轻松入门之 Struts2 V3.0

前 言

Struts2 其实并不是一个陌生的 Web 框架，Struts2 是以 Webwork 的设计思想为核心，吸收了 Struts1 的优点，因此，可以认为 Struts2 是 Struts1 和 Webwork 结合的产物。

Struts2 的使用人群逐渐增多，它在逐步吸引开发者和用户的目光，毫无疑问，大家最终都会选择于它，困为它确实是一个很优秀的框架。Struts2 方面的书籍很多，如果你是想精通 strtus2，那么将很遗憾地告知你，本书内容不适合你；如果你想花最少的时间来获取对 struts2 的最大了解程度，则它将是你最好的选择。本着学习共享的精神，将总结的资料与所有的朋友共享，希望各位朋友多多指教。

感谢周大庆和聂静宇等朋友的参与，本教程又添加了相应的内容，并对以前的细节做了调整。由于时间关系，未能全部完善，希望更多的朋友参与进来，集众人所长，所快乐分享。由于精力有限，有些内容未细细推敲，有些内容攫取于网络，如果无意之间侵犯了您的版权，请您来信告知，我将尽快删除。如果知识点有误，感谢告知。

从 2.0 发布至今已有三个多月，在这三个多月中发生了很多事情，尤其是姑姑的去世让我悲痛万分。希望身边的朋友多关注健康，重视健康。这段时间也一直忙于基于 struts2 的项目开发，直到项目临近尾声之时才抽出空儿来对本文档进行更新。希望更多的朋友参与进来，把最实用的知识展现出来，让更多的初学者朋友少走弯路，快速上手。

欢迎免费索取本资料，欢迎相互交流。本资料仍在不断完善之中，可以通过加入群、发邮件或关注 blog 来获取最新资料。

如果你的 JavaWeb 基础不够夯实，请参看 JavaWeb 书籍，如果想对 struts2 有深入了解，请与 JSF 比照学习，以下书籍向大家推荐，书籍中的案例均具有可操作性。

《JSF编程》清华出版社出版：详情请查询：http://www.china-pub.com/39347

《Java Web整合》人民邮电出版社出版：详情请查询：http://www.china-pub.com/195118

《精通JSF》人民邮电出版社出版：详情请查询：http://www.china-pub.com/195117

技术交流 QQ：53983038 文档学习 QQ 群：76434335

作者Email：qhxx@tom.com

交流blog:http://xmh517.javaeye.com/

目 录

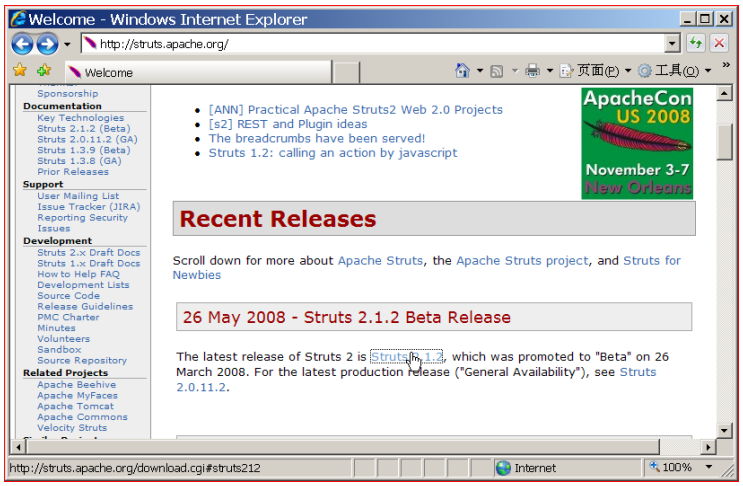
第 1 章 Struts2 入门

Struts2 的学习持续有一段时间了，也经历个几个小项目。身边也有一些朋友在逐步使用 struts2，为了减轻他们的学习曲线，我把自己的学习过程用文字记录下来，供他们参考。现在也决定把它共享出来，请大家批评指正。

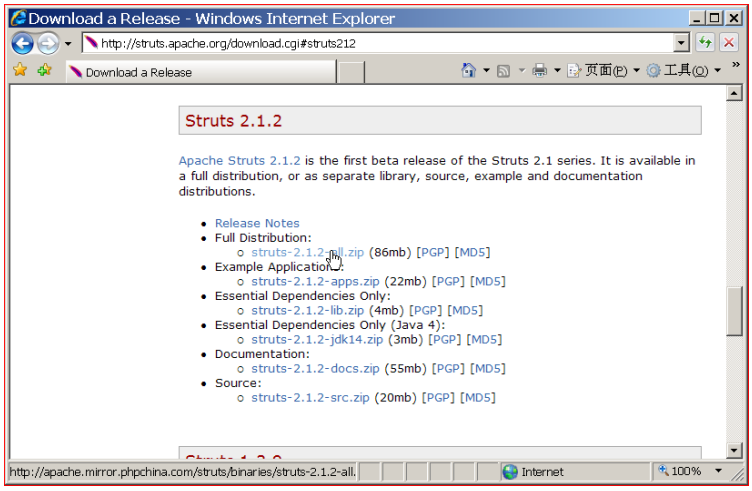
Struts2 虽然简单，但要一一描述清楚，可能会形成厚厚地一本书。所以我这里仅仅记录与工作相关的内容。有些部分请读者朋友查看相应的帮助文档或专业书籍。

一、环境说明

1. Struts 版本：2.1.2



此处下载的是 struts-2.1.2-all



2. JDK 版本：JDK1.5

3. Tomat 版本：6

4. MySql 版本：5.0

6. MyEclipse 版本：6

除无特别说明，后面的程序都基于以上环境。有些内容是基于 struts-2.1.2 所提供的案例。

当提交一个 HTML 表单到这个框架中的时候，输入并不是被发送到服务页，而是被发送到你提供的 Java 类，这些类被称为 Action。在这些 Action 执行完后，选择某一个资源来呈现返回结果，这个资源一般是页面，但也可以是 PDF 文件，或者是 Excel 文件，亦或是 Java applet 窗口。

假设你想要创建一个“Hello World”的例子来呈现欢迎信息，在你准备好开发环境后，为了创建一个“Hello World”的例子，你需要做如下三件事情：

1、创建一个 jsp 页面来呈现欢迎信息；

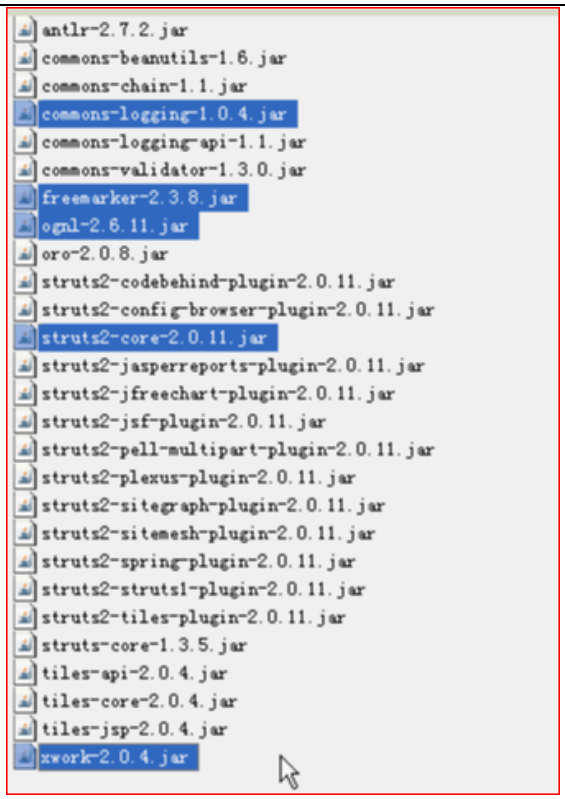
2、创建一个 Action 类来创建信息；

3、在配置文件中配置 action 和页面的映射关系。

注意：为了创建这个组件，我们将工作流分成几乎无人不晓的三部分：视图、模型和控制器。分离这三部分的原因是当系统变得越来月复杂的时候，我们能够更好的管理。

一．准备工作

建立 web 工程，其中工程名为 tutorial，在 WebRoot 下引入 struts2 的 lib 下的如下 4 个包：



在 web.xml 文件中增加 struts2 的 FilterDispatcher，修改后的 web.xml 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">
    <display-name>tutorial</display-name>
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
</web-app>
```

注意：struts2 配置了一个过滤器来执行分发功能。

二．代码

1、我们需要一个 jsp 页面来呈现信息，HelloWorld.jsp 页面代码如下所示：

```
<%@ page contentType=" text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title>Hello World!</title>
    </head>
    <body>
```

```
        <h2>
            <s:property value="message" />
        </h2>
    </body>
</html>
```

Struts2 的标签引用也简单多了，只需要一句就可以。

2、我们需要一个 Action 类来创建信息，代码如下：

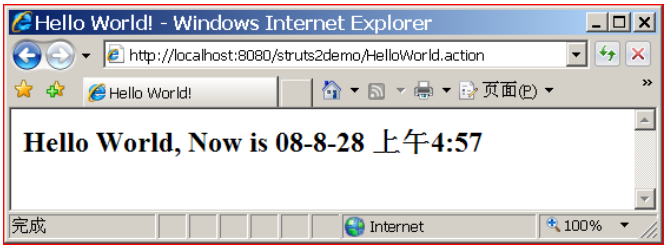
```
package org.xmh.demo;

import java.util.Date;
import java.text.DateFormat;
import com.opensymphony.xwork2.ActionSupport;

public class HelloWorld extends ActionSupport {
    private String message;

    public String getMessage() {
        return message;
    }

    public String execute() {
        message = "Hello World, Now is "
                + DateFormat.getInstance().format(new Date());
        return SUCCESS;
    }
}
```

3、我们需要在配置文件中进行相应配置来将两者联系起来。让我们编辑 src 下的 struts.xml 文件，该文件内容如下：

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="tutorial" extends="struts-default">
        <action name="HelloWorld" class="tutorial.HelloWorld">
            <result>/HelloWorld.jsp</result>
        </action>
        <!-- Add your actions here -->
    </package>
</struts>
```

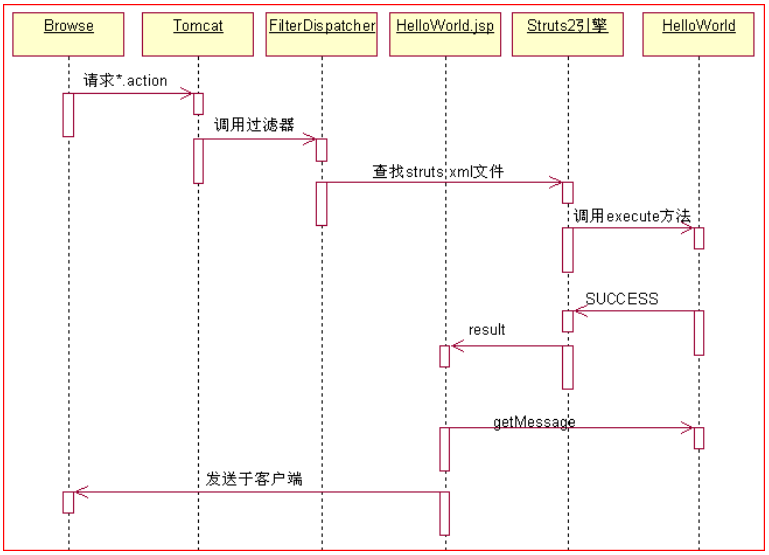
4、发布与测试。

部署该应用程序并在 IE 地址栏中输入 http://localhost:8080/struts2demo/HelloWorld.action，可以看到标题栏为：Hello World，并且显示“Struts is up and running!”的页面呈现在我们面前。



三、代码是如何工作的？

上述 Hello World 的示例用处理流程如下所示：



用文字来描述这个处理过程如下：

①浏览器请求http://localhost:8080/tutorial/HelloWorld.action，发送到web应用服务器。

②容器接收到了 Web 服务器对资源 HelloWorld.action 的请求，根据 web.xml 中的配置，服务器将包含有.action 后缀的请求转到 org.apache.struts2.dispatcher.FilterDispatcher 类进行处理。调用这个 FilterDispatcher，进入 struts2 的流程中。

③框架在 struts.xml 配置文件中查找名为 HelloWorld 的 action 对应的类。框架初始化该 Action 并且执行该 Action 类的 execute 方法；

④execute 方法将信息放入 message 变量中，并返回成功。

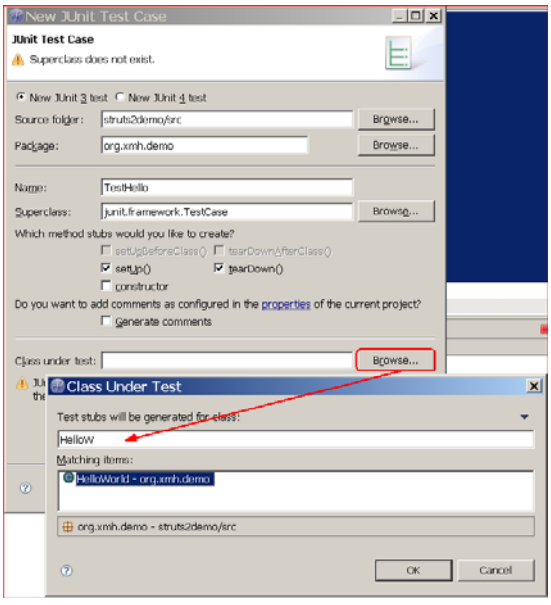
⑤框架检查配置以查看当返回成功时对应的页面。框架告诉容器来获得请求返回的结果页面 HelloWorld.jsp;

⑥<s:property value="message" />标签调用 HelloWorld 的 Action 类中的 getMessage 方法来获得 message 的值。

⑦标签显示 message 并以 HTML 格式呈现给用户；

四、测试 Action

测试 Action 很简单，以下是上述 HelloWorld 的 Action 类的测试类的代码：



```
package org.xmh.demo;

import junit.framework.TestCase;

import com.opensymphony.xwork2.ActionSupport;

public class TestHello extends TestCase {
    public void testHelloWorld() throws Exception {
        HelloWorld hello_world = new HelloWorld();
        String result = hello_world.execute();
        assertTrue(ActionSupport.SUCCESS
                .equals(result));
    }
}
```

五、需要记住的东西

本框架利用 Action 来处理 HTML 的表单以及其余请求。Action 返回一个结果的名字字符串，例如 SUCCESS、ERROR 以及 INPUT 等，从 struts.xml 中获取映射信息。一个给定的结果字符串将选择一个页面或其他资源（图片或 PDF）来返回给用户。

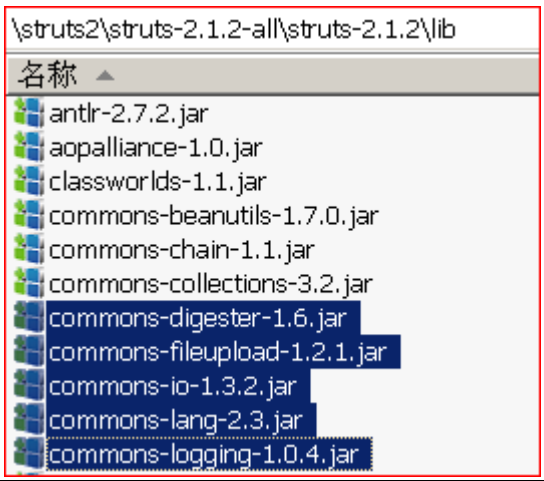
当一个 jsp 被载入的时候，通常有一些动态变化的元素需要 Action 来载入。为了更加容易的显示动态数据，本框架提供了一些可以跟 HTML 配合使用的标签。

六、问题汇总

如果你的程序无法正常运行，并且显示如下的错误：

```
严重: Exception starting filter struts2
Unable to load configuration. - bean -

jar:file:/F:/Struts2/Struts2/WebRoot/WEB-INF/lib/struts2-core-2.1.2.jar!/struts-defau
lt.xml:46:178
at
com.opensymphony.xwork2.config.ConfigurationManager.getConfiguration(ConfigurationMan
ager.java:58)
at
org.apache.struts2.dispatcher.Dispatcher.init_PreloadConfiguration(Dispatcher.java:37
1)
at org.apache.struts2.dispatcher.Dispatcher.init(Dispatcher.java:424)
……
```

原因是由于有些文件的版本不匹配造成的，请将 lib 包中相关的 commons 组件包拷至项目之中并覆盖之即可。



如果你的程序出现如下的错误信息

严重: Could not find action or result

```
There is no Action mapped for namespace / and action name login. - [unknown location]
at com.opensymphony.xwork2.DefaultActionProxy.prepare(DefaultActionProxy.java:186)
at
org.apache.struts2.impl.StrutsActionProxyFactory.createActionProxy(StrutsActionProxyFactory.java:41)
at org.apache.struts2.dispatcher.Dispatcher.serviceAction(Dispatcher.java:494)
……
at java.lang.Thread.run(Unknown Source)
```

请确保你的 struts2.xml 放在 classes 下，亦即你的 struts2.xml 是否位于 src 目录之下。

第 2 章 Struts2 晋级

Struts2 实际上是在 Webwork 基础上构建起来的 MVC 框架。从 Struts2 的源代码中可以看到，有很多都是直接使用的 xwork(Webwork 的核心技术)的包。既然从技术上来说 Struts2 是全新的框架，那么就让我们来学习一下这个新的框架的使用方法。

如果大家使用过 Struts1.x，应该对建立基于 Struts1.x 的 Web 程序的基本步骤非常清楚。让我们先来回顾一下建立基于 Struts1.x 的 Web 程序的基本步骤。

1、安装 Struts。由于 Struts 的入口点是 ActionServlet，所以得在 web.xml 中配置一下这个 Servlet。

2、编写 Action 类（一般从 org.apache.struts.action.Action 类继承）。

3、编写 ActionForm 类（一般从 org.apache.struts.action.ActionForm 类继承），这一步不是必须的，如果要接收客户端提交的数据，需要执行这一步。

4、在 struts-config.xml 文件中配置 Action 和 ActionForm。

5、如果要采集用户录入的数据，一般需要编写若干 JSP 页面，并通过这些 JSP 页面中的 form 将数据提交给 Action。

下面按编写 struts1.x 程序的这五步和 struts2.x 程序的编写过程一一对应，看看它们的异同。编写一个基于 Struts2 的 Web 程序。这个程序的功能是让用户录入两个整数，并提交给一个 Struts Action，并计算这两个数的代数和，如果代码和为非负数，则跳转到 positive.jsp 页面，否则跳转到 negative.jsp 页面。

【第 1 步】 安装 Struts2

这一步对于 Struts1.x 和 Struts2 都是必须的，只是安装的方法不同。Struts1 的入口点是一个 Servlet，而 Struts2 的入口点是一个过滤器(Filter)。因此，Struts2 要按过滤器的方式配置。下面是在 web.xml 中配置 Struts2 的代码：

```
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>
        org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

【第 2 步】 编写 Action 类

这一步和 Struts1.x 也必须进行。只是 Struts1.x 中的动作类必须从 Action 类中继承，而 Struts2.x 的动作类需要从 com.opensymphony.xwork2.ActionSupport 类继承。下面是计算两个整数代码和的

Action 类，代码如下：

```
package action;
import com.opensymphony.xwork2.ActionSupport;
public class FirstAction extends ActionSupport
{
    private int operand1;
    private int operand2;
    public String execute() throws Exception
    {
        if (getSum() >= 0)  // 如果代码数和是非负整数，跳到 positive.jsp 页面
        {
            return "positive";
        } else  // 如果代码数和是负整数，跳到 negative.jsp 页面
        {
            return "negative";
        }
    }

    public int getOperand1()
    {
        return operand1;
    }

    public void setOperand1(int operand1)
    {
        System.out.println(operand1);
          this.operand1 = operand1;
    }

    public int getOperand2()
    {
        return operand2;
    }
    public void setOperand2(int operand2)
    {
        System.out.println(operand2);
        this.operand2 = operand2;
    }
    public int getSum()
    {
        return operand1 + operand2;  // 计算两个整数的代码数和
    }
}
```

从上面的代码可以看出，动作类的一个特征就是要覆盖 execute 方法，只是 Struts2 的 execute 方法没有参数了，而 Struts1.x 的 execute 方法有四个参数。而且 execute 方法的返回值也不同的。Struts2 只返回一个 String，用于表述执行结果（就是一个标志）。上面代码的其他部分将在下面讲解。

【第 3 步】 编写 ActionForm 类

在本例中当然需要使用 ActionForm 了。在 Struts1.x 中，必须要单独建立一个 ActionForm 类（或是定义一个动作 Form），而在 Struts2 中 ActionForm 和 Action 已经二合一了。从第二步的代码可以看出，后面的部分就是应该写在 ActionForm 类中的内容。所以在第 2 步，本例的 ActionForm 类已经编写

完成（就是 Action 类的后半部分）。

【第 4 步】 配置 Action 类

这一步 struts1.x 和 struts2.x 都是必须的，只是在 struts1.x 中的配置文件一般叫 struts-config.xml（当然也可以是其他的文件名），而且一般放到 WEB-INF 目录中。而在 struts2.x 中的配置文件一般为 struts.xml，放到 WEB-INF/classes 目录中。下面是在 struts.xml 中配置动作类的代码：

```
<?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE struts PUBLIC
      "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
      "http://struts.apache.org/dtds/struts-2.0.dtd">
  <struts>
      <package name="struts2" namespace="/mystruts"
          extends="struts-default">
          <action name="sum" class="action.FirstAction">
              <result name="positive">/positive.jsp</result>
              <result name="negative">/negative.jsp</result>
          </action>
      </package>
  </struts>
```

在<struts>标签中可以有多个<package>，第一个<package>可以指定一个 Servlet 访问路径（不包括动作名），如“/mystruts”。extends 属性继承一个默认的配置文件“struts-default”，一般都继承于它，大家可以先不去管它。<action>标签中的 name 属性表示动作名，class 表示动作类名。

<result>标签的 name 实际上就是 execute 方法返回的字符串，如果返回的是“positive”，就跳转到 positive.jsp 页面，如果是“negative”，就跳转到 negative.jsp 页面。在<struts>中可以有多个<package>，在<package>中可以有多个<action>。我们可以用如下的 URL 来访问这个动作：

```
http://localhost:8080/struts2/mystruts/sum.action
```

注：Struts1.x 的动作一般都以.do 结尾，而 Struts2 是以.action 结尾。

【第 5 步】 编写用户录入接口（JSP 页面）

1、主界面（sum.jsp）

在 Web 根目录建立一个 sum.jsp，代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK" %>
  <%@ taglib prefix="s" uri="/struts-tags"%>

  <html>
      <head>
          <title>输入操作数</title>
      </head>
      <body>
           求代数和
          <br/>
          <s:form action="mystruts/sum.action" >
              <s:textfield name="operand1" label=" 操作数 1"/>
              <s:textfield name="operand2"  label=" 操作数 2" />
              <s:submit value="代数和" />
          </s:form>
      </body>
  </html>
```

在 sum.jsp 中使用了 Struts2 带的 tag。在 Struts2 中已经将 Struts1.x 的好几个标签库都统一了，在 Struts2 中只有一个标签库/struts-tags。这里面包含了所有的 Struts2 标签。但使用 Struts2 的标

签大家要注意一下。在<s:form>中最好都使用 Struts2 标签，尽量不要用 HTML 或普通文本，大家可以将 sum.jsp 的代码改为如下的形式，看看会出现什么效果：

```
        ... ...
            求代数和
          <br/>
          <s:form action="mystruts/sum.action" >
操作数 1：<s:textfield name="operand1" /><br/>
操作数 2：<s:textfield name="operand1" /><br/>
              <s:submit value="代数和" />
          </s:form>
          ... ...
```

提示一下，在<s:form>中 Struts2 使用<table>定位。

2、positive.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
  <%@ taglib prefix="s" uri="/struts-tags" %>

  <html>
    <head>
      <title>显示代数和</title>
    </head>

    <body>
      代数和为非负整数<h1><s:property value="sum" /></h1>
    </body>
  </html>
```

3、 negative.jsp

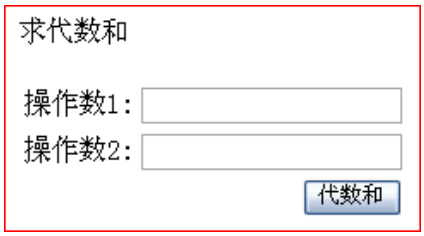
```
 <%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
  <%@ taglib prefix="s" uri="/struts-tags" %>

  <html>
    <head>
      <title>显示代数和</title>
    </head>

    <body>
      代数和为负整数<h1><s:property value="sum" /></h1>

    </body>
  </html>
```

这两个 jsp 页面的实现代码基本一样，只使用了一个<s:property>标签来显示 Action 类中的 sum 属性值。<s:property>标签是从 request 对象中获得了一个对象中得到的 sum 属性，如我们可以使用如下的代码来代替<s:property value=“sum”/>：

```
<%
 com.opensymphony.xwork2.util.OgnlValueStack ovs =
(com.opensymphony.xwork2.util.OgnlValueStack)request.getAttribute("struts.valueStack"
);
 out.println(ovs.findString("sum"));
%>
```

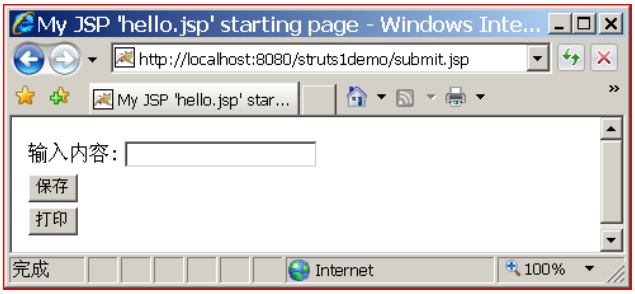
启动 Tomcat 后，在 IE 中输入如下的 URL 来测试这个例子：

http://localhost:8080/struts2/sum.jsp

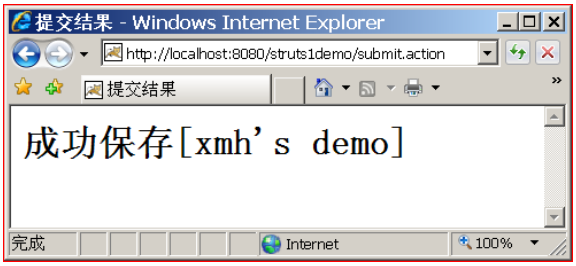


处理一个 form 多个 submit

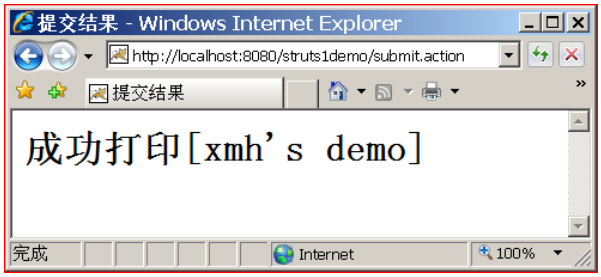
在很多 Web 应用中，为了完成不同的工作，一个 HTML form 标签中可能有两个或多个 submit 按钮，如下面的效果图所示：



当输入“xmh’s demo”单击保存按钮时：



当输入“xmh’s demo”单击打印按钮时：



由于在<form>中的多个提交按钮都向一个 action 提交，使用 Struts2 Action 的 execute 方法就无法判断用户点击了哪一个提交按钮。如果大家使用过 Struts1.x 就会知道在 Struts1.2.9 之前的版本需要使用一个 LookupDispatchAction 动作来处理含有多个 submit 的 form。但使用 LookupDispatchAction 动作需要访问属性文件，还需要映射，比较麻烦。

从 Struts1.2.9 开始，加入了一个 EventDispatchAction 动作。这个类可以通过 java 反射来调用通过 request 参数指定的动作（实际上只是判断某个请求参数是不存在，如果存在，就调用在 action 类中和这个参数同名的方法）。使用 EventDispatchAction 必须将 submit 的 name 属性指定不同的值以区分每个 submit。而在 Struts2 中将更容易实现这个功能。

当然，也可以模拟 EventDispatchAction 的方法通过 request 获得和处理参数信息。但这样比较麻烦。在 Struts2 中提供了另外一种方法，使得无需要配置可以在同一个 action 类中执行不同的方法（默认执行的是 execute 方法）。使用这种方式也需要通过请求参来来指定要执行的动作。请求参数名的格式为 action!method.action

注：由于 Struts2 只需要参数名，因此，参数值是什么都可以。

下面我就给出一个实例程序来演示如何处理有多个 submit 的 form：

【第 1 步】实现主页面

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>My JSP 'hello.jsp' starting page</title>
  </head>

  <body>
    <s:form action="submit.action" >
        <s:textfield name="msg" label="输入内容"/>
        <s:submit name="save" value="保存" align="left" method="save"/>
        <s:submit name="print" value="打印" align="left" method="print" />
    </s:form>
  </body>
</html>
```

【第 2 步】实现 Action 类

```
package org.xmh.demo;

import javax.servlet.http.HttpServletRequest;
import org.apache.struts2.interceptor.ServletRequestAware;
import com.opensymphony.xwork2.ActionSupport;

public class MoreSubmitAction extends ActionSupport implements
        ServletRequestAware {
    private String msg;
    private javax.servlet.http.HttpServletRequest request;

    // 获得 HttpServletRequest 对象
    public void setServletRequest(HttpServletRequest request) {
        this.request = request;
    }

    // 处理 save submit 按钮的动作
    public String save() throws Exception {
        request.setAttribute("result", "成功保存[" + msg + "]");
        return "save";
    }

    // 处理 print submit 按钮的动作
    public String print() throws Exception {
        request.setAttribute("result", "成功打印[" + msg + "]");
        return "print";
    }

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
```

```
        this.msg = msg;
    }
}
```

上面的代码需要注意如下两点：

save 和 print 方法必须存在，否则会抛出 java.lang.NoSuchMethodException 异常。Struts2 Action 动作中的方法和 Struts1.x Action 的 execute 不同，只使用 Struts2 Action 动作的 execute 方法无法访问 request 对象，因此，Struts2 Action 类需要实现一个 Struts2 自带的拦截器来获得 request 对象，拦截器如下：

org.apache.struts2.interceptor. ServletRequestAware

【第 3 步】配置 Struts2 的配置文件

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

<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="demo" extends="struts-default">
        <action name="submit" class="org.xmh.demo.MoreSubmitAction">
            <result name="save">/result.jsp</result>
            <result name="print">/result.jsp</result>
        </action>
    </package>

</struts>
```

【第 4 步】编写结果页

```
<%@ page pageEncoding="GBK"%>
<html>
  <head>
    <title>提交结果</title>
  </head>
  <body>
    <h1>${result}</h1>
  </body>
</html>
```

在 result.jsp 中将在 save 和 print 方法中写到 request 属性中的执行结果信息取出来，并输出到客户端。

启动 Tomcat 后，在 IE 中执行如下的 URL 来测试程序：

http://localhost:8080/struts1demo/submit..jsp

大家也可以直接使用如下的 URL 来调用 save 和 print 方法：

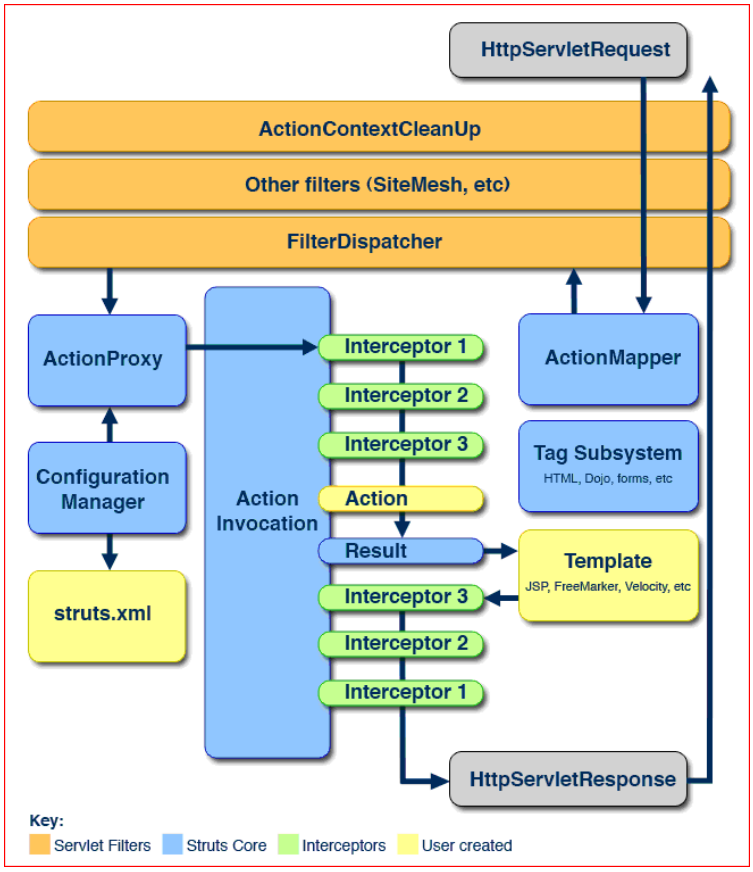
调用 save 方法：http://localhost:8080/TestS/mystruts/submit!save.action?msg=123

调用 print 方法：http://localhost:8080/TestS/mystruts/submit!print.action?msg=456

第 3 章 Struts2 核心概念

本章节将深入探讨 struts2 的核心概念，主要通过示例来说明它的使用，最后着重介绍它的三个组成部分 Action、Result、Interceptor 的原理和使用方法。

3.1 struts2 的体系结构



Struts2 的工作机制

从图可以看出，一个请求在 Struts2 框架中的处理大概分为以下几个步骤：

1、客户端初始化一个指向 Servlet 容器（例如 Tomcat）的请求；

2、这个请求经过一系列的过滤器（Filter）（这些过滤器中有一个叫做 ActionContextCleanUp 的可选过滤器，这个过滤器对于 Struts2 和其他框架的集成很有帮助，例如：SiteMesh Plugin）；

3、接着 FilterDispatcher 被调用，FilterDispatcher 询问 ActionMapper 来决定这个请求是否需要调用某个 Action；

4、如果 ActionMapper 决定需要调用某个 Action，FilterDispatcher 把请求的处理交给 ActionProxy；

5、ActionProxy 通过 Configuration Manager 询问框架的配置文件，找到需要调用的 Action 类；

6、ActionProxy 创建一个 ActionInvocation 的实例。

7、ActionInvocation 实例使用命名模式来调用，在调用 Action 的过程前后，涉及到相关拦截器（Intercepter）的调用。

8、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。返回结果通常是（但不总是，也可能是另外的一个 Action 链）一个需要被表示的 JSP 或者 FreeMarker 的模版。在表示的过程中可以使用 Struts2 框架中继承的标签。在这个过程中需要涉及到 ActionMapper。

3.2 struts2 配置文件

Struts2 相关的配置文件有 web.xml，struts.xml，struts.properties，

struts-default.xml，velocity.properties，struts-default.vm。其中 web.xml，struts.xml 是必须的，其它的配置文件可选择。它们在 web 应用中的功能如下：

web.xml：包含所有必须的框架组件的 web 部署描述符。

Struts.xml:配置包含 result/view 类型、action 映射、拦截器等的 Struts2 的主要配置文件。

Struts.properties：配置 struts2 的框架属性。

Struts-default.xml：在文件在 struts-action-x.x.jar 中，该文件是应该被包含在 struts.xml 中的缺省配置。

Welocity.properties：重写了 velocity 的配置文件。

Struts-default.vm：相对于 velocity 的缺省配置。

struts.properties 配置文件

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.

struts.properties 配置文件提供了一种改变框架默认行为的机制. 一般来讲我们没必要修改这个文件，除非你想拥有一个更加友好的开发调试环境。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.

可以被修改的属性允许改变 Freemarker 选项——改变“action-mapping”类、决定是否允许 XML 配置文件重载及确定默认用户接口主题等行为。

一个名为“default.properties”的属性文件包含在“Struts2-Core”JAR 文件中。你可以在你项目源文件路径的根目录下创建一个名为“struts.properties”的文件，来对某个属性进行修改。这样你就可以增加你想要修改的属性。新的属性值将会覆盖默认值。

以下为 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.如果需要实现自己的配置管理器,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;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,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 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.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.

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 HttpServlet Request 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,所以应该明白为什么 Struts 2 框架默认加载 struts.xml 文件了.

struts.xml 常用配置解析

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。struts.xml 内容主要包括：Action、Interceptor、Packages 和 Namespace 等，下面将会详细介绍如何配置这些元素。

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

<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="demo" extends="struts-default">
        <action name="submit" class="org.xmh.demo.MoreSubmitAction">
            <result name="save">/result.jsp</result>
            <result name="print">/result.jsp</result>
        </action>
    </package>
</struts>
```

1、使用<include>标签重用配置文件

在 Struts2 中提供了一个默认的 struts.xml 文件，但如果 package、action、interceptors 等配置比较多时，都放到一个 struts.xml 文件不太容易维护。因此，就需要将 struts.xml 文件分成多个配置文件，然后在 struts.xml 文件中使用<include>标签引用这些配置文件。这样做的优点如下：

- 结构更清晰，更容易维护配置信息。
- 配置文件可以复用。如果在多个 Web 程序中都使用类似或相同的配置文件，那么可以使用<include>标签来引用这些配置文件，这样可以减少工作量。

