

Struts 2开发技术 (高级应用)

邹国兵，博士

副教授、博导/硕导

服务计算与数据挖掘实验室

<https://scdm-shu.github.io>

目录

- Struts 2体系架构
- Struts 2工作原理
- Struts 2组件功能分析
- Struts基本应用实例
- Action实现
- Struts配置文件
- 拦截器
- 输入校验
- 文件上传
- Struts综合应用实例

用户请求的Web处理流程

HTTP请求

ActionContextCleanUP

其它过滤器(如SiteMesh等)

Struts2的核心控制器: StrutsPrepareAndExecuteFilter

Action代理

配置管理器

struts.xml

调用Action

拦截器 1

拦截器 2

拦截器 3

Action

Result

拦截器 3

拦截器 2

拦截器 1

Action映射器

Struts2标签库
如HTML,form,等

视图模板

-JSP

-FreeMarker

-等等

HTTP响应

颜色含义

Servlet过滤器

Struts2核心API

开发者定义文件

拦截器

➤ 拦截器

- Struts 2框架内置了可实现多种功能的拦截器。这些拦截器可以在Struts 2的配置文件struts-default.xml查看。

拦截器名称。	功能描述。
alias。	对不同请求中的相同参数进行命名转换。
autowiring。	框架自动寻找相应的 Bean 并完成设置工作。
chain。	构建 Action 链，当使用<result type="chain">进行配置时，当前 Action 可使用前一个已经执行结束的 Action 属性，实现 Action 链之间的数据传递。
checkbox。	负责检查 checkbox 表单控件是否被选中，当 checkbox 未被选中时，提交一个默认的值(通常是 false)。
cookie。	把带有特定名/值映射关系的 Cookie 注射到 Action 中。
conversionError。	处理类型转换时的错误信息。把 ActionContext 中的错误信息转换为相应的 Action 字段的错误信息并保存，需要时可通过视图显示相关错误信息。
createSession。	自动创建一个 HttpSession 对象，因为有些拦截器必须通过 HttpSession 对象才能正常工作(比如 TokenInterceptor)。
debugging。	负责调试，当页面中使用<s:debug>标签时，可获得值栈、上下文等信息。
execAndWait。	在后台执行 Action 并将等待画面传送给用户。
exception。	提供处理异常功能，将异常映射为结果。
fileUpload。	负责文件上传。
il8n。	把指定 Locale 信息放入 Session。
logger。	输出 Action 名称。

拦截器的应用

store.	存储或者访问实现 ValidationAware 接口的 Action 类出现的消息、错误、字段错误等。
model-driven.	如果某个 Action 实现了 ModelDriven 接口时，把 getModel()方法的结果放入值栈中。
scoped-model-driven.	如果某个 Action 实现了 ScopedModelDriven 接口，拦截器获得指定的模型，通过 setModel()方法将其传送到 Action。
params.	解析 HTTP 请求参数将其传送给 Action，设置成 Action 对应的属性值。
prepare.	处理 Action 执行之前所要执行操作。Action 需要实现 Preparable 接口，在 Action 执行之前调用 prepare()方法。
scope.	将一些公有参数信息存储到 Session 作用域或者 Application 作用域，当 Action 需要时，拦截器检查并从 Session 或 Application 中将其取出。
servletConfig.	提供对 HttpServletRequest 和 HttpServletResponse 的访问机制。
staticParams.	把定义在 XML 中的<action>元素的子元素<param>中的参数传入 Action。
roles.	检查用户是否具有 JAAS 授权，只有授权用户才可调用相应 Action。
timer.	输出 Action 的执行时间。
token.	检查传入到 Action 中的 Token 信息，防止重复提交。
tokenSession.	功能和 TokenInterceptor 相似，只不过将无效 Token 信息存放在 Session 中。
validation.	执行定义在 xxAction-validation.xml 中的校验器，完成数据校验。
workflow.	调用 Action 中的 validate()方法进行校验，校验失败返回 input 视图。
N/A.	从参数列表中删除不必要的参数。
profiling.	通过参数激活 profile。

Struts 2拦截器

✧ 拦截器的配置

定义拦截器使用<interceptor.../>元素。其格式为：

<interceptor name="拦截器名" class="拦截器实现类"></interceptor>

只要在<interceptor.>与</interceptor>之间配置<param.../>子元素即可传入相应的参数。其格式如下：

**<interceptor name="拦截器名" class="拦截器实现类 ">
 <param name="参数名">参数值</param>
</interceptor>**

**<interceptor name="myInterceptor" class="org.tool.MyInterceptor">
 <param name="参数名">参数值</param>
</interceptor>**

拦截器的配置

在struts.xml中可以配置多个拦截器，它们被包在<interceptors></interceptors>之间，例如下面的配置：

```
<?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="default" extends="struts-default">
        <interceptors>
            <interceptor name="拦截器名1" class="拦截器类1"></interceptor>
            <interceptor name="拦截器名2" class="拦截器类2"></interceptor>
            ...
            <interceptor name="拦截器名n" class="拦截器类n"></interceptor>
        </interceptors>
        ...
        //action配置
    </package>
</struts>
```

