# 第2章 Struts 2 基础

Struts 是 Apache 软件基金会赞助的一个开源项目，它最初是 Jakarta 项目中的一个子项目。Struts 2 是以 Webwork 设计思想为核心，吸收原 Struts 的优点而形成的，旨在帮助程序员更方便地运用 MVC 模式来开发 Java EE 应用。

## 2.1 Struts 2 框架开发入门

### 2.1.1 MVC 基本思想

MVC 是一种通用的 Web 软件设计模式，它强制性地把应用程序的数据处理、数据展示和流程控制分开。MVC 把应用程序分成 3 大基本模块：模型（Model，即 M）、视图（View，即 V）和控制器（Controller，即 C），它们（三者联合即 MVC）分别担当不同的任务。图 2.1 显示了这几个模块各自的职能及相互关系。

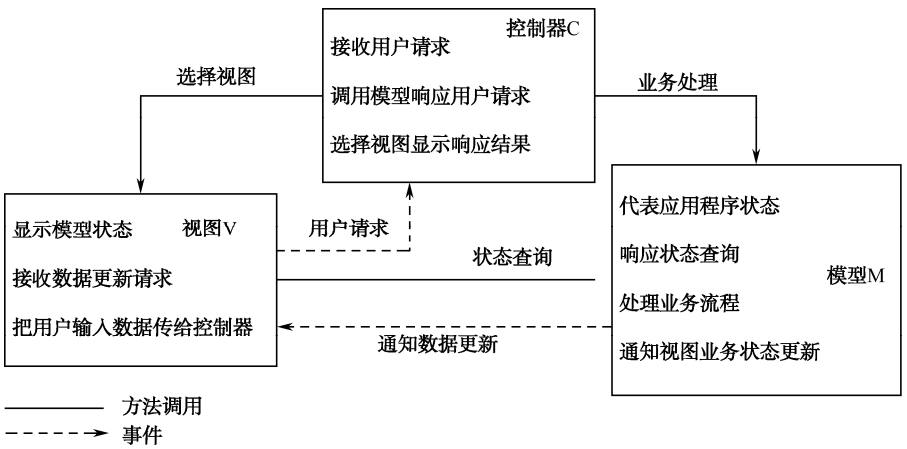



图 2.1 MVC 设计模式

- 模型：用于封装与应用程序业务逻辑相关的数据及对数据的处理方法。“模型”有对数据直接访问的权限，它不依赖“视图”和“控制器”，也就是说，模型并不关心它会被如何显示或是被如何操作。
- 视图：视图是用户看到并与之交互的界面。对老式 Web 应用程序来说，视图就是由 HTML 元素和 JSP 组成的网页；在新式 Web 应用中，HTML 和 JSP 依旧扮演着重要角色，但一些新的技术已层出不穷，包括 Macromedia Flash 以及像 HTML 5、XML/XSL、WML 等一些标识语言和 Web Services 等。
- 控制器：控制器起到不同层面间的组织作用，用于控制应用程序的流程。它处理事件并做出响应，“事件”包括用户的行为和数据模型上的改变。

MVC 的思想最早由 Trygve Reenskaug 于 1974 年提出，而作为一种软件设计模式，它是由 Xerox PARC 在 20 世纪 80 年代为 Smalltalk-80 语言发明的。采用 MVC 开发的软件模块化程度高，模块间具有低耦合、高重用性和高适用性的特点，系统易扩展、易维护，有利于软件工程化管理和缩短开发周期，所以很快

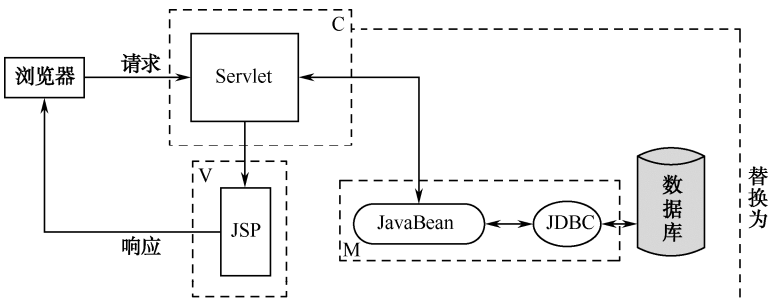
就被推荐为 Java EE 的标准设计模式，受到越来越多 Java EE 开发者的欢迎，至今已被广泛使用。

### 2.1.2 MVC 实现方式

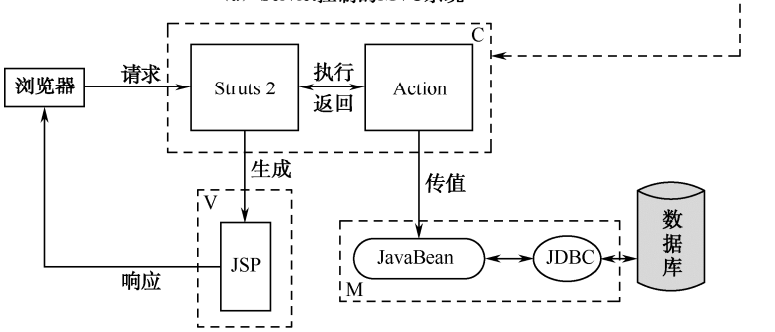
传统的 Java EE 开发采用 JSP+Servlet+JavaBean 的方式来实现 MVC（如【实例 1.1】），但它有一个缺陷：程序员在编写程序时必须继承 HttpServlet 类、覆盖 doGet()和 doPost()方法，严格遵守 Servlet 代码规范编写程序，形如：

```
package x.xx.servlet;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class XxxServlet extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
        …
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
        doGet(request,response);
    }
}
```

以上这些烦琐的代码与程序本身要实现的功能无关，仅是 Java 语言 Servlet 编程接口（API）的一部分。在开发中一旦暴露 Servlet API 就会大大增加编程的难度，为了屏蔽这种不必要的复杂性，减少用 MVC 开发 Java EE 的工作量，人们发明了 Struts 2 框架。用 Struts 2 实现的 MVC 系统与传统的用 Servlet 编写的 MVC 系统相比，两者在结构上的区别如图 2.2 所示。




（a）Servlet控制的MVC系统




（b）Struts 2控制的MVC系统

图 2.2　MVC 系统的两种实现方式

可见，Struts 2 的解决方案是：用 Struts 2 框架代替了原 Servlet 部分作为控制器，而具体的控制功能由用户自定义编写 Action 去实现，与 Struts 2 的控制核心相分离。这就进一步降低了系统中各部分组件的耦合度，也大大降低了编程难度。

目前，在 Java Web 开发领域，还有一种主流的 MVC 框架——Spring MVC，它是著名的 Spring 框架内的一个子框架，为 MVC 的应用提供了另一种优越的轻量级解决方案。有关 Spring MVC 的基础知识，本书会在讲完 Spring 之后再加以重点介绍（第 14 章）。

### 2.1.3　简单 Struts 2 开发

本节先通过一个简单实例让读者快速上手，学会 Struts 2 框架的使用。

**【实例 2.1】**将【实例 1.1】的程序改用 Struts 2 实现，即用 Struts 2 框架替换原程序中 Servlet 承担的程序流程控制职能，编写 Action 实现登录功能。

启动 MyEclipse 2017，打开已开发好的 bookManage 项目，删除 src 下 org.servlet 包及其中的 LoginServlet.java 文件（右键菜单→【Delete】），下面开始开发过程。

**1. 加载 Struts 2 包**

登录 http://struts.apache.org/，下载 Struts 2，本书使用的是 Struts 2.5.13，其官方下载页面如图 2.3 所示。


图 2.3　Struts 2 官方下载页面

在大多数情况下，使用 Struts 2 的 Web 应用并非需要用到 Struts 2 的全部特性，故这里只下载其最小核心依赖库（大小仅为 4.28 MB），单击页面中“Essential Dependencies Only”项下的“struts-2.5.13-min-lib.zip”链接即可。将下载获得的文件 struts-2.5.13-min-lib.zip 解压缩，在其目录 struts-2.5.13-min-lib\struts-2.5.13\lib 下看到有 8 个 jar 包，包括以下内容。