假设有一个配置文件，文件名为 newstruts.xml，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="demo" extends="struts-default" >
        <action name="submit"  class="action.MoreSubmitAction">
            <result name="save" >
                /result.jsp
            </result>
            <result name="print">
                /result.jsp
            </result>
        </action>
    </package>
</struts>
```

则 struts.xml 引用 newstruts.xml 文件的代码如下：

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

```
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <include file="newstruts.xml"/>
    <package name="test" extends="struts-default">
    ......
    </package>
</struts>
```

注意：

①用<include>引用的 xml 文件也必须是完成的 struts2 的配置。实际上<include>在引用时是单独解析 xml 文件，而不是将被引用的文件插入到 struts.xml 文件中。

②struts-default.xml 这个文件被包含在 struts2-core.jar 中，文件名已经可以看出这个文件的作用是 struts.xml 的默认配置，它将自动被加载后导入到 struts.xml 中去。

2、action 配置

在默认情况下，Struts2 会调用动作类的 execute 方法。但有些时候，需要在一个动作类中处理不同的动作。也就是用户请求不同的动作时，执行动作类中的不同的方法。为了达到这个目的，可以在<action>标签中通过 method 方法指定要执行的动作类的方法名，并且需要为不同的动作起不同的名子（也称为别名）。如下面代码所示：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<package name="demo" extends="struts-default" >
    <action name="test"  class="action.MyAction">
       ......
    </action>
    <action name="my"  class="action. MyAction"  method="my">
        ......
    </action>
</package>
</struts>
```

上面代码的两个动作的 class 属性都指向同一个类，name 为这个类起了两个动作别名：test 和 my。在动作 my 中，使用了 method 属性指定要要运行的方法名为 my。

在 MyAction 类中必须要有 my 方法，代码如下：

```
package action;
import com.opensymphony.xwork2.ActionSupport;
public class MyAction extends ActionSupport{
    ......
    public String execute() throws Exception {
        // 处理 test 动作的代码
    }
    public String my() throws Exception {
          // 处理 my 动作的代码
    }
    ......
}
```

除了在 struts.xml 中配置别名，还可以通过请求参数来描述指定动作（并不需要在 struts.xml 中配置）。请求参数的格式如下：

```
    http://localhost:8080/contextPath/actionName!method.action
```

关于通过请求指定动作的详细内容，请参阅前面的章节中有关处理一个 form 多个 submit 的部分。

在 struts2 中还可以为 action 指定一个或多个参数。大家还记着 struts1.x 是如何设置的 action 参数不？ 在 struts1.x 中可以使用<action>标签的 parameter 属性为其指定一个 action 参数，如果要指定多个，就只能通过逗号（,）或其他的分隔符将不同的参数隔开。而在 struts2 中可以通过<param>标签指定任意多个参数。代码如下：

```
<action name="submit"  class="action.MyAction">
    <param name="param1">value1</param>
    <param name="param2">value2</param>
    <result name="save" > /result.jsp</result>
    ......
</action>
```

当然，在 action 中读这些参数也非常简单，只需要象获取请求参数一样在 action 类中定义相应的 setter 方法即可（一般不用定义 getter 方法）。如下面的代码将读取 param1 和 param2 参数的值：

```
package action;
import com.opensymphony.xwork2.ActionSupport;
public class MyAction extends ActionSupport
{
    private String param1;
    private String param2;

    public String execute() throws Exception {
        System.out.println(param1 + param2);
    }
    public void setParam1(String param1) {
        this.param1 = param1;
    }
    public void setParam2(String param2) {
        this.param2 = param2;
    }
    ......
}
```

当 struts2 在调用 execute 之前,param1 和 param2 的值就已经是相应参数的值了,因此,在 execute 方法中可以直接使用 param1 和 param2。

3、result 配置

在默认时，<result>标签的 type 属性值是“dispatcher”（实际上就是转发，forward）。开发人员可以根据自己的需要指定不同的类型，如 redirect、stream 等。如下面代码所示：

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

如果第一个 result 的属性省略了，struts2 默认会把当作“success”。

这此 result-type 可以在 struts2-core-2.0.11.1.jar 包或 struts2 源代码中的 struts-default.xml 文件中找到，在这个文件中找到<result-types>标签，所有的 result-type 都在里面定义了。代码如下：

```
<result-types>
        <result-type name="chain" class="com.opensymphony.xwork2.ActionChainResult"/>
        <result-type name="dispatcher"
 class="org.apache.struts2.dispatcher.ServletDispatcherResult" default="true"/>
        <result-type name="freemarker"
 class="org.apache.struts2.views.freemarker.FreemarkerResult"/>
        <result-type name="httpheader"
```

```
class="org.apache.struts2.dispatcher.HttpHeaderResult"/>
       <result-type name="redirect"
 class="org.apache.struts2.dispatcher.ServletRedirectResult"/>
       <result-type name="redirectAction"
 class="org.apache.struts2.dispatcher.ServletActionRedirectResult"/>
       <result-type name="stream" class="org.apache.struts2.dispatcher.StreamResult"/>
       <result-type                                                    name="velocity"
class="org.apache.struts2.dispatcher.VelocityResult"/>
       <result-type name="xslt" class="org.apache.struts2.views.xslt.XSLTResult"/>
       <result-type                                                  name="plainText"
class="org.apache.struts2.dispatcher.PlainTextResult" />
       <!-- Deprecated name form scheduled for removal in Struts 2.1.0. The camelCase
versions are preferred. See ww-1707 -->
       <result-type name="redirect-action"
 class="org.apache.struts2.dispatcher.ServletActionRedirectResult"/>
       <result-type                                                  name="plaintext"
class="org.apache.struts2.dispatcher.PlainTextResult" />
</result-types>
```

有很多时候一个<result>被很多<action>使用，这时可以使用<global-results>标签来定义全局的<result>，代码如下：

```
<struts>
    <package name="demo" extends="struts-default">
        <global-results>
            <result name="print">/result.jsp</result>
        </global-results>
        <action name="submit" class="action.MoreSubmitAction">
        ......
        </action>
        <action name="my" class="action.MoreSubmitAction" method="my">
        ......
        </action>
    </package>
</struts>
```

如果<action>中没有相应的<result>，Struts2 就会使用全局的<result>。

4、拦截器配置

Struts2 的拦截器和 Servlet 过滤器类似。在执行 Action 的 execute 方法之前，Struts2 会首先执行在 struts.xml 中引用的拦截器，在执行完所有引用的拦截器的 intercept 方法后，会执行 Action 的 execute 方法。

在使用拦截器的时候，在 Action 里面必须最后一定要引用 struts2 自带的拦截器缺省堆栈 defaultStack,如下(这里我是引用了 struts2 自带的 checkbox 拦截器)：

```
<interceptor-ref name="checkbox">
  <param name="uncheckedValue">0</param>
</interceptor-ref>
<interceptor-ref name="defaultStack"/>
(必须加，否则出错)
```

也可以改为对全局 Action 设置自己需要的拦截器，如下：

在 struts.xml 里面定义全局的配置设置

```
    <package name="struts-shop" extends="struts-default">
    <interceptors>
      <interceptor-stack name="myStack">
```

```
        <interceptor-ref name="checkbox">
          <param name="uncheckedValue">0</param>
        </interceptor-ref>
        <interceptor-ref name="defaultStack"/>
      </interceptor-stack>
    </interceptors>
    <default-interceptor-ref name="myStack"/>（这句是设置所有 Action 自动调用的拦截器堆栈）
  </package>
```

struts-action.xml 里面配置 Action 如下：

```
  <package name="LogonAdmin" extends="struts-shop">（这里扩展 struts.xml 里面定义的配置就可以了）
  <action name="logon" class="logonAction">
     <result>/jsp/smeishop/admin/index.jsp</result>
     <result name="error">/jsp/smeishop/admin/logon.jsp</result>
     <result name="input">/jsp/smeishop/admin/logon.jsp</result>
   </action>
   <action name="logout" class="logoutAction">
     <result>/jsp/smeishop/admin/logon.jsp</result>
   </action>
 </package>
```

引用名既可以是拦截器名也可以是栈名。

5、包配置

使用 package 可以将逻辑上相关的一组 Action，Result，Interceptor 等组件分为一组，Package 有些像对象，可以继承其他的 Package，也可以被其他 package 继承，甚至可以定义抽象的 Package。

Package 的可以使用的属性：

属性	是否必须	说明
name	是	Package 的表示，为了让其他的 package 引用
extends	否	从哪个 package 继承行为
namespace	否	参考 Namespace 配置说明
abstract	否	定义这个 package 为抽象的，这个 package 中不需要定义 action

由于 struts.xml 文件是自上而下解析的，所以被继承的 package 要放在继承 package 的前边。Namespace 将 action 分成逻辑上的不同模块，每一个模块有自己独立的前缀。使用 namespace 可以有效的避免 action 重名的冲突，例如每一个 package 都可以有自己独立的 Menu 和 Help action，但是事项方式各有不同。Struts2 标签带有 namespace 选项，可以根据 namespace 的不同向服务器提交不同的 package 的 action 的请求。

“/”表示根 namespace，所有直接在应用程序上下文环境下的请求（Context）都在这个 package 中查找。

“”表示默认 namespace，当所有的 namespace 中都找不到的时候就在这个 namespace 中寻找。

例如，有如下配置：

CODE:

```
<package name="default">
    <action name="foo" class="mypackage.simpleAction>
             <result name="success" type="dispatcher">
                  greeting.jsp
             </result>
    </action>
    <action name="bar" class="mypackage.simpleAction">
```

```
        <result name="success" type="dispatcher">bar1.jsp</result>
    </action>
</package>

<package name="mypackage1" namespace="/">
   <action name="moo" class="mypackage.simpleAction">
       <result name="success" type="dispatcher">moo.jsp</result>
   </action>
</package>

<package name="mypackage2" namespace="/barspace">
   <action name="bar" class="mypackage.simpleAction">
      <result name="success" type="dispatcher">bar2.jsp</result>
   </action>
</package>
```

①如果请求为/barspace/bar.action 查找 namespace：/barspace，如果找到 bar 则执行对应的 action，否则将会查找默认的 namespace，在上面的例子中，在 barspace 中存在名字为 bar 的 action，所以这个 action 将会被执行，如果返回结果为 success，则画面将定为到 bar2.jsp

②如果请求为/moo.action，则 namespace（'/'）被查找，如果 moo .action 存在则执行，否则查询默认的 namespace，上面的例子中，根 namespace 中存在 moo action，所以该 action 被调用，返回 success 的情况下画面将定位到 moo.jsp。

3.3 Action配置

在 struts2 框架中每一个 Action 是一个工作单元。Action 负责将一个请求对应到一个 Action 处理上去，每当一个 Action 类匹配一个请求的时候，这个 Action 类就会被 Struts2 框架调用。

一个 Action 配置示例：

```
<action name="Logon" class="tutorial.Logon">
   <result type="redirect-action">Menu</result>
   <result name="input">/tutorial/Logon.jsp</result>
</action>
```

每一个 Action 可以配置多个 result，多个 ExceptionHandler，多个 Intercepter，但是只能有一个 name，这个 name 和 package 的 namespace 来唯一区别一个 Action。

每当 struts2 框架接受到一个请求的时候，它会去掉 Host，Application 和后缀等信息，得到 Action 的名字，如下的请求将得到 Welcome 这个 Action。http://www.517.org/struts2 /Welcome.action

在一个 Struts2 应用程序中，一个指向 Action 的链接通常有 Struts Tag 产生，这个 Tag 只需要指定 Action 的名字，Struts 框架会自动添加诸如后缀等的扩展，例如：

```
<s:form action="Hello">
<s:textfield label="Please enter your name" name="name"/>
<s:submit/>
</s:form
```

将产生一个如下的链接的请求: http://Hostname:post/appname/Hello.action

在定义 Action 的名字的时候不要使用“.”和“/”，最好使用英文字母和下划线。

Action 的默认入口方法由 xwork2 的 Action 接口来定义，代码清单为：

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

```
}
```

有些时候我们想指定一个 Action 的多个方法，我们可以做如下两步：

步骤 1、建立一些 execute 签名相同的方法，例如：

```
Public String forward() throws Exception
```

步骤 2、在 Action 配置的时候使用 method 属性，例如：

```
<action name="delete" class="example.CrudAction" method="delete">
```

默认情况下，当请求 HelloWorld.action 发生时，Struts 运行时(Runtime)根据 struts.xml 里的 Action 映射集(Mapping)，实例化 tutoiral.HelloWorld 类，并调用其 execute 方法。当然，可以通过以下两种方法改变这种默认调用。这个功能（Feature）有点类似 Struts 1.x 中的 LookupDispathAction。

1、在 classes/sturts.xml 中新建 Action，并指明其调用的方法；

2、访问 Action 时，在 Action 名后加上“!xxx”（xxx 为方法名）。

实现方法请参考例 2：

在 classes/ struts2demo/HelloWorld.java 中加入以下方法：

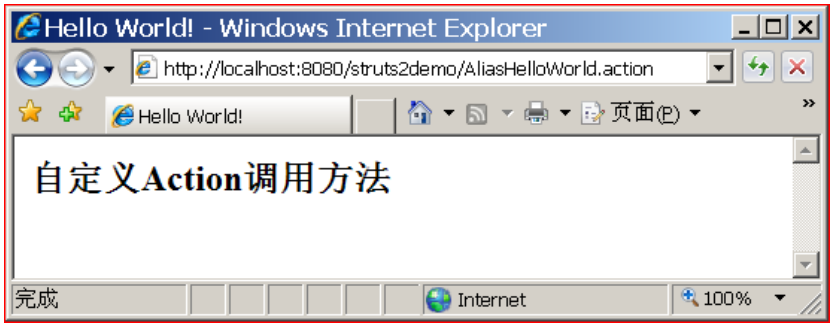
```
public String aliasAction() {
    message ="自定义 Action 调用方法";
    return SUCCESS;
}
```

在 classes/sturts.xml 中加入下面代码：

```
<action name="AliasHelloWorld" class="org.xmh.demo.HelloWorld" method="aliasAction">
   <result>/HelloWorld.jsp</result>
</action>
```

使用 http://localhost:8080/struts2demo/HelloWorld**!aliasAction**.action 地址来访问 HelloWorld Action。

在浏览器地址栏中键入 http://localhost:8080/struts2demo/AliasHelloWorld.action 或 http://localhost:8080/ struts2demo /HelloWorld!aliasAction.action，可以看到如图 2 所示页面。



通过上面的个例子，细心的朋友应该可能会发现 classes/struts2demo/HelloWorld.java 中 Action 方法（execute 和 aliasAction）返回都是 SUCCESS。这个属性变量并没有定义，所以大家应该会猜到它在 ActionSupport 或其父类中定义。没错，SUCCESS 在接口 com.opensymphony.xwork2.Action 中定义，另外同时定义的还有 ERROR, INPUT, LOGIN, NONE。

此外，在配置 Action 时都没有为 result 定义名字（name），所以它们默认都为 success。值得一提的是 Struts 2.0 中的 result 不仅仅是 Struts 1.x 中 forward 的别名，它可以实现除 forward 外的很激动人心的功能，如将 Action 输出到 FreeMaker 模板、Velocity 模板、JasperReports 和使用 XSL 转换等。这些都过 result 里的 type（类型）属性（Attribute）定义的。另外，您还可以自定义 result 类型。

下面让我们来做一个 Velocity 模板输出的例子，首先在 classes/struts.xml 中新建一个 Action 映射（Mapping），将其 result 类型设为 velocity，如以下代码所示：

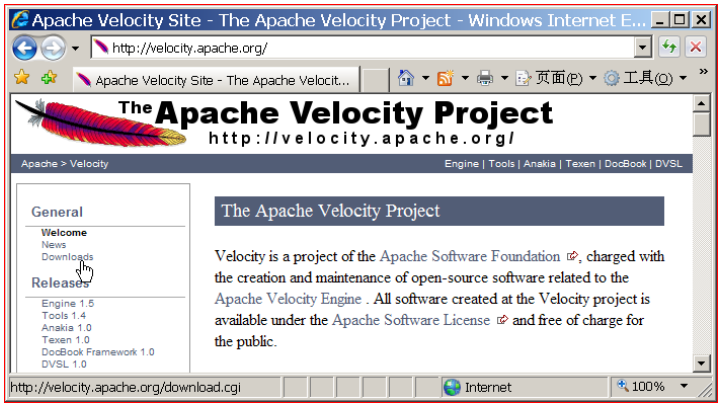
```
<action name="VMHelloWorld" class="org.xmh.demo.HelloWorld">
```

```
    <result type="velocity">/HelloWorld.vm</result>
</action>
```

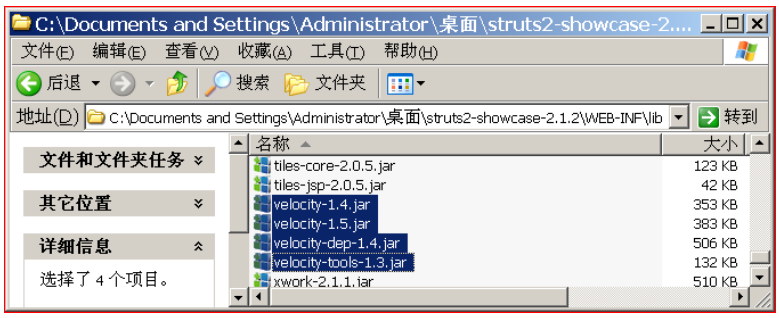
新建 HelloWorld.vm，内容如下所示：

```
<html>
  <head>
    <title>Velocity</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <h2>Message rendered in Velocity: $message</h2>
  </body>
</html>
```

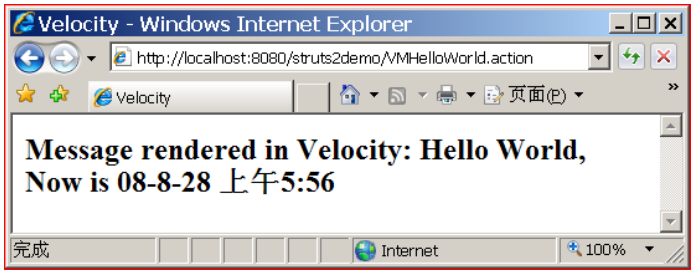
注意在使用 Velocity 模板的时候需要导入相关的包，可以在其官方网站 http://velocity.apache.org下载。



也可以直接打开它自带 showcase 案例，在其 lib 中拷贝相关的 jar 文件。



在 IE 地址栏中键入 http://localhost:8080/Struts2_Action/VMHelloWorld.action，页面输出如下图所示。



如果想对 Velocity 做深入了解，请参见我发布的相关文档。

◆ **Action 中的方法通配符**

有些时候对 Action 中方法的调用满足一定的规律，例如 edit Action 对应 edit 方法，delete Action 对应 delete 方法，这个时候我们可以使用方法通配符，例如：

<action name="*Crud" class="example.Crud" method="{1}"> 这时，editCrudAction 的引用将调用 edit 方法，同理，deleteCrudAction 的引用将调用 delete 方法。

另外一种比较常用的方式是使用下划线分割，例如：

```
<action name="Crud_*" class="example.Crud" method="{1}">
```

这样当遇到如下调用的时候可以找到对应的方法。

```
"action=Crud_input"   =>   input 方法
"action=Crud_delete"  =>   delete 方法
```

通配符和普通的配置具有相同的地位，可以结合使用框架的所有其他功能。

默认的 Action

当我们没有指定 Action 的 class 属性的时候，默认使用 com.opensymphony.xwork.ActionSupport ActionSupport 有两个方法 input 和 execute，每个方法都是简单的返回 SUCCESS。

通常情况下，请求的 Action 不存在的情况下，Struts2 框架会返回一个 Error 画面：“404 - Page not found”，有些时候或许我们不想出现一个控制之外的错误画面，我们可以指定一个默认的 Action，在请求的 Action 不存在的情况下，调用默认的 Action，通过如下配置可以达到要求：

```
<package name="Hello" extends="action-default">
<default-action-ref name="UnderConstruction">
<action name="UnderConstruction">
   <result>/UnderConstruction.jsp</result>
</action>
<action name="*" >
   <result>/{1}.jsp</result>
</action>
```

每个 Action 将会被映射到以自己名字命名的 JSP 上。

第 4 章 表单验证

前面，花了不少的时间讨论 Action 的输出。其实 web 程序搞来搞去，有何新意？程序无非就是输入、操作和输出。因此，接下来讨论一下输入——表单输入。

使用 Struts 2.0，表单数据的输入将变得非常方便，和普通的 POJO 一样在 Action 编写 Getter 和 Setter，然后在 JSP 的 UI 标志的 name 与其对应，在提交表单到 Action 时，就可以取得其值。

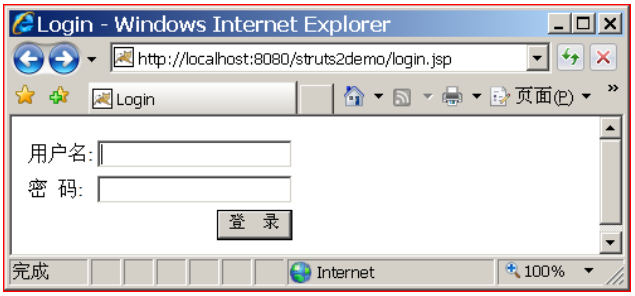
让我们看一个例子，新建 Login Action，它通过 Login.jsp 的表单获得用户名和密码，验查用户名是否为“xmh”，密码是否则为“xmh”。如果，两者都符合，就在 HelloWorld 中显示“Welcome, xmh”，否则显示“Invalid user or Password”。

```
package org.xmh.demo;

import com.opensymphony.xwork2.ActionSupport;

public class Login extends ActionSupport {
    private String name;
    private String password;
    private String message;

    ……省略 set/get 方法

    public String getMessage() {
        return message;
    }
```

```
    @Override
    public String execute() {
        if ("xmh".equals(name) && "xmh".equals(password)) {
            message = "Welcome, " + name;
        } else {
            message = "Invalid user or password";
        }
        return SUCCESS;
    }
}
```

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
    <title>Login</title>
</head>
<body>
<s:form action="Login" method="POST">
    <s:textfield name="name" label="用户名"/>
    <s:password name="password" label="密 码"/>
    <s:submit value="登  录"/>
</s:form>
</body>
</html>
```

http://localhost:8080/struts2demo/login.jsp，出现如图 4 所示页面。



分别在 User name 中输入“xmh”和“xmh”，点击“登录”按钮，出现如图所示页面。

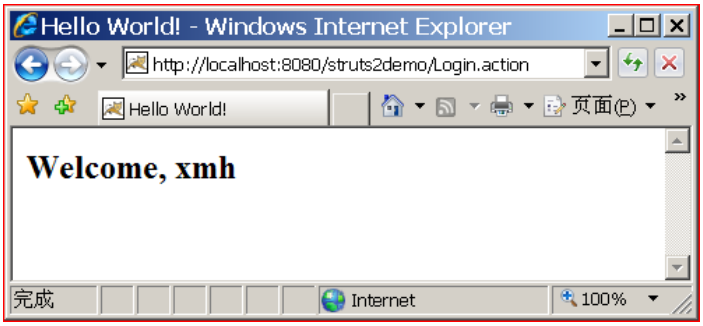
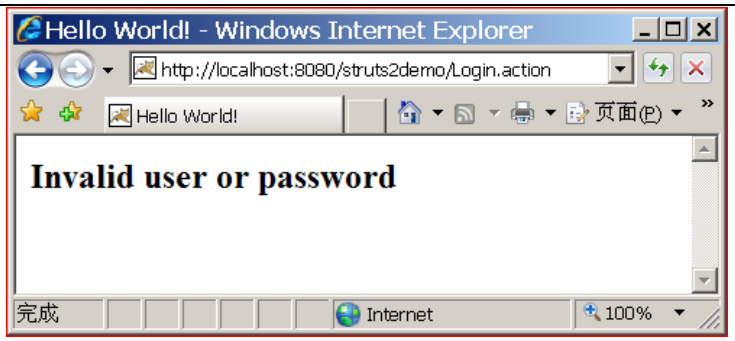


图 5 Login 成功页面

在浏览器地址栏中键入 http://localhost:8080/Struts2demo/Login.jsp，分别在 User name 中输入“123”和“123”，点击“登录”按钮，出现如图所示页面。



Struts 2.0 支持更高级的 POJO 访问，如 user.getPassword()。可以用另一写法实现。首先，将 name 和 password 从 Login 类中分离出来，到新建类 User 中。这样对开发多层系统尤其有用。它可以使系统结构更清晰。

在上例的基础上调整如下：

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
    <title>Login</title>
</head>
<body>
<s:form action="Login" method="POST">
    <s:textfield name="user.name" label="用户名"/>
    <s:password name="user.password" label="密 码"/>
    <s:submit value="登  录"/>
</s:form>
</body>
</html>
```

```
package org.xmh.demo;

public class User {
    private String name;
    private String password;
    ……省略相应的 set/get 方法
}
```

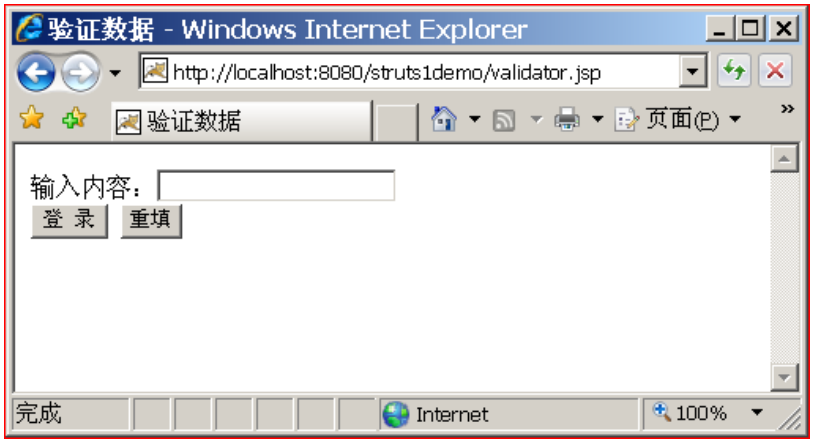
```
package org.xmh.demo;

import com.opensymphony.xwork2.ActionSupport;

public class Login extends ActionSupport {
    private User user;
    private String message;
    public String getMessage() {
        return message;
    }
    @Override
    public String execute() {
        if ("xmh".equals(user.getName()) && "xmh".equals(user.getPassword())) {
            message = "Welcome, " + user.getName();
```

```
        } else {
            message = "Invalid user or password";
        }
        return SUCCESS;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
}
```

4.1 手动完成输入校验

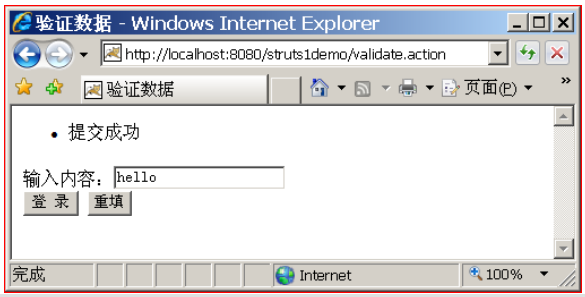
1.重写 validate()方法

也是把校验条件写在 validate()方法里面.这样可以减少 execute()或是自定义的 action 方法的代码.struts2 在调用 execte()方法时,会先执行 validate()方法.如果没通过 validate 则会抛出 fieldError,并返回 name="input"中定义的视图.否则继续执行 execute()方法.




```
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title>验证数据</title>
    </head>
    <body>
        <s:actionerror />
        <s:actionmessage />
        <s:form action="validate.action" theme="simple">
        输入内容：<s:textfield name="msg" />
            <s:fielderror>
                <s:param>msg.hello</s:param>
            </s:fielderror>
            <br />
            <s:submit value="登 录" />
            <s:reset value="重填" />
        </s:form>
```

```
    </body>
</html>
```




```
package org.xmh.demo;

import com.opensymphony.xwork2.ActionSupport;
public class ValidateAction extends ActionSupport {
    private String msg;
    public String execute() {
        System.out.println(SUCCESS);
        return SUCCESS;
    }

    public void validate() {
        if (!msg.equalsIgnoreCase("hello")) {
            System.out.println(INPUT);
            this.addFieldError("msg.hello", "必须输入 hello!");
            this.addActionError("处理动作失败!");
        } else {
            this.addActionMessage("提交成功");
        }
    }

    public String getMsg() {
        return msg;
    }
    public void setMsg(String msg) {
        this.msg = msg;
    }
}
```

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

<!DOCTYPE struts PUBLIC
```

```
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="demo" extends="struts-default">
        <action name="validate" class="org.xmh.demo.ValidateAction">
            <result name="success">/validator.jsp</result>
            <result name="input">/validator.jsp</result>
        </action>
    </package>
</struts>
```

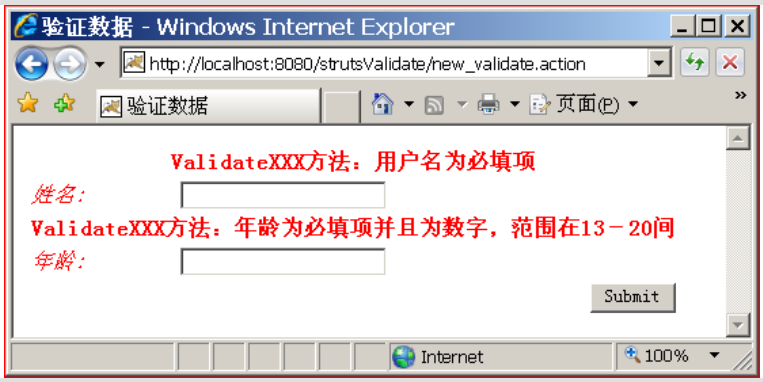
2.重写 validateXXX()方法

如果输入校验只想校验某个处理逻辑，也就是仅仅校验某个处理方法，则重写 validate 方法显然不够，validate()方法无法知道需要校验哪个处理逻辑。实际上，如果重写了 Action 的 validate 方法，则该方法会校验所有的处理逻辑。

为了实现校验指定处理逻辑的功能，struts2 的 Action 提供了一个 validateXXX 方法。XXX 即是 Action 对应的处理逻辑方法。

例：

```
public void validateSaveOrder(){
 System.out.println("\n test--------------------");
 if(orderMain.getOrderTypeId() == 0){
    addFieldError("test","test");
 }
}
```




```
package org.xmh.demo;

import java.util.regex.Pattern;

import com.opensymphony.xwork2.ActionSupport;

public class NewValidateAction extends ActionSupport {
    private String msg; // 必须输入
    private Integer age; // 在 13 和 20 之间

    ……省略相应的 set/get 方法

    public void validateRegist(){
        System.out.println("\\d 匹配数字 "+"8".matches("\\d"));

        if(msg==null||"".equals(msg)){
            addFieldError("msg","ValidateXXX 方法：用户名为必填项");
```

```
        }
        if(age==null||age<13||age>20){
            addFieldError("age","ValidateXXX 方法：年龄为必填项并且为数字，范围在 13－
20 间");
        }
    }

    public String regist(){
        return SUCCESS;
    }
}
```

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

<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
     <package name="demo" extends="struts-default">
        <action name="new_validate" class="org.xmh.demo.NewValidateAction"
 method="regist">
            <result name="input">/validate_form.jsp</result>
            <result name="success">/validate_form.jsp</result>
        </action>
    </package>

</struts>
```

- **基本输入校验**

使用 validate 方法来验证客户端提交的数据，但如果使用 validate 方法就会将验证代码和正常的逻辑代码混在一起，但这样做不利于代码维护，而且也很难将过些代码用于其他程序的验证。在 Struts2 中为我们提供了一个 Validation 框架，这个框架和 Struts1.x 提供的 Validation 框架类似，也是通过 XML 文件进行配置。

4.2 struts2 框架实现数据校验

服务端验证

下面将给出一个例子来演示如何使用 Struts2 的 validation 框架来进行服务端验证。我们可以按着如下四步来编写这个程序：

【第 1 步】建立 Action 类

创建 action 类,重写 execute()方法.

例如: ValidationAction.java

```
import com.opensymphony.xwork2.ActionSupport;
public class ValidationAction extends ActionSupport {
    private String requiredString;

   public String getRequiredString() {
  return requiredString;
```

```
}

public void setRequiredString(String requiredString) {
 this.requiredString = requiredString;
}

@Override
   public String execute() {
       return SUCCESS;
  }
```

下面我们来验证 msg 和 age 属性。

【第 2 步】编写验证规则配置文件

这是一个基于 XML 的配置文件，和 struts1.x 中的 validator 框架的验证规则配置文件类似。但一般放到和要验证的.class 文件在同一目录下，而且配置文件名要使用如下两个规则中的一个来命名：

```
    <ActionClassName>-validation.xml
    <ActionClassName>-<ActionAliasName>-validation.xml
```

其中<ActionAliasName>就是 struts.xml 中<ation>的 name 属性值。在本例中我们使用第一种命名规则，所以文件名是 NewValidateAction-validation.xml。文件的内容如下：

在 action 的包下创建 action-validation.xml 文件.