拦截器的应用

- 使用拦截器，步骤如下：

(1) 创建使用拦截器的Action类

```
public class TimerAction extends ActionSupport {  
    public String execute() throws Exception {  
        System.out.println("使用拦截器的例子");  
        return SUCCESS;  
    }  
}
```


拦截器的应用

(2) 部署拦截器

```
<struts>  
  <package name="default" extends="struts-default">  
    <interceptors>  
      <!-- 部署timer拦截器 -->  
      <interceptor name="timer"  
        class="com.opensymphony.xwork2.interceptor.  
          TimerInterceptor" />  
    </interceptors>  
    <!-- 省略配置Action的代码 -->  
  </package>  
</struts>
```

拦截器的应用

(3) 为Action添加拦截器

要在某个Action类中使用拦截器，必须为该Action添加指定的拦截器。

```
<struts>
  <package name="default" extends="struts-default">
    <!-- 省略部署拦截器的代码-->
    <!-- 添加拦截器 -->
    <action name="timerAction"
      class="com.interceptor.TimerAction">
      <result>/success.jsp</result>
      <interceptor-ref name="timer" />
    </action>
  </package>
</struts>
```

拦截器栈

✧ 拦截器配置

通常情况下，一个Action要配置不仅一个拦截器，往往多个拦截器一起使用来进行过滤。这时就会把需要配置的几个拦截器组成一个拦截器栈。

定义拦截器栈用<interceptor-stack name="拦截器栈名"/>元素，由于拦截器栈是由各拦截器组合而成的，所以需要在该元素下面配置<interceptor-ref .../>子元素来对拦截器进行引用。其格式如下：

```
<interceptor-stack name="拦截器栈名">  
  <interceptor-ref name="拦截器一"></interceptor-ref>  
  <interceptor-ref name="拦截器二"></interceptor-ref>  
  <interceptor-ref name="拦截器三"></interceptor-ref>  
</interceptor-stack>
```

拦截器栈配置

下面是拦截器栈的配置方法：

```
<package name="包名">
  <interceptors>
    <interceptor name="拦截器一" class="拦截器实现类"></interceptor>
    <interceptor name="拦截器二" class="拦截器实现类"></interceptor>
    <interceptor-stack name="拦截器栈名">
      <interceptor-ref name="拦截器一"></interceptor-ref>
      <interceptor-ref name="拦截器二"></interceptor-ref>
    </interceptor-stack>
  </interceptors>
</package>
```

Struts 2 默认拦截器栈

拦截器栈名称。	所包含拦截器。	功能描述。
basicStack。	exception , servletConfig , prepare , checkbox , params , conversionError。	用于异常处理、HTTP 对象处理、参数传递和转换错误处理等，它是 Struts 2 提供的最常用的拦截器栈。
validationWorkflowStack。	basicStack , validation , workflow。	包含 basicStack 提供的所有功能，此外还添加对验证和工作流的支持。
fileUploadStack。	fileUpload , basicStack。	包含 basicStack 提供的所有功能，此外还添加对文件上传的支持。
modelDrivenStack。	modelDriven , basicStack。	包含 basicStack 提供的所有功能，此外还添加对模型驱动的支持。
chainStack。	chain , basicStack。	包含 basicStack 提供的所有功能，此外还添加对 action 链的支持。
i18nStack。	i18n , basicStack。	包含 basicStack 提供的所有功能，此外还添加国际化的支持。
paramsPrepareParamsStack。	exception , alias , params , servletConfig , prepare , i18n , chain , modelDriven , fileUpload , checkbox , staticParams , params , conversionError , validation。	当 Action 的 prepare()方法被调用时，用于载入请求参数并用请求参数来重写一些其他载入的数据。它的一个典型应用是更新对象。
defaultStack。	exception , alias , servletConfig , prepare , i18n , chain , debugging , profiling , .. , scopedModelDriven , modelDriven , fileUpload , checkbox , staticparams , .. , params , conversionError , validation , workflow。	默认拦截器栈，为 Struts 2 框架中绝大多数的应用提供了所需的功能。
completeStack。	defaultStack。	defaultStack 的别名，用于向后兼容 Web Work 的应用。
executeAndWaitStack。	execAndWait , defaultStack。	包含 defaultStack 提供的所有功能，此外还添加异步支持 Action 的功能。