（1）Struts 2 的 4 个基本类库

struts2-core-2.5.13.jar

ognl-3.1.15.jar

log4j-api-2.8.2.jar

freemarker-2.3.23.jar

（2）附加的 4 个库

commons-io-2.5.jar

commons-lang3-3.6.jar

javassist-3.20.0-GA.jar

commons-fileupload-1.3.3.jar

将它们一起复制到项目的\WebRoot\WEB-INF\lib 路径下。在工作区视图中，右击项目名，从弹出菜单中选择【Refresh】刷新。打开项目树，展开“Web App Libraries”项可看到这 8 个 jar 包，如图 2.4 所示，表明 Struts 2 加载成功了。

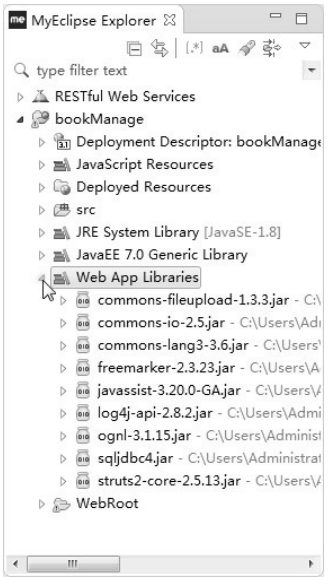



图 2.4　Struts 2 加载成功

其中，主要类描述如下。

struts2-core-2.5.13.jar：Struts 2.5 的主框架类库。

ognl-3.1.15.jar：OGNL 表达式语言。

log4j-api-2.8.2.jar：管理程序运行日志的 API 接口。

freemarker-2.3.23.jar：所有的 UI 标记模板。

Struts 2 从 2.3 升级到 2.5 版，有比较大的变化，主要体现在：

① 将原 xwork-core 库整合进核心 struts2-core 库。早期的 Struts 2 是基于 WebWork 框架发展起来的，后者对应于 xwork-core 库，但自 2.5 版起，Struts 2 不再提供独立的 xwork-core 库，相关的功能全部合并到主框架核心库中，这也标志着 Struts 与 WebWork 两大框架的真正融合。

② 以 log4j-api 取代原 commons-logging 库。log4j 提供了用户创建日志需要实现的适配器组件，比之原先 commons-logging 的通用日志处理功能更为强大，支持灵活的日志定制，并且版本越高，可选的显示信息的种类就越丰富。

### 2. 配置 web.xml

Struts 2 框架需要在项目 web.xml 文件中配置，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_9" 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">
    <filter>
        <filter-name>struts-prepare</filter-name>
        <filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareFilter</filter-class>
    </filter>
    <filter>
        <filter-name>struts-execute</filter-name>
        <filter-class>org.apache.struts2.dispatcher.filter.StrutsExecuteFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts-prepare</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <filter-mapping>
        <filter-name>struts-execute</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

```
    <display-name>bookManage</display-name>
    <welcome-file-list>
        <welcome-file>login.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

请读者删掉 web.xml 中原来的配置代码，以上面的代码取而代之。这其实是在配置 Struts 2 的过滤器，有关内容会在后面讲解。

### 3. 实现控制器 Action

基于 Struts 2 框架的 Java EE 应用程序使用自定义的 Action（控制器）来处理深层控制逻辑，完成用户想要完成的功能。本例定义名为“login”的控制器，判断登录用户名和密码的正确性。在项目 src 下建立包 org.action，在包里创建 LoginAction 类。

LoginAction.java 代码如下：

```
package org.action;
import java.util.*;
import org.model.*;
import org.dao.*;
import com.opensymphony.xwork2.*;
public class LoginAction extends ActionSupport{
    private Login login;
    //处理用户请求的 execute 方法
    public String execute() throws Exception{
        //该类为项目与数据的接口（DAO 接口），用于处理数据与数据库表的一些操作
        LoginDao loginDao = new LoginDao();
        Login l = loginDao.checkLogin(login.getName(), login.getPassword());
        if(l!=null){                                          //如果登录成功
            //获得会话，用来保存当前登录用户的信息
            Map session = ActionContext.getContext().getSession();
            session.put("login", l);                          //把获取的对象保存在 Session 中
            return SUCCESS;        //验证成功返回字符串 SUCCESS（此时 Session 中已有用户对象）
        }else{
            return ERROR;                                     //验证失败返回字符串 ERROR
        }
    }
    //属性 login 的 get/set 方法
    public Login getLogin() {
        return login;
    }
    public void setLogin(Login login) {
        this.login = login;
    }
}
```

加黑部分代码用到了 Action 模型传值，其机制将在后面介绍。

### 4. 配置 struts.xml

在编写好 Action（控制器）的代码之后，还需要进行配置才能让 Struts 2 识别 Action。在 src 下创建文件 struts.xml（注意文件位置和大小写），输入如下的配置代码：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
```

```
        "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"
        "http://struts.apache.org/dtds/struts-2.5.dtd">
<!-- START SNIPPET: xworkSample -->
<struts>
    <package name="default" extends="struts-default">
    <!--用户登录-->
     <action name="login" class="org.action.LoginAction">
          <result name="success">main.jsp</result>
          <result name="error">error.jsp</result>
     </action>
    </package>
    <constant name="struts.i18n.encoding" value="gb2312"/>
</struts>
<!-- END SNIPPET: xworkSample -->
```

这表示凡是请求“name”为“login”的都要用“class”对应的类来处理。而“result”中表示如果类中返回“success”就跳转到“main.jsp”页面，返回“error”则跳转到“error.jsp”页面。

### 5. 编写 JSP

本例 login.jsp（登录页）、main.jsp（欢迎主页）这两个 JSP 文件均使用 Struts 2 的标签进行了重新改写。

登录页 login.jsp，代码如下：

```
<%@ page language="java" pageEncoding="gb2312"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
    <title>图书管理系统</title>
</head>
<body bgcolor="#71CABF">
<s:form action="login" method="post" theme="simple">
    <table>
        <caption>用户登录</caption>
        <tr>
            <td>登录名<s:textfield name="login.name" size="20"/></td>
        </tr>
        <tr>
            <td>密&amp;nbsp;&amp;nbsp;码<s:password name="login.password" size="21"/></td>
        </tr>
        <tr>
            <td>
                <s:submit value="登录"/>
                <s:reset value="重置"/>
            </td>
        </tr>
    </table>
</s:form>
</body>
</html>
```

其中加黑部分代码使用到了 Struts 2 的标签库，各标签的具体用法将在第 3 章详细介绍，这里暂不展开。该页面表单提交给的对象是“login”（Action 控制器名）。

欢迎主页 main.jsp，代码如下：

```
<%@ page language="java" pageEncoding="gb2312"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
    <title>欢迎使用</title>
</head>
<body>
    <s:set var="login" value="#session['login']"/>
    <s:property value="#login.name"/>，您好！欢迎使用图书管理系统。
</body>
</html>
```

其中加黑部分代码用到了 Struts 2 的 OGNL 表达式，将在第 3 章介绍。

JSP 文件 error.jsp（出错处理页）代码不变，从略。

最后，部署运行程序，效果与【实例 1.1】完全一样，如图 1.53～图 1.55 所示。

对于上面这个实例，读者如果对其代码中有些语句不是很明白也没关系，后面会对所有的内容进行详细讲解，这里主要让读者能够明白 Struts 2 框架的用途和基本的使用方法。