例:ValidationAction-validation.xml(黑色部分代表相应的 action 类)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC
          "-//OpenSymphony Group//XWork Validator 1.0//EN"
          "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd" >
<validators>
    <field name ="requiredString" >
        <field-validator type ="requiredstring">
            <message> This string is required 必填字段</message>
        </field-validator>
    </field>
</validators>
```

这个文件使用了两个规则：requiredstring（必须输入）和 int（确定整型范围）。关于其他更详细的验证规则，请读者访问 docs/validation.html 来查看。

【第 3 步】配置 Action 类，struts.xml 的代码如下：

在 struts.xml 定义 action 类

```
 <action name ="ValidationAction" class ="com.xishi.action.ValidationAction" >
     <result>Output.jsp</result >
     <result name ="input">Input.jsp</result>
 </action>
```

注:input 参数代表校验失败应返回的页面.

在 Input.jsp 页面增加<s:fielderror/>，这个是必需的,否则不会提示.

注:<s:fielderror/>的用法:

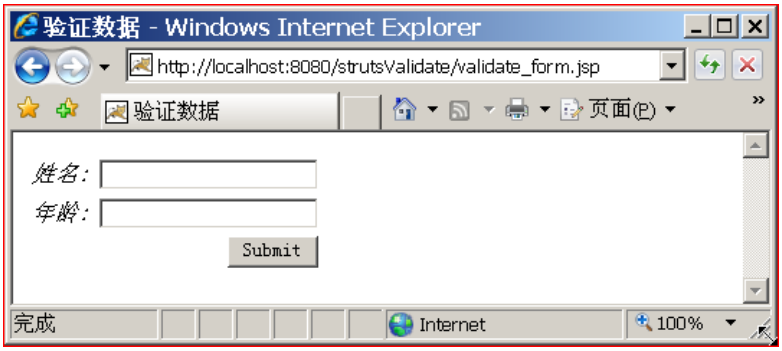
<s:fielderror/>表示显示全部校验信息

<s:fielderror><s:param>字段 1</s:param></s:fielderror>表示只显示字段 1 的校验信息

至于 Input.jsp Output.jsp 就是普通 struts2 页面.

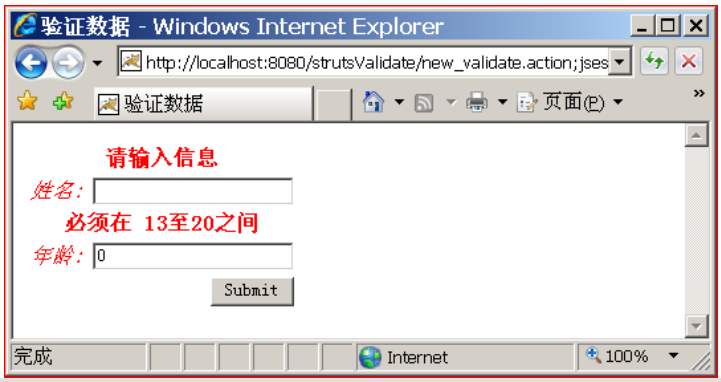
【第 4 步】编写数据录入 JSP 页。

在 Web 根目录中建立一个 validate_form.jsp 文件，代码如下：



在上面的程序中还使用了一个 styles.css 来定制错误信息的风格。具体代码参见 styles.css 文件。使用如下的 URL 来测试这个程序：

http://localhost:8080/strutsValidate/validate_form.jsp




```
package org.xmh.demo;

import com.opensymphony.xwork2.ActionSupport;

public class NewValidateAction extends ActionSupport {
    private String msg; // 必须输入
    private int age; // 在 13 和 20 之间

    ……省略相应的 set/get 方法

}
```

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

<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
     <package name="demo" extends="struts-default">
        <action name="new_validate" class="org.xmh.demo.NewValidateAction">
            <result name="input">/validate_form.jsp</result>
            <result name="success">/validate_form.jsp</result>
        </action>
    </package>

</struts>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    <field name="msg">
        <field-validator type="requiredstring">
            <message>请输入信息</message>
        </field-validator>
    </field>
    <field name="age">
        <field-validator type="int">
            <param name="min">13</param>
            <param name="max">20</param>
            <message>
                必须在 13 至 20 之间
            </message>
        </field-validator>
    </field>
</validators>
```

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<link rel="stylesheet" type="text/css"
    href="<s:url value="/styles.css"/>">
<html>
    <head>
        <title>验证数据</title>
    </head>
    <body>
        <s:form action="new_validate">
            <s:textfield name="msg" label="姓名" />

            <s:textfield name="age" label="年龄" />

            <s:submit />
        </s:form>
    </body>
</html>
```

总结：struts2 数据校验是在 web server 上的实现,因此有可能增加 web server 的负荷。所以纯粹的空值校验，可以用 JavaScript 实现的。当然如果客户端禁用 Javascript 或是不支持 JavaScript，那就是另外一回事了。另外使用 struts 作为数据校验，应尽可能地使用框架进行校验，可以实现 action 与 validation 分离，减少代码量。只有在校验框架满足不了需求时，才要重写 validate()方法.最后的选择是重写 validateXXX()方法。

第 5 章 国际化实现

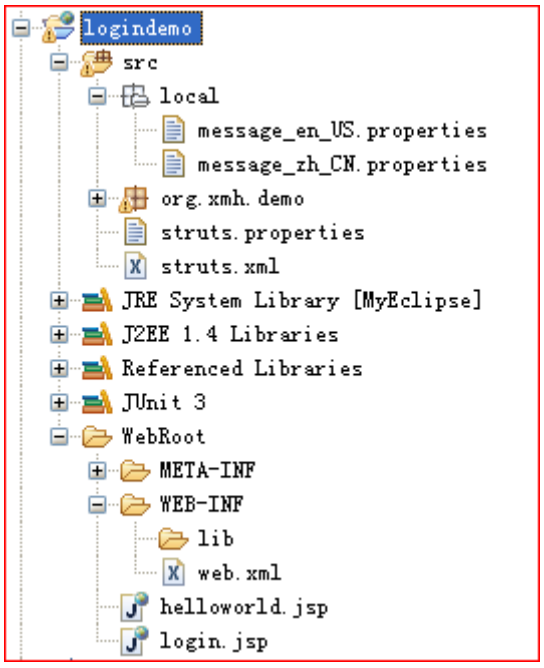
有关国际化问题，在 jsp 和 struts 的学习中都为之做了详细的讲解。此处不再赘述。因为程序国际化是一个企业应用必须实现的功能，所以还是希望读者朋友要重视本章节的学习。Struts 2 提供了很好的程序国际化支持。

程序国际化的设计的主要思想是：程序界面中需要输出国际化信息的地方，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。

5.1 页面的国际化

Struts2 的国际化实现更加方便，下面通过修改 login 示例来说明它的相关用法。

步骤 1、准备资源属性文件。在 src 目录下新建一个 local 包，把所有的资源文件放置其中，这里只创建两种语言的资源文件：英文和中文的资源文件。如下图所示。



message_en_US.properties 的内容如下所示：

```
login.name=name
login.pass=pass
login.submit=submit
login.succmessage=welcome
login.erromessage=Invalid user or password
```

步骤 2：页面上使用国际化功能。struts2 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，全局属性加载资源文件就是通过 struts.properties 文件来完成。

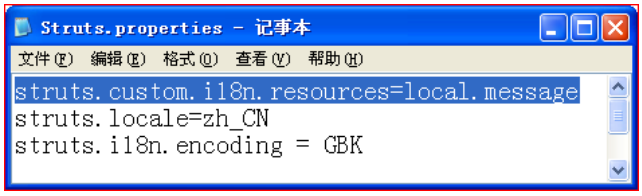
①在页面中临时加载

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title>Login</title>
    </head>
    <body>
        <s:i18n name="local.message">
            <s:form action="Login" method="POST">
                <s:textfield name="user.name" label="%{getText('login.name')}" />
                <s:password name="user.password" label="%{getText('login.pass')}" />
                <s:submit value="%{getText('login.submit')}" />
```

```
            </s:form>
        </s:i18n>
    </body>
</html>
```

②全局加载

在 classes 文件夹下新建 Struts.properties 文件，设置全局属性加载资源的路径，如下图所示。



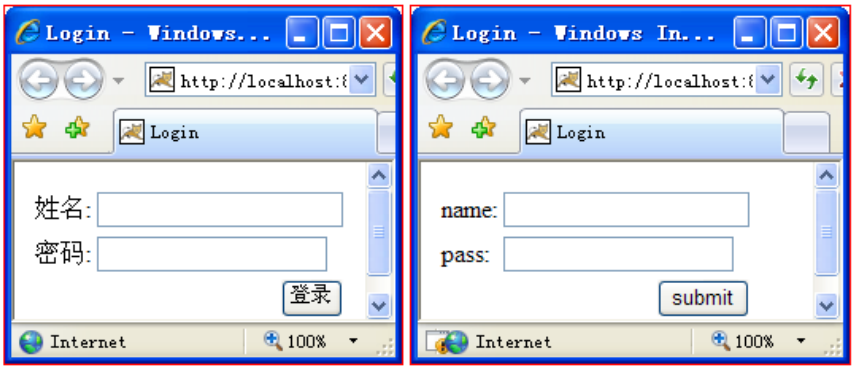
或者打开 struts.xml，在<struts>的下面加如下一行配置：

```
<constant name="struts.custom.i18n.resources" value="message" />
```

页面部分就毋须再使用<s:i18n/>标签，上面的页面部分就修改为：

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title>Login</title>
    </head>
    <body>
            <s:form action="Login" method="POST">
                <s:textfield name="user.name" label="%{getText('login.name')}" />
                <s:password name="user.password" label="%{getText('login.pass')}" />
                <s:submit value="%{getText('login.submit')}" />
            </s:form>
    </body>
</html>
```

发布应用程序，在 IE 中输入访问地址，调整 IE 所用的语言，当首选语言分别设置为“中文”和“英文”时，显示的效果图如下所示。



注意：如果在 struts 的属性文件中设置 struts.locale=zh_CN 或 struts.local=en_US 时，不需要再调整 IE 所用的语言也可以获得如上的效果。

5.2 Action的国际化

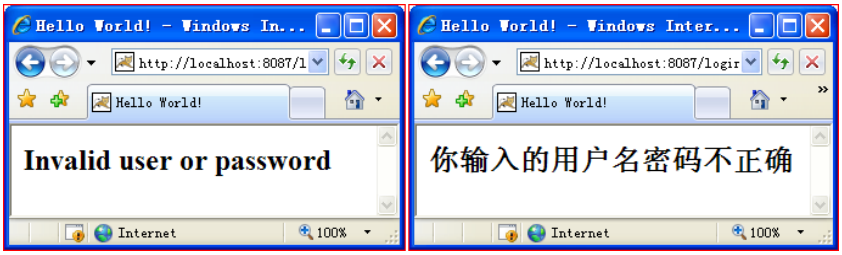
在 struts2 中，当由一个 Action 跳转到一个页面中时，就可以在该页面获取这个 Action 的资源包中信息字符串。比如在登录这个示例中，如果用户登录成功应该返回欢迎的提示信息，否则就提示用户名或密码错误的提示信息，前面是直接将字符串硬编码在程序中，现在要用到国际化，需要将字符串配

置于资源文件中，然后在 Action 中取出所对应的字符串值存储于变量中即可。

```
package org.xmh.demo;
import com.opensymphony.xwork2.ActionSupport;
public class Login extends ActionSupport {
    private User user;
    private String message;
    public String getMessage() {
        return message;
    }

    @Override
    public String execute() {
        if ("xmh".equals(user.getName()) && "xmh".equals(user.getPassword())) {
            message = getText("login.succmessage")+", " + user.getName();
        } else {
            message = getText("login.erromessage");
        }
        return SUCCESS;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
}
```

发布应用程序，在 IE 中输入访问地址，调整 IE 所用的语言，登录成功时中文和英文显示的效果如下图所示。

登录失败时中文和英文显示的效果如下图所示。



5.3 验证信息的国际化

在前一章节学习了验证，怎么实现验证信息的国际化呢？

修改资源文件，内容如下：

```
login.name=name
login.pass=pass
login.submit=submit
login.succmessage=welcome
login.erromessage=Invalid user or password
error.username=user's name should not be null
error.userpass=password should not be null
```

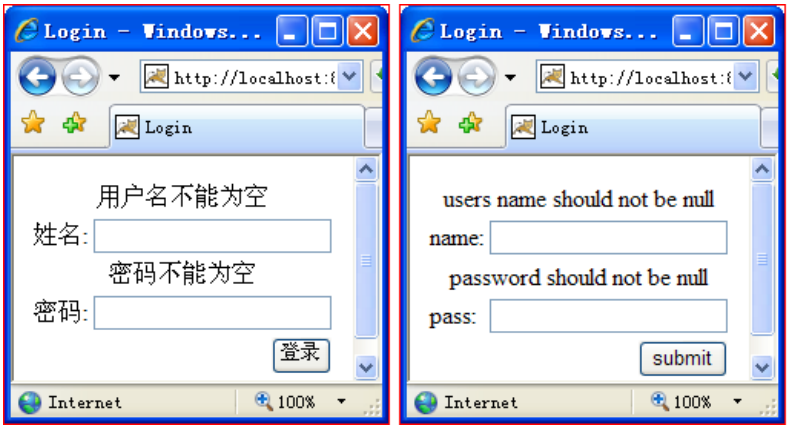
修改配置文件：

```
<struts>
    <package name="tutorial" extends="struts-default">
        <action name="Login" class="org.xmh.demo.Login">
            <result name="success">/helloworld.jsp</result>
              <result name ="input">login.jsp</result>
        </action>
    </package>
</struts>
```

在 Action 的同级目录添加 Login-validation.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC
          "-//OpenSymphony Group//XWork Validator 1.0//EN"
          "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd" >
<validators>
    <field name ="user.name" >
        <field-validator type ="requiredstring">
            <message key="error.username">not null</message>
        </field-validator>
    </field>
        <field name ="user.password" >
        <field-validator type ="requiredstring">
            <message key="error.userpass">not null</message>
        </field-validator>
    </field>
</validators>
```

发布项目，运行 tomcat，输入访问 URL，然后直接用鼠标单击登录按钮，然后再调整 IE 首选语言，效果如下图所示：



注意：国际化文件的查找顺序

假设我们在某个 ChildAction 中调用了 getText("user.title")，Struts 2.0 的将会执行以下的操作：

1、查找 ChildAction_xx_XX.properties 文件或 ChildAction.properties；

2、查找 ChildAction 实现的接口，查找与接口同名的资源文件 MyInterface.properties；

3、查找 ChildAction 的父类 ParentAction 的 properties 文件，文件名为 ParentAction.properties；

4、判断当前 ChildAction 是否实现接口 ModelDriven。如果是，调用 getModel()获得对象，查找与其同名的资源文件；

5、查找当前包下的 package.properties 文件；

6、查找当前包的父包，直到最顶层包；

7、在值栈（Value Stack）中，查找名为 user 的属性，转到 user 类型同名的资源文件，查找键为 title 的资源；

8、查找在 struts.properties 配置的默认的资源文件，参考例 1；

9、输出 user.title。

第 6 章 拦截器浅析

6.3 拦截器基础

Struts2 的拦截器和 Servlet 过滤器类似。在执行 Action 的 execute 方法之前，Struts2 会首先执行在 struts.xml 中引用的拦截器，在执行完所有引用的拦截器的 intercept 方法后，会执行 Action 的 execute 方法。

Struts2 拦截器类必须从 com.opensymphony.xwork2.interceptor.Interceptor 接口继承，在 Intercepter 接口中有如下三个方法需要实现：

```
void destroy();
void init();
String intercept(ActionInvocation invocation) throws Exception;
```

其中 intercept 方法是拦截器的核心方法，所有安装的拦截器都会调用之个方法。在 Struts2 中已经在 struts-default.xml 中预定义了一些自带的拦截器，如 timer、params 等。如果在<package>标签中继承 struts-default，则当前 package 就会自动拥有 struts-default.xml 中的所有配置。代码如下：

```
    <package name="demo" extends="struts-default" > ... </package>
```

在 struts-default.xml 中有一个默认的引用，在默认情况下（也就是<action>中未引用拦截器时）会自动引用一些拦截器。这个默认的拦截器引用如下：

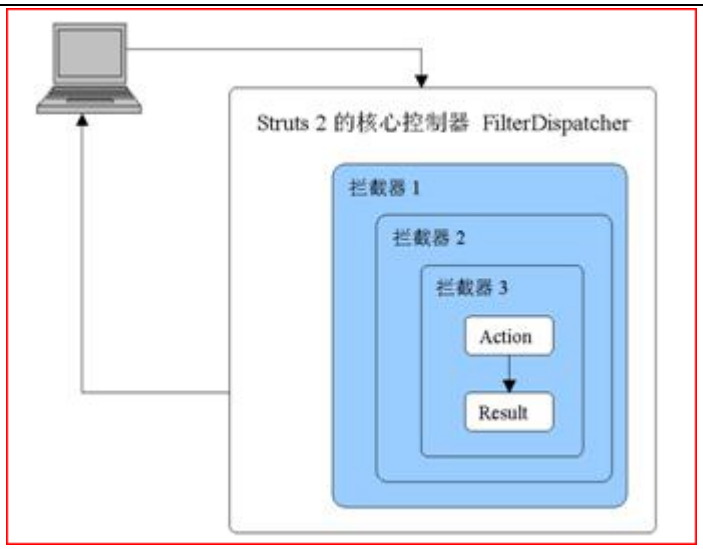
```
    <default-interceptor-ref name="defaultStack"/>
    <interceptor-stack name="defaultStack">
        <interceptor-ref name="exception"/>
        <interceptor-ref name="alias"/>
        <interceptor-ref name="servletConfig"/>
        <interceptor-ref name="prepare"/>
        <interceptor-ref name="i18n"/>
        <interceptor-ref name="chain"/>
        <interceptor-ref name="debugging"/>
        <interceptor-ref name="profiling"/>
        <interceptor-ref name="scopedModelDriven"/>
        <interceptor-ref name="modelDriven"/>
        <interceptor-ref name="fileUpload"/>
        <interceptor-ref name="checkbox"/>
        <interceptor-ref name="staticParams"/>
        <interceptor-ref name="params">
            <param name="excludeParams">dojo\..*</param>
        </interceptor-ref>
        <interceptor-ref name="conversionError"/>
        <interceptor-ref name="validation">
              <param name="excludeMethods">input,back,cancel,browse</param>
         </interceptor-ref>
         <interceptor-ref name="workflow">
               <param name="excludeMethods">input,back,cancel,browse</param>
         </interceptor-ref>
    </interceptor-stack>
```

上面在 defaultStack 中引用的拦截器都可以在<action>中不经过引用就可以使用(如果在<action>中引用了任何拦截器后，要使用在 defaultStack 中定义的拦截器，也需要在<action>中重新引用，在后面将详细讲解)。

6.2 使用拦截器

Struts2 中自带的默认拦截器栈已经能够满足大多数应用的需要了，大多数的应用都不必再增加自己的拦截器或者改变已有的拦截器栈。在 Struts2 中，许多 action 通常都有一些共同的需要关心的问题，比如有一些 action 它们都需要对页面上的输入进行验证，有一些 action 需要对上传的文件进行一下预处理，还有一些 action 可能需要防止重复提交(double submit)，还有很多 action 需要在页面显示之前将下拉列表和者其它的一些控件事先装好值。通过使用“拦截器”策略，Struts2 框架使得共享这些问题的解决方案变得十分容易。当请求一个 action 的时候，框架会调用这个 action 对象，但是，在 action 执行之前，这个调用可能会被另外一个对象所拦截，。在 action 执行之后，这个调用可能被一个对象再次拦截，如果读者朋友对 AOP 有所了解，则一切都会变得简单明了。

拦截器能在 action 被调用之前和被调用之后执行一些“代码”。Struts2 框架的大部分核心功能都是通过拦截器来实现的，如防止重复提交、类型转换、对象封装、校验、文件上传、页面预装载等等，都是在拦截器的帮助下实现的。每一个拦截器都是独立装载的(pluggable)，我们可以根据实际的需要为每一个 action 配置它所需要的拦截器，例如，一个 action 需要用来类型装换、文件上传，那么我们可以给它设置相应的两个拦截器。



在有些情况下，拦截器可能阻止一个 action 执行，例如重复提交的情况下或者校验没有通过的情况下，除此之外，拦截器还能在一个 action 执行之前改变它的状态。拦截器被定义在栈中，这个栈指定了拦截器的执行顺序，在有些情况下，拦截器在栈中的顺序是非常重要的。下面是一个简单的拦截器配置的例子：

要实现一个日志的功能，在某个 Action 执行的前后分别打印出相关的日志信息。在 struts2 中已经提供了一个日志的拦截器，如下图所示。

struts-default.xml

```
<interceptor name="logger" class="com.opensymphony.xwork2.interceptor.LoggingIntercep
<interceptor name="modelDriven" class="com.opensymphony.xwork2.interceptor.ModelDrive
<interceptor name="scopedModelDriven" class="com.opensymphony.xwork2.interceptor.Scop
<interceptor name="params" class="com.opensymphony.xwork2.interceptor.ParametersInter
<interceptor name="actionMappingParams" class="org.apache.struts2.interceptor.ActionM
<interceptor name="prepare" class="com.opensymphony.xwork2.interceptor.PrepareInterce
<interceptor name="staticParams" class="com.opensymphony.xwork2.interceptor.StaticPar
<interceptor name="scope" class="org.apache.struts2.interceptor.ScopeInterceptor"/>
<interceptor name="servletConfig" class="org.apache.struts2.interceptor.ServletConfig
<interceptor name="sessionAutowiring" class="org.apache.struts2.spring.interceptor.Se
```

这里仍然以登录的程序 login 为例，来实同此功能。

修改 Action 文件：

```
public String execute() {
        try {
            System.out.println("=======这是正常输出内容=======");
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } // 延迟 1 秒
        return SUCCESS;

    }
```

修改配置文件：

```
<struts>
    <package name="tutorial" extends="struts-default">
        <action name="Login" class="org.xmh.demo.Login">
            <interceptor-ref name="logger" />
            <result>/helloworld.jsp</result>
        </action>
    </package>
</struts>
```

运行程序，在 IE 中输入“http://localhost:8087/logindemo/Login.action” ，控制台显示的效果如下图所示：

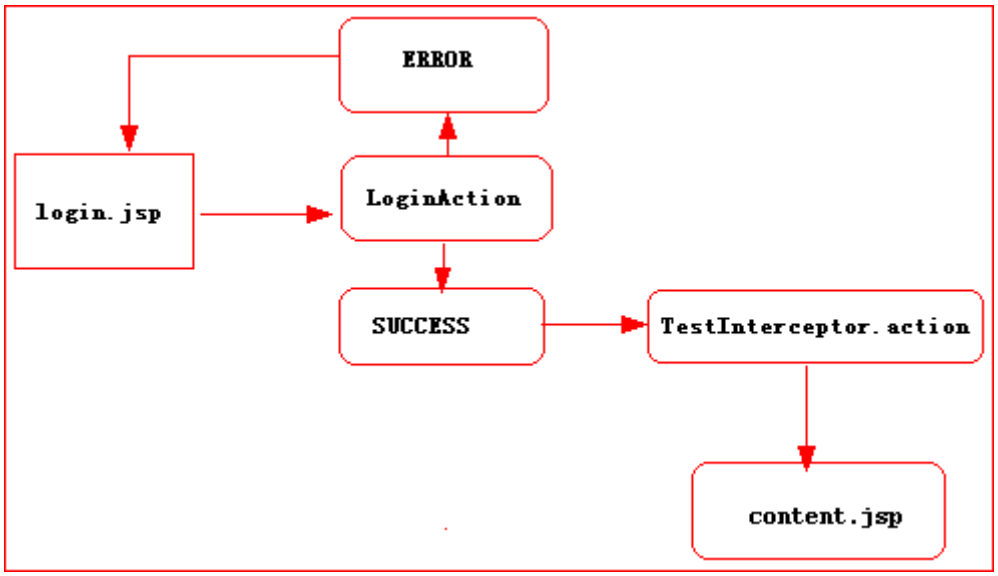
```
信息: Starting execution stack for action //Login
=======这是正常输出内容=======
2008-12-23 10:06:29 com.opensymphony.xwork2.util.logging.commons
警告: Could not find property [org.apache.catalina.jsp_file]
2008-12-23 10:06:31 com.opensymphony.xwork2.util.logging.commons
信息: Finishing execution stack for action //Login
```

图中用红线标注的地方就是日志信息，下面通过源代码来分析一下相应的功能。。

```
public class LoggingInterceptor extends AbstractInterceptor {
    private static final Log log = LogFactory.getLog(LoggingInterceptor.class);
    private static final String FINISH_MESSAGE = "Finishing execution stack for action
";
    private static final String START_MESSAGE = "Starting execution stack for action ";

    public String intercept(ActionInvocation invocation) throws Exception {
        logMessage(invocation, FINISH_MESSAGE);
        String result = invocation.invoke();
        logMessage(invocation, START_MESSAGE);
        return result;
    }
    private void logMessage(ActionInvocation invocation, String baseMessage) {
        if (log.isInfoEnabled()) {
            StringBuffer message = new StringBuffer(baseMessage);
            String namespace = invocation.getProxy().getNamespace();

            if ((namespace != null) && (namespace.trim().length() > 0)) {
                message.append(namespace).append("/");
            }

            message.append(invocation.getProxy().getActionName());
            log.info(message.toString());
        }
    }
}
```

6.3 自定义拦截器

上例学习了 struts 提供的拦截器，本小节将学习如何自定义一个拦截器。在 JSP 的学习过程中，曾处理过如何防止非法用户登录的示例。下面将使用拦截器来改写这个示例。当用户访问 login.jsp 页面，并输入正确的用户名和密码时，可以到注册成功页，并且可以访问 content.jsp 页面内容。如果用户未登录直接访问 loginAction 或 TestInterceptor.action 则返回到 login.jsp 页面。



自定义拦截器的步骤如下：

步骤 1、编写拦截器代码

```
package org.xmh.demo;
import java.util.Map;
import org.apache.struts2.interceptor.SessionAware;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction extends ActionSupport implements SessionAware {
    private static final long serialVersionUID = 1L;
    private UserBean userBean;
    private Map<String, String> session;
    public UserBean getUserBean() {
        return userBean;
    }
    public void setUserBean(UserBean userBean) {
        this.userBean = userBean;
    }

    @Override
    public String execute() {
        if (userBean == null) {
         //这里的 LOGIN 对应 Action.java 中的
         //public static final String LOGIN = "login";
            return LOGIN;
        }
        String name = userBean.getName();
        String pwd = userBean.getPassword();
        if (name.equals("xmh") && pwd.equals("xmh")) {
            session.put("LOGIN", name);
            return SUCCESS;
        } else {
            return ERROR;
        }
    }
    public void setSession(Map session) {
        this.session = session;
```

```
    }
}
```

UserBean 的代码比较简单，清单如下所示：

```
package org.xmh.demo;

public class UserBean {
   private String name;
   private String password;
   public String getName() {
       return name;
   }
   public void setName(String name) {
       this.name = name;
   }
   public String getPassword() {
       return password;
   }
   public void setPassword(String password) {
       this.password = password;
   }
}
```

LoginAction 的代码清单如下所示：

```
package org.xmh.demo;
import java.util.Map;
import org.apache.struts2.interceptor.SessionAware;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction extends ActionSupport implements SessionAware {
   private static final long serialVersionUID = 1L;
   private UserBean userBean;
   private Map<String, String> session;
   public UserBean getUserBean() {
       return userBean;
   }

   public void setUserBean(UserBean userBean) {
       this.userBean = userBean;
   }

   @Override
   public String execute() {
       if (userBean == null) {
           return LOGIN;
       }

       String name = userBean.getName();
       String pwd = userBean.getPassword();

       if (name.equals("xmh") && pwd.equals("xmh")) {

           session.put("LOGIN", name);
```

```
                return SUCCESS;
            } else {
                return ERROR;
            }
        }
        public void setSession(Map session) {
            this.session = session;
        }
    }
```

TestInterceptorAction 的代码清单如下所示：

```
    public class TestInterceptorAction extends ActionSupport {

        private static final long serialVersionUID = 1L;

        @Override
        public String execute() {
            return SUCCESS;
        }

    }
```

步骤 2、配置拦截器：

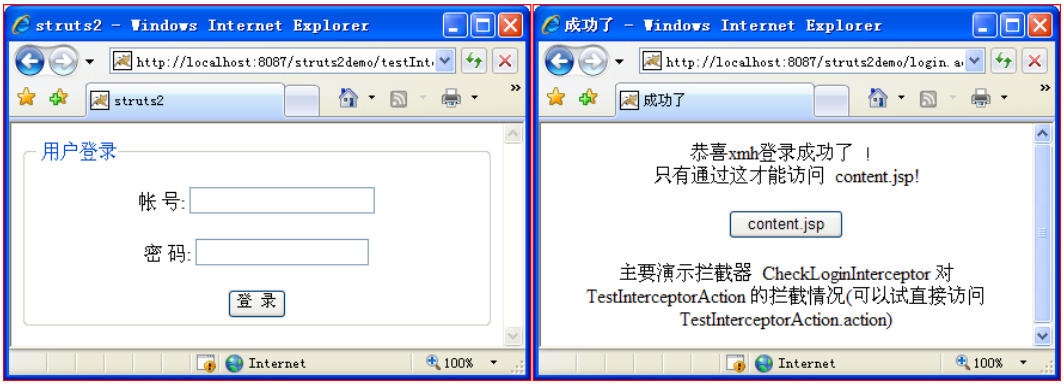
```
    <!DOCTYPE struts PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
        "http://struts.apache.org/dtds/struts-2.0.dtd">
    <struts>
        <!-- 加载默认的 struts2 配置文件 -->
        <include file="struts-default.xml" />
        <!-- 继承默认的 struts2 配置文件 -->
        <package name="default" extends="struts-default">
            <!-- 定义一个名为 checkLogin 的拦截器 -->
            <interceptors>
                <!-- 定义权限检查拦截器 -->
                <interceptor name="checkLogin"
                    class="org.xmh.demo.CheckLoginInterceptor" />
                <!-- 定义一个包含权限检查的拦截器栈 -->
                <interceptor-stack name="myDefaultStack">
                    <!-- 定义拦截器栈包含 checkLogin 拦截器 -->
                    <interceptor-ref name="checkLogin"></interceptor-ref>
                    <interceptor-ref name="defaultStack"></interceptor-ref>
                </interceptor-stack>
            </interceptors>
            <!-- 设置全局 全局默认的拦截器栈-->
            <default-interceptor-ref
name="myDefaultStack"></default-interceptor-ref>

            <!-- 定义全局 Result -->
            <global-results>
                <!-- 当返回 login 视图名时，转入/login.jsp 页面 -->
                <result name="login">/login.jsp</result>
            </global-results>
```

```
            <!--
                action 标签里 name 属性代表我们要处理的.action 的前面部分
                action 标签里 class 属性代表我们需要哪个类来处理
                result 标签的 name 属性代表 action 类的执行方法的返回值,
                action 类的默认执行方法是 public String execute()
            -->
            <action name="login"
                class="org.xmh.demo.LoginAction">
                <result name="success">success.jsp</result>
                <result name="error">error.jsp</result>
                <result name="login">login.jsp</result>
                <!--拦截器一般配置在 result 元素之后  -->
            </action>

            <action name="testInterceptor"
                class="org.xmh.demo.TestInterceptorAction">
                <result name="success">content.jsp</result>
                <result name="login">login.jsp</result>
            </action>
        </package>
    </struts>
```

XML 中的<default-interceptor-ref name="defaultStack"/>就是对 interceptor 或者是 interceptor 栈的引用。从 tag 的命名 default-interceptor-ref 可以知道这个 interceptor(interceptor 栈)是默认的设置,也就是所有的 action 在没有定义自己特有的 interceptor 引用的时候，都会使用这个默认的 interceptor 引用；其它页面的代码请参看清代码部分。

步骤 3、运行示例，查看效果

在 IE 中输入访问地址，输入正确的帐号和密码，则可以获得如下图所示的效果。



6.4 综合示例

第 7 章 探讨 Ioc 模式

很多时候我的同事会问我：“如果我要取得 Servlet API 中的一些对象，如 request、response 或 session 等，应该怎么做？这里的 execute 不像 Struts 1.x 的那样在参数中引入。”开发 Web 应用程序

当然免不了跟这些对象打交道。在 Strutx 2.0 你可以有两种方式获得这些对象：非 IoC（控制反转 Inversion of Control）方式和 IoC 方式。

非 IoC 方式

要获得上述对象，关键 Struts 2.0 中 com.opensymphony.xwork2.ActionContext 类。我们可以通过它的静态方法 getContext() 获取当前 Action 的上下文对象。另外，org.apache.struts2.ServletActionContext 作为辅助类（Helper Class），可以帮助您快捷地获得这几个对象。

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

如果你只是想访问 session 的属性（Attribute），你也可以通过 ActionContext.getContext().getSession()获取或添加 session 范围（Scoped）的对象。

IoC 方式

要使用 IoC 方式，我们首先要告诉 IoC 容器（Container）想取得某个对象的意愿，通过实现相应的接口做到这点。

```
package org.xmh.demo;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.struts2.ServletActionContext;

import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class NonIoCServlet extends ActionSupport {
    private String message;

    public String getMessage() {
        return message;
    }

    @Override
    public String execute() {
        ActionContext.getContext().getSession().put("msg", "Hello World from
Session!");

        HttpServletRequest request = ServletActionContext.getRequest();
        HttpServletResponse response = ServletActionContext.getResponse();
        HttpSession session = request.getSession();

        StringBuffer sb =new StringBuffer("Message from request: ");
        sb.append(request.getParameter("msg"));
        sb.append("<br>Response Buffer Size: ");
        sb.append(response.getBufferSize());
        sb.append("<br>Session ID: ");
        sb.append(session.getId());

        message = sb.toString();
```

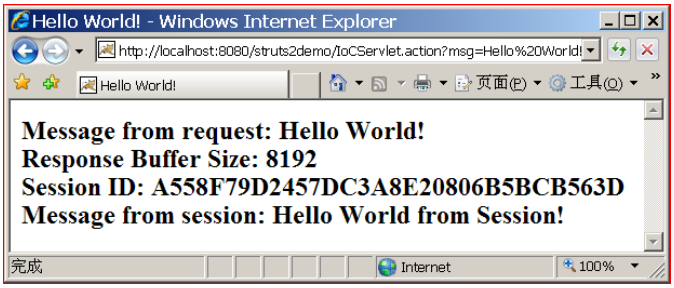
```
        return SUCCESS;
    }
}
```

```
package org.xmh.demo;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.struts2.interceptor.ServletRequestAware;
import org.apache.struts2.interceptor.ServletResponseAware;
import org.apache.struts2.interceptor.SessionAware;

import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class IoCServlet extends ActionSupport implements SessionAware,
ServletRequestAware, ServletResponseAware {
    private String message;
    private Map att;
    private HttpServletRequest request;
    private HttpServletResponse response;

    public String getMessage() {
        return message;
    }

    public void setSession(Map att) {
        this.att = att;
    }

    public void setServletRequest(HttpServletRequest request) {
        this.request = request;
    }

    public void setServletResponse(HttpServletResponse response) {
        this.response = response;
    }

    @Override
    public String execute() {
        att.put("msg", "Hello World from Session!");
        HttpSession session = request.getSession();

        StringBuffer sb =new StringBuffer("Message from request: ");
        sb.append(request.getParameter("msg"));
        sb.append("<br>Response Buffer Size: ");
        sb.append(response.getBufferSize());
```

```
        sb.append("<br>Session ID: ");
        sb.append(session.getId());

        message = sb.toString();
        return SUCCESS;
    }
}
```

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
    <title>Hello World!</title>
</head>
<body>
    <h2>
        <s:property value="message" escape="false"/>
        <br>Message from session: <s:property value="#session.msg"/>
    </h2>
</body>
</html>
```

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="xmh" extends="struts-default">
        <action name="NonIoCServlet"
            class="org.xmh.demo.NonIoCServlet">
            <result>/Servlet.jsp</result>
        </action>
        <action name="IoCServlet" class="org.xmh.demo.IoCServlet">
            <result>/Servlet.jsp</result>
        </action>
    </package>
</struts>
```



在 struts1.x Action 类的 execute 方法中，有四个参数，其中两个就是 response 和 request。而在 Struts2 中，并没有任何参数，因此，就不能简单地从 execute 方法获得 HttpServletResponse 或 HttpServletRequest 对象了。但在 Struts2 Action 类中仍然有很多方法可以获得这些对象。下面就列出四种获得这些对象的方法。

【方法 1】使用 Struts2 Aware 拦截器

这种方法需要 Action 类实现相应的拦截器接口。如我们要获得 HttpServletResponse 对象，需要实现 org.apache.struts2.interceptor.ServletResponseAware 接口，代码如下：

```
package action;

import com.opensymphony.xwork2.ActionSupport;
import javax.servlet.http.*;
import org.apache.struts2.interceptor.*;

public class MyAction extends ActionSupport implements ServletResponseAware
{
    private javax.servlet.http.HttpServletResponse response;
    // 获得 HttpServletResponse 对象
    public void setServletResponse(HttpServletResponse response)
    {
        this.response = response;
    }
    public String execute() throws Exception
    {
        response.getWriter().write("实现 ServletResponseAware 接口");
    }
}
```

在上面的代码中，MyAction 实现了一个 ServletResponseAware 接口，并且实现了 setServletResponse 方法。如果一个动作类实现了 ServletResponseAware 接口，Struts2 在调用 execute 方法之前，就会先调用 setServletResponse 方法，并将 response 参数传入这个方法。如果想获得 HttpServletRequest、HttpSession 和 Cookie 等对象，动作类可以分别实现 ServletRequestAware、SessionAware 和 CookiesAware 等接口。这些接口都在 org.apache.struts2.interceptor 包中。

如果要获得请求参数，动作类可以实现 org.apache.struts2.interceptor. ParameterAware 接口，但如果只想判断某个参数是否存在，也可以实现 com.opensymphony.xwork2.interceptor. ParameterNameAware 接口。这个接口有一个 acceptableParameterName 方法，当 Struts2 获得一个请求参数时，就会调用一次。读者可以在这个方法中将所有的请求参数记录下来，以便以后使用。这个方法的定义如下：

```
boolean acceptableParameterName(String parameterName);
```

【方法 2】使用 RequestAware 拦截器

这种方法和第 1 种方法类似。动作类需要实现一个 org.apache.struts2.interceptor.RequestAware 接口。所不同的是 RequestAware 将获得一个 com.opensymphony.xwork2.util.OgnlValueStack 对象，这个对象可以获得 response、request 及其他的一些信息。代码如下所示：

```
package action;

import java.util.Map;
import org.apache.struts2.*;
import com.opensymphony.xwork2.ActionSupport;
import javax.servlet.http.*;
import com.opensymphony.xwork2.util.*;
import org.apache.struts2.interceptor.*;

public class FirstAction extends ActionSupport implements RequestAware
{
    private Map request;
private HttpServletResponse response;
```

```
    public void setRequest(Map request)
    {
        this.request = request;
    }
    public String execute() throws Exception
    {
        java.util.Set<String> keys = request.keySet();
        // 枚举所有的 key 值。实际上只有一个 key：struts.valueStack
        for(String key: keys)
            System.out.println(key);
        // 获得 OgnlValueStack 对象
        OgnlValueStack stack = (OgnlValueStack)request.get("struts.valueStack");
        // 获得 HttpServletResponse 对象
        response =
(HttpServletResponse)stack.getContext().get(StrutsStatics.HTTP_RESPONSE);
        response.getWriter().write("实现 RequestAware 接口");
    }
}
```

我们也可以使用 StrutsStatics.HTTP_REQUEST、StrutsStatics.PAGE_CONTEXT 来获得 HttpServletRequest 和 PageContext 对象。这种方法有些麻烦，一般很少用，读者可以作为一个参考。

【方法 3】使用 ActionContext 类

这种方法比较简单，我们可以通过 org.apache.struts2.ActionContext 类的 get 方法获得相应的对象。代码如下：

```
    HttpServletResponse response(HttpServletResponse) =
ActionContext.getContext().get(org.apache.struts2.StrutsStatics.HTTP_RESPONSE);

HttpServletRequest request(HttpServletRequest) =
ActionContext.getContext().get(org.apache.struts2.StrutsStatics.HTTP_REQUEST);
```

【方法 4】使用 ServletActionContext 类

Struts2 为我们提供了一种最简单的方法获得 HttpServletResponse 及其他对象。这就是 org.apache.struts2.ServletActionContext 类。我们可以直接使用 ServletActionContext 类的 getRequest、getResponse 方法来获得 HttpServletRequest、HttpServletResponse 对象。代码如下：

```
    HttpServletResponse response = ServletActionContext.getResponse()
    response.getWriter().write("hello world");
```