拦截器栈的应用

- Struts 2中封装请求参数到action、数据验证和文件上传等功能是由系统默认的defaultStack中的拦截器实现的。所以，我们定义的拦截器需要引用系统默认的拦截器defaultStack。
- 如果希望包下所有action都使用自定义的拦截器，可通过<default-interceptor-ref name=“自定义拦截器名”/>，把拦截器定义为默认拦截器。
- **注意：**每个包只能指定一个默认拦截器。另外，一旦我们为包中的某个action显式指定了某个拦截器，则默认拦截器不会起作用。

自定义拦截器

```
<package name="itcast" namespace="/test" extends="struts-default">
  <interceptors>
    <interceptor name="permission"
      class="cn.itcast.aop.PermissionInterceptor" />
    <interceptor-stack name="permissionStack">
      <interceptor-ref name="defaultStack" />
      <interceptor-ref name="permission" />
    </interceptor-stack>
  </interceptors>
  <action name="helloworld_*"
    class="cn.itcast.action.HelloWorldAction" method="{1}">
    <result name="success">/WEB-INF/page/hello.jsp</result>
    <interceptor-ref name="permissionStack"/>
  </action>
</package>
```

自定义拦截器实现接口

Struts 2提供了一些接口或类供程序员自定义拦截器。例如，Struts 提供了`com.opensymphony.xwork2.interceptor.Interceptor`接口，程序员只要实现该接口就可自定义拦截器类编写。该接口代码如下：

```
public interface Interceptor extends Serializable {  
    void init();  
    String intercept(ActionInvocation invocation) throws Exception;  
    void destroy();  
}
```

该接口中有如下三种方法。

- **init()**：该方法在拦截器被实例化之后、拦截器执行之前调用。该方法只被执行一次，主要用于初始化资源。
- **intercept(ActionInvocation invocation)**：该方法用于实现拦截的动作。该方法有个参数，用该参数调用`invoke()`方法，将控制权交给下一个拦截器，或者交给Action类的方法。
- **destroy()**：该方法与`init()`方法对应，拦截器实例被销毁之前调用。用于销毁在`init()`方法中打开的资源。

自定义拦截器抽象类

除了Interceptor接口之外，Struts 2框架还提供了AbstractInterceptor类，该类实现了Interceptor接口，并提供了init()方法和destroy()方法的空实现。在一般的拦截器实现中，都会继承该类，因为一般实现的拦截器是不需要打开资源的，故无须实现这两种方法，继承该类会更简洁。该类的代码实现为：

```
public abstract class AbstractInterceptor implements Interceptor {  
    public void init() {  
    }  
    public void destroy() {  
    }  
    public abstract String intercept(ActionInvocation invocation) throws Exception;  
}
```

显而易见，实现Interceptor接口要实现三个方法，在不需要重写init()与destroy()这两个方法时，继承AbstractInterceptor类更简单。

拦截器的举例

拦截器举例1:

测试拦截器与Action执行先后关系。

拦截器举例（一）

首先创建项目（InterceptorTest）、加载Struts 2类库及修改web.xml，创建自定义拦截器类“MyInterceptor.java”，编写代码如下：

```
package org.interceptor;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;

public class MyInterceptor extends AbstractInterceptor {
    public String intercept(ActionInvocation arg0) throws Exception {
        System.out.println("我在Action前执行---->");
        String result=arg0.invoke();
        System.out.println("我在Action后执行---->");
        return result;
    }
}
```

拦截器举例（一）

创建Action类“TestAction.java”，编写代码如下：

```
package org.action;
import com.opensymphony.xwork2.ActionSupport;

public class TestAction extends ActionSupport {
    public String execute() throws Exception {
        System.out.println("我在Action中执行---->");
        return NONE; //不做任何跳转
    }
}
```