## 2.2　Struts 2 原理及工作流程

### 2.2.1　Struts 2 工作原理

Struts 2 框架内部是基于一种称为“过滤器”的机制运作的，其工作原理图如图 2.5 所示，该图出自 Struts 2 官方发布的技术文档，清楚地概括了 Struts 2 的整个工作过程。

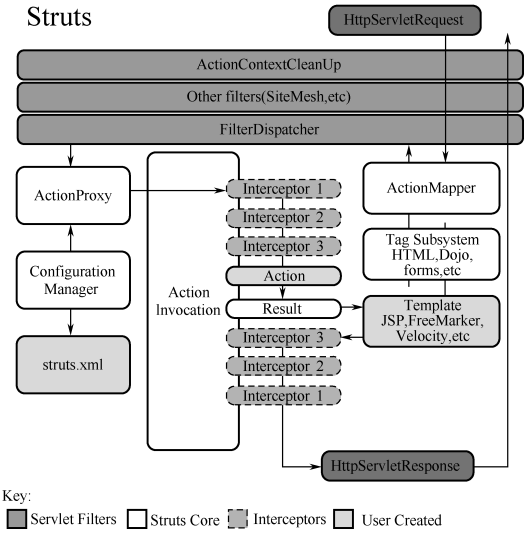



图 2.5　Struts 2 的工作原理图

一般来说，从客户端发出请求直至最终生成响应并返回给客户端一个页面的全部过程，大致可分为如下几个步骤。

① 客户端提交一个（HttpServletRequest）请求。

② 请求被提交到一系列（主要是 3 层）的过滤器（Filter），如（ActionContextCleanUp、其他过

滤器、FilterDispatcher）。注意，这里是有顺序的，先是 ActionContextCleanUp，再是其他过滤器（SiteMesh 等），最后才到 FilterDispatcher。

③ FilterDispatcher 接收到请求后，询问 ActionMapper 是否需要调用某个 Action 来处理这个（HttpServletRequest）请求，如果 ActionMapper 决定需要调用某个 Action，则 FilterDispatcher 把请求的处理交给 ActionProxy。

④ ActionProxy 通过 Configuration Manager（struts.xml）询问框架的配置文件，找到需要调用的 Action 类（该 Action 类一般是程序员自定义的处理请求的类）。

⑤ ActionProxy 创建一个 ActionInvocation 实例，同时 ActionInvocation 通过代理模式调用 Action。但在调用之前，ActionInvocation 会根据配置加载 Action 相关的所有 Interceptor（拦截器）。

⑥ 一旦 Action 执行完毕，ActionInvocation 负责根据 struts.xml 中的配置找到对应的返回结果 Result。然后根据结果返回对应的视图（JSP、FreeMarker 等）呈现给客户端。

在项目中，Struts 2 框架主要作为控制器使用，除此之外，它还内置有标签子系统（Tag Subsystem），为前端 JSP 应用提供一些标签。Struts 2 有效地把它们整合在一起，从而增强了规范性。

### 2.2.2 Struts 2 项目运行流程

了解了 Struts 2 的工作原理，再结合【实例 2.1】就可以总结出一个基于 Struts 2 开发的项目的运行流程了，如图 2.6 所示。

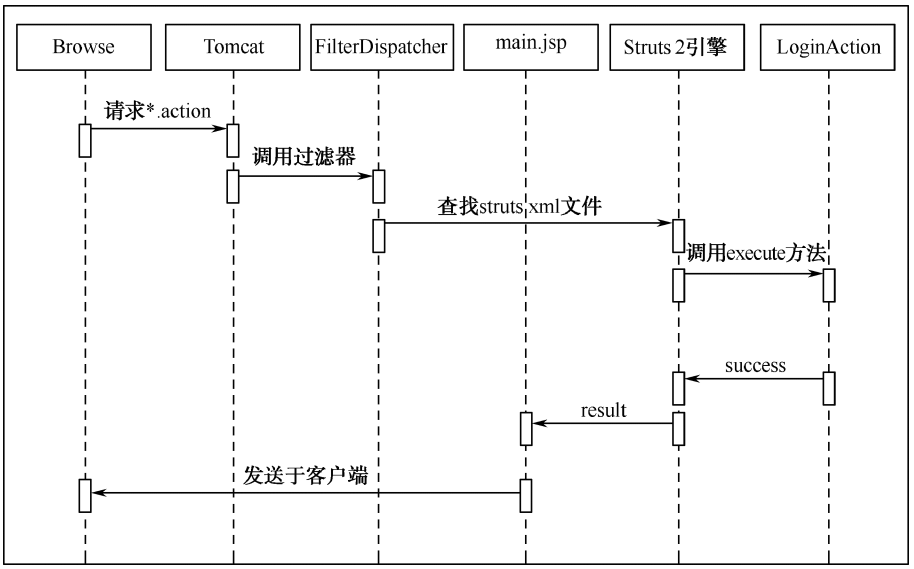



图 2.6 Struts 2 项目运行流程

其运行流程如下所述。

① 浏览器请求“http://localhost:8080/bookManage/login.jsp”，发送到 Web 应用服务器。

② 容器接收到了 Web 服务器对 JSP 页面中 login.action 的请求，根据 web.xml 中的配置，服务器将包含.action 后缀的请求转到“org.apache.struts2.dispatcher.FilterDispatcher”类进行处理，进入 Struts 2 的流程中。Struts 2.1.3 以上版本开始使用更先进的“org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter”类；而在 Struts 升级到 2.5 版后，又将这个类的功能一分为二（分别置于“org.apache.struts2.dispatcher.filter.StrutsPrepareFilter”和“org.apache.struts2.dispatcher.filter.StrutsExecuteFilter”类中），故新版 Struts 2.5 的使用要求用户配置两个过滤器，名称分别为 struts-prepare 和 struts-execute。

③ 框架在 struts.xml 配置文件中查找名为“login”的 action 对应的类。框架初始化该 Action（对数据进行了封装，并把数据放入值栈中）并且执行该 Action 类的 execute 方法（如果配置文件中指定了特定方法则会执行对应的方法，默认执行 execute 方法），该方法可以进行一些数据处理等操作，然后返回（【实例 2.1】项目验证用户名成功返回“success”）。

④ 框架检查配置以查看当返回成功时对应的页面。框架告诉容器来获得请求返回的结果页面 main.jsp（欢迎主页），在该页面中用 OGNL 表达式输出存在值栈中的值（这里也可以用 Struts 2 提供的标签来输出）。

在 Struts 2 框架中，Action 类的调用是通过代理类 ActionProxy 来完成的，代理类再创建一个 ActionInvocation 对象，来调用程序员自定义的 Action 类，在调用之前会先根据配置加载 Action 相关的所有 Interceptor（拦截器）。Struts 2 就是通过一系列的拦截器来工作的，关于拦截器会在第 5 章详细介绍。

## 2.3 Struts 2 的控制器 Action 类

Struts 2 的控制器 Action 可以是一个简单的 POJO（Plain Ordinary Java Object）类，但在实际应用中一般自定义的 Action 类都会继承 Struts 2 框架提供的 ActionSupport 类，下面具体介绍该类。

### 2.3.1 使用 ActionSupport

继承 ActionSupport 类能够帮助程序员更好地完成一些工作，它实现了 5 个接口并包含了一组默认的实现。如果编程中想要用到该类提供的某些功能，只要重写它提供的方法就可以了，例如，要实现验证功能，只需在自定义 Action 类中重写 validate()方法即可。

ActionSupport 类的声明如下：

```
public class ActionSupport implements Action, Validateable, ValidationAware,
        TextProvider, LocaleProvider,Serializable{
}
```

可以看出它实现了 5 个接口，下面分别简要地介绍它们。

#### 1. Action 接口

该接口提供了 5 个常量及一个 execute()方法。代码如下：

```
public interface Action {
        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";
        public String execute() throws Exception;
}
```