从这四种方法来看，最后一种是最简单的，读者可以根据自己的需要和要求来选择使用哪一种方法来获得这些对象。

第 8 章 Struts2 标签

为了使从一个页面中链接一个动态数据变得简单，Struts2 框架提供了一系列的标签。

Struts2 标签的一种用法是创建链接到其他 Web 资源，特别是针对那些在本地应用中的资源。

1. 普通链接

Web 程序中最普通的应用是链接到其他页面，下面看 Welcome.jsp。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
```

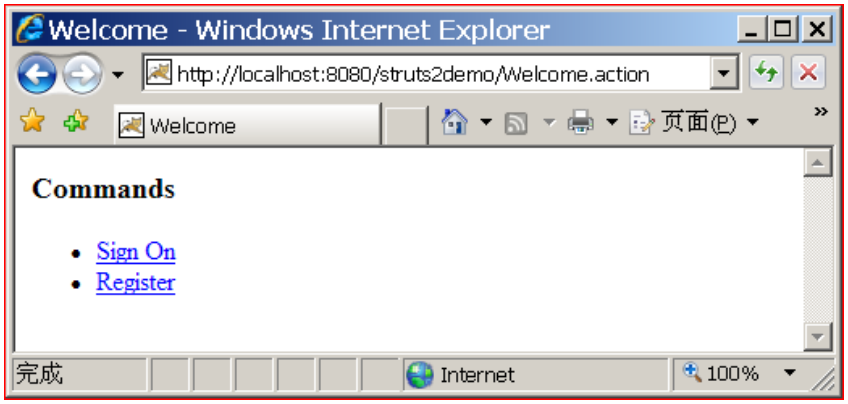
```
    <title>Welcome</title>
    <link href="<s:url value="/css/tutorial.css"/>" rel="stylesheet"
          type="text/css"/>
</head>

<body>
<h3>Commands</h3>
<ul>
    <li><a href="<s:url action="Login_input"/>">Sign On</a></li>
    <li><a href="<s:url action="Register"/>">Register</a></li>
</ul>

</body>
</html>
```

注册 action，在 struts.xml 中注册一个 action 来显示 welcome.jsp。

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

<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="TestEMO" extends="struts-default">
        <action name="Welcome">
            <result>/Welcome.jsp</result>
        </action>
    </package>
</struts>
```

在地址栏中敲入 http://localhost:8080/struts2demo/Welcome.action（struts2demo 是 project 名），会导向到 Welcome.jsp。

效果页是：



因为此页面其它的连接并无配置相关的 action，所以单击其它连接，将出现错误页面导航的提示。此页面解析后的代码如下图所示：

```
<html>
<head>
    <title>Welcome</title>
    <link href="/struts2demo/css/tutorial.css" rel="stylesheet"
        type="text/css"/>
</head>

<body>
<h3>Commands</h3>
<ul>
    <li><a href="/struts2demo/Login_input.action">Sign On</a></li>
    <li><a href="/struts2demo/Register.action">Register</a></li>
</ul>

</body>
</html>
```

页面代码分析如下：

1. <%@ taglib prefix="s" uri="/struts-tags" %>

此句表示导入 struts 标签，并以 s 为前缀。即以 s 为前缀的标签均来自 struts 标签库。

2. <link href="<s:url value="/css/tutorial.css"/>" rel="stylesheet" type="text/css"/>

此句表示利用 url 标签导入一个路径，链接到一个文件，注意此路径为项目下的绝对路径。

3. <a href="<s:url action="Login_input"/>">Sign On

此句表示利用 url 标签链接到一个 action。

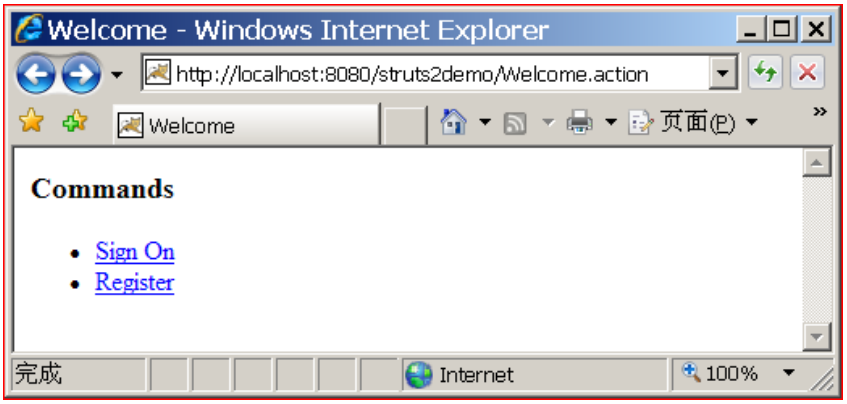
2. 使用通配符

对于上面的 action 注册，我们也可以用下面的语句代替。

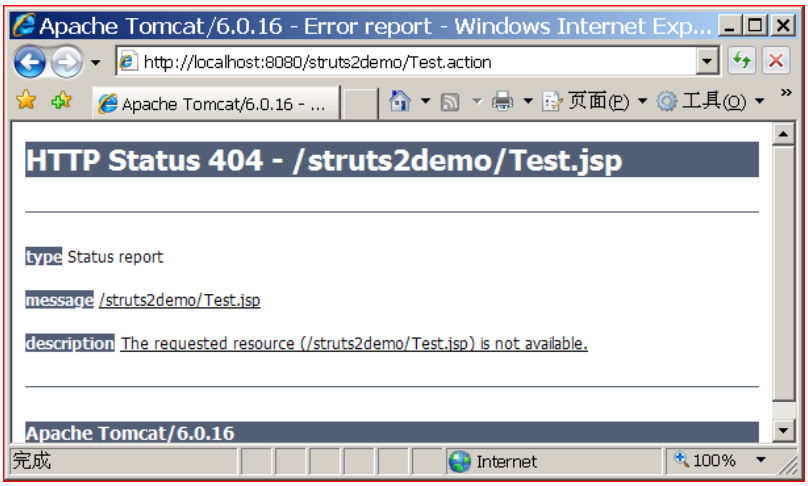
```
<action name="*">
        <result>/example/{1}.jsp</result>
</action>
```

此句的意思是，如果在没有找到匹配的 action 名称的情况下，默认调用 action 名称.jsp。第一句中星号指任意，而第二句中{1}指代第一句中星号指代的内容。

如果在地址栏中敲入 http://localhost:8080/struts2demo/Welcome.action，则系统查找 struts.xml，如果找不到 name 为 Welcome 的 action，则调用 name 为星号的这个 action，根据此 action，将输出/ Welcome.jsp。



如果在地址栏中敲入 http://localhost:8080/struts2demo/Test.action，则系统查找 struts.xml，如果找不到 name 为 Test 的 action，则调用 name 为星号的这个 action，根据此 action，将输出/ Test.jsp，因为没有这个页面，所以系统将会报错找不相应的页面。到 Login_input.jsp 和 Register.jsp。因为这两个 action 还没有注册，也没有相应的 jsp 文件。

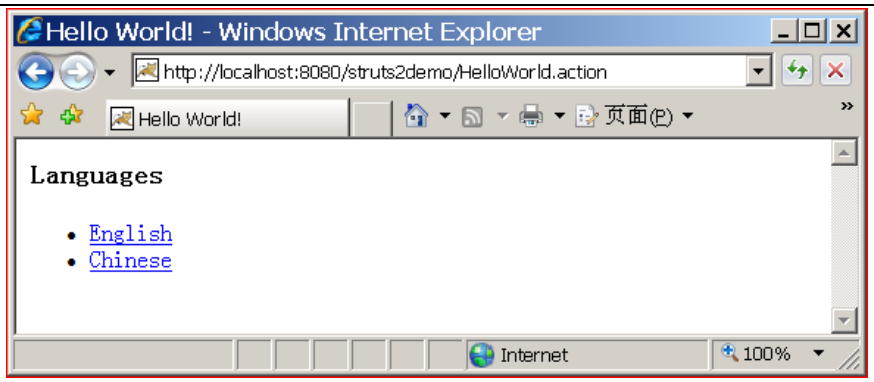


3．带参数的链接

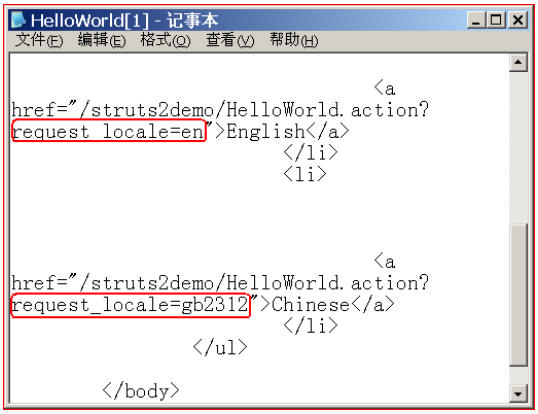
超链接后面带有参数大家不会陌生，诸如 http://www.apache.com/?language=ch。这个链接后面带有一个 language 参数，其值为 ch。你可以通过 request.getParameter(“language”)找到参数值。下面演示在 struts2 中如何设置带参数的链接。看 HelloWorld.jsp。

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

<html>
    <head>
        <title>Hello World!</title>
    </head>
    <body>
        <h2>
            <s:property value="message" />
        </h2>
        <h3>
            Languages
        </h3>
        <ul>
            <li>
                <s:url id="url" action="HelloWorld">
                    <s:param name="request_locale">en</s:param>
                </s:url>
                <s:a href="%{url}">English</s:a>
            </li>
            <li>
                <s:url id="url" action="HelloWorld">
                    <s:param name="request_locale">gb2312</s:param>
                </s:url>
                <s:a href="%{url}">Chinese</s:a>
            </li>
        </ul>

    </body>
</html>
```

运行效果如下图所示：



查看源代码如下所示：



说明

<s:url id="url" action="HelloWorld"><s:param name="request_locale">en</s:param></s:url>

此段表示设置一个 url 标签指向名为 HelloWorld 的 action，此标签带一个 id 取名为 url，后面会用到。带一个参数 request_locale，其值为 en。

<s:a href="%{url}">English</s:a>

此句用到了 struts2 的超链接标签，连接的地址即为 1 中 url，点击 English，发出的信息为：http://localhost:8080/ struts2demo /HelloWorld.action?request_locale=en

注册 action 到 struts.xml

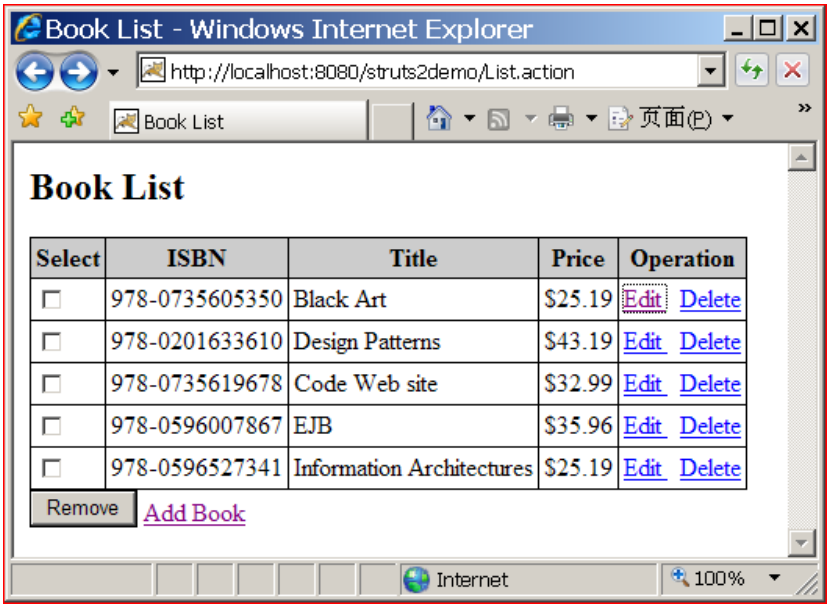
```
<package name="example"  extends="struts-default">
        <action name="HelloWorld" >
                <result>/ HelloWorld.jsp</result>
        </action>
```

在 Struts 2 中实现 CRUD

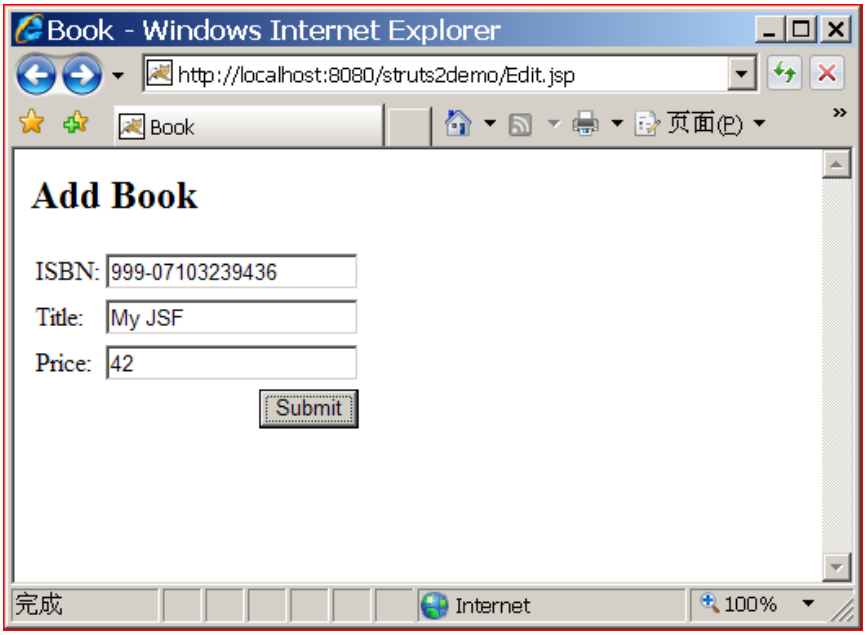
CRUD 是 Create（创建）、Read（读取）、Update（更新）和 Delete（删除）的缩写，它是普通应用程序的缩影。如果您掌握了某框架的 CRUD 编写，那么意味可以使用该框架创建普通应用程序了，所以大家使用新框架开发 OLTP（Online Transaction Processing）应用程序时，首先会研究一下如何编写 CRUD。这类似于大家在学习新编程语言时喜欢编写“Hello World”。

本文旨在讲述 Struts 2 上的 CRUD 开发，所以为了例子的简单易懂，我不会花时间在数据库的操作上。取而代之的是一个模拟数据库的哈希表（Hash Map）。

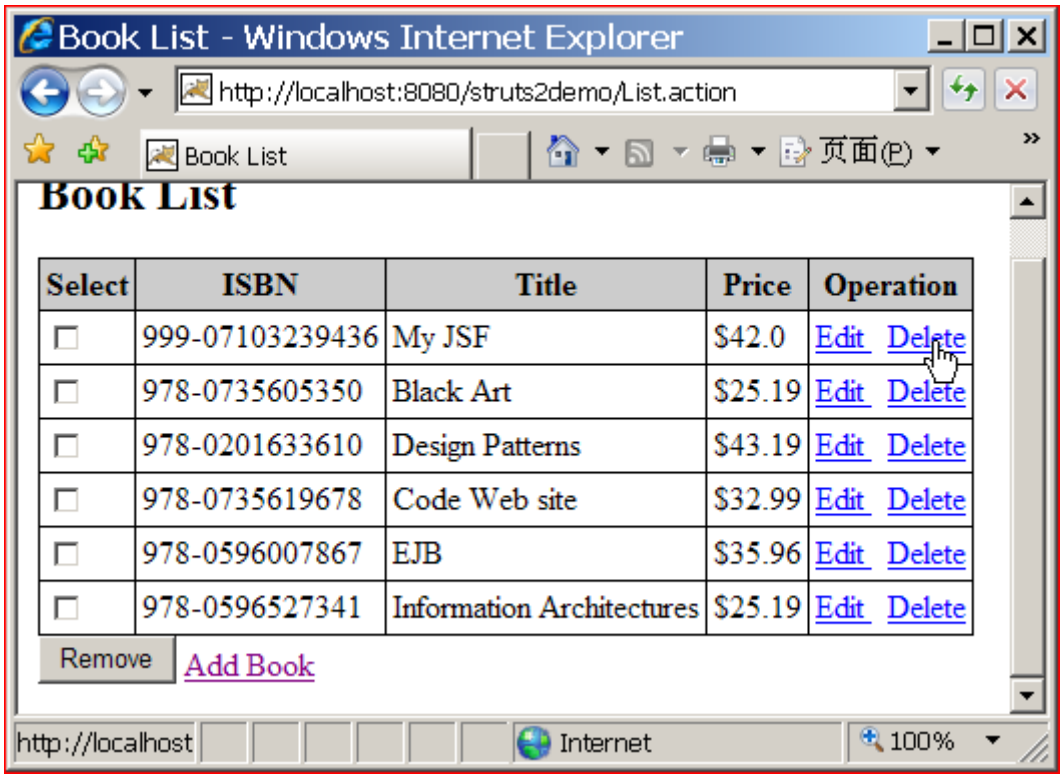
这是一个列表页：



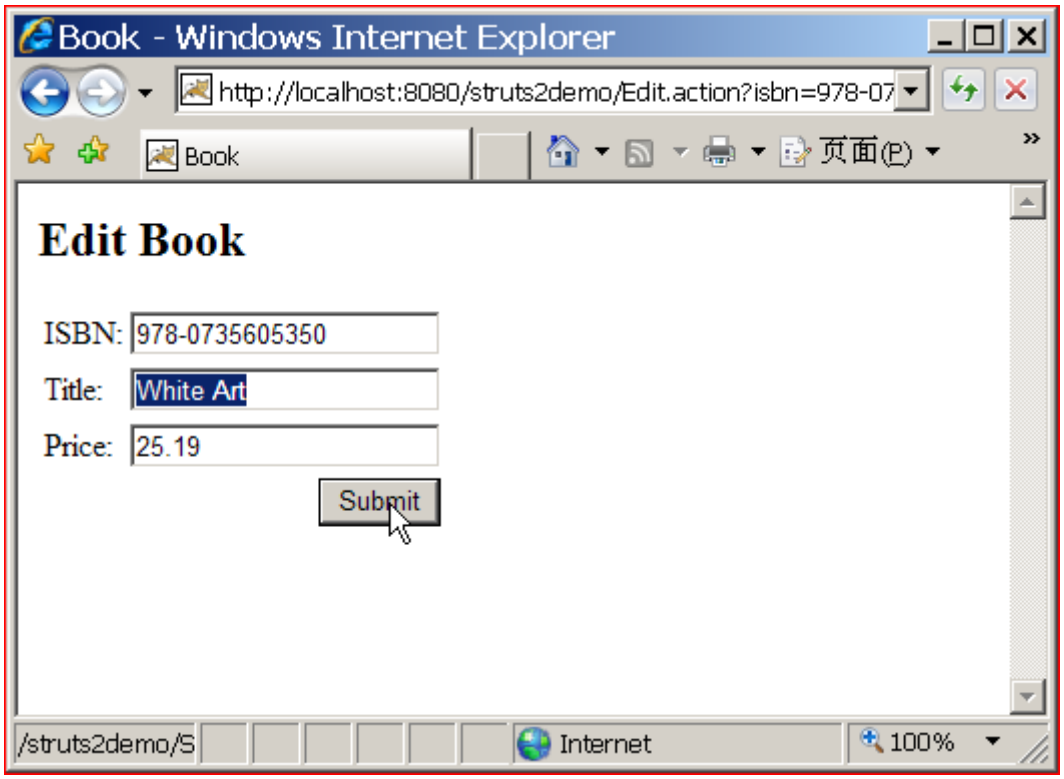
当单击添加时，显示添加页面。



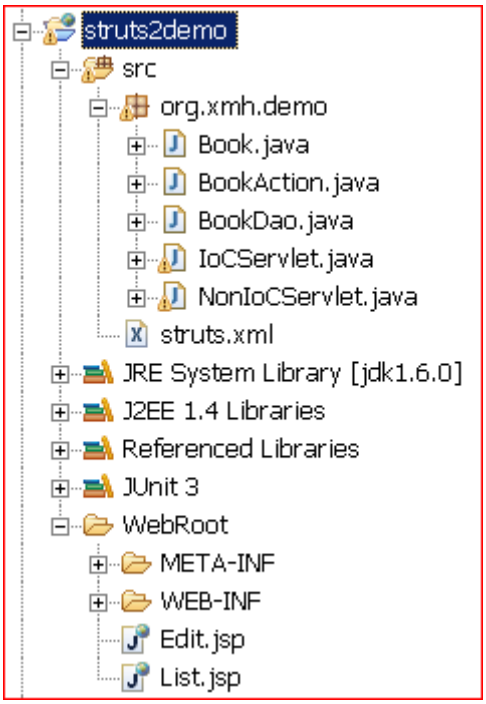
单击提交按钮，新添加的数据将在列表页显示出来。



执行删除命令，将删除新添加的这一条数据。然后单击“Black Art”所对应的“Edit”按钮。将显示数据更新页。



程序的布局如下图所示：



首先，看看数据模型 Book 类。Book 类有三个属性 isbn,、title 和 price 分别代表书籍的编号、名称和价格，其中编号用于唯一标识书籍（相当数据库中的主键）。

```
package org.xmh.demo;

public class Book {
    private String isbn;
    private String title;
    private double price;

    public Book() {
    }

    public Book(String isbn, String title, double price) {
        this.isbn = isbn;
        this.title = title;
        this.price = price;
    }

    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
```

```
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

Book 对象，这主要是为了方便检索和保存 Book 对象；另外，我还

接下来看看的 DAO（Data Access Object，数据访问对象），使用 ConcurrentMap 数据结构存储，将 data 变量设为静态唯一来模拟应用程序的数据库。代码如下：

```
package org.xmh.demo;
import java.util.Collection;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;

public class BookDao {
    private static final BookDao instance;
    private static final ConcurrentMap<String, Book> data;

    static {
        instance = new BookDao();
        data = new ConcurrentHashMap<String, Book>();
        data.put("978-0735619678", new Book("978-0735619678", "Code Web site", 32.99));
        data.put("978-0596007867", new Book("978-0596007867", "EJB", 35.96));
        data.put("978-0201633610",  new  Book("978-0201633610",  "Design  Patterns",
43.19));
        data.put("978-0596527341",     new     Book("978-0596527341",     "Information
Architectures", 25.19));
        data.put("978-0735605350", new Book("978-0735605350", "Black Art", 25.19));
    }

    private BookDao() {}

    public static BookDao getInstance() {
        return instance;
    }

    public Collection<Book> getBooks() {
        return data.values();
    }

    public Book getBook(String isbn) {
        return data.get(isbn);
    }

    public void storeBook(Book book) {
        data.put(book.getIsbn(), book);
    }
```

```
    public void removeBook(String isbn) {
        data.remove(isbn);
    }

    public void removeBooks(String[] isbns) {
        for(String isbn : isbns) {
            data.remove(isbn);
        }
    }
}
```

然后，我们再来看看 Action 类的代码：

```
package org.xmh.demo;

import java.util.Collection;

import com.opensymphony.xwork2.ActionSupport;

public class BookAction extends ActionSupport {
    private static final long serialVersionUID = 872316812305356L;

    private String isbn;
    private String[] isbns;
    private Book book;
    private Collection<Book> books;
    private BookDao dao =  BookDao.getInstance();

    public Book getBook() {
        return book;
    }

    public void setBook(Book book) {
        this.book = book;
    }

    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    public String[] getIsbns() {
        return isbns;
    }

    public void setIsbns(String[] isbns) {
        this.isbns = isbns;
    }
```

```
    public Collection<Book> getBooks() {
        return books;
    }

    public void setBooks(Collection<Book> books) {
        this.books = books;
    }

    public String load() {
        book = dao.getBook(isbn);
        return SUCCESS;
    }

    public String list() {
        books = dao.getBooks();
        return SUCCESS;
    }

    public String store() {
        dao.storeBook(book);
        return SUCCESS;
    }

    public String remove() {
        if(null != isbn) {
            dao.removeBook(isbn);
        } else {
            dao.removeBooks(isbns);
        }
        return SUCCESS;
    }
}
```

BookAction 类中属性 isbn 用于表示待编辑或删除的书籍的编号，属性 isbns 用于表示多个待删除的书籍的编号数组，属性 book 表示当前书籍，属性 books 则表示当前的书籍列表。BookAction 有四个 Action 方法分别是 load、list、store 和 remove，也即是 CRUD 都集中在 BookAction 中实现。

再下来是 Action 的配置代码：

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

<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
    <package name="CRUDDEMO" extends="struts-default">
        <action name="List" class="org.xmh.demo.BookAction" method="list">
            <result>List.jsp</result>
        </action>
        <action name="Edit" class="org.xmh.demo.BookAction" method="load">
            <result>Edit.jsp</result>
```

```
        </action>
        <action name="Store" class="org.xmh.demo.BookAction" method="store">
            <result type="redirect">List.action</result>
        </action>
        <action name="Remove" class="org.xmh.demo.BookAction" method="remove">
            <result type="redirect">List.action</result>
        </action>
    </package>
</struts>
```

以上的配置中，我使用了四个 Action 定义。它们都在“/Book”名值空间内。这样我就可以分别通过“http://localhost:8080/Struts2demo/Book/List.action”、“http://localhost:8080/Struts2demo /Book/Edit.action”、“http://localhost:8080/ Struts2demo/Book/Store.action” 和“http://localhost:8080/Struts2demo/Book/Remove.action”来调用 BookAction 的四个 Action 方法进行 CRUD 操作。当然，这只是个人喜好，你大可以只定义一个 Action（假设其名称为“Book”），之后通过“http://localhost:8080/ Struts2demo/Book!list.action”的方式来访问，详细做法请参考《Struts 2.0 的 Action 讲解》。另外，我由于希望在完成编辑或删除之后回到列表页，所以使用类型为 redirect（重定向）的 result。

下面是列表页面的代码：

```
<%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Book List</title>
    <style type="text/css">
        table {
            border: 1px solid black;
            border-collapse: collapse;
        }

        table thead tr th {
            border: 1px solid black;
            padding: 3px;
            background-color: #cccccc;
        }

        table tbody tr td {
            border: 1px solid black;
            padding: 3px;
        }
    </style>
</head>
<body>
    <h2>Book List</h2>
    <s:form action="Remove" theme="simple">
        <table cellspacing="0">
```

```
            <thead>
                <tr>
                    <th>Select</th>
                    <th>ISBN</th>
                    <th>Title</th>
                    <th>Price</th>
                    <th>Operation</th>
                </tr>
            </thead>
            <tbody>
                <s:iterator value="books">
                    <tr>
                        <td><input  type="checkbox"  name="isbns"  value='<s:property
value="isbn" />' /></td>
                        <td><s:property value="isbn" /></td>
                        <td><s:property value="title" /></td>
                        <td>$<s:property value="price" /></td>
                        <td>
                            <a   href='<s:url   action="Edit"><s:param   name="isbn"
value="isbn" /></s:url>'>
                                Edit
                            </a>
                             
                            <a   href='<s:url   action="Remove"><s:param   name="isbn"
value="isbn" /></s:url>'>
                                Delete
                            </a>
                        </td>
                    </tr>
                </s:iterator>
            </tbody>
        </table>
        <s:submit value="Remove" /><a href="Edit.jsp">Add Book</a>
    </s:form>
</body>
</html>
```

以上代码，值得注意的是在<s:form>标签，我设置了 theme 属性为“simple”，这样可以取消其默认的表格布局。之前，有些朋友问我“如果不希望提交按钮放在右边应该怎样做？”，上述做汗是答案之一。当然，更佳的做法自定义一个 theme，并将其设为默认应用到整个站点，如此一来就可以得到统一的站点风格。我会在以后的文章中会对此作详细的描述。

```
<%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>

<!DOCTYPE     html     PUBLIC     "-//W3C//DTD     XHTML     1.0     Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Book</title>
</head>
<body>
    <h2>
```

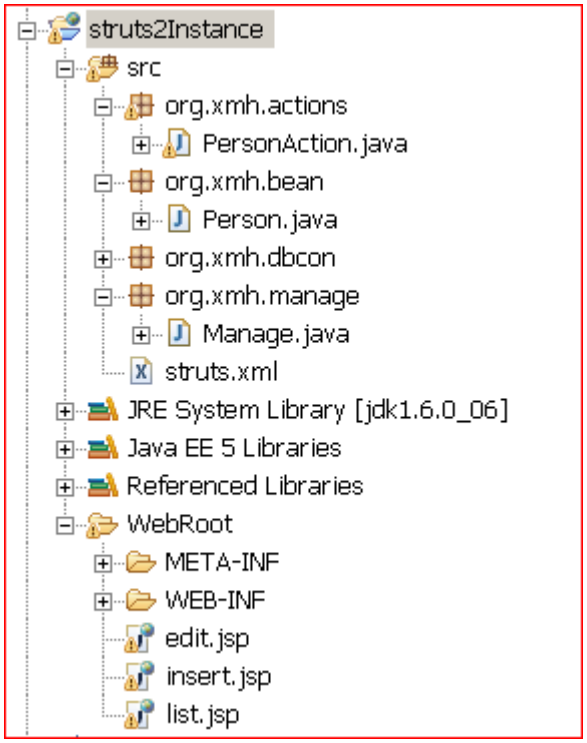
```
        <s:if test="null == book">
            Add Book
        </s:if>
        <s:else>
            Edit Book
        </s:else>
    </h2>
    <s:form action="Store" >
        <s:textfield name="book.isbn" label="ISBN" />
        <s:textfield name="book.title" label="Title" />
        <s:textfield name="book.price" label="Price" />
        <s:submit />
    </s:form>
</body>
</html>
```

如果 book 为 null，则表明该页面用于添加书籍，反之则为编辑页面。

完成上面这个练习后，接下来以我们一个 Old demo 为例来学习基于数据库的 CRUD。
首先数据库设置为：

	id	name	classes	score
	3	小冰球	517	86
	5	赵六	11	445
	10	朱爱华	34	98
	16	zah	07	334

工程布局图如下：



代码清单如下：

```
    package org.xmh.actions;
```

```
import java.util.List;

import org.xmh.bean.Person;
import org.xmh.manage.Manage;

import com.opensymphony.xwork2.ActionSupport;

public class PersonAction extends ActionSupport {
    Person person;
    private String id;
    List<Person> persons;
    Manage dao = new Manage();

    public String get() {
        person = dao.get(id);
        return SUCCESS;
    }

    public String list() {
        persons = dao.getAll();
        return SUCCESS;
    }

    public String save() {
        dao.save(person.getId(), person.getName(), person.getClasses(), person
                .getScore());
        return SUCCESS;
    }

    public String dele() {
        dao.delete(id);
        return SUCCESS;
    }

    public Person getPerson() {
        return person;
    }

    public void setPerson(Person person) {
        this.person = person;
    }

    public List getPersons() {
        return persons;
    }

    public void setPersons(List persons) {
        this.persons = persons;
    }
```

```
    public Manage getDao() {
        return dao;
    }

    public void setDao(Manage dao) {
        this.dao = dao;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }
}
```

```
package org.xmh.bean;

public class Person {
    private String name;
    private String classes;
    private String score;
    private String id;
    public String getScore() {
        return score;
    }

    public String getName() {
        return name;
    }

    public String getId() {
        return id;
    }

    public void setClasses(String classes) {
        this.classes = classes;
    }

    public void setScore(String score) {
        this.score = score;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setId(String id) {
        this.id = id;
```

```
    }

    public String getClasses() {
        return classes;
    }

}
```

```
package org.xmh.dbcon;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBcon {
    public DBcon() {
    }

    public Connection getcon() {
        Connection con = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        }
        try {
            String url = "jdbc:mysql://localhost:3306/studb"
                    + "?useUnicode=true&characterEncoding=GBK";
            con = DriverManager.getConnection(url, "root", "3239436");

        } catch (SQLException ex1) {
            ex1.printStackTrace();
        }
        return con;
    }
    public static void main(String args[]){
        DBcon db = new DBcon();
        System.out.println(db.getcon().toString());
    }
}
```

```
package org.xmh.manage;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
```

```
import org.xmh.bean.Person;
import org.xmh.dbcon.DBcon;

public class Manage {
    public Manage() {
    }

    boolean issuc = false;
    DBcon dbc = new DBcon();

    public boolean save(String id ,String name, String classes, String score) {
        Connection con = dbc.getcon();
        Statement s = null;
        String sql="";
        if(id==null||"".equals(id))
            sql="insert into stuInfo (name,classes,score)values('"
                + name + "','" + classes + "','" + score + "')";
        else
            sql="update stuInfo set name='" + name
            + "',classes='" + classes + "',score='" + score
            + "' where id='" + id + "'";
        try {
            s = con.createStatement();
            int count = s
                    .executeUpdate(sql);
            if (count > 0) {
                issuc = true;

            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        } finally {
            close(s, con);
        }
        return issuc;
    }

    public List<Person> getAll() {
        List<Person> list = new ArrayList<Person>();
        Connection con = dbc.getcon();
        Statement s = null;
        ResultSet rs = null;
        try {
            String sql = "select * from stuInfo";
            s = con.createStatement();
            rs = s.executeQuery(sql);
            while (rs.next()) {
                Person mybean = new Person();
                mybean.setId(rs.getString("id"));
                mybean.setName(rs.getString("name"));
                mybean.setClasses(rs.getString("classes"));
```

```
                mybean.setScore(rs.getString("score"));
                list.add(mybean);
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        } finally {
            close(rs, s, con);
        }

        return list;
    }

    public boolean delete(String id) {
        Connection con = dbc.getcon();
        Statement s = null;
        try {
            s = con.createStatement();
            int count = s.executeUpdate("delete from stuInfo where id='" + id
                    + "'");
            if (count > 0) {
                issuc = true;
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        } finally {
            close(s, con);
        }
        return issuc;

    }

    public Person get(String id) {
        Connection con = dbc.getcon();
        Person mybean = new Person();
        Statement s = null;
        ResultSet rs = null;
        try {
            String sql = "select * from stuInfo where id='" + id + "'";
            s = con.createStatement();
            rs = s.executeQuery(sql);
            if (rs.next()) {

                mybean.setId(rs.getString("id"));
                mybean.setName(rs.getString("name"));
                mybean.setClasses(rs.getString("classes"));
                mybean.setScore(rs.getString("score"));
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        } finally {
            close(rs, s, con);
```

```
        }
        return mybean;

    }

    public void close(ResultSet rs, Statement s, Connection con) {
        try {
            rs.close();
            s.close();
            con.close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }

    public void close(Statement s, Connection con) {
        try {
            s.close();
            con.close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }

    }
}
```

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

<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
    <package name="Struts2_CRUD_DEMO" extends="struts-default">
        <action name="List" class="org.xmh.actions.PersonAction"
            method="list">
            <result>/list.jsp</result>
        </action>
        <action name="Edit" class="org.xmh.actions.PersonAction"
            method="get">
            <result>/edit.jsp</result>
        </action>
        <action name="Save" class="org.xmh.actions.PersonAction"
            method="save">
            <result type="redirect">/List.action</result>
        </action>
        <action name="Delete" class="org.xmh.actions.PersonAction"
            method="dele">
            <result type="redirect">List.action</result>
        </action>
```

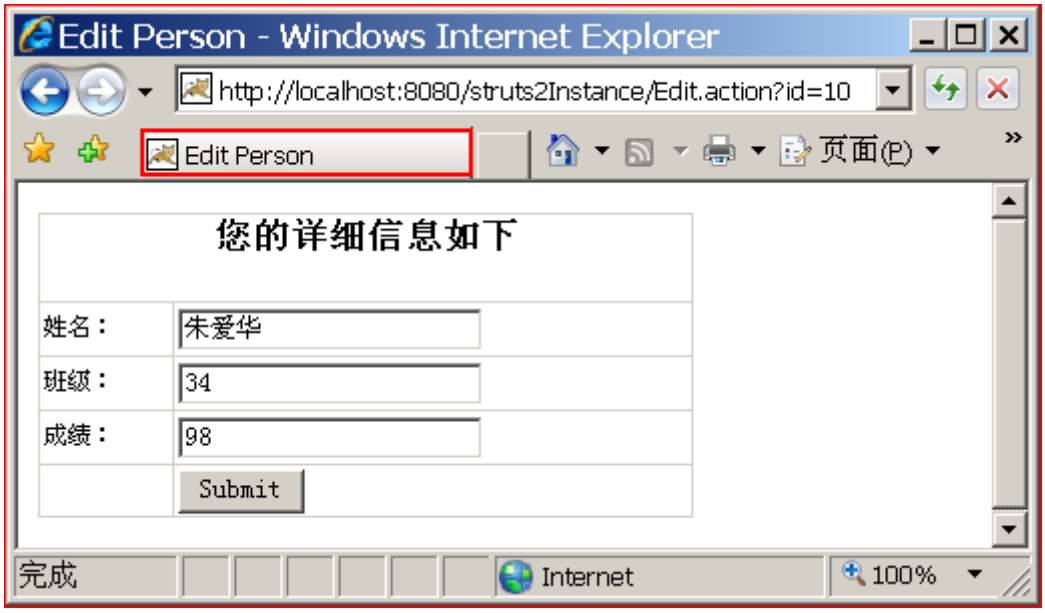
```
    </package>
</struts>
```


```
<%@page contentType="text/html; charset=GBK" import="java.util.*"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title>select</title>
        <style>
.xmh {
    font-size: 12px
}
</style>
    </head>
    <body bgcolor="#ffffff">
        <s:form action="Remove" theme="simple">
            <table width="442" border="1" cellpadding="1" cellspacing="0"
                style="border-style: inherit" bordercolordark="#FFFFFF" class="xmh">
                <tr>
                    <td>
                        <strong>姓名</strong>
                    </td>
                    <td>
                        <strong>班级</strong>
                    </td>
                    <td>
                        <strong>成绩</strong>
                    </td>
                    <td>
                        <strong>修改</strong>
                    </td>
                    <td>
                        <strong>删除</strong>
                    </td>
                    <td>
                        <strong>添加学员</strong>
                    </td>
                </tr>
```

