>
<!ELEMENT interceptor (param*)>
<!ATTLIST interceptor
  name CDATA #REQUIRED
  class CDATA #REQUIRED
>
<!ELEMENT interceptor-stack (interceptor-ref+)>
<!ATTLIST interceptor-stack
  name CDATA #REQUIRED
>
<!ELEMENT interceptor-ref (param*)>
<!ATTLIST interceptor-ref
  name CDATA #REQUIRED
>
<!ELEMENT default-interceptor-ref (param*)>
<!ATTLIST default-interceptor-ref
  name CDATA #REQUIRED
>
<!ELEMENT default-action-ref (param*)>
<!ATTLIST default-action-ref
  name CDATA #REQUIRED
>
<!ELEMENT global-results (result+)>
<!ELEMENT global-exception-mappings (exception-mapping+)>
<!ELEMENT action (param|result|interceptor-ref|exception-mapping)*>
<!ATTLIST action
  name CDATA #REQUIRED
  class CDATA #IMPLIED
  method CDATA #IMPLIED
  converter CDATA #IMPLIED
>
<!ELEMENT param (#PCDATA)>
<!ATTLIST param
  name CDATA #REQUIRED
>
<!ELEMENT result (#PCDATA|param)*>
<!ATTLIST result
  name CDATA #IMPLIED
  type CDATA #IMPLIED
>
<!ELEMENT exception-mapping (#PCDATA|param)*>
<!ATTLIST exception-mapping
  name CDATA #IMPLIED
  exception CDATA #REQUIRED
  result CDATA #REQUIRED
>
<!ELEMENT include (#PCDATA)>
<!ATTLIST include
  file CDATA #REQUIRED
```

```
>
<!ELEMENT bean (#PCDATA)>
<!-- bean
  type CDATA #IMPLIED
  name CDATA #IMPLIED
  class CDATA #REQUIRED
  scope CDATA #IMPLIED
  static CDATA #IMPLIED
  optional CDATA #IMPLIED
-->
<!ELEMENT constant (#PCDATA)>
<!-- constant
  name CDATA #REQUIRED
  value CDATA #REQUIRED
-->
```

通过上面给出的 `struts.xml` 配置文件范例，以及 Struts 2 配置文件的 DTD 信息，我们已经可以全面掌握 Struts 2 配置文件的文件结构了，但 `struts.xml` 文件各配置元素的意义、各属性的相关含义依然不能深入了解。关于 Struts 2 配置文件各元素及各属性将在下一章深入介绍。

3.6 本章小结

因为 Struts 2 是以 Struts 1 和 WebWork 为基础的，故本章大致介绍了 Struts 1 和 WebWork 框架的 MVC 实现，并在此基础上侧重于介绍 Struts 1 和 WebWork 与 Struts 2 的延续性，以及 Struts 2 框架与 Struts 1 和 WebWork 的类似性。通过阅读本章，读者应该明白：Struts 2 是通过 WebWork 发展起来的，而不是通过 Struts 1 发展起来的。

本章还介绍了如何通过 Eclipse IDE 工具来开发 Struts 2 应用，Struts 2 的配置文件，以及通过 `web.xml` 文件加载 Struts 2 框架。详细介绍了 `struts.properties` 文件，以及 `struts.xml` 配置文件的结构。关于深入配置 `struts.xml` 文件的元素，则放在下一章介绍。