如果 Action 类继承了 ActionSupport 类，就可以直接应用这几个常量，比如在【实例 2.1】中的 Action 代码：

```
if(l!=null){
        Map session = ActionContext.getContext().getSession();
        session.put("login", l);
        return SUCCESS;                 //验证成功返回 SUCCESS，实际返回的是字符串"success"
}else{
        return ERROR;                   //验证失败返回 ERROR，实际返回的是字符串"error"
}
```

因为这几个字符串是在程序中经常用到的，所以 Struts 2 框架提供了它们对应的常量，如果有需要，完全可以自定义返回值，只要与 struts.xml 配置的“result”中对应就可以了。

2. Validateable 接口

该接口提供了一个 validate()方法用于校验表单数据，在实际应用中只要在 Action 类中重写该方法即可。【实例 2.1】的程序并没有对填入的数据进行任何判断，即使用户未输入任何内容，提交后也会查询数据库，在一般情况下是不会允许的。这时就可以在 Action 类中重写 validate()方法，然后在该方法中对取得的数据进行判断，如果为空或其他不允许的情况就可以保存错误信息。该方法是在执行 execute()方法之前执行的。

3. ValidationAware 接口

该接口定义了一些方法用来对 Action 执行过程中产生的信息进行处理。例如，该接口中提供了 addFieldError(String fieldname，String errorMessage)方法用来在验证出错时保存错误信息。

4. TextProvider 接口

该接口中提供了一系列 getText()方法，用于获得对应的国际化信息资源。在 Struts 2 中的国际化信息资源都是以 key-value 对出现的，通过使用该接口中的 getText()方法可以用 key 来获得相应的 value 值（国际化内容会在第 5 章讲解）。

5. LocaleProvider 接口

该接口提供了一个 getLocale()方法，用于国际化时获得语言/地区信息。

## 2.3.2 Action 传值

Action 可以通过其属性获取页面上表单文本框中用户输入的值，代码如下：

```
package org.action;
…
public class LoginAction extends ActionSupport {
        private String name;                    //登录名
        private String password;                //密码
        //处理用户请求的 execute 方法
        public String execute() throws Exception{
                …
        }
        //属性 name 的 get/set 方法
          public String getName() {
                return name;
          }
          public void setName(String name) {
                this.name = name;
          }
        //属性 password 的 get/set 方法
          public String getPassword() {
                return password;
          }
          public void setPassword(String password) {
                this.password = password;
          }
}
```

其中有两个属性“name”和“password”，并且生成了它们的 get 和 set 方法。在运行 login.jsp 的时候，Struts 2 框架会根据页面的文本框名在 Action 类中寻找其 set 方法来对其进行赋值。例如，页面的文本框名为“name”的字段就会直接在 Action 类中找相应的 setName(String name)方法为其赋值，而不是找其对应的在 Action 类中的属性“name”，所以在 Action 类中的属性名不一定要和页面中的文本框名相对应，但是在 Action 类中必须是页面的文本框名对应的 set 和 get 方法，而不是 Action 类中属性的 get 和 set 方法。也就是说，在 Action 类中，name 属性可以不叫 name，可以叫“username”或其他，但是里面必须有页面的文本框名“name”对应的 setName()方法，而不是 setUsername()方法。但一般情况下，程序员都会把它们统一起来，让 Action 类中的属性就对应页面中的输入文本框名，这样会方便很多。

前面这种传值方式称为字段传值方式，如果一个表单中字段比较多，而在 Action 中就要写很多属性，若在不同的 Action 中都要用到还要重复写，非常麻烦，也不利于维护，所以 Struts 2 框架还提供了另一种传值方式即模型传值方式。使用该传值方式首先要把字段封装成一个类，并生成其 get 和 set 方法，就是通常说的 JavaBean 了。因【实例 2.1】的项目中已编写好了 JavaBean 类 Login.java，故只需在 Action 中改变写法：

```
package org.action;
…
public class LoginAction extends ActionSupport{
    private Login login;
    //处理用户请求的 execute 方法
    public String execute() throws Exception{
        …
    }
    //属性 login 的 get/set 方法
    public Login getLogin() {
        return login;
    }
    public void setLogin(Login login) {
        this.login = login;
    }
}
```

可以发现，这样简单了很多。还要注意的是，传值页面（登录页 login.jsp）的“属性名”也要做小小的修改，要把以前的“属性名”改为“模型对象名.属性名”。例如，这里要把以前的“<input name="name" type="text" size="20"/>”中的“name”及“<input name="password" type="password" size="21"/>”中的 name 值“password”修改为“login.name”和“login.passoword”，而欢迎界面的取值也要相应地修改为“login.name”。这样修改后，再重新启动 Tomcat 服务器，运行项目会得到相同的结果。

在 Struts 2 中，Action 类的属性总是在调用 execute()方法之前被设置（通过 get/set 方法），这意味着在 execute()中可以直接使用，因为在 execute()执行之前，它们已经被赋予了正确的值。

**注意：**

当项目的“.java 文件”或者一些配置文件如“*.xml”经过修改后，一定要重新启动 Tomcat 服务器，而若仅修改了 JSP 页面则只需刷新页面即可。

### 2.3.3 Action 访问 Servlet API

Struts 2 中没有与任何的 Servlet API 关联，这样大大降低了程序的耦合性，但是有时候在写程序时

需要用到 Servlet 的一些 API，如“request”、“response”、“application”、“session”等。为此，Struts 2 提供了两种方式来访问 Servlet API，介绍如下。

### 1. 通过 ActionContext

ActionContext 类提供了一个静态的 getContext()方法来获得 ActionContext 对象，然后根据其对象来获得一些 Servlet API 的对象。例如：

```
ActionContext ac=ActionContext.getContext();                    //获得 ActionContext 对象
Map session=ac.getSession();                                    //获得 session 对象
Map application=ac.getApplication();                            //获得 application 对象
Map request=ac.get();                                           //获得 request 对象
```

大家可能有些奇怪，这些方法得到的都是 Map 类型，而不是要求的“HttpSession”或“Application”，其实 Struts 2 把 Map 对象模拟成了“HttpSession”等对象，从而将 Servlet 从 Action 中分离出来。

由于“request”和“response”比较特殊，也是在开发中经常会用到的，所以 Struts 2 提供了专门的类来获取，即“ServletActionContext”。

```
HttpServletRequest request=ServletActionContext.getRequest();          //获得 HttpServletRequest 对象
HttpServletResponse response =ServletActionContext.getResponse();      //获得 HttpServletResponse 对象
HttpSession session=request.getSession();                              //获得 HttpSession 对象
```

除了这种方法外，还可以用如下方法得到：

```
ActionContext ac=ActionContext.getContext();
//获得 HttpServletRequest 对象
HttpServletRequest request=(HttpServletRequest) ac.get(ServletActionContext.HTTP_REQUEST);
//获得 HttpServletResponse 对象
HttpServletResponse response=(HttpServletResponse)ac.get(ServletActionContext.HTTP_RESPONSE);
//获得 HttpSession 对象
HttpSession session=request.getSession();
```

### 2. 通过实现*Aware 接口

Struts 2 中提供了一系列的*Aware 接口，如表 2.1 所示。

表 2.1　*Aware 接口及获得对象的方法

| 接 口 名 称 | 获得 Servlet 对象的方法 |
|---|---|
| ApplicationAware | void setApplication(Map application) |
| CookiesAware | void setCookies (Map cookies) |
| RequestAware | void setRequest(Map request) |
| ServletRequestAware | void setServletRequest(HttpServletRequest request) |
| ServletResponseAware | void setServletResponse(HttpServletResponse response) |
| SessionAware | void setSession(Map session) |

例如，要获得 Application 对象，Action 类就可以编写如下：

```
import java.util.Map;
import org.apache.struts2.interceptor.ApplicationAware;
public class TestApplication implements ApplicationAware{
        private Map application;
        public void setApplication(Map application) {
                this.application=application;
        }
        public String execute() throws Exception{
```

```
            //...其他内容，这里可以直接应用 application
        }
    }
```