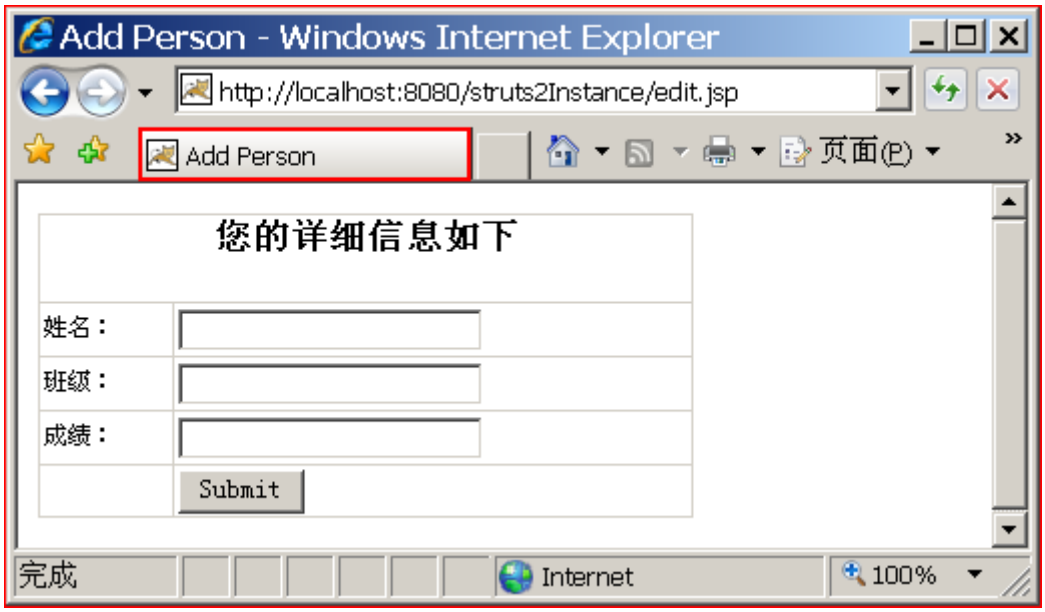
```
                <s:iterator value="persons">
                    <tr>
                        <td>
                            <s:property value="name" />
                        </td>
                        <td>
                            <s:property value="classes" />
                        </td>
                        <td>
                            <s:property value="score" />
                        </td>
                        <td>
                            <a
                                href='<s:url      action="Edit"><s:param      name="id"
value="id" /></s:url>'>编辑信息</a>
                        </td>
                        <td>
                            <a
                                href='<s:url     action="Delete"><s:param     name="id"
value="id" /></s:url>'>删除信息</a>
                        </td>
                        <td>
                            <a href="edit.jsp">添加信息</a>
                        </td>
                    </tr>
                </s:iterator>
            </table>
        </s:form>
    </body>
</html>
```

```
<%@page contentType="text/html; charset=GBK"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title><s:if test="null == book">
            Add Person
        </s:if> <s:else>
            Edit Person
        </s:else>
        </title>
        <style>
.xmh {
    font-size: 12px
}
</style>
    </head>

    <body>
        <s:form action="Save" theme="simple">
            <table width="328" border="1" cellpadding="1" cellspacing="0"
```

```
                style="border-style: inherit" bordercolordark="#FFFFFF" class="xmh">

                <tr>
                    <td colspan="2">
                        <h3 align="center">
                            您的详细信息如下
                        </h3>
                    </td>
                </tr>
                <tr>
                    <td>
                        姓名：
                    </td>
                    <td>
                        <s:textfield name="person.name" />
                    </td>
                </tr>
                <tr>
                    <td>
                        班级：
                    </td>
                    <td>
                        <s:textfield name="person.classes" />
                    </td>
                </tr>
                <tr>
                    <td>
                        成绩：
                    </td>
                    <td>
                        <s:textfield name="person.score" />
                    </td>
                </tr>
                <tr>
                    <td>
                        <s:hidden name="person.id" />
                    </td>
                    <td>
                        <s:submit />
                    </td>
                </tr>
            </table>
        </s:form>
    </body>
</html>
```



添加的页面效果如下：




```
<%@page contentType="text/html; charset=GBK"%>
<html>
    <head>
        <title>insert</title>
        <style>
.xmh {
    font-size: 12px
}
</style>
    </head>
    <body>
        <form action="isinsert.jsp" method="post">
            <table width="402" border="1" cellpadding="1" cellspacing="0"
                style="border-style: inherit" bordercolordark="#FFFFFF" class="xmh">
                <tr>
                    <td width="115">
```

```
                    姓名：
                </td>
                <td width="280">
                    <label>
                        <input type="text" name="username" />
                    </label>
                </td>
            </tr>
            <tr>
                <td>
                    班级：
                </td>
                <td>
                    <label>
                        <input type="text" name="classes" />
                    </label>
                </td>
            </tr>
            <tr>
                <td>
                    成绩：
                </td>
                <td>
                    <label>
                        <input type="text" name="score" />
                    </label>
                </td>
            </tr>
            <tr>
                <td>
                     
                </td>
                <td>
                    <input type="submit" name="Submit" value="提交" />
                    <input type="submit" name="Submit2" value="重置" />
                    <a href="select.jsp">查看信息</a>
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

第 9 章 表达式 OGNL

OGNL 的全称是 Object Graph Navigation Language(对象图导航语言)，它是一种强大的表达式语言，让你通过简单一致的表达式语法来读取和设置 Java 对象的属性值，调用对象的方法，遍历整个对象的结构图，实现字段类型转换等功能。

为什么使用 OGNL？

相对于其它的表达式语言，OGNL 的功能更为强大，它提供了很多高级而必需的特性，例如强大的类型转换功能、静态或实例方法的执行、跨集合投影，以及动态 lambda 表达式定义等。

9.1 OGNL概述

OGNL 表达式的计算都是围绕 OGNL 上下文来进行的，OGNL 上下文实际上就是一个 Map 对象，由 ognl.OgnlContext 类(实现了 java.util.Map 接口)来表示。OGNL 上下文可以包含一个或多个 JavaBean 对象，在这些对象中有一个是特殊的，这个对象就是上下文的根(root)对象。如果在写表达式的时候，没有指定使用上下文中的哪一个对象，那么根对象将被假定为表达式所依据的对象。

下面我们给出一个例子程序，看看在 OGNL 中如何根据上下文来计算表达式的值。

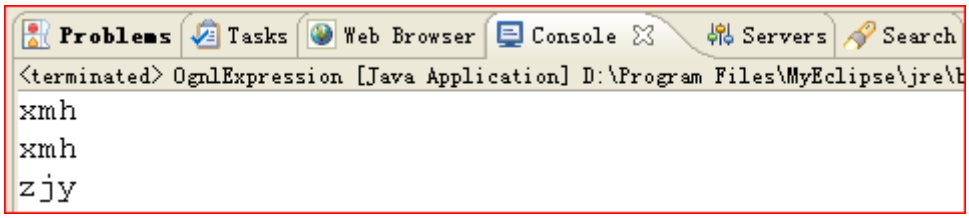
```
public class Employee{
     private String name;
     public String getName(){
          return name;
     }
     public void setName(String name){
          this.name=name;
     }
}
```

```
public class Manager{
     private String name;
     public String getName(){
          return name;
     }
     public void setName(String name){
          this.name=name;
     }
}
```

```
import ognl.Ognl;
import ognl.OgnlContext;
import ognl.OgnlException;


public class OgnlExpression{
     private Object expression;
     public OgnlExpression(String expressionString) throws OgnlException{
          expression=Ognl.parseExpression(expressionString);
     }
     public Object getExpression(){
          return expression;
     }
     public Object getValue(OgnlContext context, Object rootObject)throws OgnlException{
          return Ognl.getValue(getExpression(), context, rootObject);
     }
     public void setValue(OgnlContext context, Object rootObject, Object value) throws
OgnlException{
          Ognl.setValue(getExpression(), context,rootObject, value);
     }
     public static void main(String[] args) throws OgnlException{
          Employee employee=new Employee();
```

```
            employee.setName("xmh");
            Manager manager=new Manager();
            manager.setName("zjy");
            //创建 OGNL 上下文
            OgnlContext context=new OgnlContext();
            //将 employee 对象和 manager 对象放到 OGNL 上下文中
            context.put("employee",employee);
            context.put("manager",manager);
            //将 employee 对象设置为 OGNL 上下文的根
            context.setRoot(employee);

            //表达式 name，将执行 employee.getName()，因为 employee 对象是根对象
            OgnlExpression exp=new OgnlExpression("name");
            System.out.println(exp.getValue(context, context.getRoot()));

            //表达式#employee.name，将执行 employee.getName()
            exp=new OgnlExpression("#employee.name");
            System.out.println(exp.getValue(context, context.getRoot()));

            //表达式#manager.name,将执行 manager.getName()
            //如果你访问的不是 OGNL 上下文中的根对象，那么必须在前面加上一个名称空间，例如#manager
            exp=new OgnlExpression("#manager.name");
            System.out.println(exp.getValue(context, context.getRoot()));
        }
}
```

控制台显示的结果如下所示：



在 OGNL 上下文中，只能有一个根对象，如果你访问根对象，那么在写表达式的时候，直接写对象的属性就可以了；否则，你需要使用“#key”前缀，例如表达式#namager.name。

9.2 OGNL基础

OGNL 表达式

OGNL 表达式的基础单元就是导航链，通常简称为链(chain)。最简单的链由下列部分组成：

1、属性名：如 name 和 manager.name；

2、方法调用：如 manager.hashCode()，返回 manager 对象的散列码；

3、数组索引：如 emals[0]，返回当前对象的邮件列表中的第一个邮件地址。

所有 OGNL 表达式的计算都是在当前对象的上下文中，一个链简单地使用链中先前链接的结果作为下一步计算的当前对象。我们看如下所示的链：

name.toCharArray()[0].numericValue.toString()

这个表达式按照下列的步骤进行计算：

1、获取根对象的 name 属性；

2、在 String 结果上调用 toCharArray()方法；

3、从 char 数组中提取第一个字符；

4、从提取的字符对象上行到 numericValue 属性(这个字符被表示为 Character 对象，Character 类有一个 getNumericValue()方法)；

5、在 Integer 对象结果上调用 toString()方法。

这个表达式最终结果是最后返回的 toString()方法调用返回的字符串。

常量

OGNL 支持的所有常量类型：

1、字符串常量：以单引号或双引号括起来的字符串。如"hello"，'hello'。不过要注意的是，如果是单个字符的字符串常量，必须使用双引号。

2、字符常量：以单引号括起来的字符。如'a'。

3、数值常量：除了 Java 中的 int、long、float 和 double 外，OGNL 还让你使用“b”或“B”后缀指定 BigDecimal 常量，用“h”“H”后缀指定 BigInteger 常量。

4、布尔常量： true 和 false。

5、null 常量。

操作符

OGNL 除了支持所有的 Java 操作符外，还支持以下几种：

1、逗号,

与 C 语言中的逗号操作符类似。

2、花括号{}

用于创建列表，元素之间用逗号分隔。

3、in 和 not in

用于判断一个值是否在集合中。

访问 JavaBean 的属性

假如有一个 employee 对象作为 OGNL 上下文的根对象，那对于下面的表达式：

```
1、name              对应的 java 代码是 employee.getName();
2、address.country   对应的 java 代码是 employee.getAddress().getCountry();
```

访问静态方法和静态字段

```
@class@method(args)          //调用静态方法
@class@field                 //调用静态字段
```

其中 class 必须给出完整的类名(包括包名),如果省略 class，那么默认使用的类是 java.util.Math，如：

```
@@min(5,3)
@@max(5,3)
@@PI
```

索引访问

OGNL 支持多种索引方式的访问。

1、数组和列表索引

在 OGNL 中，数组和列表可以大致看成是一样的。

如：array[0]、list[0]。表达式：{'zhangsan','lisi','wangwu'}[1]等。

2、JavaBean 的索引属性

要使用索引属性，需要提供两对 setter 和 getter 方法，一对用于数组，一对用于数组中的元素。

如：有一个索引属性 interest，它的 getter 和 setter 如下

```
public String[] interest;
public String[] getInterest(){ return interest;}
```

```
public void setInterest(String[] interest){ this.interest=interest;}
public String getInterest(int i){ return interest[i]}
public void setInterest(int i, String newInterest){ interest[i]=newInterest;}
```

对于表达式 interest[2]，OGNL 可以正确解释这个表达式，调用 getInterest(2)方法。如果是设置的情况下，会调用 setInterest(2,value)方法。

3、OGNL 对象的索引属性

JavaBean 的索引属性只能使用整型作为索引，OGNL 扩展了索引属性的概念，可以使用任意的对象来作为索引。

对集合进行操作

1、创建集合：

创建列表 使用花括号将元素包含起来，元素之间使用逗号分隔。如{'zhangsan','lisi','wangwu'}

创建数组 OGNL 中创建数组与 Java 语言中创建数组类似。

创建 Map

Map 使用特殊的语法来创建 #{"key":value, ...}

如果想指定创建的 Map 类型，可以在左花括号前指定 Map 实现类的类名。如：

#@java.util.LinkedHashMap@{"key":"value",....}

Map 通过 key 来访问，如 map["key"]或 map.key。

2、投影

OGNL 提供了一种简单的方式在一个集合中对每一个元素闻调用相同的方法，或者抽取相同的属性，并将结果保存为一个新的集合，称之为投影。

假如 employees 是一个包含了 employee 对象的列表，那么

#employees.{name}将返回所有雇员的名字的列表。

在投影期间，使用#this 变量来引用迭代中的当前元素。如：

```
objects.{#this instanceof String? #this: #this.toString()}
```

3、选择

OGNL 提供了一种简单的方式来使用表达式从集合中选择某些元素，并将结果保存到新的集合中，称为选择。如：

```
#employees.{?#this.salary>3000}将返回薪水大于 3000 的所有雇员的列表。
#employees.{^#this.salary>3000} 将返回第一个薪水大于 3000 的雇员的列表。
#employees.{$#this.salary>3000} 将返回最后一个薪水大于 3000 的雇员的列表。
```

lambda 表达式

lambda 表达式的语法是：:[...]。OGNL 中的 lambda 表达式只能使用一个参数，这个参数通过#this 引用。如：

```
#fact= :[ #this<=1 ? 1 : #this* #fact ( #this-1) ], #fact(30)
#fib= :[#this==0 ? 0 : #this==1 ? 1 : #fib(#this-2)+#fib(#this-1)], #fib(11)
```

9.3 struts2 中OGNL

OGNL 是通常要结合 Struts 2 的标志一起使用，如<s:property value="xx" /> 等。大家经常遇到的问题是#、%和$这三个符号的使用。

◆ “#”主要有三种用途：

1、访问 OGNL 上下文和 Action 上下文，#相当于 ActionContext.getContext()；下表有几个 ActionContext 中有用的属性：

名称	作用	例子
parameters	包含当前HTTP请求参数的Map	#parameters.id[0]作用相当于request.getParameter("id")
request	包含当前HttpServletRequest的属性（attribute）的Map	#request.userName相当于request.getAttribute("userName")
session	包含当前HttpSession的属性（attribute）的Map	#session.userName相当于session.getAttribute("userName")
application	包含当前应用的ServletContext的属性（attribute）的Map	#application.userName相当于application.getAttribute("userName")
attr	用于按request > session > application顺序访问其属性（attribute）	#attr.userName相当于按顺序在以上三个范围（scope）内读取userName属性，直到找到为止

2、用于过滤和投影（projecting)集合，如 books.{?#this.price<100} ；

3、构造 Map，如#{'foo1':'bar1', 'foo2':'bar2'} 。

◆ “%”符号的用途是在标志的属性为字符串类型时，计算 OGNL 表达式的值。例如在 Ognl.jsp 中加入以下代码：

```
< hr />
    < h3 > %的用途 </ h3 >
    < p >< s:url value ="#foobar['foo1']" /></ p >
    < p >< s:url value ="%{#foobar['foo1']}" /></ p >
```

◆ “$”有两个主要的用途

用于在国际化资源文件中，引用 OGNL 表达式（请参见相应章节）；

在 Struts 2 配置文件中，引用 OGNL 表达式，如

```
< action name ="AddPhoto" class ="addPhoto" >
            < interceptor-ref name ="fileUploadStack" />
            < result type ="redirect" > ListPhotos.action?albumId=${albumId} </ result
>
        </ action >
```

9.4 OGNL使用示例

```
package org.xmh.demo;
public class Student {
    private String name;
    private int age;
    private int score;

    public Student() {
    }

    public Student(String name, int age, int score) {
        this.name = name;
        this.age = age;
        this.score = score;
    }

……省略相应的 set/get 方法
}
```

Action 类代码清单：

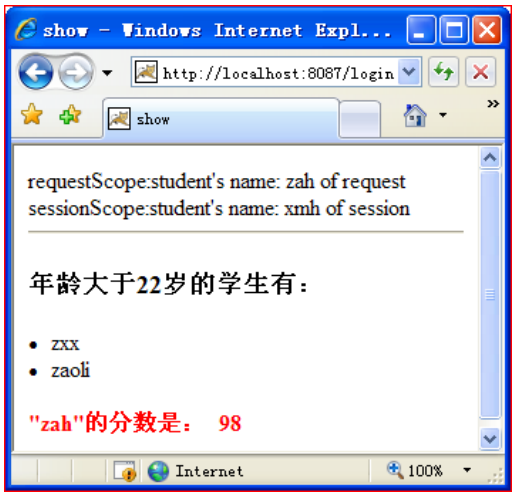
```
package org.xmh.demo;
import java.util.ArrayList;
```

```
import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.apache.struts2.interceptor.ServletRequestAware;

import com.opensymphony.xwork2.ActionSupport;

public class OnglDemoAction extends ActionSupport implements
        ServletRequestAware {
    private List<Student> stus = new ArrayList<Student>();
    HttpServletRequest request;
    private static final long serialVersionUID = 1L;

    public void setServletRequest(HttpServletRequest request) {
        this.request = request;
    }

    public String execute() throws Exception {
        HttpSession session = request.getSession();
        request.setAttribute("name", "zah of request");
        session.setAttribute("name", "xmh of session");
        stus.add(new Student("zxx", 23, 63));
        stus.add(new Student("wangwu", 19, 60));
        stus.add(new Student("zaoli", 26, 87));
        stus.add(new Student("xuke", 22, 57));
        stus.add(new Student("aihua", 21, 98));
        return SUCCESS;
    }

    public List<Student> getStus() {
        return stus;
    }

    public void setStus(List<Student> stus) {
        this.stus = stus;
    }
}
```

配置文件要添加如下内容：

```
    <action name="show" class="org.xmh.demo.OnglDemoAction">
            <result>/showstu.jsp</result>
        </action>
```

运行服务，在 IE 中输入“http://localhost:8087/logindemo/show.action”，显示的页面效果如下图所示：



第 10 章 上传下载

◆ 上传单个文件

上传文件是很多 Web 程序都具有的功能。在 Struts1.x 中已经提供了用于上传文件的组件。而在 Struts2 中提供了一个更为容易操作的上传文件组件。所不同的是，Struts1.x 的上传组件需要一个 ActionForm 来传递文件，而 Struts2 的上传组件是一个拦截器（这个拦截器不用配置，是自动装载的）。在本文中先介绍一下如何用 struts2 上传单个文件，最后介绍一下用 struts2 上传任意多个文件。

要用 Struts2 实现上传单个文件的功能非常容易实现，只要使用普通的 Action 即可。但为了获得一些上传文件的信息，如上传文件名、上传文件类型以及上传文件的 Stream 对象，就需要按着一定规则来为 Action 类增加一些 getter 和 setter 方法。

在 Struts2 中，用于获得和设置 java.io.File 对象（Struts2 将文件上传到临时路径，并使用 java.io.File 打开这个临时文件）的方法是 getUpload 和 setUpload。获得和设置文件名的方法是 getUploadFileName 和 setUploadFileName，获得和设置上传文件内容类型的方法是 getUploadContentType 和 setUploadContentType。上传需要的 JAR 文件：commons-io-1.1.jar，commons-fileupload-1.1.1.jar

下面是用于上传的动作类的完整代码：

```
package action;
import java.io.*;
import com.opensymphony.xwork2.ActionSupport;
public class UploadAction extends ActionSupport
{
    private File upload;
    private String fileName;
    private String uploadContentType;

    public String getUploadFileName()
    {
        return fileName;
    }
    public void setUploadFileName(String fileName)
    {
        this.fileName = fileName;
```

```
    }
    public File getUpload()
    {
        return upload;
    }
    public void setUpload(File upload)
    {
        this.upload = upload;
    }
    public void setUploadContentType(String contentType)
    {
        this.uploadContentType=contentType;
    }
    public String getUploadContentType()
    {
        return this.uploadContentType;
    }
    public String execute() throws Exception
    {
        java.io.InputStream is = new java.io.FileInputStream(upload);
        java.io.OutputStream os = new java.io.FileOutputStream("d:\\upload\\" +
fileName);
        byte buffer[] = new byte[8192];
        int count = 0;
        while((count = is.read(buffer)) > 0)
        {
            os.write(buffer, 0, count);
        }
        os.close();
        is.close();
        return SUCCESS;
    }
}
```

在execute方法中的实现代码就很简单了,只是从临时文件复制到指定的路径(在这里是d:\upload)中。上传文件的临时目录的默认值是 javax.servlet.context.tempdir 的值，但可以通过struts.properties(和struts.xml在同一个目录下)的struts.multipart.saveDir属性设置。Struts2上传文件的默认大小限制是 2M（2097152 字节），也可以通过 struts.properties 文件中的struts.multipart.maxSize 修改，如 struts.multipart.maxSize=2048 表示一次上传文件的总大小不能超过2K字节。

下面的代码是上传文件的JSP页面代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title>上传单个文件</title>
    </head>
    <body>
        <s:form action="upload" namespace="/test"
            enctype="multipart/form-data">
            <s:file name="upload" label="输入要上传的文件名" />
```

```
            <s:submit value="上传" />
        </s:form>
    </body>
</html>
```

也可以在 success.jsp 页中通过<s:property>获得文件的属性（文件名和文件内容类型），代码如下：

```
    <s:property value="uploadFileName"/>
```

◆ 上传任意多个文件

在 Struts2 中，上传任意多个文件也非常容易实现。首先，要想上传任意多个文件，需要在客户端使用 DOM 技术生成任意多个<input type=”file” />标签。name 属性值都相同。代码如下：

```
<html>
    <head>
        <script language="javascript">
function addComponent()
{
        var uploadHTML = document.createElement( "<input type='file'  name='upload'/>");
        document.getElementById("files").appendChild(uploadHTML);
        uploadHTML = document.createElement( "<p/>");
        document.getElementById("files").appendChild(uploadHTML);
}
</script>
    </head>
    <body>
        <input type="button" onclick="addComponent();" value="添加文件" />
        <br />
        <form onsubmit="return true;" action="/struts2/test/upload.action"
            method="post" enctype="multipart/form-data">
            <span id="files"> <input type='file' name='upload' />
                <p />
            </span>
            <input type="submit" value="上传" />
        </form>
    </body>

</html>
```

上面的 javascript 代码可以生成任意多个<input type=’file’>标签，name 的值都为 file（要注意的是，上面的 javascript 代码只适合于 IE 浏览器，firefox 等其他浏览器需要使用他的代码）。至于 Action 类，和上传单个文件的 Action 类基本一至，只需要将三个属性的类型改为 List 即可。代码如下：

```
package action;

import java.io.*;
import com.opensymphony.xwork2.ActionSupport;

public class UploadMoreAction extends ActionSupport
{
    private java.util.List<File> uploads;
    private java.util.List<String> fileNames;
    private java.util.List<String> uploadContentTypes;
```

```
    public java.util.List<String> getUploadFileName()
    {
        return fileNames;
    }
    public void setUploadFileName(java.util.List<String> fileNames)
    {
        this.fileNames = fileNames;
    }
    public java.util.List<File> getUpload()
    {
        return uploads;
    }

    public void setUpload(java.util.List<File> uploads)
    {
        this.uploads = uploads;
    }

    public void setUploadContentType(java.util.List<String> contentTypes)
    {
        this.uploadContentTypes = contentTypes;

    }

    public java.util.List<String> getUploadContentType()
    {
        return this.uploadContentTypes;
    }

    public String execute() throws Exception
    {
        if (uploads != null)
        {
            int i = 0;
            for (; i < uploads.size(); i++)
            {
                java.io.InputStream is = new java.io.FileInputStream(uploads.get(i));
                java.io.OutputStream os = new java.io.FileOutputStream(
                        "d:\\upload\\" + fileNames.get(i));
                byte buffer[] = new byte[8192];
                int count = 0;
                while ((count = is.read(buffer)) > 0)
                {
                    os.write(buffer, 0, count);
                }
                os.close();
                is.close();
            }
        }
        return SUCCESS;
    }
```

```
}
```

在 execute 方法中，只是对 List 对象进行枚举，在循环中的代码和上传单个文件时的代码基本相同。如果读者使用过 struts1.x 的上传组件，是不是感觉 Struts2 的上传功能更容易实现呢？在 Struts1.x 中上传多个文件时，可是需要建立带索引的属性的。而在 Struts2 中，就是这么简单就搞定了。图 1 是上传任意多个文件的界面。

第 11 章 视图浅析

第 12 章 集成 Ajax

12.1 JSON概述

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，易于阅读和编写，同时也易于机器解析和生成。它基于 ECMA262 语言规范（1999-12 第三版）中 JavaScript 编程语言的一个子集。JSON 采用与编程语言无关的文本格式，但是也使用了类 C 语言（包括 C， C++， C#， Java， JavaScript， Perl， Python 等）的习惯，这些特性使 JSON 成为理想的数据交换格式。

JSON 的结构基于下面两点

1、 "名称/值"对的集合 不同语言中，它被理解为对象(object)，记录(record)，结构(struct)，字典(dictionary)，哈希表(hash table)，键列表(keyed list)等

2、值的有序列表 多数语言中被理解为数组(array)

JSON 以一种特定的字符串形式来表示 JavaScript 对象。如果将具有这样一种形式的字符串赋给任意一个 JavaScript 变量，那么该变量会变成一个对象引用，而这个对象就是字符串所构建出来的，好像有点拗口，我们还是用实例来说明。

这里假设我们需要创建一个 User 对象，并具有以下属性

用户 ID
用户名
用户 Email
可以使用以下 JSON 形式来表示 User 对象：

```
{"UserID":11, "Name":"Truly", "Email":"zhuleipro◎hotmail.com"};
```

然后如果把这一字符串赋予一个 JavaScript 变量，那么就可以直接使用对象的任一属性了。
完整代码:

```
<script>
var User = {"UserID":11, "Name":"Truly", "Email":"zhuleipro◎hotmail.com"};
alert(User.Name);
</script>
```

实际使用时可能更复杂一点，比如为 Name 定义更详细的结构，使它具有 FirstName 和 LastName：

```
{"UserID":11,                    "Name":{"FirstName":"Truly","LastName":"Zhu"},
"Email":"zhuleipro◎hotmail.com"}
```

完整代码:

```
<script>
var User = {"UserID":11, "Name":{"FirstName":"Truly","LastName":"Zhu"},
 "Email":"zhuleipro◎hotmail.com"};
alert(User.Name.FirstName);
```

```
</script>
```

现在增加一个新的需求，我们某个页面需要一个用户列表，而不仅仅是一个单一的用户信息，那么这里就需要创建一个用户列表数组。

下面代码演示了使用 JSON 形式定义这个用户列表：

```
[
{"UserID":11,                          "Name":{"FirstName":"Truly","LastName":"Zhu"},
"Email":"zhuleipro◎hotmail.com"},
{"UserID":12,                    "Name":{"FirstName":"Jeffrey","LastName":"Richter"},
"Email":"xxx◎xxx.com"},
{"UserID":13, "Name":{"FirstName":"Scott","LastName":"Gu"}, "Email":"xxx2◎xxx2.com"}
]
```

完整代码：

```
<script>
var UserList = [
{"UserID":11,                          "Name":{"FirstName":"Truly","LastName":"Zhu"},
"Email":"zhuleipro◎hotmail.com"},
{"UserID":12,                    "Name":{"FirstName":"Jeffrey","LastName":"Richter"},
"Email":"xxx◎xxx.com"},
{"UserID":13, "Name":{"FirstName":"Scott","LastName":"Gu"}, "Email":"xxx2◎xxx2.com"}
];
alert(UserList[0].Name.FirstName);
</script>
```

事实上除了使用"."引用属性外，我们还可以使用下面语句：

```
alert(UserList[0]["Name"]["FirstName"]); 或者 alert(UserList[0].Name["FirstName"]);
```

现在读者应该对 JSON 的使用有点认识了，归纳为以下几点：

对象是属性、值对的集合。一个对象的开始于“{”，结束于“}”。每一个属性名和值间用“:”提示，属性间用“,”分隔。

数组是有顺序的值的集合。一个数组开始于"["，结束于"]"，值之间用","分隔。

值可以是引号里的字符串、数字、true、false、null，也可以是对象或数组。这些结构都能嵌套。字符串和数字的定义和 C 或 Java 基本一致。

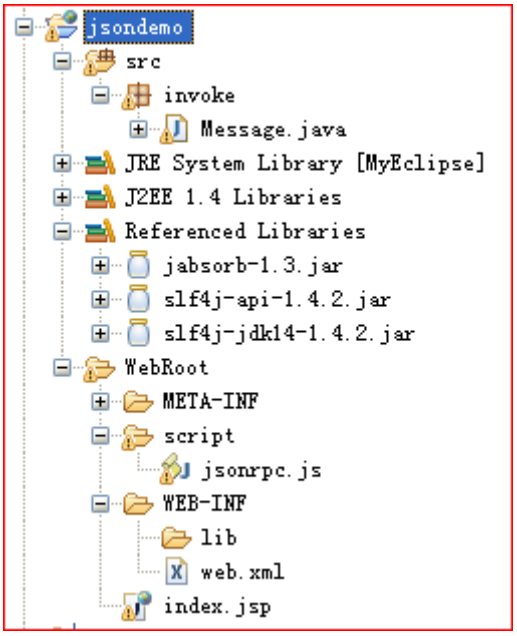
12.2 JSON-RPC概述

JSON-RPC 是一种轻量级的远程过程调用协议，它允许从 Javascript 进行客户-服务器交互。通用机制包含两个建立数据连接的对等体。在连接 的生存期内，对等体可以调用另一个对等体所提供的方法。要调用远程方法，就要发送请求。请求必须使用应答进行响应，除非该请求是一个通知。

这有什么好处呢？使用 JSON-RPC，可以创建定制的 Java 对象，并可通过 Javascript 和 Ajax 实现对其方法的完全访问。此外，这些对象在会话的整个生存期中都将存留！

这需要两个库：需要驻留在服务器应用程序上的 jsonrpc.jar 以及 jsonrpc.js，该 JavaScript 库应下载到客户端。

12.3 JSON示例



配置文件内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
    <servlet>
        <servlet-name>JSONRPCServlet</servlet-name>
        <servlet-class>org.jabsorb.JSONRPCServlet</servlet-class>
        <init-param>
            <param-name>gzip_threshold</param-name>
            <param-value>0</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>JSONRPCServlet</servlet-name>
        <url-pattern>/JSON-RPC</url-pattern>
    </servlet-mapping>
</web-app>
```

导入 JavaScript 库

接下来是要将 jsonrpc.js 库包含到 jsp 中。这将链接到一个文本文件。复制该文本并将其 粘贴到项目中一个新的 JavaScript 文件中，可以称之为 jsonrpc.js。要确保将库导入到所有希望在其中使用 jsonrpc 的页面中，这可通 过在 jsp 的<HEAD>部分添加下面的代码行来实现：

```
<script type="text/javascript"  src="/path/to/jsonrpc.js"</>
```

为了让客户端与服务器通信，需要使用<jsp:useBean>标签创建一个 bridge。将下属代码行放在接近 JSP 顶部的某处：

```
<jsp:useBean id="JSONRPCBridge"  scope="session"
class="com.metaparadigm.jsonrpc.JSONRPCBridge" />
```

然后使用该 bridge 注册一个对象。该对象可以是一个普通 Java 对象，也可以是您自己创建的对象。

```
<% JSONRPCBridge.registerObject("myObject",new  com.package.myobjects.MyObject()); %>
```

另一种替代方案是，为了与页面流通信，可以让页面流初始化一个对象，传递所需的数据，并通过页面流上的一个 getter 方法使该对象对 JSP 可用。然后在 JSP 上执行 getData，以将该对象放在 JSONRPC

可以获取的 pageContext 上：

```
<netui-data:getData resultId="myObject"  value="{pageFlow.myObject}" />
<% JSONRPCBridge.registerObject("myObject",  pageContext.getAttribute("myObject")); %>
```

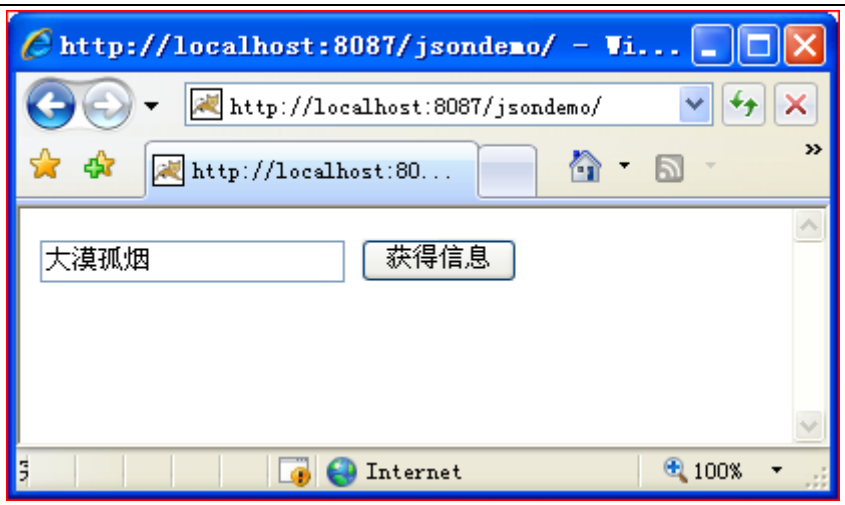
Index.jsp 的代码清单如下所示：

```
<%@ page language="java" import="java.util.*" pageEncoding="GB18030"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <script type="text/javascript" src="script/jsonrpc.js"></script>
        <script type="text/javascript">
function asyc(result,e){
if(e == null)
alert(result);
}
function onLoad(){
jsonrpc = new JSONRpcClient("JSON-RPC");
}
window.onload = onLoad;
function invoke(){
var text = document.getElementById("text");
var result = jsonrpc.msg.getMessage(asyc, text.value);
alert(result);
}
</script>
        <jsp:useBean id="JSONRPCBridge" scope="session"
            class="org.jabsorb.JSONRPCBridge" />
        <jsp:useBean id="message" scope="session" class="invoke.Message" />
        <%
            JSONRPCBridge.registerObject("msg", message);
        %>
    </head>
    <body>
        <input type="text" id="text" />
        <input type="button" value="获得信息" onclick="invoke()" />
    </body>
</html>
```

编写 Message 类及相应的方法

```
public class Message implements java.io.Serializable {
    public String getMessage(String s) {
        return "你好 " + s;
    }
}
```

部署应用程序，启动服务，输入相应的访问地址将显示如下效果：



输入“大漠孤烟”单击“获得信息”按钮，将弹出如下图所示的对话框。

12.4 struts2 与JSON示例

JSON 插件提供了一种名为 json 的 ResultType，一旦为某个 Action 指定了一个类型为 json 的 Result，则该 Result 无需映射到任何视图资源。因为 JSON 插件会负责将 Action 里的状态信息序列化成 JSON 格式的数据，并将该数据返回给客户端页面的 JavaScript。

简单地说，JSON 插件允许我们在 JavaScript 中异步调用 Action，而且 Action 不再需要使用视图资源来显示该 Action 里的状态信息，而是由 JSON 插件负责将 Action 里的状态信息返回给调用页面——通过这种方式，就可以完成 Ajax 交互。

Struts2 提供了一种可插拔方式来管理插件，安装 Struts2 的 JSON 插件与安装普通插件并没有太大的区别，一样只需要将 Struts2 插件的 JAR 文件复制到 Web 应用的 WEB-INF/lib 路径下即可。

（1）登陆 http://code.google.com/p/jsonplugin/downloads/list 站点，下载 Struts2 的 JSON 插件的最新版本，当前最新版本是 0.7，我们可以下载该版本的 JSON 插件。将 json-plugin.jar 文件复制到 Web 应用的 WEB-INF 路径下，即可完成 JSON 插件的安装。

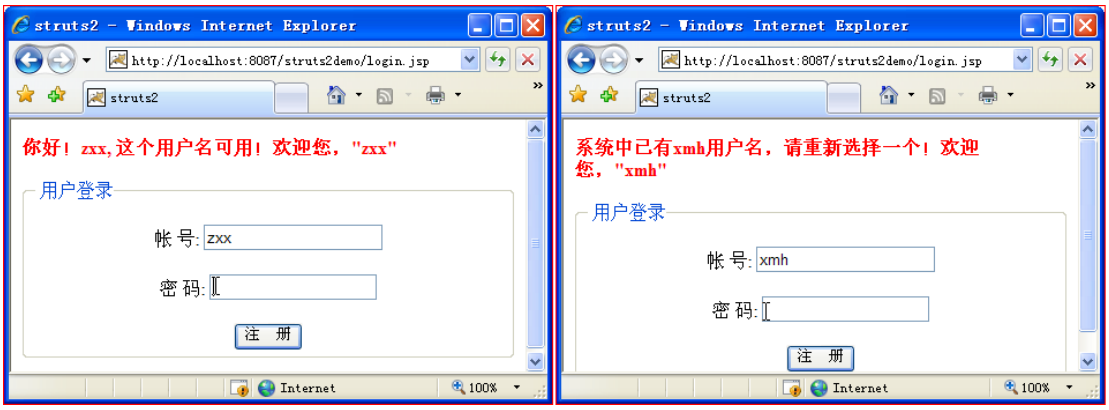
（2）把 JSON.js、prototype.js 复制到 js 文件夹下

在配置文件中有两个值得注意的地方：

第一个地方是配置 struts.i18n.encoding 常量时，不再是使用 GBK 编码，而是 UTF-8 编码，这是因为 Ajax 的 POST 请求都是以 UTF-8 的方式进行编码的。

第二个地方是配置包时，自己的包继承了 json-default 包，而不再继承默认的 default 包，这是因为只有在该包下才有 json 类型的 Result。

在 struts 2 中，要结合 json 的话，其实是不错的选择，下面通过做一个登陆系统来演示 struts2 与 json 的用法。当用户输入姓名并移出焦点时，系统将根据用户输入的信息返回相关的提示。



步骤 1、编写页面代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>struts2</title>
        <script type="text/javascript" src="js/json.js"></script>
        <script type="text/javascript" src="js/prototype.js"></script>
        <script language="JavaScript">
function validateName()
            {
                //请求的地址
                var url = 'validateName.action';
                var params = Form.Element.serialize('userBean.name');
                //创建 Ajax.Request 对象，对应于发送请求
                var myAjax = new Ajax.Request(
                url,
                {
                    //请求方式：POST
                    method:'post',
                    //请求参数
                    parameters:params,
                    //指定回调函数
                    onComplete: processResponse,
                    //是否异步发送请求
                    asynchronous:true
                });
            }
            function processResponse(request)
            {

                var jsonObje = request.responseText.parseJSON();
                $("tip").innerHTML = jsonObje.tip;
                $("tip2").innerHTML='欢迎您，'+JSON.stringify(jsonObje.userBean.name);
            }
  </script>
```

```
    </head>
    <body>
        <span id="tip" style="color: red; font-weight: bold"> </span>
        <span id="tip2" style="color: red; font-weight: bold"> </span>
        <form action="" method="post" name="form">
            <fieldset>
                <legend>
                    用户登录
                </legend>
                <p align="center">
                    帐  号:
                    <input type="text" name="userBean.name" onblur="validateName();" />
                </p>
                ……
            </fieldset>
        </form>
    </body>
</html>
```

步骤 2、编写处理 Action 的代码：

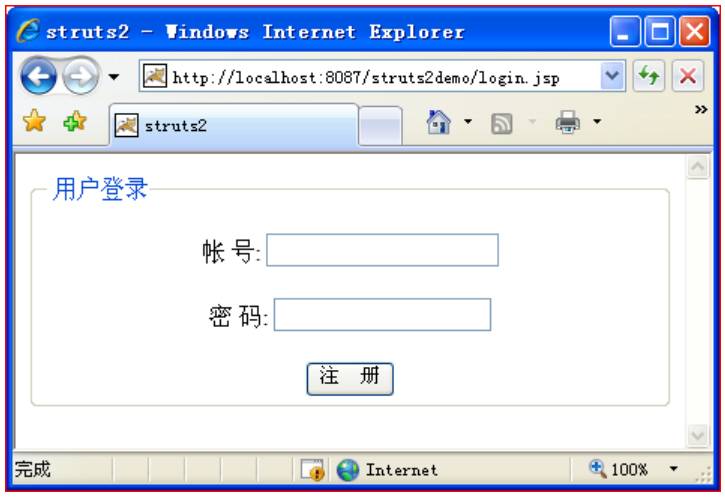
```
public class ValidateNameAction extends ActionSupport {
    private UserBean userBean;
    private String tip;
    @JSON
    public UserBean getUserBean() {
        return userBean;
    }
    public void setUserBean(UserBean userBean) {
        this.userBean = userBean;
    }
    @JSON
    public String getTip() {
        return tip;
    }
    public void setTip(String tip) {
        this.tip = tip;
    }
    public String execute() {
        try {
            if (validateName(userBean)) {
                tip = "你好！" + userBean.getName() + ", 这个用户名可用！";
            } else {
                tip = "系统中已有" + userBean.getName() + "用户名，请重新选择一个！";
            }
        } catch (Exception e) {
            tip = e.getMessage();
```

```
        }
        return SUCCESS;
    }
    public boolean validateName(UserBean userBean) {
        if ("xmh".equals(userBean.getName()))
            return false;
        else
            return true;
    }
}
```

步骤 3、修改配置文件

```
……
<struts>
    <!-- 加载默认的 struts2 配置文件 -->
    <include file="struts-default.xml" />
    <!-- 继承默认的 struts2 配置文件 -->
    <package name="default" extends="json-default" >
        <action name="validateName" class="org.xmh.demo.ValidateNameAction">
            <result type="json" />
        </action>
    </package>
</struts>
```

步骤 4、发布应用程序，输入相应的访问网址，效果图如下所示：



关于本示例更多的知识请参看本书所配光盘源码部分。

第 13 章 集成 Hibernate

本章节主要通过一个示例来学习 strtus2 与 hibernate 的集成运用。

13.1 系统总体设计图

本系统采用了四层架构，分别为视图层、控制器层、数据访问层、持久化层。客户端不直接与数据

库交互，而是通过控制器与数据访问层建立连接，再由数据访问层与数据库交互。

表现层采用了 JSP，控制层采用 struts2，数据访问层使用 Hibernate 封装了对底层数据库的相关操作，数据库采用了 MySQL 数据库存放数据。

13.2 系统用例图

本程序比较简单，在管理员成功登陆之后对书籍进行增加、删除、修改、查询，故而根据程序的场景分析绘制出如图 1 所示的系统用例图：

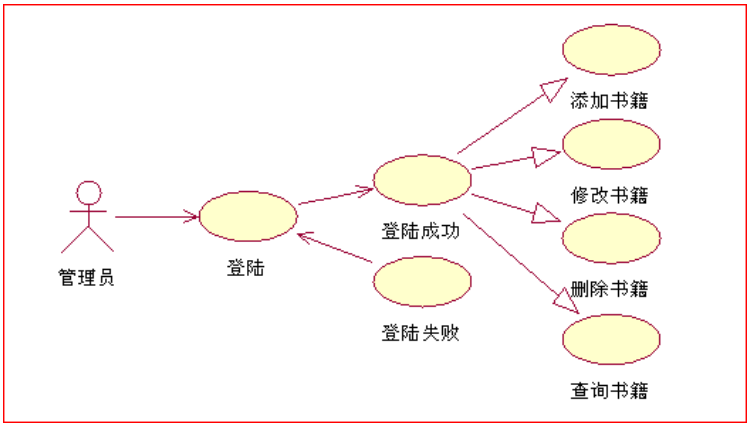


图 1 系统用例图

13.3 数据库

系统有管理员表、书籍表，表结构如图 2 所示：

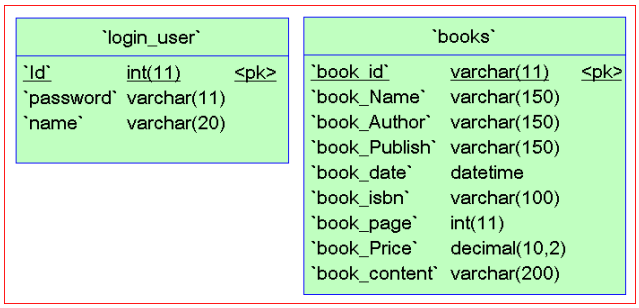


图 2 数据库表效果图

可以使用如下代码创建上图所示数据库表：

```
create database if not exists `bookshell`;
USE `bookshell`;
DROP TABLE IF EXISTS `books`;
CREATE TABLE `books` (
  `book_id` varchar(11) NOT NULL,
  `book_Name` varchar(150) default NULL,
  `book_Author` varchar(150) default NULL,
  `book_Publish` varchar(150) default NULL,
  `book_date` datetime default NULL,
  `book_isbn` varchar(100) default NULL,
```

```
  `book_page` int(11) default NULL,
  `book_Price` decimal(10,2) default NULL,
  `book_content` varchar(200) default NULL,
  PRIMARY KEY  (`book_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
DROP TABLE IF EXISTS `login_user`;
CREATE TABLE `login_user` (
  `Id` int(11) NOT NULL auto_increment,
  `password` varchar(11) default NULL,
  `name` varchar(20) default NULL,
  PRIMARY KEY  (`Id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

13.4 系统效果图展示

◆ **登陆视图**

对于一个系统，安全是必不可缺少的，管理员必须登陆之后才能进行相关的操作。登陆页下如图 3 所示：

图 3 登陆页效果图

主要代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>用户登录</title>
        <meta http-equiv="pragma" content="no-cache">
    </head>
    <body>
        <h2>管理员登陆</h2>
        <s:form action="/Login.action">
```

```
            <s:textfield name="user.name" label="用户名" />
            <s:password  name="user.password" label="密    码"/>
            <s:submit value="登录"/>
        </s:form>
    </body>
</html>
```

◆ 4.2 登陆失败视图

在登陆页输入”用户名”和”密码”，单击【登陆】按钮，如果用户名和密码不正确，则提示“登陆失败”，如图 4 所示：

图 4 登陆失败页效果图

页面主要代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>登录失败</title>
  </head>

  <body>
    登录失败!您的用户名或密码有误！<a href="login.jsp">返回</a>
  </body>
</html>
```

◆ 4.3 登陆成功视图

如果在登陆页输入”用户名”和”密码”正确，系统跳转到欢迎页，如图 5 所示：

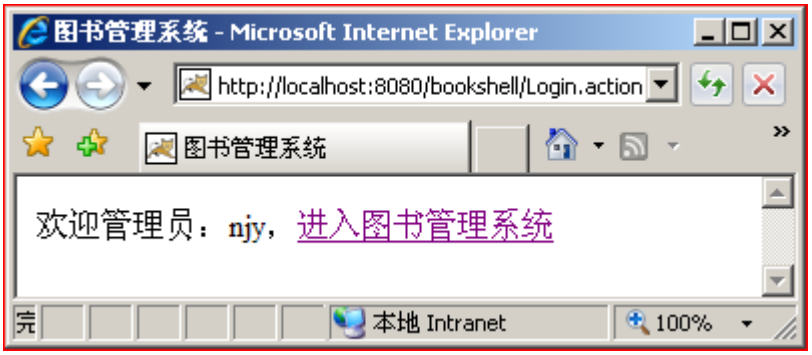


图 5 登陆成功页效果图

页面主要代码如下：

```
<%@page contentType="text/html; charset=UTF-8"%>
```

```
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=GBK" />
        <title>图书管理系统</title>
    </head>
    <body>

        欢迎管理员：${name}，<a href="<s:url action="list" />">进入图书管理系统</a>
    </body>
</html>
```

◆ 4.4 书籍列表视图

在登陆成功之后的欢迎页，点击【进入图书管理系统】系统会进入书籍列表页，书籍列表--顾名思义该页用来显示所有的书籍信息，在后台将数据查询出来，并在页面上迭代出数据。如图 6 所示：

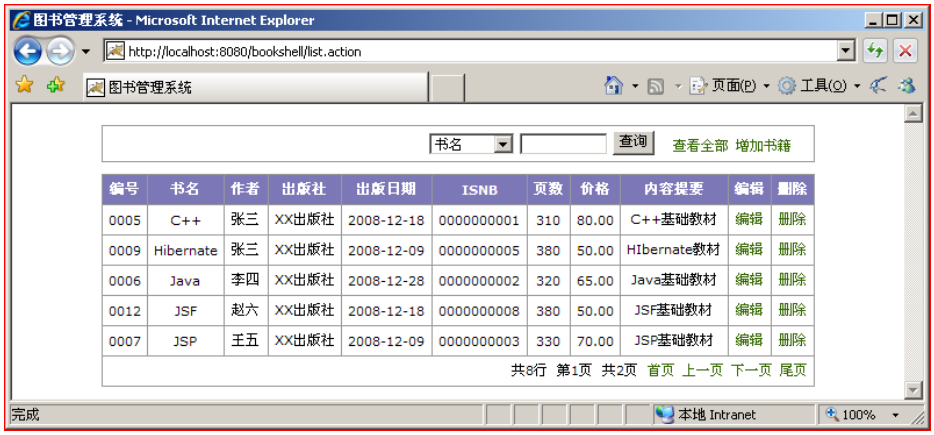


图 6 图书列表页效果图

主要代码如下：

```
<%@page pageEncoding="gb2312" contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
<head><title>图书管理系统</title></head>
<link    rel="stylesheet"    type="text/css"    href="${pageContext.request.contextPath}/styles.css"
media="screen">
    <script language="JavaScript">
        function doSearch(){
            if(document.all.searchValue.value=="")
            {
                alert("请输入查询关键字!");
            }else{

    window.location.href="bookAdmin/list.action?queryName="+document.all.searchName.value+"&&queryV
alue="+document.all.searchValue.value;
            }
        }
```

```
    </script>
<body>
<br/>
<table align="center">
<tr align="center">
    <td align="right" height="20">
        <select name="searchName">
            <option value="bookName">书名</option>
            <option value="bookAuthor">作者</option>
            <option value="bookPublish">出版社</option>
            <option value="bookDate">出版日期</option>
            <option value="bookIsbn">ISNB</option>
            <option value="bookPage">页数</option>
        </select>
        <input type="text" name="searchValue" value="" size="10"/>
        <input type="button" value="查询" onClick="doSearch();">
          
        <a href="<s:url action="list" includeParams="none"/>">查看全部</a>
        <a href='<s:url action="edit" ></s:url>'>增加书籍</a>
          
    </td>
</tr>
</table>
<table align="center">
   <tr align="center">
    <th>编号</th>
       <th>书名</th>
       <th>作者</th>
       <th>出版社</th>
       <th>出版日期</th>
       <th>ISNB</th>
       <th>页数</th>
       <th>价格</th>
       <th>内容提要</th>
       <th>编辑</th>
       <th>删除</th>
   </tr>
   <tbody>
   <s:iterator value="availableItems" status="stuts">
       <tr align="center">
        <td><s:property value="bookId"/></td>
          <td><s:property value="bookName"/></td>
          <td><s:property value="bookAuthor"/></td>
          <td><s:property value="bookPublish"/></td>
          <td><s:text name="format.date"><s:param value="bookDate"/></s:text></td>
```

```
            <td><s:property value="bookIsbn" /></td>
              <td><s:property value="bookPage" /></td>
            <td><s:property value="bookPrice"/></td>
            <td><s:property value="bookContent"/></td>
            <td><a href='<s:url action="edit"><s:param name="bookId" value="bookId" /></s:url>'>编辑
</a></td>
            <td><a href='<s:url action="delete"><s:param name="bookId" value="bookId" /></s:url>'>删
除</a></td>
      </tr>
    </s:iterator>
    <tr align="right">
     <td colspan="11">
          共<s:property value="totalRows"/>行 
          第<s:property value="currentPage"/>页 
          共<s:property value="pager.getTotalPages()"/>页 
          <a href="<s:url value="list.action">
               <s:param name="currentPage" value="currentPage"/>
               <s:param name="pagerMethod" value="'first'"/>
          </s:url>">首页</a>
          <a href="<s:url value="list.action">
               <s:param name="currentPage" value="currentPage"/>
               <s:param name="pagerMethod" value="'previous'"/>
          </s:url>">上一页</a>
          <a href="<s:url value="list.action">
               <s:param name="currentPage" value="currentPage"/>
               <s:param name="pagerMethod" value="'next'"/>
          </s:url>">下一页</a>
          <a href="<s:url value="list.action">
               <s:param name="currentPage" value="currentPage"/>
               <s:param name="pagerMethod" value="'last'"/>
          </s:url>">尾页</a>
     </td>
    </tr>
    </tbody>
</table>
</body>
</html>
```

◆ 4.5 修改书籍视图

当需要修改某条信息时，在列表页单击对应信息的【编辑】，系统将先查询并显示出该信息，在信息修改完之后单击【修改】即可将修改后的数据保存到数据库，并返回到列表页将所有的书籍信息显示出来。编辑页如图 7 所示：

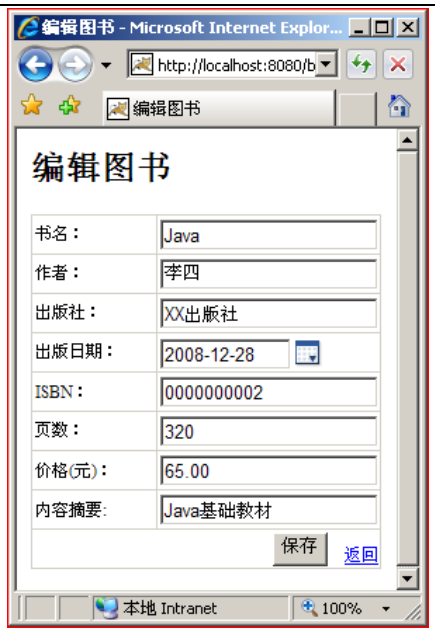


图 7 编辑图书信息效果图

页面主要代码如下：

```
<%@page pageEncoding="UTF-8" contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
    <head>
        <title>编辑图书</title>
        <!-- 为了下面时间的插件正常显示，此句不可少 -->
        <s:head theme="ajax"/>
        <style>
td {
    font-size: 12px
}
</style>
    </head>
    <body>
        <h2>
            <s:if test="null == book">
           增加图书
       </s:if>
            <s:else>
           编辑图书
       </s:else>
        </h2>
        <s:form name="editForm" action="save" validate="true" theme="simple">
            <table width="250" height="253" border="1" cellpadding="1"
                cellspacing="0" style="border-style: inherit"
                bordercolordark="#FFFFFF">
                <tr>
                    <td width="36%">书名：</td>
```

```
            <td><s:textfield name="book.bookName" /></td>
        </tr>
        <tr>
            <td>作者： </td>
            <td><s:textfield name="book.bookAuthor" /></td>
        </tr>
        <tr>
            <td>出版社： </td>
            <td><s:textfield name="book.bookPublish" /></td>
        </tr>
        <tr>
            <td>出版日期： </td>
            <td><s:datetimepicker name="book.bookDate"></s:datetimepicker>
            </td>
        </tr>
        <tr>
            <td>ISBN： </td>
            <td><s:textfield name="book.bookIsbn" /></td>
        </tr>
        <tr>
            <td>页数： </td>
            <td><s:textfield name="book.bookPage" /></td>
        </tr>
        <tr>
            <td>价格(元)： </td>
            <td><s:textfield name="book.bookPrice" /></td>
        </tr>
        <tr>
            <td>内容摘要: </td>
            <td><s:textfield name="book.bookContent" /></td>
        </tr>
        <tr>
            <td colspan="2" align="right">
                <s:if test="null == book">
                    <s:hidden name="book.bookId" value="%{bookId}" />
                </s:if>
                <s:else>
                    <s:hidden name="book.bookId" />
                </s:else>
                <s:hidden name="queryName" />
                <s:hidden name="queryValue" />
                <s:submit value="%{getText('保存')}" />  
                <a href="<s:url action="list"/>">返回</a>
            </td>
        </tr>
```

```
            </table>
        </s:form>
    </body>
</html>
```

细心的你应该已经发现该页其实还有肩负添加图书的责任，不同的是添加的时候没有数据存在。

◆ 4.6 图书查询视图

作为管理员要管理的书籍肯定很多，对于不计其数的书籍想要记住全部的信息是不太现实的，为了满足这个需求，需要提供强大的搜索功能。前面书籍列表页已经提到了，如图 8 所示：

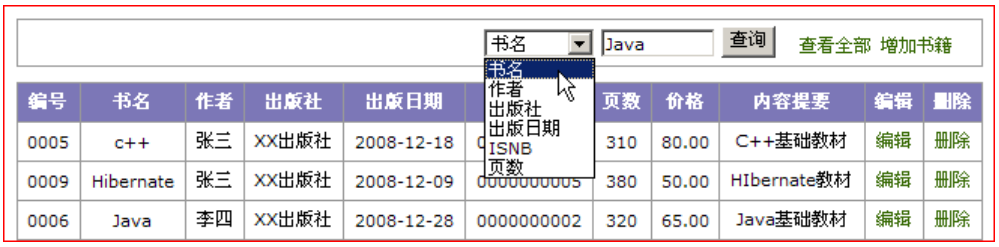


图 8 搜索页效果图

选择搜索类型、搜索关键字，单击【查询】，系统将返回数据库中保存的书籍结果。若查询书名为 Java，图 9 展示了搜索结果：

图 9 搜索结果效果图

13.5 代码清单

◆ 5.1 Action 的实现

AbstractAction.java

```
package com.ssh.commons;


import com.opensymphony.xwork2.ActionSupport;
//该类继承 ActionSupport 被作为超类使用
public class AbstractAction extends ActionSupport {
}
```

BookAction.java 该 Action 处理有关书籍的操作

```
package com.ssh.books.web.actions;
//省略导入的相关包
public class BooksAction extends AbstractAction {


    private BooksService booksService;
    private PagerService pagerService;
```

```
    private Books book;
    private Pager pager;

    protected Collection availableItems;
    protected String currentPage;
    protected String pagerMethod;
    protected String totalRows;
    protected String bookId;
    protected String queryName;
    protected String queryValue;
    protected String searchName;
    protected String searchValue;
    protected String queryMap;

    public String list() throws Exception {
        if (queryMap != null || !queryMap.equals("")) {
            String[] str = queryMap.split("~");
            this.setQueryName(str[0]);
            this.setQueryValue(str[1]);
        }

        int totalRow = booksService.getRows(this.getQueryName(), this
                .getQueryValue());
        pager = pagerService.getPager(this.getCurrentPage(), this
                .getPagerMethod(), totalRow);
        this.setCurrentPage(String.valueOf(pager.getCurrentPage()));
        this.setTotalRows(String.valueOf(totalRow));
        availableItems = booksService.getBooks(this.getQueryName(), this
                .getQueryValue(), pager.getPageSize(), pager.getStartRow());

        this.setQueryName(this.getQueryName());
        this.setQueryValue(this.getQueryValue());

        this.setSearchName(this.getQueryName());
        this.setSearchValue(this.getQueryValue());

        return SUCCESS;
    }

    public String load() throws Exception {
        if (bookId != null)
            book = booksService.getBook(bookId);
        else
            bookId = booksService.getMaxID();
        return SUCCESS;
```

```
    }

    public String save() throws Exception {
        if (this.getBook().getBookPrice().equals("")) {
            this.getBook().setBookPrice("0.0");
        }

        String id = this.getBook().getBookId();
        Books book = booksService.getBook(id);

        if (book == null)
            booksService.addBook(this.getBook());
        else
            booksService.updateBook(this.getBook());

        this.setQueryName(this.getQueryName());
        this.setQueryValue(this.getQueryValue());

        if (this.getQueryName() == null || this.getQueryValue() == null
                || this.getQueryName().equals("")
                || this.getQueryValue().equals("")) {
        } else {
            queryMap = this.getQueryName() + "~" + this.getQueryValue();
        }
        return SUCCESS;
    }

    public String delete() throws Exception {
        booksService.deleteBook(this.getBookId());

        if (this.getQueryName() == null || this.getQueryValue() == null
                || this.getQueryName().equals("")
                || this.getQueryValue().equals("")) {
        } else {
            queryMap = this.getQueryName() + "~" + this.getQueryValue();
        }
        return SUCCESS;
    }
    //省略相关属性的 setter/getter 方法
}
```

LoginAction.java

```
package com.ssh.books.web.actions;
//省略导入的相关包
public class LoginAction implements Action, ServletRequestAware {
    private User user;
```

```
    private UserService userService;
    HttpServletRequest request = null;

    public String execute() throws Exception {
        String retVal = ERROR;
        boolean isLogin = userService.isLogin(user);
        if (isLogin)
            retVal = SUCCESS;
        request.setAttribute("name", user.getName());
        return retVal;
    }
    //省略属性的 setter/getter 方法
}
```

◆ 5.2 Service 的实现

BooksServce.java

```
package com.ssh.books.services.iface;

import java.util.List;
import com.ssh.books.model.Books;

public interface BooksService {
    public List getAll();// 获得所有记录

    public List getBooks(int pageSize, int startRow);// 获得所有记录

    public int getRows();// 获得总行数

    public int getRows(String fieldname, String value);// 获得总行数

    public List queryBooks(String fieldname, String value);// 根据条件查询

    public List getBooks(String fieldname, String value, int pageSize,int startRow);// 根据条件查询

    public Books getBook(String bookId);// 根据 ID 获得记录

    public String getMaxID();// 获得最大 ID 值

    public void addBook(Books pd);// 添加记录

    public void updateBook(Books pd);// 修改记录

    public void deleteBook(String bookId);// 删除记录
}
```

UserService.java

```
package com.ssh.books.services.iface;

import com.ssh.books.model.User;

public interface UserService {
     public boolean isLogin(User user);
     public boolean isLogin(String name, String password);
}
```

◆ 5.3 ServiceImpl 的实现

BooksServiceImpl.java

```
package com.ssh.books.services;

import java.util.List;
import com.ssh.books.dao.iface.BooksDao;
import com.ssh.books.model.Books;
import com.ssh.books.services.iface.BooksService;

public class BooksServiceImpl implements BooksService{
     private BooksDao booksDao;
     public BooksServiceImpl(){}

     /**
      * 函数说明：添加信息
      * 参数说明：对象
      * 返回值：
      */
     public void addBook(Books book) {
          booksDao.addBook(book);
     }

     /**
      * 函数说明：删除信息
      * 参数说明： 对象
      * 返回值：
      */
     public void deleteBook(String bookId) {
          Books book=booksDao.getBook(bookId);
          booksDao.deleteBook(book);
     }

     /**
      * 函数说明：获得所有的信息
      * 参数说明：
```

```
     * 返回值：信息的集合
     */
    public List getAll() {
        return booksDao.getAll();
    }

    /**
     * 函数说明：获得总行数
     * 参数说明：
     * 返回值：总行数
     */
    public int getRows() {
        return booksDao.getRows();
    }

    /**
     * 函数说明：获得所有的信息
     * 参数说明：
     * 返回值：信息的集合
     */
    public List getBooks(int pageSize, int startRow) {
        return booksDao.getBooks(pageSize, startRow);
    }

    /**
     * 函数说明：获得一条的信息
     * 参数说明： ID
     * 返回值：对象
     */
    public Books getBook(String bookId) {
        return booksDao.getBook(bookId);
    }

    /**
     * 函数说明：获得最大 ID
     * 参数说明：
     * 返回值：最大 ID
     */
    public String getMaxID() {
        return booksDao.getMaxID();
    }

    /**
     * 函数说明：修改信息
     * 参数说明： 对象
```

```
     * 返回值:
     */
    public void updateBook(Books book) {
        booksDao.updateBook(book);
    }

    /**
     * 函数说明：查询信息
     * 参数说明： 集合
     * 返回值:
     */
    public List queryBooks(String fieldname,String value) {
        return booksDao.queryBooks(fieldname, value);
    }

    /**
     * 函数说明：获得总行数
     * 参数说明:
     * 返回值：总行数
     */
    public int getRows(String fieldname,String value) {
        return booksDao.getRows(fieldname, value);
    }

    /**
     * 函数说明：查询信息
     * 参数说明： 集合
     * 返回值:
     */
    public List getBooks(String fieldname,String value,int pageSize, int startRow) {
        return booksDao.getBooks(fieldname, value,pageSize,startRow);
    }

    public BooksDao getBooksDao() {
        return booksDao;
    }

    public void setBooksDao(BooksDao booksDao) {
        this.booksDao = booksDao;
    }
}
```

UserServiceImpl.java

```
package com.ssh.books.services;
```

```
import com.ssh.books.dao.iface.UserDao;
import com.ssh.books.model.User;
import com.ssh.books.services.iface.UserService;

public class UserServiceImpl implements UserService {
    private UserDao userDao;

    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }

    public boolean isLogin(User user) {
        boolean isLogin = false;
        try {
            User u = userDao.find(user.getName(), user.getPassword());
            if (u != null) {
                isLogin = true;
            }
        } catch (Exception e) {
            System.out.println("isLogin error\n" + e.getMessage());
        }
        return isLogin;
    }

    public boolean isLogin(String name, String password) {
        boolean isLogin = false;
        try {
            User u = userDao.find(name, password);
            if (u != null) {
                isLogin = true;
            }
        } catch (Exception e) {
            System.out.println("isLogin error\n" + e.getMessage());
        }
        return isLogin;
    }
}
```

◆ 5.4 Dao 的实现

BooksDao.java

```
package com.ssh.books.dao.iface;

import java.util.List;
import com.ssh.books.model.Books;
```

```
public interface BooksDao {
    public List getAll();// 获得所有记录

    public List getBooks(int pageSize, int startRow);// 获得所有记录

    public int getRows();// 获得总行数

    public int getRows(String fieldname, String value);// 获得总行数

    public List queryBooks(String fieldname, String value);// 根据条件查询

    public List getBooks(String fieldname, String value, int pageSize,
            int startRow);// 根据条件查询

    public Books getBook(String bookId);// 根据 ID 获得记录

    public String getMaxID();// 获得最大 ID 值

    public void addBook(Books book);// 添加记录

    public void updateBook(Books book);// 修改记录

    public void deleteBook(Books book);// 删除记录
}
```

UserDao.java

```
package com.ssh.books.dao.iface;

import com.ssh.books.model.User;

public interface UserDao {
    public User find(String name, String password);
}
```

◆ 5.5 DaoImpl 的实现

BooksDaoImpl.java

```
package com.ssh.books.dao.hibernate;

import java.sql.SQLException;
import java.util.Iterator;
import java.util.List;
import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.springframework.orm.hibernate3.HibernateCallback;
```

```
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import com.ssh.books.dao.iface.BooksDao;
import com.ssh.books.model.Books;
import com.ssh.commons.PublicUtil;


/**
 * @author cwf
 *
 */
public class BooksDaoImpl extends HibernateDaoSupport implements BooksDao {

    public BooksDaoImpl(){}

    /**
     * 函数说明：添加信息
     * 参数说明：对象
     * 返回值：
     */
    public void addBook(Books book) {
        this.getHibernateTemplate().save(book);
    }

    /**
     * 函数说明：删除信息
     * 参数说明： 对象
     * 返回值：
     */
    public void deleteBook(Books book) {
        this.getHibernateTemplate().delete(book);
    }

    /**
     * 函数说明：获得所有的信息
     * 参数说明：
     * 返回值：信息的集合
     */
    public List getAll() {
        String sql="FROM Books ORDER BY bookName";
        return this.getHibernateTemplate().find(sql);
    }

    /**
     * 函数说明：获得总行数
     * 参数说明：
```

```
     * 返回值：总行数
     */
    public int getRows() {
        String sql="FROM Books ORDER BY bookName";
        List list=this.getHibernateTemplate().find(sql);
        return list.size();
    }

    /**
     * 函数说明：获得所有的信息
     * 参数说明：
     * 返回值：信息的集合
     */
    public List getBooks(int pageSize, int startRow) throws HibernateException {
        final int pageSize1=pageSize;
        final int startRow1=startRow;
        return this.getHibernateTemplate().executeFind(new HibernateCallback(){

            public List doInHibernate(Session session) throws HibernateException, SQLException {
                // TODO 自动生成方法存根
                Query query=session.createQuery("FROM Books ORDER BY bookName");
                query.setFirstResult(startRow1);
                query.setMaxResults(pageSize1);
                return query.list();
            }
        });
    }

    /**
     * 函数说明：获得一条的信息
     * 参数说明： ID
     * 返回值：对象
     */
    public Books getBook(String bookId) {
        return (Books)this.getHibernateTemplate().get(Books.class,bookId);
    }

    /**
     * 函数说明：获得最大 ID
     * 参数说明：
     * 返回值：最大 ID
     */
    public String getMaxID() {
        String date=PublicUtil.getStrNowDate();
        String sql="SELECT MAX(bookId)+1 FROM Books  ";
```

```
        String noStr = null;
        List ll = (List) this.getHibernateTemplate().find(sql);
        Iterator itr = ll.iterator();
        if (itr.hasNext()) {
            Object noint = itr.next();
          if(noint == null){
            noStr = "1";
          }else{
            noStr = noint.toString();
          }
        }

        if(noStr.length()==1){
            noStr="000"+noStr;
        }else if(noStr.length()==2){
            noStr="00"+noStr;
        }else if(noStr.length()==3){
            noStr="0"+noStr;
        }else{
            noStr=noStr;
        }
        return noStr;
    }

    /**
     * 函数说明：修改信息
     * 参数说明： 对象
     * 返回值：
     */
    public void updateBook(Books pd) {
        this.getHibernateTemplate().update(pd);
    }

    /**
     * 函数说明：查询信息
     * 参数说明： 集合
     * 返回值：
     */
    public List queryBooks(String fieldname,String value) {
        String sql="FROM Books where "+fieldname+" like '%"+value+"%'"+"ORDER BY bookName";
        return this.getHibernateTemplate().find(sql);
    }

    /**
     * 函数说明：获得总行数
```

```
     * 参数说明：
     * 返回值：总行数
     */
    public int getRows(String fieldname,String value) {
        String sql="";
        if(fieldname==null||fieldname.equals("")||fieldname==null||fieldname.equals(""))
            sql="FROM Books ORDER BY bookName";
        else
            sql="FROM Books where "+fieldname+" like '%"+value+"%'"+"ORDER BY bookName";
        List list=this.getHibernateTemplate().find(sql);
        return list.size();
    }

    /**
     * 函数说明：查询信息
     * 参数说明： 集合
     * 返回值：
     */
    public List getBooks(String fieldname,String value,int pageSize, int startRow) {
        final int pageSize1=pageSize;
        final int startRow1=startRow;
        final String queryName=fieldname;
        final String queryValue=value;
        String sql="";

        if(queryName==null||queryName.equals("")||queryValue==null||queryValue.equals(""))
            sql="FROM Books ORDER BY bookName";
        else
            sql="FROM Books where "+fieldname+" like '%"+value+"%'"+"ORDER BY bookName";

        final String sql1=sql;
        return this.getHibernateTemplate().executeFind(new HibernateCallback(){

            public List doInHibernate(Session session) throws HibernateException, SQLException {
                // TODO 自动生成方法存根
                Query query=session.createQuery(sql1);
                query.setFirstResult(startRow1);
                query.setMaxResults(pageSize1);
                return query.list();
            }
        });
    }
}
```

UserDaoImpl.java

```
package com.ssh.books.dao.hibernate;

import java.util.List;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import com.ssh.books.dao.iface.UserDao;
import com.ssh.books.model.User;

public class UserDaoImpl extends HibernateDaoSupport implements UserDao {

    public User find(String name, String password) {

        List<User> list = this.getHibernateTemplate().findByNamedParam(
                "FROM User AS u WHERE u.name =:name AND u.password =:password",
                new String[] { "name", "password" },
                new String[] { name, password });

        if (list.size() == 0)
            return null;
        else
            return list.get(0);
    }
}
```

◆ 5.6 其他功能分析

Pager.java 分页的 JavaBean

```
package com.ssh.commons;

public class Pager {
    private int totalRows; //总行数
    private int pageSize = 5; //每页显示的行数
    private int currentPage; //当前页号
    private int totalPages; //总页数
    private int startRow; //当前页在数据库中的起始行

    public Pager() {
    }

    public Pager(int _totalRows) {
        totalRows = _totalRows;
        totalPages=totalRows/pageSize;
        int mod=totalRows%pageSize;
        if(mod>0){
            totalPages++;
        }
        currentPage = 1;
```

```
        startRow = 0;
    }

    public int getStartRow() {
        return startRow;
    }
    public int getTotalPages() {
        return totalPages;
    }
    public int getCurrentPage() {
        return currentPage;
    }
    public int getPageSize() {
        return pageSize;
    }
    public void setTotalRows(int totalRows) {
        this.totalRows = totalRows;
    }
    public void setStartRow(int startRow) {
        this.startRow = startRow;
    }
    public void setTotalPages(int totalPages) {
        this.totalPages = totalPages;
    }
    public void setCurrentPage(int currentPage) {
        this.currentPage = currentPage;
    }
    public void setPageSize(int pageSize) {
        this.pageSize = pageSize;
    }
    public int getTotalRows() {
        return totalRows;
    }
    public void first() {
        currentPage = 1;
        startRow = 0;
    }
    public void previous() {
        if (currentPage == 1) {
            return;
        }
        currentPage--;
        startRow = (currentPage - 1) * pageSize;
    }
    public void next() {
```

```
            if (currentPage < totalPages) {
                currentPage++;
            }
            startRow = (currentPage - 1) * pageSize;
        }
        public void last() {
            currentPage = totalPages;
            startRow = (currentPage - 1) * pageSize;
        }
        public void refresh(int _currentPage) {
            currentPage = _currentPage;
            if (currentPage > totalPages) {
                last();
            }
        }
}
```

PagerService.java