### 2.3.4 Action 返回结果

在一个 Action 类中，有时会返回多个结果，如判断一件事情，如果为真就返回 SUCCESS，否则返回 ERROR（或"error"）。在【实例 2.1】中，对 DAO 接口执行 checkLogin()方法获取到 Login 对象 l 的值进行判断，然后返回不同结果。

```
Login l = loginDao.checkLogin(login.getName(), login.getPassword());
if(l!=null){
    …
    return SUCCESS;          //验证成功返回
}else{
    return ERROR;            //验证失败返回
}
```

这里判断 l 对象不为空（数据库中有这个用户信息）就返回成功，然后根据配置文件的返回跳转到欢迎页面，如果 l 为空则返回出错页面，所以还要在 struts.xml 文件中配置两种不同的返回结果跳转到的页面，如下：

```
<!-- 用户登录 -->
<action name="login" class="org.action.LoginAction">
    <result name="success">main.jsp</result>
    <result name="error">error.jsp</result>
</action>
```

当然，在方法中还可以返回更多的值（可以为任意字符串），但不管返回什么值，都要在配置文件中进行对应，且不能返回两个或几个结果相同的值。对于继承了 ActionSupport 的 Action 类可以返回前面提到的 5 个常量。

### 2.3.5 在 Action 中定义多方法

大家可以想象这样一种情况：有一个用户登录，定义了一个 LoginAction 类，如果现在程序还需有一个注册功能，是否还要定义一个 RegistAction 类呢？当然是可以的，但这并不是最好的办法。如果程序中功能越来越多，那就要定义越来越多的 Action 类，所以一般不采取这样的方式，而是把相关的功能定义在同一个 Action 类中，用多个方法来实现不同的功能。

例如，在 LoginAction 类中定义两个方法：

```
package org.action;
…
public class LoginAction extends ActionSupport{
    private Login login;
    //处理用户请求的 execute 方法（实现登录验证功能）
    public String execute() throws Exception{
        …
    }
    // regist()方法（实现注册功能）
    public String regist() throws Exception{
        //注册新用户
        …
        return "regist";
```

```
        }
        //省略属性 login 的 get/set 方法
    }
```

这里暂时列举在 Action 类中定义不同方法，并没有说明请求时如何映射到具体的方法，不同的请求怎么对应到相应的处理方法会在 2.5.1 节的 action 配置中讲解。

## 2.4 解密 Struts 2 程序文件

### 2.4.1 web.xml 文件

在前面开发例子的过程中，首先就配置了 web.xml，它是一个正规的 XML 文件，包括版本及编码信息，然后就是<web-app>标签。这里具体讲解在<web-app>里面配置的信息。

```
…
    <filter>
        <filter-name>struts-prepare</filter-name>
        <filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareFilter</filter-class>
    </filter>
    <filter>
        <filter-name>struts-execute</filter-name>
        <filter-class>org.apache.struts2.dispatcher.filter.StrutsExecuteFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts-prepare</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <filter-mapping>
        <filter-name>struts-execute</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
…
```

这里面配置了两个过滤器，那么就先来介绍过滤器的使用。

Filter 过滤器是 Java 项目开发中的一种常用技术。过滤器是用户请求和处理程序之间的一层处理程序。这层程序可以对用户请求和处理程序响应的内容进行处理。过滤器可以用于权限控制、编码转换等场合。

Servlet 过滤器是在 Java Servlet 规范中定义的，它能够对过滤器关联的 URL 请求和响应进行检查和修改。Servlet 过滤器能够在 Servlet 被调用之后检查 response 对象，修改 response Header 对象和 response 内容。Servlet 过滤器过滤的 URL 资源可以是 Servlet、JSP、HTML 文件，或是整个路径下的任何资源。多个过滤器还可构成一个过滤器链，当请求过滤器关联的 URL 的时候，过滤器就会逐个发生作用。

所有过滤器必须实现 java.Serlvet.Filter 接口，这个接口中含有 3 个过滤器类必须实现的方法。

- init(FilterConfig)：这是 Servlet 过滤器的初始化方法。Servlet 容器创建 Servlet 过滤器实例后将调用这个方法。
- doFilter(ServletRequest,ServletResponse,FilterChain)：这个方法完成实际的过滤操作。当用户请求与过滤器关联的 URL 时，Servlet 容器将先调用过滤器的 doFilter 方法，返回响应之前也会调用此方法。FilterChain 参数用于访问过滤器链上的下一个过滤器。
- destroy()：Servlet 容器在销毁过滤器实例前调用该方法。这个方法可以释放 Servlet 过滤器占用

的资源。

过滤器类编写完成后，必须在 web.xml 中进行配置，格式如下：

```
<filter>
    <!-- 自定义的名称 -->
    <filter-name>过滤器名</filter-name>
    <!-- 自定义的过滤器类，注意，如果类放在指定包下，要加完整包名 -->
    <filter-class>过滤器对应类</filter-class>
      <init-param>
          <!-- 类中参数名称 -->
          <param-name>参数名称</param-name>
          <!-- 对应参数的值 -->
          <param-value>参数值</param-value>
      </init-param>
</filter>
```

过滤器必须和特定的 URL 关联才能发挥作用，过滤器的关联方式有 3 种：与一个 URL 关联、与一个 URL 目录下的所有资源关联、与一个 Servlet 关联。

与一个 URL 资源关联：

```
<filter-mapping>
        <!-- 这里与上面配置所起的名称要相同 -->
        <filter-name>过滤器名</filter-name>
        <!-- 与该 url 资源关联-->
        <url-pattern>xxx.jsp</url-pattern>
</filter-mapping>
```

与一个 URL 目录下的所有资源关联：

```
<filter-mapping>
        <filter-name>过滤器名</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

与一个 Servlet 关联：

```
<filter-mapping>
        <filter-name>过滤器名</filter-name>
        <url-pattern>Servlet 名称</url-pattern>
</filter-mapping>
```

通过上面的讲解，相信大家对 web.xml 文件中配置的内容已经很清楚了。Struts 2 框架的 web.xml 文件中配置的就是过滤器，新版 Struts 2.5 的两个过滤器对应的类分别是框架中的“org.apache.struts2.dispatcher.filter.StrutsPrepareFilter”和“org.apache.struts2.dispatcher.filter.StrutsExecuteFilter”，“url-pattern”指定为“/*”，表示该 URL 目录下的所有请求都交由 Struts 2 处理，这就把 Web 应用与 Struts 2 框架关联起来了。

### 2.4.2　struts.xml 文件

Struts 2.5.13 版本下的 struts.xml 的大体格式如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
      "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"
      "http://struts.apache.org/dtds/struts-2.5.dtd">
<!-- START SNIPPET: xworkSample -->
<struts>
```

```
    <package name="default" extends="struts-default">
    <!-- 配置一个个 Action -->
     ...
    </package>
    <constant name="struts.i18n.encoding" value="gb2312"/>
</struts>
<!-- END SNIPPET: xworkSample -->
```

在该版本下，这个格式是不变的，其前面是编码格式及一些头文件信息，接着是<struts>...</struts>，该文件中的其他配置都包含在其中。该标签下可以编写下面几个子标签。

- include：用于导入其他 xml 配置文件。
- constant：配置一些常量信息。
- bean：由容器创建并注入的组件。
- package：配置包信息。

这几个子标签可以并排编写在<struts>下，而其他一些配置大都写在<package>中。<package>将在稍后的 2.5.3 节详细讲解，而 bean 标签不常用，所以本节主要讲解其他两个标签。