```
    package com.ssh.commons;

public class PagerService {
    public Pager getPager(String currentPage, String pagerMethod, int totalRows) {
        // 定义 pager 对象，用于传到页面
        Pager pager = new Pager(totalRows);
        // 如果当前页号为空，表示为首次查询该页
        // 如果不为空，则刷新 pager 对象，输入当前页号等信息
        if (currentPage != null) {
            pager.refresh(Integer.parseInt(currentPage));
        }
        // 获取当前执行的方法，首页，前一页，后一页，尾页。
        if (pagerMethod != null) {
            if (pagerMethod.equals("first")) {
                pager.first();
            } else if (pagerMethod.equals("previous")) {
                pager.previous();
            } else if (pagerMethod.equals("next")) {
                pager.next();
            } else if (pagerMethod.equals("last")) {
                pager.last();
            }
        }
        return pager;
    }
}
```

Struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
    <constant name="struts.enable.DynamicMethodInvocation"
        value="false" />
    <constant name="struts.devMode" value="true" />
    <constant name="struts.i18n.encoding" value="UTF-8" />

    <!-- Add packages here -->
    <package name="login" extends="struts-default">
        <action name="Login" class="loginAction">
            <result name="success">/index.jsp</result>
            <result name="error">/error.jsp</result>
        </action>
    </package>
    <package name="products" extends="struts-default">
        <!--default-interceptor-ref name="validation"/-->
        <!-- Add actions here -->
        <action name="list" class="bookAction" method="list">
            <result>/list.jsp</result>
        </action>

        <action name="delete" class="bookAction" method="delete">
            <result type="redirect">
                list.action?queryMap=${queryMap}
            </result>
        </action>

        <action name="*" class="com.ssh.commons.AbstractAction">
            <result>/{1}.jsp</result>
        </action>

        <action name="edit" class="bookAction" method="load">
            <result>/editBook.jsp</result>
        </action>

        <action name="save" class="bookAction" method="save">
            <interceptor-ref name="params" />
            <interceptor-ref name="validation" />
            <result name="input">/editBook.jsp</result>
            <result type="redirect">
```

```
                list.action?queryMap=${queryMap}
            </result>
        </action>
    </package>
</struts>
```

applicationContext.xml

```
    <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
    <!-- dataSource config -->
    <bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url"
            value="jdbc:mysql://localhost:3306/bookshell" />
        <property name="username" value="root" />
        <property name="password" value="root" />
    </bean>
    <bean id="exampleHibernateProperties"
        class="org.springframework.beans.factory.config.PropertiesFactoryBean">
        <property name="properties">
            <props>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.query.substitutions">
                    true 'T', false 'F'
                </prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.c3p0.minPoolSize">5</prop>
                <prop key="hibernate.c3p0.maxPoolSize">20</prop>
                <prop key="hibernate.c3p0.timeout">600</prop>
                <prop key="hibernate.c3p0.max_statement">50</prop>
                <prop key="hibernate.c3p0.testConnectionOnCheckout">
                    false
                </prop>
            </props>
        </property>
    </bean>
    <!-- Hibernate SessionFactory -->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="dataSource">
```

```
            <ref bean="dataSource" />
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
                <prop key="show_sql">true</prop>
            </props>
        </property>
        <property name="mappingResources">
            <list>
                <value>com/ssh/books/model/Books.hbm.xml</value>
                <value>com/ssh/books/model/User.hbm.xml</value>

            </list>
        </property>
    </bean>
    <!-- TransactionManager  声明式事务定义-->
    <bean id="transactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
    </bean>

    <bean id="transactionInterceptor"
        class="org.springframework.transaction.interceptor.TransactionInterceptor">
        <property name="transactionManager" ref="transactionManager" />
        <property name="transactionAttributes">
            <props>
                <prop key="isLogin*">
                    PROPAGATION_REQUIRED,readOnly
                </prop>
                <prop key="update*">PROPAGATION_REQUIRED</prop>
                <prop key="add*">PROPAGATION_REQUIRED</prop>
                <prop key="delete*">PROPAGATION_REQUIRED</prop>
            </props>
        </property>
    </bean>
    <bean id="proxyUserService"
        class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
        <property name="beanNames">
            <list>
                <value>userService</value>
```

```
                <value>bookService</value>
            </list>
        </property>
        <property name="interceptorNames"
            value="transactionInterceptor" />
    </bean>
    <bean id="bookAction" class="com.ssh.books.web.actions.BooksAction"
        singleton="false">
        <property name="booksService">
            <ref bean="booksService" />
        </property>
        <property name="pagerService">
            <ref bean="pagerService" />
        </property>
    </bean>
    <!-- Services -->
    <bean id="booksService"
        class="com.ssh.books.services.BooksServiceImpl">
        <property name="booksDao">
            <ref bean="booksDao" />
        </property>
    </bean>
    <!-- DAO -->
    <bean id="booksDao"
        class="com.ssh.books.dao.hibernate.BooksDaoImpl">
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
    </bean>
    <bean id="pagerService" class="com.ssh.commons.PagerService" />
    <!-- view -->
    <bean id="loginAction"
        class="com.ssh.books.web.actions.LoginAction">
        <property name="userService" ref="userService" />
    </bean>
    <!-- Services -->
    <bean id="userService"
        class="com.ssh.books.services.UserServiceImpl">
        <property name="userDao" ref="userDao" />
    </bean>
    <!-- DAO -->
    <bean id="userDao"
        class="com.ssh.books.dao.hibernate.UserDaoImpl">
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>
```

```
</beans>
```

更详细的代码请参看程序源代码，这里不再一一列出。

13.6 代码树形图

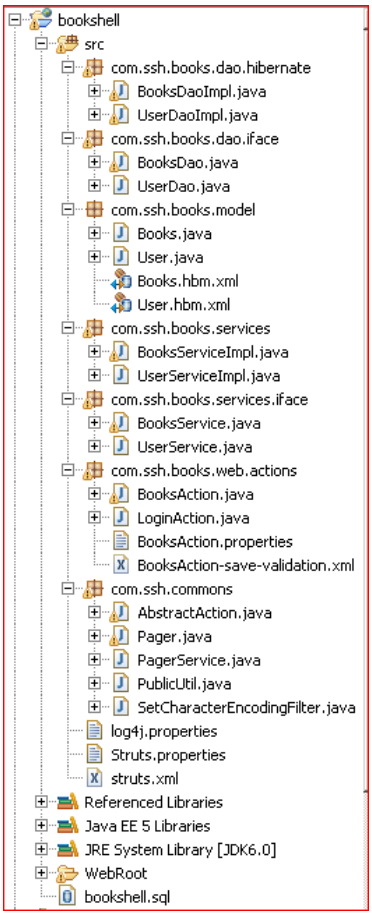


图 10 代码树形效果图

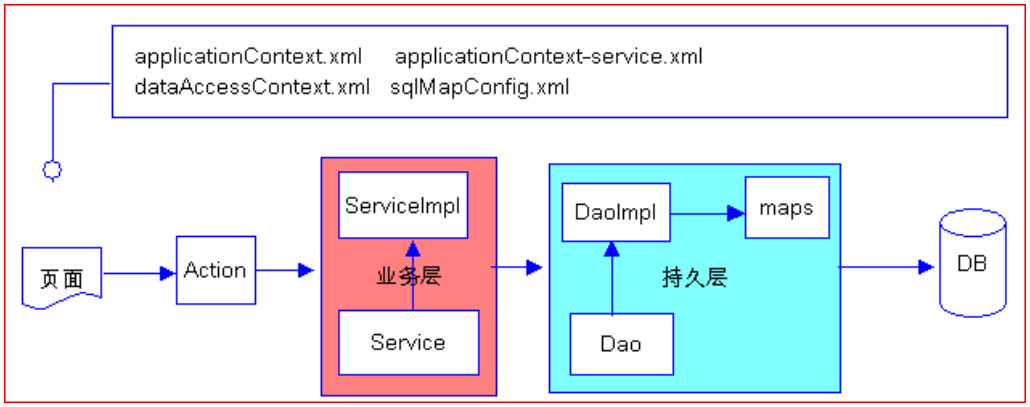
第 14 章 集成 Spring

第 15 章 集成 ibatis

iBatis 是又一个 O/R Mapping 解决方案，j2ee 的 O/R 方案真是多，和 Hibernate 相比，iBatis 最大的特点就是小巧，上手很快。如果你不需要太多复杂的功能，iBatis 是能满足你的要求又足够灵活的最简单的解决方案。

iBatis 最大的特点是简单，最新版本 2.0，只要有 SQL 基础，这一章节即便没有 iBatis 基础也可

以阅读。有关 struts2、spring、ibatis 的集成方案如下图所示：



当服务容器启动的时候，会根据 web.xml 中配置的 spring 路径，初始化 spring 环境。Spring 会加载配置的 ibaits 环境。

ibstis 的配置主要由两种文件：

1、有关项目的总体配置，如连接的数据源，连接池，缓存等的配置，也即 sqlmapconfig.xml 文件的配置。

2、sqlmap.xml 文件的配置，也即对象与表的操作映射的配置。

下面分别来讲述这两个配置文件的内容。

➢ sqlmapconfig.xml 首先查找 spring 配置文件，查找 sqlmapconfig.xml 文件的位置

```
<bean id="sqlMapClient"
        class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
        <property name="configLocation"
            value="WEB-INF/classes/com/xmh/config/sqlMapConfig.xml" />
        <property name="dataSource" ref="dataSource" />
    </bean>
```

SqlMapClient 对于 ibatis 的意义类似于 Session 与 Hibernate 以及 Connection 与 JDBC，这里的 sqlMapClient 节点实际上配置了一个 sqlMapClient 的创建工厂类。configLocation 属性配置了 ibatis 映射文件的名称。

sqlMapConfig.xml 文件包括了 DataSource 的详细配置信息。可以把相关的数据库配置信息写在属性文件中，然后通过<properties resource="路径/SqlMapConfig.properties"/>来获取配置文件。这样做在属性文件中定义的属性可以作为变量在 SqlMap 配置文件中以及多包含的 SqlMap 映射文件中使用。

通过 Settings 元素优化 SqlMapClient 实例的各选项比如：cacheModelsEnabled 属性是否启用 SqlMapClient 的 model 的缓存处理，enhancementEnabled 属性，是否运行时增强字节码，lazyLoadingEnabled 属性是否启用所有的延迟加载，调试程序时使用，maxRequests 属性同时执行 SQL 语句的最大线程数，大于这个值的线程将阻塞直到另一个线程执行结束，不同的 DBMS 有不同的限定值，减少这个参数值能提高性能，通常是 maxTransations 数值的 10 倍，maxTransations 属性，同时进入 SqlMapClient.startTransaction()的最大线程数，大于这个线程的数值将被阻塞，直到另一个线程的结束。不同的 DBMS 有不同的限制，这个参数值应该小于或等于 maxSessions 数值 -->

```
<settings
cacheModelsEnabled="true"
enhancementEnabled="true"
lazyLoadingEnabled="true"
maxSessions="64"
maxTransactions="12"
maxRequests="128"
/>
```

最后为 SqlMap 配置事务管理服务，type 用来指定事务管理的类型，这个属性值可以是一个类名，也可以是一个别名。包含在框架中的事务管理器分别是 JDBC、JTA、EXTERNAL。

```
<transactionManager type="JDBC">
<!-- datasource 元素为 SqlMap 数据源配置一系列的属性信息 -->
<datasource type="JNDI">
   <property name="DBJNDIContext" value="java:comp/env/jdbc/webpublish"/>
</datasource>
</transactionManager>
<sqlMap resource="路径/映射文件"/>
</sqlMapConfig>
```

因为数据源的配置与事务的配置均由 spring 负责，所以在 spring 与组合的框架中，sqlMapConfig.xml 文件的内容被简化，只需要设置 sqlMap 即可。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMapConfig PUBLIC "-//iBATIS.com//DTD SQL Map Config 2.0//EN"
    "http://www.ibatis.com/dtd/sql-map-config-2.dtd">
<sqlMapConfig>
    <sqlMap resource="com/xmh/dao/impl/maps/SBook.xml"/>
</sqlMapConfig>
```

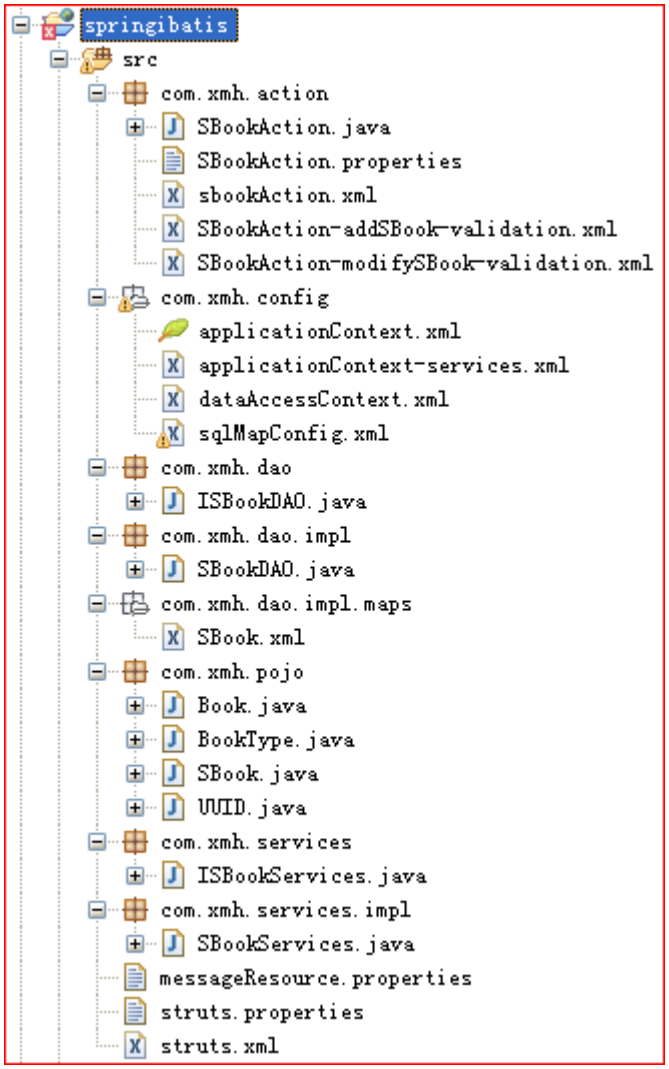
- 映射文件，亦即 sql 语句与对象间的桥梁文件。

```
    <insert id="saveBook" parameterClass="sbook">
        <selectKey keyProperty="id" resultClass="int">
            <![CDATA[
                SELECT LAST_INSERT_ID() AS VALUE
            ]]>
        </selectKey>
        <![CDATA[
            INSERT INTO sbook(title,author,total,price,isbn,publisher)
            VALUES(#title#,#author#,#total#,#price#,#isbn#,#publisher#)
        ]]>
    </insert>
```

如果对上述内容还有疑义，请参考相关的 ibatis 文档。

下面通过一个简单的 CRUD 示例来了解 spring、struts2、ibatis 的组合运用。

项目的布局如下：



➢ 持久层代码分析

首先从 maps 下的 Sbook.xml 文件入手，代码清单如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMap
    PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-2.dtd">
<sqlMap>
    <typeAlias alias="sbook" type="com.xmh.pojo.SBook" />
    <!-- 添加一本新书 -->
    <insert id="saveBook" parameterClass="sbook">
        <selectKey keyProperty="id" resultClass="int">
            <![CDATA[
                SELECT LAST_INSERT_ID() AS VALUE
            ]]>
        </selectKey>
        <![CDATA[
            INSERT INTO sbook(title,author,total,price,isbn,publisher)
            VALUES(#title#,#author#,#total#,#price#,#isbn#,#publisher#)
        ]]>
    </insert>
```

```
    <!-- 删除图书 -->
    <delete id="deleteBook" parameterClass="int">
        <![CDATA[
            DELETE FROM SBOOK WHERE ID=#id#
        ]]>
    </delete>

    <!-- 通过出版社名称查找此出版社出版的图书  -->
    <select id="findBookByPublisher" parameterClass="string"
        resultClass="sbook">
        <![CDATA[
            SELECT * FROM sbook WHERE publisher=#publisher#
        ]]>
    </select>
    <!-- 通过图书唯一的 ISBN 号码查找图书 -->
    <select id="findBookByISBN" parameterClass="string"
        resultClass="sbook">
        <![CDATA[
            SELECT * FROM sbook WHERE isbn=#isbn#
        ]]>
    </select>
    <!-- 查找所有的图书 -->
    <select id="findAllBook" resultClass="sbook">
        <![CDATA[
            SELECT * FROM sbook
        ]]>
    </select>
    <!-- 更新图书信息 -->
    <update id="updateBook" parameterClass="sbook">
        <![CDATA[
            UPDATE SBOOK SET

    title=#title#,author=#author#,price=#price#,total=#total#,isbn=#isbn#,publisher=#
publisher#
            WHERE id=#id#
        ]]>
    </update>
    <!-- 查找特定图书 -->
    <select id="findBookById" parameterClass="int" resultClass="sbook">
        <![CDATA[
            SELECT * FROM sbook WHERE ID=#id#
        ]]>
    </select>
</sqlMap>
```

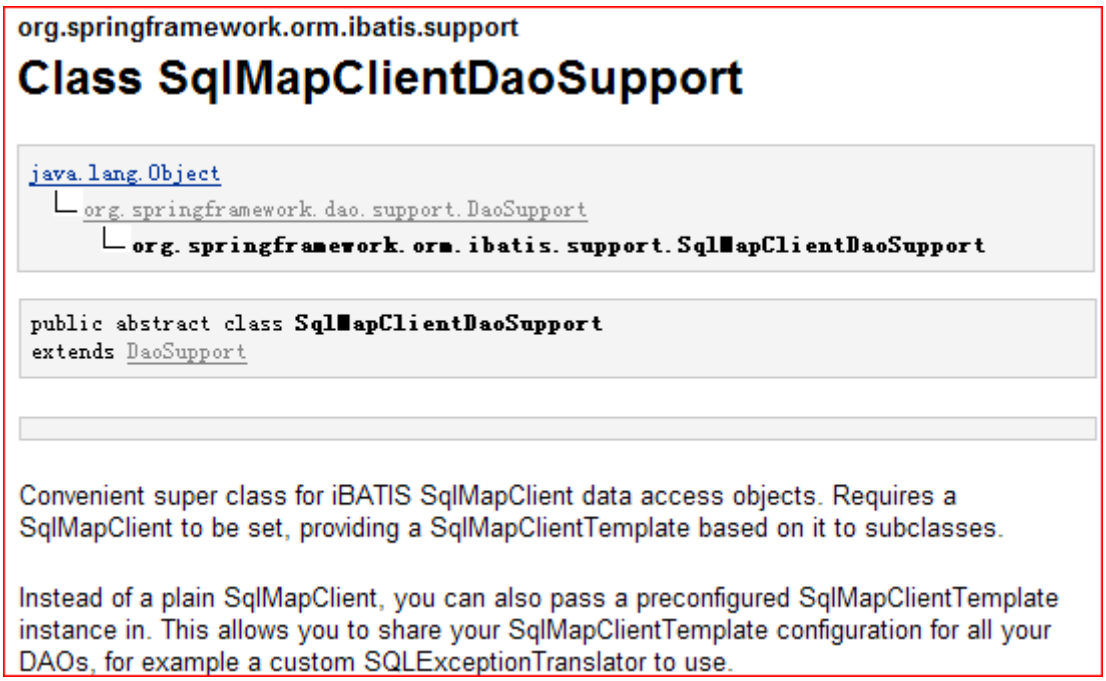
持久化层的核心代码如下：

```
package com.xmh.dao.impl;
……
public class SBookDAO extends SqlMapClientDaoSupport implements ISBookDAO {
    public SBook findBookByISBN(String isbn) throws RuntimeException {
        return (SBook) this.getSqlMapClientTemplate().queryForObject(
                "findBookByISBN", isbn);
```

```
    }

    @SuppressWarnings("unchecked")
    public List<SBook> findBooksByPublisher(String publisher)
            throws RuntimeException {
        return this.getSqlMapClientTemplate().queryForList(
                "findBookByPublisher", publisher);
    }

    public void saveBook(SBook book) throws RuntimeException {
        this.getSqlMapClientTemplate().insert("saveBook", book);
    }

    public void deleteBook(int id) throws RuntimeException {
        this.getSqlMapClientTemplate().delete("deleteBook", id);
    }

    public void updateBook(SBook book) throws RuntimeException {
        this.getSqlMapClientTemplate().update("updateBook", book);
    }

    @SuppressWarnings("unchecked")
    public List findAllBook() throws RuntimeException {
        return this.getSqlMapClientTemplate().queryForList("findAllBook");
    }

    public SBook findBookById(int id) throws RuntimeException {
        return (SBook) this.getSqlMapClientTemplate().queryForObject(
                "findBookById", id);
    }
}
```

SqlMapClientDaoSupport 这个类为 Spring 的 ibatis 模版类。它提供很多模版方法，并且 Spring 提供了异常处理，使用比较方便。

SqlMapClientDaoSupport 的 api 文档截图如下：



org.springframework.orm.ibatis.support

Class SqlMapClientDaoSupport

java.lang.Object
  └ org.springframework.dao.support.DaoSupport
    └ **org.springframework.orm.ibatis.support.SqlMapClientDaoSupport**

```
public abstract class SqlMapClientDaoSupport
extends DaoSupport
```

Convenient super class for iBATIS SqlMapClient data access objects. Requires a SqlMapClient to be set, providing a SqlMapClientTemplate based on it to subclasses.

Instead of a plain SqlMapClient, you can also pass a preconfigured SqlMapClientTemplate instance in. This allows you to share your SqlMapClientTemplate configuration for all your DAOs, for example a custom SQLExceptionTranslator to use.

从文档中可以看出使用这个类必须要为其注入 SqlMapClient，所以在 spring 的配置文件中，需要如下设置。

```
    <!-- 注入 BookDAO 层 -->
    <bean id="sbookDAO" class="com.xmh.dao.impl.SBookDAO">
        <property name="sqlMapClient" ref="sqlMapClient" />
    </bean>
```

- 业务层调用持久层，业务层的代码如下：

```
package com.xmh.services.impl;
……
public class SBookServices implements ISBookServices {
    private ISBookDAO sbookDAO;

    @SuppressWarnings("unchecked")
    public List getAllBook() throws RuntimeException {
        return sbookDAO.findAllBook();
    }

    public SBook getBookByISBN(String isbn) throws RuntimeException {
        return sbookDAO.findBookByISBN(isbn);
    }

    public List<SBook> getBooksByPublisher(String publisher)
            throws RuntimeException {
        return sbookDAO.findBooksByPublisher(publisher);
    }

    ……
    public void setSbookDAO(ISBookDAO sbookDAO) {
        this.sbookDAO = sbookDAO;
    }
```

```
}
```

Spring 容器为其注入持久层对象，所以 spring 的配置文件中需要设置：

```
<bean id="sbookServices" class="com.xmh.services.impl.SBookServices">
        <property name="sbookDAO" ref="sbookDAO"/>
</bean>
```

➢ Strtus2 的控制层调用业务层，控制层的代码如下：

```
package com.xmh.action;
……
public class SBookAction extends ActionSupport{
    private static final long serialVersionUID = 1L;
    private ISBookServices sbookServices;
    private SBook sbook;
    private String tips;
    private String bookId;
    @SuppressWarnings("unchecked")
    private List bookList;
    ……

    public void setSbookServices(ISBookServices sbookServices) {
        this.sbookServices = sbookServices;
    }

    @SuppressWarnings("unchecked")
    public List getBookList() {
        return bookList;
    }
}
```

在整合的框架之下，struts2 中的 Action 是交由容器管理的。如下代码所示：

```
<bean id="sbookAction" class="com.xmh.action.SBookAction">
        <property name="sbookServices" ref="sbookServices"/>
</bean>
```

在 struts2 的配置文件中，action 就不再需要完整的路径。

```
<action name="addSBook" class="sbookAction" method="addSBook">
            <result name="success">/addBook.jsp</result>
            <result name="input">/addBook.jsp</result>
            <result name="error">/addBook.jsp</result>
        </action>
```

➢ 事务处理的方式：

pring 通过 AOP 来拦截方法的调用，从而在这些方法上面添加声明式事务处理的能力。典型配置如下：

```
   <!-- 配置事务特性 -->
    <tx:advice id="txAdvice" transaction-manager="事务管理器名称">
        <tx:attributes>
           <tx:method name="add*" propagation="REQUIRED"/>
           <tx:method name="del*" propagation="REQUIRED"/>
           <tx:method name="update*" propagation="REQUIRED"/>
           <tx:method name="*" read-only="true"/>
       </tx:attributes>
    </tx:advice>
```

配置哪些类的方法需要进行事务管理

```
    <aop:config>
        <aop:pointcut id="allManagerMethod" expression="execution(*
 com.ibatis.manager.*.*(..))"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="allManagerMethod"/>
    </aop:config>
```

下面对 execution 进行说明：

```
execution(modifier-pattern?
          ret-type-pattern
          declaring-type-pattern?
          name-pattern(param-pattern)
          throws-pattern?)
```

括号中各个 pattern 分别表示修饰符匹配（modifier-pattern?）、返回值匹配（ret-type-pattern）、类路径匹配（declaring-type-pattern?）、方法名匹配（name-pattern）、参数匹配（(param-pattern)）、异常类型匹配（throws-pattern?），其中后面跟着“?”的是可选项。

在各个 pattern 中可以使用“*”来表示匹配所有。在(param-pattern)中，可以指定具体的参数类型，多个参数间用“,”隔开，各个也可以用“*”来表示匹配任意类型的参数，如(String)表示匹配一个 String 参数的方法；(*,String)表示匹配有两个参数的方法，第一个参数可以是任意类型，而第二个参数是 String 类型；可以用(..)表示零个或多个任意参数。

现在来看看几个例子：

1）execution(* *(..))表示匹配所有方法

2）execution(public * com. savage.service.UserService.*(..)) 表示匹配 com.savage.server.UserService 中所有的公有方法

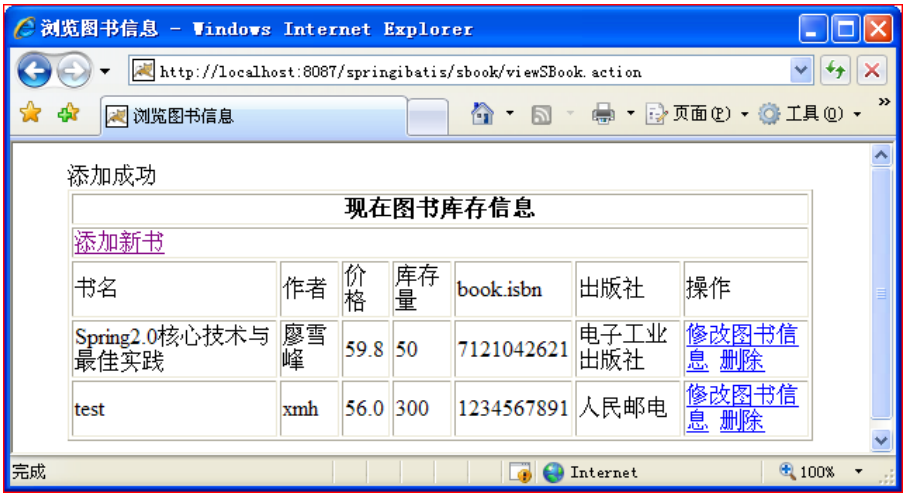
3）execution(* com.savage.server..*.*(..))表示匹配 com.savage.server 包及其子包下的所有方法

注意：这些事务都是声明在业务逻辑层的对象上的。

创建一个事务管理器，对事务进行管理。

```
    <bean id="txManager" class="包名.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
    </bean>
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost/数据库名"/>
        <property name="username" value="用户名"/>
        <property name="password" value="密码"/>
    </bean>
```

其它部分代码请参见本书源代码部分，部署此工程，输入相应的访问 URL，将会获得如下图所示的效果。



第 16 章 集成 Jquery

16.1 什么是Jquery

jQuery 是一个 JavaScript 库，它有助于简化 JavaScript™ 以及 Asynchronous JavaScript + XML (Ajax) 编程。与类似的 JavaScript 库不同，jQuery 具有独特的基本原理，可以简洁地表示常见的复杂代码。学习 jQuery 基本原理，探索其特性和功能，执行一些常见的 Ajax 任务并掌握如何使用插件扩展 jQuery。

jQuery 由 John Resig 创建于 2006 年初，对于任何使用 JavaScript 代码的程序员来说，它是一个非常有用的 JavaScript 库。无论您是刚刚接触 JavaScript 语言，并且希望获得一个能解决文档对象模型（Document Object Model，DOM）脚本和 Ajax 开发中一些复杂问题的库，还是作为一个厌倦了 DOM 脚本和 Ajax 开发中无聊的重复工作的资深 JavaScript 专家，jQuery 都会是您的首选。

jQuery 能帮助您保证代码简洁易读。您再也不必编写大堆重复的循环代码和 DOM 脚本库调用了。使用 jQuery，您可以把握问题的要点，并使用尽可能最少的代码实现您想要的功能。

jQuery 可以跨浏览器进行 DOM 操作、事件处理、样式更换和外部通信。它超轻巧（20KB），并且执行速度超快，所以赢得了众多 JS 开发者的青睐，至少，在我刚完成的一个彩票投注系统中就大量的应用到 jQuery，也使得我有机会在此把工作中的应用体会补充在本文档之中。

16.2 Jquery操作CSS

Jquery 对 css 的操作相当方便，能很方便我们去通过 js 修改 css。传统 javascript 对 css 的操作相当繁琐，比如

```
<div id="a" style="background:blue">css</div>
```

取它的 background 语法是：

```
document.getElementById("a").style.background
```

而 jQuery 对 css 更方便的操作：

```
$("#a").background();
$("#a").background( “red” )
```

说明如下

$("#a")得到 jQuery 对象[<div id="a" … /div>]

$("#a").background()将取出该对象的 background 样式。

$("#a").background("red")将该对象的 background 样式设为 red。

jQuery 提供了以下方法，来操作 css:

background ()	width()
background (val)	width(val)
color()	left()
color(val)	left(val)
css(name)	overflow()
css(prop)	overflow(val)
css(key, value)	position()
float()	position(val)
float(val)	top()
height()	top(val)
height(val)	

这里需要讲解一下 css(name);css(prop);css(key, value)，其他的看名字都知道什么作用了！

```
<div id="a" style="background:blue; color:red">css</div><P id="b">test</P>
```

css(name):获取样式名为 name 的样式

$("#a").css("color"):将得到样式中 color 值 red，("#a").css("background ")将得到 blue

css(prop):prop 是一个 hash 对象，用于设置大量的 css 样式

$("#b").css({ color: "red", background: "blue" });

最终效果是

test

{ color: "red", background: "blue" }，hash 对象，color 为 key，"red"为 value，

css(key, value) 用于设置一个单独得 css 样式

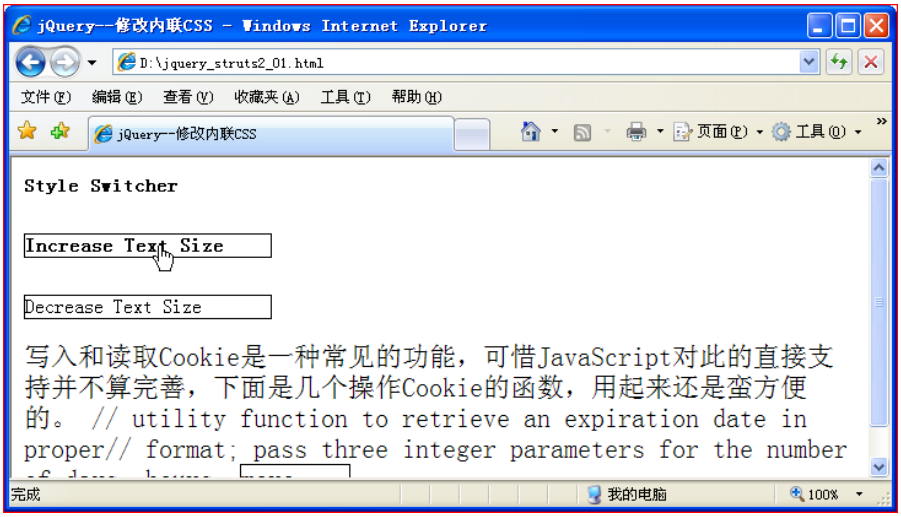
$("#b").css("color","red");

最终效果是

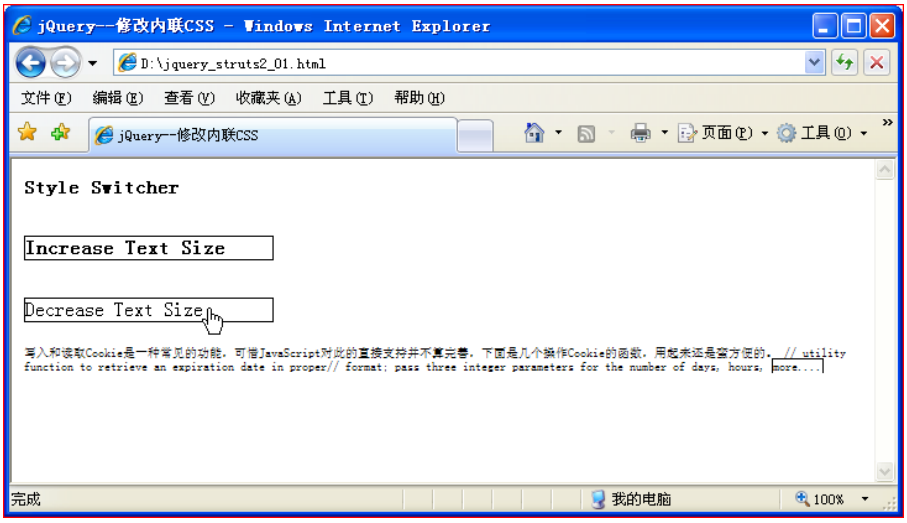
test

下面通过用 jquery 来控制文字大小来学习 jquery 对 css 的操控：

单击上面的按钮，文字变大：



单击下面的按钮，文字变小：



代码如下所示：

```
<script language="javascript">
$(document).ready(function(){
    $('div.button').click(function(){
       //增大或减小字体
      var $speech=$('div.speech');
      var currentSize=$speech.css('fontSize');
      //alert(currentSize);
      var num=parseFloat(currentSize,10);
      //alert(num);//这是返回的是数字
      var unit=currentSize.slice(-2);
      //alert(unit);//这里反回的是 PX..
         if(this.id=='switcher-large'){
             num*=1.5;
          }else if(this.id=='switcher-small'){
             num/=1.5;
          }

      $speech.css('fontSize',num+unit);//num+unit 就等价于数字+px(例如 22px)
      });
    });
</script>
```

详细代码请参看源码部分。

16.3 Jquery操作DOM

在 HTML 中是经常可见的代码。在原始的 javascript 里面可以用 var o=document.getElementById('a')取的 id 为 a 的节点对象，在用 o.src 来取得或修改该节点的 scr 属性，在 jQuery 里$("#a")将得到 jQuery 对象[ ]，然后可以用 jQuery 提供的很多方法来进行操作，如$("#a").scr()将得到 5.jpg,$("#a").scr("1.jpg")将该对象 src 属性改为 1,jpg。下面我们来讲 jQuery 提供的众多 jQuery 方法，方便大家快速对 DOM 对象进行操作

```
herf()    herf(val)
```

说明：对 jQuery 对象属性 herf 的操作。

例子：

未执行 jQuery 前

```
<a href="1.htm" id="test" onClick="jq()">jQuery</a>
```

jQuery 代码及功能：

```
function jq(){
   alert($("#test").href());
   $("#test").href("2.html");
}
```

运行：先弹出对话框显示 id 为 test 的连接 url，在将其 url 改为 2.html，当弹出对话框后会看到转向到 2.html

同理，jQuery 还提供类似的其他方法，大家可以分别试验一下：

herf()　herf(val)	html()　html(val)	id()　id (val)
name()　name (val)	rel()　rel (val)	src()　src (val)
title()　title (val)	val()　val(val)	

操作

◆ after(html)　在匹配元素后插入一段 html

```
<a href="#" id="test" onClick="jq()">jQuery</a>
```

jQuery 代码及功能：

```
function jq(){
      $("#test").after("<b>Hello</b>");
}
```

执行后相当于：

```
<a href="#" id="test" onClick="jq()">jQuery</a><b>Hello</b>
```

◆ after(elem)　after(elems)　将指定对象 elem 或对象组 elems 插入到在匹配元素后

```
<p id="test">after</p><a href="#" onClick="jq()">jQuery</a>
```

jQuery 代码及功能

```
function jq(){
     $("a").after($("#test"));
}
```

执行后相当于

```
<a href="#" onClick="jq()">jQuery</a><p id="test">after</p>
```

◆ append(html)在匹配元素内部，且末尾插入指定 html

```
<a href="#" id="test" onClick="jq()">jQuery</a>
```

jQuery 代码及功能：

```
function jq(){
     $("#test").append("<b>Hello</b>");
}
```

执行后相当于

```
<a href="#" onClick="jq()">jQuery<b>Hello</b></a>
```

同理还有 append(elem)　append(elems) before(html) before(elem) before(elems)请执行参照 append 和 after 的方来测试、理解！

◆ appendTo(expr)　与 append(elem)相反

```
<p id="test">after</p><a href="#" onClick="jq()">jQuery</a>
```

jQuery 代码及功能

```
function jq(){
      $("a"). appendTo ($("#test"));
```

```
}
```

执行后相当于

```
<p id="test">after<a href="#" onClick="jq()">jQuery</a> </p>
```

◆ clone() 复制一个 jQuery 对象

```
<p id="test">after</p><a href="#" onClick="jq()">jQuery</a>
```

jQuery 代码及功能：

```
function jq(){
     $("#test").clone().appendTo($("a"));
}
```

复制$("#test")然后插入到<a>后,执行后相当于

```
<p id="test">after</p><a href="#" onClick="jq()">jQuery</a><p id="test">after</p>
```

◆ empty() 删除匹配对象的所有子节点

```
<div id="test">
  <span>span</span>
  <p>after</p>
</div>
<a href="#" onClick="jq()">jQuery</a>
```

jQuery 代码及功能：

```
function jq(){
    $("#test").empty();
}
```

执行后相当于

```
<div id="test"></div><a href="#" onClick="jq()">jQuery</a>
```

◆ insertAfter(expr) insertBefore(expr)

insertAfter(expr)相当于 before(elem),insertBefore(expr)相当于 after (elem)

◆ prepend (html) prepend (elem) prepend (elems) 在匹配元素的内部且开始出插入

通过下面例子区分 append(elem) appendTo(expr) prepend (elem)

```
<p id="a">p</p>
<div>div</div>
```

执行$("#a").append($("div")) 后相当于

```
<p id="a">
  P
  <div>div</div>
</p>
```

执行$("#a").appendTo($("div")) 后 相当于

```
<div>
   div
   <p id="a">p</p>
</div>
```

执行$("#a").prepend ($("div")) 后 相当于

```
<p id="a">
   <div>div</div>
   P
</p>
```

◆ remove() 删除匹配对象

注意区分 empty()，empty()移出匹配对象的子节点，remove()，移出匹配对象

◆ wrap(htm) 将匹配对象包含在给出的 html 代码内

```
<p>Test Paragraph.</p> <a href="#" onClick="jq()">jQuery</a>
```

jQuery 代码及功能：

```
function jq(){
      $("p").wrap("<div class='wrap'></div>");
}
```

执行后相当于

```
<div class='wrap'><p>Test Paragraph.</p></div>
```

◆ wrap(elem) 将匹配对象包含在给出的对象内

```
<p>Test Paragraph.</p><div id="content"></div>
<a href="#" onClick="jq()">jQuery</a>
```

jQuery 代码及功能：

```
function jq(){
      $("p").wrap( document.getElementById('content') );
}
```

执行后相当于

```
<div id="content"><p>Test Paragraph.</p></div>
```

遍历、组合

◆ add(expr) 在原对象的基础上在附加符合指定表达式的 jquery 对象

```
<p>Hello</p><p><span>Hello Again</span></p>
<a href="#" onClick="jq()">jQuery</a>
```

jQuery 代码及功能：

```
function jq(){
     var f=$("p").add("span");
     for(var i=0;i < $(f).size();i++){
     alert($(f).eq(i).html());}
}
```

执行$("p")得到匹配 的对象，有两个，add("span")是在("p")的基础上加上匹配的对象，所有一共有 3 个，从上面的函数运行结果可以看到$("p").add("span")是 3 个对象的集合，分别是[Hello]，[Hello Again]，[Hello Again]。

◆ add(el) 在匹配对象的基础上在附加指定的 dom 元素。

```
$("p").add(document.getElementById("a"));
```

◆ add(els) 在匹配对象的基础上在附加指定的一组对象，els 是一个数组。

```
<p>Hello</p><p><span>Hello Again</span></p>
```

jQuery 代码及功能：

```
function jq(){
     var f=$("p").add([document.getElementById("a"), document.getElementById("b")])
     for(var i=0;i < $(f).size();i++){
             alert($(f).eq(i).html());}
}
```

注意 els 是一个数组，这里的[ ]不能漏掉。

◆ ancestors () 一依次以匹配结点的父节点的内容为对象,根节点除外。

```
<div>
    <p>one</p>
    <span>
    <u>two</u>
    </span>
</div>
```

jQuery 代码及功能：

```
function jq(){
```

```
    var f= $("u").ancestors();
    for(var i=0;i < $(f).size();i++){
     alert($(f).eq(i).html());}
}
```

第一个对象是以<u>的父节点的内容为对象，[<u>two</u>]

第一个对象是以<u>的父节点的父节点(div)的内容为对象,[one <u>two</u>]

一般一个文档还有<body>和<html>，依次类推下去。

◆ ancestors (expr) 在 ancestors ( ) 的基础上之取符合表达式的对象

如上各例子讲 var f 改为 var f= $("u").ancestors(“div”),则只返回一个对象：

[one <u>two</u>]

◆ children() 返回匹配对象的子介点

```
<p>one</p>
<div id="ch">
    <span>two</span>
</div>
```

jQuery 代码及功能：

```
function jq(){
    alert($("#ch").children().html());
}
```

$("#ch").children()得到对象[two].所以.html()的结果是” two”

◆ children(expr) 返回匹配对象的子介点中符合表达式的节点

```
<div id="ch">
     <span>two</span>
     <span id="sp">three</span>
</div>
```

jQuery 代码及功能

```
function jq(){
    alert($("#ch").children(“#sp”).html());
}
```

$("#ch").children()得到对象[twothree].