### 1. <include>标签

在实际开发中，可能会把所有的配置信息都放在一个 struts.xml 文件中，但这仅限于一些小的项目。如果一个项目很大，需要配置的信息很多（可能有成千上万行的代码！），这么大一个配置文件，无论是管理还是维护都是相当不容易的，所以这时就要“分而治之”，把不同方面的信息分别编写配置文件。例如，在学生信息系统中，把学生信息的配置、课程信息的配置、成绩信息的配置分别放入各自的配置文件 xs.xml、kc.xml 和 cj.xml 中，然后再用 Struts 2 提供的<include>标签把它们导入到 struts.xml 中。

例如，xs.xml 为：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"
    "http://struts.apache.org/dtds/struts-2.5.dtd">
<struts>
    <package name="xs" extends="struts-default">
        <action name="addXs" class="org.action.XsAction">
            ...
        </action>
    </package>
</struts>
```

在 struts.xml 中导入应为：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"
    "http://struts.apache.org/dtds/struts-2.5.dtd">
<!-- START SNIPPET: xworkSample -->
<struts>
    <include file="xs.xml"/>
    ...
</struts>
<!-- END SNIPPET: xworkSample -->
```

这样就成功导入了。

**注意：**

这里 xs.xml 也是放在 classes 文件夹下的，即在 MyEclipse 中直接放在 src 下即可，如果 xs.xml 放在其他包（如“org.xs”）中，那么 struts.xml 中引入的时候还要加上包名，如下：

```
<struts>
    <include file="/org/xs/xs.xml"/>
    ...
</struts>
```

2. <constant>标签

<constant>是用来在 struts.xml 中定义常量属性的，如设置编码形式、开发模式等。该标签里面有两个属性：name 和 value。例如，可以做下面的设置：

```
<struts>
    <!-- 设置开发模式 -->
    <constant name="struts.devMode" value="true"/>
    <!-- 设置编码格式 GB2312 -->
    <constant name="struts.i18n.encoding" value="gb2312"/>
    ...
</struts>
```

其中，name 的值均为 Struts 2 的常量信息，这些常量信息并不多，不仅可以在 struts.xml 中设置，也可以在 Struts 2 的另一个配置文件 struts.properties 中单独设置。下节将介绍该配置文件。

### 2.4.3 struts.properties 文件

struts.properties 文件是一个标准的 properties 文件，该文件中存放一系列的 key-value 对，每个 key 就是一个 Struts 2 属性，而其对应的 value 值就是一个 Struts 2 的属性值。struts.properties 文件和 struts.xml 一样，要放在项目的 classes 文件夹下，Struts 2 框架会自动加载该文件。

通常情况下，项目并不需要这个配置文件，因为 Struts 2 框架已经提供了一个默认的配置文件 default.properties，但是在开发中，有时为了方便可以更改默认信息，这时就要应用 struts.properties，把要修改的属性的 key-value 对放入其中，这样新的配置信息就会覆盖系统默认的值。struts.properties 文件中所包含的所有属性都可以在 web.xml 配置文件中使用“init-param”标签进行配置，或者在 struts.xml 文件中使用“constant”标签进行配置。

下面介绍 Struts 2 中常量配置信息。

- 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.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。如果需要实现自己的配置管理器，可以编写一个实现 Configuration 接口的类，该类可以自己加载 Struts 2 配置文件。
- 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。前面说过，Struts 2 会自动加载 struts.xml 文件，这里就是最好的解释。
- 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.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.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.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 flase 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.dispatcher.parametersWorkaround：对于某些 Java EE 服务器，不支持 HttpServlet Request 调用 getParameterMap()方法，此时可以设置该属性值为 true 来解决这一问题。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.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.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。通常无须修改该属性值。

● struts.i18n.encoding：该属性指定 Web 应用的默认编码集。一般当获取中文请求参数值时会将该属性值设置为 GBK 或者 GB 2312。该属性默认值为 UTF-8。

● 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（允许国际化文件重载）。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。开发阶段将该属性设置为 true，将可以在每次请求时都重新加载国际化资源文件，从而可以看到实时开发效果。产品发布阶段将该属性设置为 false，是为了提高响应性能，每次请求都重新加载资源文件会大大降低应用的性能。

● struts.locale：指定 Web 应用的默认 Locale。

● 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.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>配置

的 name 属性完成映射。

● struts.objectFactory：指定 Struts 2 默认的 ObjectFactoryBean，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，支持 tiger 和 notiger 两个属性值。

● 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，其默认属性值是 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，如果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.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.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（如 Url 标签），该属性才提供 Web 应用的默认端口。

● 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.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.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。该属性允许使用 ftl、vm 或 jsp，分别对应 FreeMarker、Velocity 和 JSP 模板。

● 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.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。

## 2.5　Struts 2 配置详解

### 2.5.1　<action>配置详解

Struts 2 的核心功能是 Action。对于开发人员来说，使用 Struts 2 框架，主要的编码工作就是编写 Action 类。当开发好 Action 类后，就需要配置 Action 映射，以告诉 Struts 2 框架，针对某个 URL 的请求应该交由哪一个 Action 进行处理。这就是 struts.xml 中 action 配置要起的作用。在【实例 2.1】中：

```
<action name="login" class="org.action.LoginAction">
    <result name="success">main.jsp</result>
    <result name="error">error.jsp</result>
</action>
```

上例表示名为“login”的请求，交由“org.action.LoginAction”这个类来处理，返回结果为“success”时跳转到“main.jsp”页面，结果为“error”时则跳转到“error.jsp”页面。

### 1. <action>属性

<action>有以下属性。

name：该属性是必需的，对应请求的 Action 的名称。

class：该属性不是必需的，指明处理类的具体路径，如“org.action.LoginAction”。

method：该属性不是必需的，若 Action 类中有不同的方法，该属性指定请求对应应用哪个方法。

converter：该属性不是必需的，指定 Action 使用的类型转换器（类型转换内容会在第 4 章讲解）。

在一般情况下，都会为<action>设置 name 和 class 属性，如果没有设置 method 属性，系统会默认调用 Action 类中的 execute 方法。若在 Action 中存在多个方法，请求其某个方法的时候就要通过这里的 method 属性来指定所请求的方法名。例如，在 2.3.5 节的 Action 类中有 execute 和 regist 两个方法，如果要在请求中应用 regist 方法，就要在相应的 action 中配置 method 属性：

```
<action name="login" class="org.action.LoginAction" method="regist">
    ...
</action>
```

加黑部分代码就是要配置的指定的方法，表示应用 LoginAction 类中的 regist 方法。

### 2. 在<action>中应用通配符

前面讲过，可以在<action>中指定 method 属性来决定应用 Action 类中的哪个方法，但这样有些麻烦，应用两个不同的方法就要配置两个<action>，Struts 2 中提供了通配符的使用，可以应用通配符只配置一个 Action 就可以根据通配符来识别应用 Action 类中的哪个方法。

<action>配置要修改为：

```
<action name="*" class="org.action.LoginAction" method="{1}">
    ...
</action>
```

其中“{1}”就是取前面“*”的值。例如，如果要应用 Action 类中的 regist 方法，请求就为：

```
<form action="regist.action" method="post">
    ...
</form>
```

“regist”就会与“*”匹配，得出“*”为“regist”，则<action>中 method 属性的值就为“regist”，就会应用 Action 类中的 regist 方法。

不仅方法可以使用通配符这样匹配，返回的值也可以。例如，如果应用 regist 方法返回“error”时就跳转到“regist.jsp”界面，<action>配置修改为：

```
<action name="*" class="org.action.LoginAction" method="{1}">
    ...
    <result name="error">{1}.jsp</result>
</action>
```

使用通配符可以很大程度地减少 struts.xml 的配置内容，但是可以发现，在编写时也会对 Action 类中的方法命名有限制，必须和请求名称对应，返回视图的名称也同样要对应。所以在实际开发中，要根据实际情况来决定是否使用通配符。

### 3. 访问 Action 类中方法的其他方式

仍以前面的 LoginAction 为例，<action>配置可以用正常情况，只需配置 name 和 class：

```
<action name="login" class="org.action.LoginAction">
    ...
</action>
```

这样配置完全不知道要访问 LoginAction 类中的哪个方法，但是可以在请求中指明，请求的 form 表单要改为：

```
<form action="login!regist.action" method="post">
    ...
</form>
```

其中，“login!regist.action”中“!”前面的“login”对应<action>中的 name 属性值，“!”后面的“regist”对应要使用的 LoginAction 类中的方法名。

此方式是在请求中指定应用 Action 类中的哪个方法，还有一种办法是在提交按钮中设置的，<action>不用做任何改变，不过提交按钮需要用 Struts 2 的标签来实现，并且指定 method：

```
<s:form action="login" method="post">
    ...
    <s:submit value="登录"/>
    <s:submit value="注册" method="regist"/>
</s:form>
```

该 form 中有两个按钮，一个是登录按钮，另一个是注册按钮，其中注册按钮特别指明了要用的方法，不会产生冲突。这里大家只要知道这种用法就行了，Struts 2 的标签库会在后面专门讲解。

### 4. 使用默认类

如果未指明 class 属性，则系统将会自动引用<default-class-ref>标签中所指定的类，即默认类。在 Struts 2 中，系统默认类为 ActionSupport，当然也可以自己定义默认类，例如：

```
<package name="default" extends="struts-default">
    <default-class-ref class="org.action.LoginAction"/>
    <action name="regist">
        ...
    </action>
    …
</package>
```

上面代码中定义了默认类，则在没有指定 class 属性的请求中都会应用该默认类，若指定了自己的 class 属性，则默认类在该 action 中将不起作用。

## 2.5.2 <result>配置详解

<result>是为 Action 类的返回值指定跳转方向的，在 Struts 2 框架中，一个完整的<result>配置为：

```
<result name="Action 类对应返回值" type="跳转结果类型">
    <param name="参数名">参数值</param>
</result>
```

<result>包含两个属性 name 和 type。name 属性与 Action 类中返回的值进行匹配，type 属性指定将要跳转的结果类型，在实际应用中不一定都要跳转到一个页面，有可能会从一个 action 跳转到另一个 action，这时就要指定 type 属性。<param>是为返回结果设置参数的。

Struts 2 中支持多种结果类型，在下载的 Struts 2 完整版的 struts-default.xml 文件中可以找到所有的支持类型，该文件的位置在“\struts-2.5.13-all\struts-2.5.13\src\core\src\main\resources”下，打开

该文件可以找到如下代码：

```
<result-types>
    <result-type name="chain" class="com.opensymphony.xwork2.ActionChainResult"/>
    <result-type name="dispatcher" class="org.apache.struts2.result.ServletDispatcherResult" default="true"/>
    <result-type name="freemarker" class="org.apache.struts2.views.freemarker.FreemarkerResult"/>
    <result-type name="httpheader" class="org.apache.struts2.result.HttpHeaderResult"/>
    <result-type name="redirect" class="org.apache.struts2.result.ServletRedirectResult"/>
    <result-type name="redirectAction" class="org.apache.struts2.result.ServletActionRedirectResult"/>
    <result-type name="stream" class="org.apache.struts2.result.StreamResult"/>
    <result-type name="velocity" class="org.apache.struts2.result.VelocityResult"/>
    <result-type name="xslt" class="org.apache.struts2.views.xslt.XSLTResult"/>
    <result-type name="plainText" class="org.apache.struts2.result.PlainTextResult" />
    <result-type name="postback" class="org.apache.struts2.result.PostbackResult" />
</result-types>
```

下面简要介绍这些类型的作用范围。

- chain：用来处理 Action 链。
- dispatcher：用来转向页面，通常处理 JSP，该类型也为默认类型。
- freemarker：处理 FreeMarker 模板。
- httpheader：控制特殊 http 行为的结果类型。
- redirect：重定向到一个 URL。
- redirectAction：重定向到一个 Action。
- stream：向浏览器发送 InputStream 对象，通常用来处理文件下载，还可用于返回 AJAX 数据。
- velocity：处理 Velocity 模板。
- xslt：处理 XML/XSLT 模板。
- plainText：显示原始文件内容，如文件源代码等。
- postback：用来把请求参数作为表单提交到指定的目的地。

其实，<result>的两个属性都有默认值，如果没有指明就是默认值，name 属性的默认值为“success”，type 属性的默认值为“dispatcher”，就是跳转到 JSP 页面。下面详细讲解几个常用的结果类型。

### 1. dispatcher 类型

该结果类型是默认的结果类型，从“struts-default.xml”中也可以看出，其定义为“default="true"”。定义该类型时，物理视图为 JSP 页面，并且该 JSP 页面必须和请求信息处于同一个 Web 应用中。还有一点值得注意的是，请求转发时地址栏不会改变，也就是说，属于同一请求，所以请求参数及请求属性等数据不会丢失，该跳转类似于 JSP 中的“forward”。从前面的例子中也可以看出，跳转到“main.jsp”页面后，仍可以取出“name”的值。在应用该类型时，一般都会省略不写。配置该类型后，<result>可以指定以下两个参数。

- location：指定请求处理完成后跳转的地址，如“main.jsp”。
- parse：指定是否允许在 location 参数值中使用表达式，如“main.jsp?name=${name}”，在实际运行时，这个结果信息会替换为用户输入的“name”值，该参数默认值是 true。

### 2. redirect 类型

该结果类型可以重定向到 JSP 页面，也可以重定向到另一个 Action。该类型是与 dispatcher 类型相对的，当 Action 处理用户请求结束后，将重新生成一个请求，转入另一个界面。例如，在【实例 2.1】中，当用默认值“dispatcher”时，请求完成，转向“main.jsp”界面，如图 2.7 所示。可以发现，请求

没变，还是“login.action”，但页面已经跳转到“main.jsp”，并且可以取出“name”的值。如果把<result>中的内容改为：

```
<result name="success" type="redirect">main.jsp</result>
```

则请求完成，重定向到 main.jsp 欢迎主页，如图 2.8 所示，可以看出 URL 已变为“main.jsp”。配置 redirect 类型后，<result>也可指定 location 和 parse 两个参数。

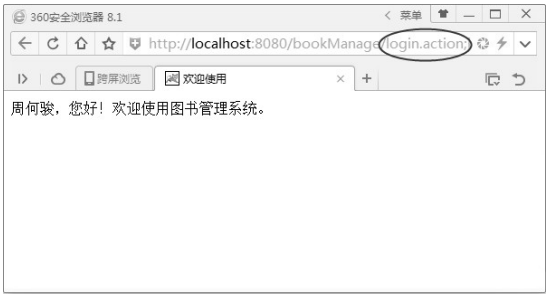



图 2.7　dispatcher 类型时的跳转界面

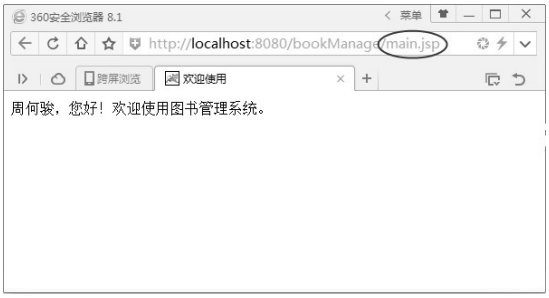



图 2.8　redirect 类型重定向的界面

### 3. redirectAction 类型

该结果类型与 redirect 类似，都是重定向而不是转发，该类型一般都为了重定向到一个新的 action 请求，而非 JSP 页面。配置该类型时，<result>可以配置如下两个参数。

- actionName：该参数指定重定向的 action 名。
- namespace：该参数指定需要重定向的 action 所在的命名空间（命名空间会在后面讲解）。