$("#ch").children(“#sp”)过滤得到[three]

◆ parent () parent (expr)取匹配对象父节点的。参照 children 帮助理解

◆ contains(str) 返回匹配对象中包含字符串 str 的对象

This is just a test. So is this

jQuery 代码及功能：

```
function jq(){
    alert($("p").contains("test").html());
}
```

$("p")得到两个对象，而包含字符串”test”只有一个。所有$("p").contains("test")返回

[This is just a test.]

◆ end() 结束操作,返回到匹配元素清单上操作前的状态.

◆ filter(expr) filter(exprs) 过滤现实匹配符合表达式的对象 exprs 为数组，注意添加“[ ]”

Hello Hello Again And Again

jQuery 代码及功能：

```
function jq(){
    alert($("p").filter(".selected").html())
```

```
}
```

$("p")得到三个对象，$("p").contains("test")只返回 class 为 selected 的对象。

◆ find(expr) 在匹配的对象中继续查找符合表达式的对象

```
<p>Hello</p><p id="a">Hello Again</p><p class="selected">And Again</p>
```

Query 代码及功能：

```
function jq(){
    alert($("p").find("#a").html())
}
```

在$("p")对象中查找 id 为 a 的对象。

◆ is(expr) 判断对象是否符合表达式,返回 boolen 值

```
<p>Hello</p><p id="a">Hello Again</p><p class="selected">And Again</p>
```

Query 代码及功能：

```
function jq(){
    alert($("#a").is("p"));
}
```

◆ next() next(expr) 返回匹配对象剩余的兄弟节点

```
<p>Hello</p><p id="a">Hello Again</p><p class="selected">And Again</p>
```

jQuery 代码及功能

```
function jq(){
        alert($("p").next().html());
        alert($("p").next(".selected").html());
}
```

$("p").next()返回 [<p id="a">Hello Again</p> , <p class="selected">And Again</p>]两个对象；$("p").next(".selected)只返回 [And Again]一个对象。prev () prev (expr) 参照 next 理解

◆ not(el) not(expr) 从 jQuery 对象中移出匹配的对象，el 为 dom 元素，expr 为 jQuery 表达式。

```
<p>one</p><p id="a">two</p>
<a href="#" onclick="js()">jQuery</a>
```

jQuery 代码及功能：

```
function js(){
     alert($("p").not(document.getElementById("a")).html());
     alert($("p").not("#a").html());
}
```

$("p")由两个对象，排除后的对象为[one]

◆ siblings () siblings (expr) jquery 匹配对象中其它兄弟级别的对象

```
<p>one</p>
<div>
  <p id="a">two</p>
</div>
<a href="#" onclick="js()">jQuery</a>
```

jQuery 代码及功能：

```
function js(){
       alert($("div").siblings().eq(1).html());
}
```

$("div").siblings() 的结果实返回两个对象 [one , jQuery]

alert($("div").siblings("a")返回一个对象[jQuery]

其他

addClass(class)　为匹配对象添加一个 class 样式
removeClass (class)　将第一个匹配对象的某个 class 样式移出

◆ attr (name)　获取第一个匹配对象的属性

```
<img src="test.jpg"/><a href="#" onclick="js()">jQuery</a>
```

jQuery 代码及功能：

```
function js(){
    alert($("img").attr("src"));
}
```

返回 test.jpg

◆ attr (prop)　为第一个匹配对象的设置属性，prop 为 hash 对象，用于为某对象批量添加众多属性

```
<img/><a href="#" onclick="js()">jQuery</a>
```

jQuery 代码及功能：

```
function js(){
    $("img").attr({ src: "test.jpg", alt: "Test Image" });
}
```

运行结果相当于

```
<img src="test.jpg" alt="Test Image"/>
```

◆ attr (key,value)　为第一个匹配对象的设置属性，key 为属性名，value 为属性值

```
<img/><a href="#" onclick="js()">jQuery</a>
```

jQuery 代码及功能

```
function js(){
    $("img").attr(“src”,” test.jpg”);
}
```

运行结果相当于

◆ removeAttr (name)　将第一个匹配对象的某个属性移出

```
<img alt="test"/><a href="#" onclick="js()">jQuery</a>
```

jQuery 代码及功能：

```
function js(){
    $("img"). removeAttr("alt");
}
```

运行结果相当于<img />

◆ toggleClass (class)　将当前对象添加一个样式，不是当前对象则移出此样式，返回的是处理后的对象

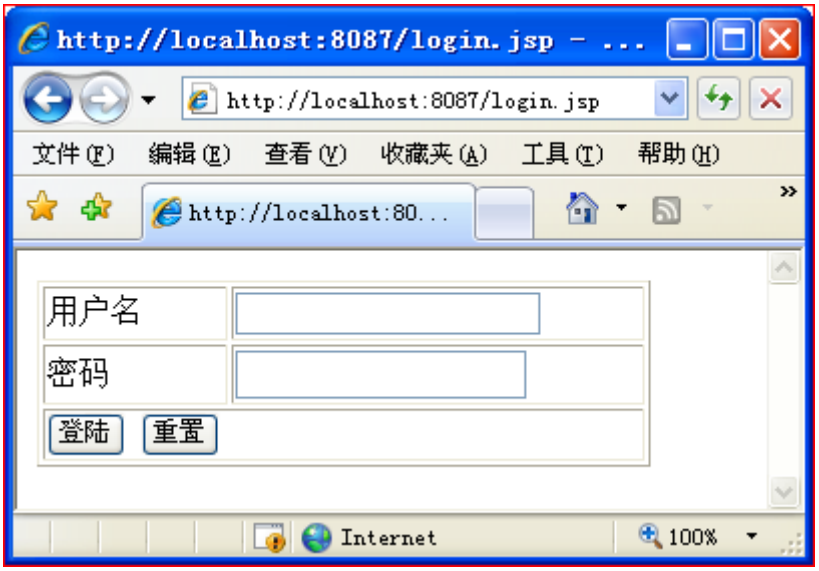
```
<p>Hello</p><p class="selected">Hello Again</p><a href="#" onclick="js()">jQuery</a>
```

$("p")的结果是返回对象 [Hello , Hello Again]

$("p").toggleClass("selected")的结果是实返回对象 [Hello , Hello Again]

16.4 Jquery处理text

经常会在页面上运用异步效果，下面就来看看用 jquery 实现登录效果。



代码如下所示：

```
      <script type="text/javascript" language="javascript"
 src="/js/jquery-1.2.6.min.js"></script>
          <!--以下是修改功能代码//-->
        <script type="text/javascript">
     function sendInsertRequest() {
         var username = $("#username").val();
         var pass = $("#pass").val();
         $.ajax({
             type: "POST",// 指定是 post 还是 get,当然此处要提交,当然就要用 post 了
             cache: "false",//(默认: true,dataType 为 script 时默认为 false) jQuery 1.2
新功能，设置为 false 将不会从浏览器缓存中加载请求信息。
             url:  "ajax?method=insertCustomer",//发送请求的地址。
             data: "username=" + username + "&pass=" + pass ,//发送到服务器的数据
             dataType: "text",//返回纯文本字符串
             timeout:10000,// 设置请求超时时间（毫秒）。
             error: function (XMLHttpRequest, textStatus, errorThrown) {//请求失败时
调用函数。
                 $("#infoSpan").html("<img
src='/assets/images/standard_msg_error.gif'/><span    class='Estilo11'    style='color:
#aaaaaa'>请求失败！</span>");
             } ,
             success:responseInsertRequest//请求成功后回调函数。
         });
         return false;
     }
     function responseInsertRequest(message) {
         alert(message);
        if(message=="ok") {
             window.location.href = "/welcome.jsp";
        }else{
             window.location.href = "/insert.jsp";
        }
     }
     </script>
```

16.5 Jquery处理xml

16.4 Jquery处理json

第 17 章 投票管理系统

第 18 章 无纸化办公管理系统

系统的登录窗口：

办公系统个人通讯录部分列表页面效果：

添加联系人：

修改联系人资料：

首先，注意在截图的中，大家会发现访问的地址后缀为.do 格式，在前面的示例中都采用的是.action 方式，如何来进行相关的设置呢？

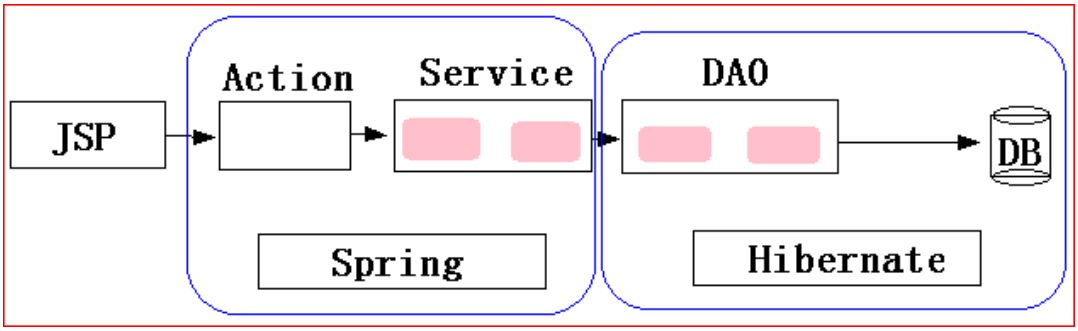
struts.properties 文件

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。

struts.properties 配置文件提供了一种改变框架默认行为的机制。一般来讲我们没必要修改这个文件，除非你想拥有一个更加友好的开发调试环境。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。



dao 只是针对数据的操作层，service 是业务逻辑，因为业务逻辑很复杂就涉及到事务的管理，而且他还可以提供对外的服务，比如 webservice 服务，都是由 service 做的， action 只负责请求转发，因为我们的业务需求简单，读者朋友请不要把 dao 和 service 做的事情视为一样，这里要强调一下：事务在 service，业务逻辑却在 service。

在 SSH2 的整合过程中，我建议采用如上图所示的组合方式：将 Action 的创建交由 Spring 管理，在 struts.xml 中直接引用创建的对象即可。并且 Action 所需的对象直接通过设置注入即可。在 spring 中调用 hibernate 建议采用继承 HibernateDaoSupport 的方式进行。

下面来看看相关内容清单：

Login.jsp

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title><s:text name="login.page.title" />
        </title>
    </head>
    <body>

        <form name="form1" action="login.do" method="post">
            <table width="300" border="1">
                <tr>
                    <td colspan="2">
                        <s:text name="login.page.title" />
                    </td>
                </tr>
                <tr>
                    <td>
                        <s:text name="login.page.username" />
                    </td>
                    <td>
                        <s:textfield name="username" />
```

```
                        <s:fielderror>
                            <s:param>username</s:param>
                        </s:fielderror>
                    </td>
                </tr>
                <tr>
                    <td>
                        <s:text name="login.page.password" />
                    </td>
                    <td>
                        <s:password name="password" />
                        <s:fielderror>
                            <s:param>password</s:param>
                        </s:fielderror>
                    </td>
                </tr>
                <tr>
                    <td colspan="2">
                        <s:submit key="login.page.login" />
                        <a  href="register!init.do"><s:text  name="login.page.register"
/>
                        </a>
                    </td>
                </tr>
            </table>
            <br>
            <br>
            <s:actionerror />
            <s:actionmessage />
        </form>
    </body>
</html>
```

<s:text../>是从资源文件中读取相关数据，查看配置文件，login 对应的是 LoginAction，<s:textfield >、<s:password>标签分别是绑定 LoginAction 相应的 set 方法。<s:fielderror>是当验证出错时，要显示的相关设置。

当用户单击“登录”时，将提交 login.do 请求，由 loginAction 处理。这里未指明具体的方法，所以请求将交由 execute()方法处理。

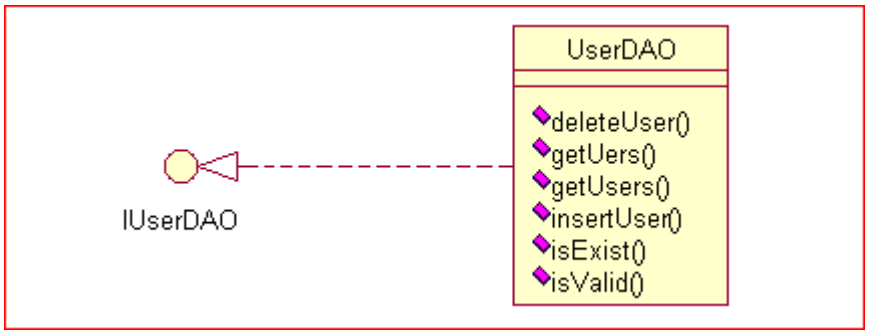
```
<action name="login" class="loginAction">
            <result name="success">welcome.jsp</result>
            <result name="input">login.jsp</result>
        </action>
```

处理成功将返回 welcome.jsp 页面，否则将返回 login.jsp 页面。

```
package com.demo.struts2.actions;
……

public class LoginAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    private UserDAO userDAO;
    private String username;
    private String password;
    public void validate() {
```

```
        clearErrorsAndMessages();
        if (username == null || username.equals("")) {
            addFieldError("username", getText("login.error.username"));
        }
        if (password == null || password.equals("")) {
            addFieldError("password", getText("login.error.password"));
        }
    }

    public String execute() throws Exception {
        if (this.userDAO.isValid(username, password)) {
            ActionContext.getContext().getSession().put(Constants.USERNAME_KEY,
                    username);
            return SUCCESS;
        } else {
            super.addActionError(super.getText("login.message.failed"));
            return INPUT;
        }
    }
    public UserDAO getUserDAO() {
        return userDAO;
    }
    public void setUserDAO(UserDAO userDAO) {
        this.userDAO = userDAO;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

execute()方法在执行前会先调用 validate()方法。



在 IUserDAO 是一个接口，它里面定义了与底层数据库打交道的方法，代码清单如下所示：

```
package com.demo.hibernate.dao;
```

```
import java.util.List;

import com.demo.hibernate.beans.User;

public interface IUserDAO {

    public boolean isValid(final String username, final String password);

    public boolean isExist(String username);

    public void insertUser(User user);

    public User getUser(String userid);

    public List getUsers();

    public void deleteUser(String userid);
}
```

在这儿只讨论 isValid 方法，查看其实现类的代码如下所示：

```
public class UserDAO extends BaseDao implements IUserDAO {

    public boolean isValid(final String username, final String password) {
        List list = (List) getHibernateTemplate().execute(
            new HibernateCallback() {
            public Object doInHibernate(Session session)
                    throws HibernateException {
                List result = session.createCriteria(User.class).add(
                        Restrictions.eq("username", username)).add(
                        Restrictions.eq("password", password)).list();
                return result;
            }
        });
        if (list.size() > 0) {
            return true;
        } else {
            return false;
        }
    }
……
}
```

可能读者朋友对 BaseDao 感觉陌生。它的代码清单如下所示：

```
package com.demo.hibernate.dao;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
public class BaseDao extends HibernateDaoSupport {
}
```

因为在 spring 的配置文件中，要对底层的实现类注入 SessionFactory 对象，如果实现类较多的话，配置文件看上去就显得不够简炼：

```
    <bean id="userDAO" class="com.demo.hibernate.dao.UserDAO">
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
```

```
    </bean>
    <bean id="addressDAO" class="com.demo.hibernate.dao.AddressDAO">
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
    </bean>
```

如果采用一个类文件过渡一下，则只需要给父类注入 SessionFactory 对象即可，代码看上去清爽多了，如下所示：

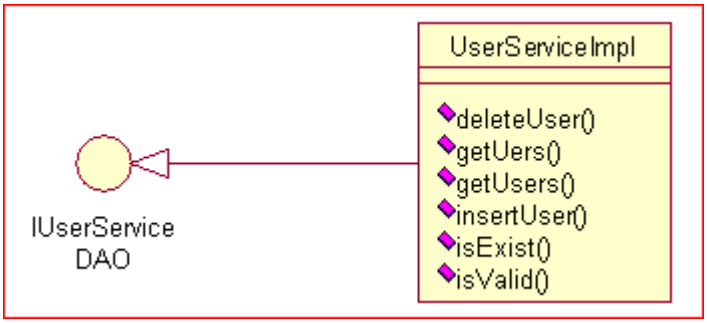
```
    <!-- 定义 DAO -->
<bean id="baseDao" class="com.demo.hibernate.dao.BaseDao">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>
 <bean id="userDAO" class="com.demo.hibernate.dao.UserDAO" parent="baseDao" />
 <bean id="addressDAO" class="com.demo.hibernate.dao.AddressDAO" parent="baseDao" />
```

在编码的时候推荐使用回调函数，采用这种做法的好处是不用关心事务。session 的创建和销毁，一切都在程序内部完成。它的名字 doInHibernate 即已说明它的功能。只是代码看起来有些凌乱。

getHibernateTemplate().getSessionFactory()有 getCurrentSession () 和 openSession()两个方法，前者表示使用当前的 session，后者表示重新建立一个新的 session。但是这种做法，是需要自己去手动关闭 session 的。所以你需要配置 openSessioninview，个人认为不推荐使用！再一次温故回调函数的写法。

```
public boolean isExist(final String username) {
        List list = (List) getHibernateTemplate().execute(new HibernateCallback() {
            public Object doInHibernate(Session session)
                    throws HibernateException {
                List result = session.createCriteria(User.class).add(
                        Restrictions.eq("username", username)).list();
                return result;
            }
        });
        if (list.size() > 0) {
            return true;
        } else {
            return false;
        }
    }
```

业务接口与业务层关系如下：



具体的代码参见源文件。在前面已表述 service 层的作用，请读者朋友不要将 service 层与 dao 层的作用混为一谈。

下面看看 Action 部分的代码：

```
......
```

```
public class AddressAction extends PageAction {
    private IAddressServiceDAO addressService=null;
    private static final long serialVersionUID = 1L;

    protected String id = null;

    protected String username = null;

    protected String name = null;

    protected String sex = null;

    protected String mobile = null;

    protected String email = null;

    protected String qq = null;

    protected String company = null;

    protected String address = null;

    protected String postcode = null;

    // 新增或修改时执行表单验证
    public void validate() {
        // 清除错误消息
        clearErrorsAndMessages();

        // 取得请求参数
        String queryString = getRequestPath();
        if (queryString.indexOf("addressadd!insert") != -1
                || queryString.indexOf("addressedit!update") != -1) {
            // 检查表单字段 name
            if (name == null || name.equals("")) {
                addFieldError("name", getText("address.error.name"));
            }

            // 检查表单字段 mobile
            if (mobile != null && !mobile.equals("")) {
                Pattern p_mobile = Pattern.compile(RegExpression.REG_mobile);
                Matcher m_mobile = p_mobile.matcher(mobile);
                if (!m_mobile.find()) {
                    addFieldError("mobile", getText("address.error.mobile"));
                }
            }

            // 检查表单字段 email
            if (email != null && !email.equals("")) {
                Pattern p_email = Pattern.compile(RegExpression.REG_email);
                Matcher m_email = p_email.matcher(email);
```

```
                if (!m_email.find()) {
                    addFieldError("email", getText("address.error.email"));
                }
            }

            // 检查表单字段 postcode
            if (postcode != null && !postcode.equals("")) {
                Pattern p_postcode = Pattern
                        .compile(RegExpression.REG_postcode);
                Matcher m_postcode = p_postcode.matcher(postcode);
                if (!m_postcode.find()) {
                    addFieldError("postcode", getText("address.error.postcode"));
                }
            }
        }
    }

    // 请求 addressInit.do 的处理函数
    public String init() throws Exception {
        // 清除错误消息
        clearErrorsAndMessages();

        // 重设分页参数
        super.pageSize = Constants.pageSize;
        super.pageNo = Constants.pageNo;

        // 取得当前分页数据
        super.pager =addressService.findPagerByUsername(super
                .getLoginUsername(), super.pageSize, super.pageNo);

        // 保存分页数据
        setSession(Constants.PAGER_ADDRESS, super.pager);

        return Constants.LIST_KEY;
    }

    // 请求 addressList.do 的处理函数
    public String list() throws Exception {
        // 清除错误消息
        clearErrorsAndMessages();

        // 取得当前分页数据
        super.pager = addressService.findPagerByUsername(super
                .getLoginUsername(), super.pageSize, super.pageNo);

        // 保存分页数据
        setSession(Constants.PAGER_ADDRESS, super.pager);

        return Constants.LIST_KEY;
    }
```

```
    // 请求 addressAdd.do 的处理函数
    public String add() throws Exception {
        // 清除错误消息
        clearErrorsAndMessages();

        // 重设各表单字段
        reset();
        return Constants.ADD_KEY;
    }

    // 重设各表单字段
    private void reset() {
        setId(null);
        setUsername(null);
        setName(null);
        setSex(null);
        setMobile(null);
        setEmail(null);
        setQq(null);
        setCompany(null);
        setAddress(null);
        setPostcode(null);
    }

    // 给表单字段赋值
    private void bean2Form(Address address) {
        setId(address.getId().toString());
        setUsername(address.getUsername());
        setName(address.getName());
        setSex(address.getSex());
        setMobile(address.getMobile());
        setEmail(address.getEmail());
        setQq(address.getQq());
        setCompany(address.getCompany());
        setAddress(address.getAddress());
        setPostcode(address.getPostcode());
    }

    // 请求 addressEdit.do 的处理函数
    public String edit() throws Exception {
        // 清除错误消息
        clearErrorsAndMessages();

        // id 为空时返回错误
        if (this.getId() == null) {
            saveActionError("address.message.edit.notexist");
            return Constants.LIST_KEY;
        } else {
            // 查询数据表
            Address address =addressService.findById(id);
```

```
            // 不存在时返回错误
            if (address == null) {
                saveActionError("address.message.edit.notexist");
                return Constants.LIST_KEY;
            } else {
                // 给表单字段赋值
                bean2Form(address);
                return Constants.EDIT_KEY;
            }
        }
    }

    // 请求 addressInsert.do 的处理函数
    public String insert() throws Exception {
        // 清除错误消息
        clearErrorsAndMessages();

        // 判断姓名是否已经存在
        boolean b = addressService.isExist(super.getLoginUsername(), this.getName());
        if (!b) {
            // 插入数据表
            Address address = new Address();
            address.setUsername(super.getLoginUsername());
            address.setName(this.getName());
            address.setSex(this.getSex());
            address.setMobile(this.getMobile());
            address.setEmail(this.getEmail());
            address.setQq(this.getQq());
            address.setCompany(this.getCompany());
            address.setAddress(this.getAddress());
            address.setPostcode(this.getPostcode());
            addressService.insert(address);
        }

        // 取得缓存的分页参数
        Pager pagerSession = (Pager) getSession(Constants.PAGER_ADDRESS);
        super.pageSize = pagerSession.getPageSize();
        super.pageNo = pagerSession.getPageNo();

        // 查询当前页的数据
        super.pager = addressService.findPagerByUsername(super
                .getLoginUsername(), super.pageSize, super.pageNo);

        if (!b) {
            // 保存成功信息
            saveActionMessage("address.message.add.success");
            return Constants.LIST_KEY;
        } else {
            // 保存失败信息
            saveActionError("address.message.add.failed");
            return Constants.ADD_KEY;
```

```
        }

    }

    // 请求 addressUpdate.do 的处理函数
    public String update() throws Exception {
        // 清除错误消息
        clearErrorsAndMessages();

        // 更新数据表
        Address address = new Address();
        address.setId(new Integer(id));
        address.setUsername(super.getLoginUsername());
        address.setName(this.getName());
        address.setSex(this.getSex());
        address.setMobile(this.getMobile());
        address.setEmail(this.getEmail());
        address.setQq(this.getQq());
        address.setCompany(this.getCompany());
        address.setAddress(this.getAddress());
        address.setPostcode(this.getPostcode());
        addressService.update(address);

        // 给表单字段赋值
        bean2Form(address);

        // 取得缓存的分页参数
        Pager pagerSession = (Pager) getSession(Constants.PAGER_ADDRESS);
        super.pageSize = pagerSession.getPageSize();
        super.pageNo = pagerSession.getPageNo();

        // 查询当前页的数据
        super.pager = addressService.findPagerByUsername(super
                .getLoginUsername(), super.pageSize, super.pageNo);

        saveActionMessage("address.message.edit.success");
        return Constants.LIST_KEY;

    }

    // 请求 addressDelete.do 的处理函数
    public String delete() throws Exception {
        // 清除错误消息
        clearErrorsAndMessages();

        // id 为空时返回错误
        if (this.getId() == null) {
            saveActionError("address.message.edit.notexist");
        } else {
            // 删除数据
            addressService.delete(id);
```

```
            saveActionMessage("address.message.delete.success");
        }
        // 取得当前页的数据
        super.pager = addressService.findPagerByUsername(super
                .getLoginUsername(), super.pageSize, super.pageNo);

        return Constants.LIST_KEY;
    }
    ......
}
```

现在看看 PageAction 的代码：

```
package com.demo.struts2.common;

import com.demo.struts2.util.Constants;
import com.demo.struts2.util.Pager;

public class PageAction extends BaseAction {

    private static final long serialVersionUID = -7509152655359967096L;

    protected int pageSize = Constants.pageSize;

    protected int pageNo = Constants.pageNo;

    protected Pager pager;

    public int getPageNo() {
        return pageNo;
    }

    public void setPageNo(int pageNo) {
        this.pageNo = pageNo;
    }

    public Pager getPager() {
        return pager;
    }

    public void setPager(Pager pager) {
        this.pager = pager;
    }

    public int getPageSize() {
        return pageSize;
    }

    public void setPageSize(int pageSize) {
        this.pageSize = pageSize;
    }
}
```

BaseAction 里到底有什么呢？

```
package com.demo.struts2.common;

import org.apache.struts2.ServletActionContext;

import com.demo.service.dao.IAddressServiceDAO;
import com.demo.service.dao.IUserServiceDAO;
import com.demo.struts2.util.Constants;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class BaseAction extends ActionSupport {

    private static final long serialVersionUID = -7509152655359967096L;

    // 取得当前登录的用户名
    protected String getLoginUsername() {
        return (String) ActionContext.getContext().getSession().get(
                Constants.USERNAME_KEY);
    }

    // 判断当前用户是否超时
    protected boolean isTimeout() {
       if (ActionContext.getContext().getSession().get(Constants.USERNAME_KEY) == null)
{
            return true;
        } else {
            return false;
        }
    }

    // 检查 Session 对象是否存在
    protected boolean isExistSession(String key) {
        if (ActionContext.getContext().getSession().get(key) != null) {
            return true;
        } else {
            return false;
        }
    }

    // 保存 Session 对象
    protected void setSession(String key, Object obj) {
        ActionContext.getContext().getSession().put(key, obj);
    }

    // 取得 Session 对象
    protected Object getSession(String key) {
        return ActionContext.getContext().getSession().get(key);
    }

    // 保存一条错误
    protected void saveActionError(String key) {
```

```
        super.addActionError(super.getText(key));
    }

    // 保存一个消息
    protected void saveActionMessage(String key) {
        super.addActionMessage(super.getText(key));
    }

    // 取得查询的 URL
    protected String getRequestPath() {
        return (String)ServletActionContext.getRequest().getServletPath();
    }

}
```

分页的解决方案：

```
package com.demo.struts2.util;

import java.util.ArrayList;
import java.util.List;

public class Pager {
    // 页面大小
    protected int[] pageSizeList = { 10, 25, 50, 100, 200, 300, 500 };
    // 一页显示的记录数
    protected int pageSize = Constants.pageSize;
    // 当前页码
    protected int pageNo = Constants.pageNo;
    // 记录总数
    protected int rowCount = 0;
    // 总页数
    protected int pageCount = 1;
    // 起始行数
    protected int startIndex = 1;
    // 结束行数
    protected int endIndex = 1;

    protected int firstPageNo = 1;
    protected int prePageNo = 1;
    protected int nextPageNo = 1;
    protected int lastPageNo = 1;

    // 结果集存放 List
    protected List resultList;

    public Pager(int pageSize, int pageNo, int rowCount, List resultList) {
        this.pageSize = pageSize;
        this.pageNo = pageNo;
        this.rowCount = rowCount;
        this.resultList = resultList;

        if (rowCount % pageSize == 0) {
```

```
            this.pageCount = rowCount / pageSize;
        } else {
            this.pageCount = rowCount / pageSize + 1;
        }
        this.startIndex = pageSize * (pageNo - 1);
        this.endIndex = this.startIndex + resultList.size();

        this.lastPageNo = this.pageCount;
        if (this.pageNo > 1) this.prePageNo = this.pageNo -1;
        if (this.pageNo == this.lastPageNo){
            this.nextPageNo = this.lastPageNo;
        } else {
            this.nextPageNo = this.pageNo + 1;
        }
    }
    //把分页页面显示设定数据封装于一对象数组之中。
    public Object[] getPageSizeIndexs() {
        List result = new ArrayList(pageSizeList.length);
        for (int i = 0; i < pageSizeList.length; i++) {
            result.add(String.valueOf(pageSizeList[i]));
        }
        Object[] indexs = (result.toArray());
        return indexs;
    }
    //把页数封闭于一对象数组之中
    public Object[] getPageNoIndexs() {
        List result = new ArrayList(pageCount);
        for (int i = 0; i < pageCount; i++) {
            result.add(String.valueOf(i + 1));
        }
        Object[] indexs = (result.toArray());
        return indexs;
    }

    public int getEndIndex() {
        return endIndex;
    }

    public void setEndIndex(int endIndex) {
        this.endIndex = endIndex;
    }

    public int getPageCount() {
        return pageCount;
    }

    public void setPageCount(int pageCount) {
        this.pageCount = pageCount;
    }

    public int getPageNo() {
```

```
        return pageNo;
    }

    public void setPageNo(int pageNo) {
        this.pageNo = pageNo;
    }

    public int getPageSize() {
        return pageSize;
    }

    public void setPageSize(int pageSize) {
        this.pageSize = pageSize;
    }

    public int[] getPageSizeList() {
        return pageSizeList;
    }

    public void setPageSizeList(int[] pageSizeList) {
        this.pageSizeList = pageSizeList;
    }

    public List getResultList() {
        return resultList;
    }

    public void setResultList(List resultList) {
        this.resultList = resultList;
    }

    public int getRowCount() {
        return rowCount;
    }

    public void setRowCount(int rowCount) {
        this.rowCount = rowCount;
    }

    public int getStartIndex() {
        return startIndex;
    }

    public void setStartIndex(int startIndex) {
        this.startIndex = startIndex;
    }

    public int getFirstPageNo() {
        return firstPageNo;
    }
```

```
    public void setFirstPageNo(int firstPageNo) {
        this.firstPageNo = firstPageNo;
    }

    public int getLastPageNo() {
        return lastPageNo;
    }

    public void setLastPageNo(int lastPageNo) {
        this.lastPageNo = lastPageNo;
    }

    public int getNextPageNo() {
        return nextPageNo;
    }

    public void setNextPageNo(int nextPageNo) {
        this.nextPageNo = nextPageNo;
    }

    public int getPrePageNo() {
        return prePageNo;
    }

    public void setPrePageNo(int prePageNo) {
        this.prePageNo = prePageNo;
    }

}
```

第 19 章 某数据采集系统

STRUTS2 整理成册

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<s:iterator id="jxnews" value="#request.jxNewList">
<s:set id="jxtitle" value="#jxnews.title"/>
<div style="padding:5px">
<img src="../Images/Icons/33.gif" width="4" height="7" align="absMiddle"> 
<a       href="/homepage/Info/viewnew.action?news.id=<s:property       value='id'/>"><c:out
```

```
value="${fn:substring(jxtitle,0,14)}..."/></a></div>
</s:iterator>
```

第二种方案：

```
<s:iterator id="news" value="#request.lstssq">
<div style="padding:5px"><img src="Images/Icons/33.gif" width="4" height="7"
align="absMiddle"> 
<a
        href="/homepage/Info/viewnew.action?news.id=<s:property value='#news.id'/>">
<s:property value="@com.sunloto.lottery.common.Utils@cutStr(#news.title,10)"/></a>
</div>
</s:iterator>
```