注意这些参数是可选配置的，不是必需的，在实际情况中可以根据需要配置。

看下面一段代码：

```
...
<package name="test1" extends="struts-default">
        <action name="regist" class="org.action.LoginAction" method="regist">
                <result name="success" type="redirectAction">
                        <param name="actionName">login</param>
                        <param name="namespace">/test2</param>
                </result>
        </action>
</package>
<package name="test2" extends="struts-default" namespace="/test2">
        <action name="login" class="org.action.LoginAction" method="login">
                <result name="success">main.jsp</result>
        </action>
</package>
...
```

上面代码就是 redirectAction 类型应用的体现。首先对“test1”包中的“regist”进行请求，通过 LoginAction 类中的 regist 方法来处理请求，完成后用 redirectAction 结果类型来重新定向到“test2”包中的“login”，然后用 LoginAction 类中的 login 方法处理，完成后跳转到“main.jsp”页面，由于“test2”包指定了命名空间“namespace”，所以必须配置参数指定：

```
<param name="actionName">login</param>
<param name="namespace">/test2</param>
```

分别指定重定向的 action 名及该 action 所在的命名空间。

4. chain 类型

前面的 redirect 及 redirectAction 虽然都可以重定向到另外的 action，但是它们都不能实现数据的传递，在重定向过程中，请求属性等都会丢失，这样有的时候就不利于编程了。因此，Struts 2 又提供了另一种结果类型“chain”，用来实现 action 之间的跳转，而非重定向，意思就是可以跳转到另外的 action 而且数据不丢失，通过设置 chain 类型，可以组成一条 action 链，不用担心数据的丢失，这样就大大方便了编程。action 跳转可以共享数据的原理是处于同一个 action 链的 action 都共享同一个值栈，每个 action 执行完毕后都会把数据压入值栈，如果需要就直接到值栈中提取。

5. 全局结果

从前面的例子中可以看出，<result>都是包含在<action>...</action>中的，这配置的是局部结果，只对当前 action 请求返回的结果起作用。假如都返回到同一页面，而且在不同的 action 请求中都会用到，那么配置局部结果就显得冗余了。所以，Struts 2 提供了全局结果的配置，例如，如果返回“error”，都跳转到错误页面：

```
...
<struts>
        <package name="default" extends="struts-default">
            <global-results>
                <result name="error">error.jsp</result>
            </global-results>
            <action name="login" class="org.action.LoginAction">
                ...
            </action>
            <action name="test2" class="org.action.TestAction">
                ...
            </action>
        </package>
</struts>
```

上面代码的加黑部分定义了一个全局结果，当用户请求处理完成后，如果返回“error”，就会到当前 action 配置的返回中寻找，如果没有找到就会到全局结果中寻找。也就是说，局部结果配置的优先级大于全局结果。

## 2.5.3 <package>配置详解

从前面的例子可以看出，Struts 2 的 action 都是放在<package>...</package>中的。package 元素用于定义 struts.xml 中的包配置，<package>中可以定义 action 和拦截器等。用 package 定义包配置时可以指定 4 个属性和 8 个标签。

1. 可配置的属性

（1）name 属性

该属性必须指定，代表包的名称，由于 struts.xml 中可以定义不同的<package>，而且它们之间还可以互相引用，所以必须指定名称。

（2）extends 属性

该属性是可选的，表示当前定义的包继承其他的包，继承了其他包，就可以继承其他包中的 action、拦截器等。由于包信息的获取是按照配置文件中的先后顺序进行的，所以父包必须在子包之前被定义。

在一般情况下，定义包时都会继承一个名为“struts-default”的包，该包是 Struts 2 内置的，定义

在 struts-default.xml 这个文件中。这个文件的位置在前面讲解<result>结果类型的时候已经说过，打开该文件找到这个包，可以发现该包下定义了一些结果类型、拦截器及拦截器栈，结果类型在前面已经讲解，拦截器会在后面详细讲解。

（3）namespace 属性

该属性是可选的，用来指定一个命名空间，如在前面讲 redirectAction 类型时已经用到了，定义命名空间非常简单，只要指定“namespace="/*"”即可，其中“*”是我们自定的，如果直接指定“"/"”，表示设置命名空间为根命名空间。如果不指定任何 namespace，则使用默认的命名空间，默认的命名空间为“" "”。

当指定了命名空间后，相应的请求也要改变，例如：

```
<action name="login" class="org.action.LoginAction" namespace="/user">
    ...
</action>
```

请求就不能是“login”，而必须改为“user/login”。当 Struts 2 接收到请求后，会将请求信息解析为 namespace 名和 action 名两部分，然后根据 namespace 名在 struts.xml 中查找指定命名空间的包，并且在该包中寻找与 action 名相同的配置，如果没有找到，就到默认的命名空间中寻找与 action 名称相同的配置，如果还没找到，就给出错误信息。看下面的代码：

```
<package name="default">
    <action name="foo" class="org.TestAction">
        <result name="success">foo.jsp</result>
    </action>
    <action name="bar" class=" org.TestAction ">
        <result name="success">bar.jsp</result>
    </action>
</package>
<package name="mypackage1" namespace="/">
    <action name="moo" class=" org.TestAction ">
        <result name="success">moo.jsp</result>
    </action>
</package>
<package name="mypackage2" namespace="/barspace">
    <action name="bar" class=" org.TestAction ">
        <result name="success">bar.jsp</result>
    </action>
</package>
```

如果页面中请求为 barspace/bar.action，框架将首先在命名空间为/barspace 的包中查找 bar 这个 action 配置，如果找到了，则执行 bar.action；如果没有找到，则到默认的命名空间中继续查找。在本例中，/barspace 命名空间中有名为 bar 的 Action，因此它会被执行。

如果页面中请求为 barspace/foo.action，框架会在命名空间为/barspace 的包中查找 foo 这个 action 配置。如果找不到，框架会到默认命名空间中去查找。在本例中，/barspace 命名空间中没有 foo 这个 action，因此默认的命名空间中的/foo.action 将会被找到并执行。

如果页面中请求为 moo.action，框架会在根命名空间“/”中查找 moo.action，如果没有找到，再到默认命名空间中查找。

（4）abstract 属性

该属性是可选的，如果定义该包是一个抽象包，则该包不能包含<action>配置信息，但可以被继承。

2. 可配置的标签

<package>主要包含以上 4 个属性，<package>下面还可配置以下几个标签。

● <action>：action 标签，其作用前面已经详细讲解。

● <default-action-ref>：配置默认 action。如果配置了默认 action，则当请求的 action 名在包中找不到与之匹配的名称时就会应用默认 action。

● <default-class-ref>：配置默认类。

● <default-interceptor-ref>：配置默认拦截器。

● <global-exception-mappings>：配置发生异常时对应的视图信息，为全局信息，与之对应还有局部异常配置，局部异常配置要配置在<action>标签中，局部异常配置用<exception-mapping>进行配置。配置异常信息格式如下：

```
<package name="default" extends="struts-default">
    <global-exception-mappings>
        <exception-mapping result="逻辑视图" exception="异常类型"/>
    </global-exception-mappings>
    <action name="action 名称">
        <exception-mapping result="逻辑视图" exception="异常类型"/>
    </action>
</package>
```

<exception-mapping>中可以指定 3 个属性，分别为 name：可选属性，用来标识该异常配置信息；result：该属性必须指定，指定发生异常时显示的视图信息，必须配置为逻辑视图；exception：该属性必须指定，用来指定异常类型。

● <global-results>：配置全局结果，前面已经讲述。

● <interceptors>：配置拦截器。

● <result-types>：配置结果类型。

拦截器知识会在第 5 章拦截器部分讲解，读者在这里了解即可。

## 习 题 2

1. 简述 MVC 及其优点。
2. MVC 有哪几种实现方式？
3. 简述 Struts 2 的工作流程。
4. 应用 Struts 2 框架，开发一个加法器，采用两个页面：一个输入数据，另一个输出结果。