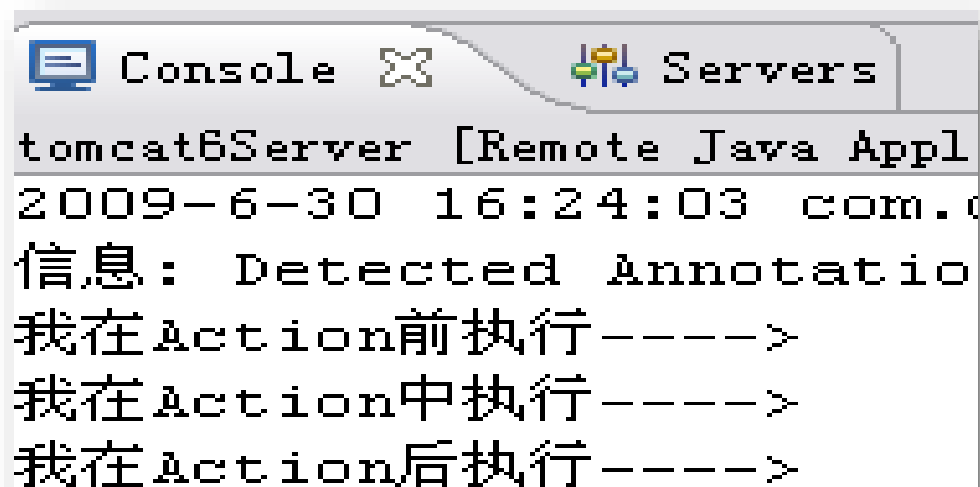
拦截器举例 (一)

在struts.xml中配置action配置及拦截器配置，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="default" extends="struts-default">
      <interceptors>
        <interceptor name="myInterceptor"
          class="org.interceptor.MyInterceptor"/>
      </interceptors>
      <action name="test" class="org.action.TestAction">
        <interceptor-ref name="defaultStack"></interceptor-ref>
        <interceptor-ref name="myInterceptor"></interceptor-ref>
      </action>
    </package>
  </struts>
```

拦截器举例（一）

做这些编程工作后，部署项目并启动Web服务器，在浏览器中输入“<http://localhost:8080/InterceptorTest/test.action>”请求，再查看控制台，出现如图所示的界面，可见，自定义拦截器起了作用，在Action执行前后分别输出了一句话。



The screenshot shows the Eclipse IDE's console window. The title bar includes 'Console' and 'Servers'. The console output is as follows:

```
tomcat6Server [Remote Java Appl  
2009-6-30 16:24:03 com.o  
信息: Detected Annotatio  
我在Action前执行---->  
我在Action中执行---->  
我在Action后执行---->
```

Eclipse演示

拦截器的举例

拦截器举例2:

测试拦截器控制Action执行流程。

拦截器举例（二）

下面来配置拦截器，如果输入框中输入的内容是“hello”，返回当前页面。实现该功能只需要在原项目的基础上配置拦截器即可。首先编写拦截器实现类，代码如下：

```
package org.tool;
import org.action.StrutsAction;
import com.opensymphony.xwork2.Action;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;
public class MyInterceptor extends AbstractInterceptor {
    public String intercept(ActionInvocation arg0) throws Exception {
        // 得到StrutsAction类对象
        StrutsAction action=(StrutsAction)arg0.getAction();
        // 如果Action类中的name属性的值为“hello”，返回错误页
        if(action.getName().equals("hello")) {
            return Action.ERROR;
        }
        // 继续执行其他拦截器或Action类中的方法
        return arg0.invoke();
    }
}
```


拦截器举例（二）

在struts.xml配置文件中拦截器配置，修改后的代码如下：

```
<?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>
    <package name="default" extends="struts-default">
        <interceptors>
            <interceptor name="myInterceptor" class="org.tool.MyInterceptor"></interceptor>
        </interceptors>
        <default-interceptor-ref name=""></default-interceptor-ref>
        <action name="struts" class="org.action.StrutsAction">
            <result name="success">/welcome.jsp</result>
            <result name="error">/hello.jsp</result>
            <result name="input">/hello.jsp</result>
            <!--拦截配置在result后面-->
            <!--使用系统默认拦截器栈-->
            <interceptor-ref name="defaultStack"></interceptor-ref>
            <!--配置拦截器-->
            <interceptor-ref name="myInterceptor"></interceptor-ref>
        </action>
    </package>
</struts>
```

拦截器举例（二）

经过配置后，重新部署项目，在运行界面输入“hello”，也会经过拦截返回到当前页面，分别如左右图所示。

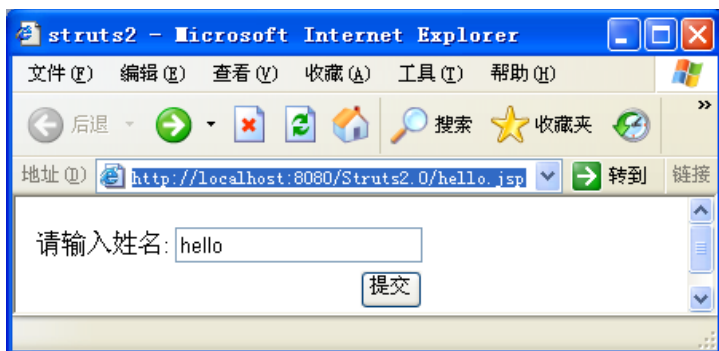


图 运行界面

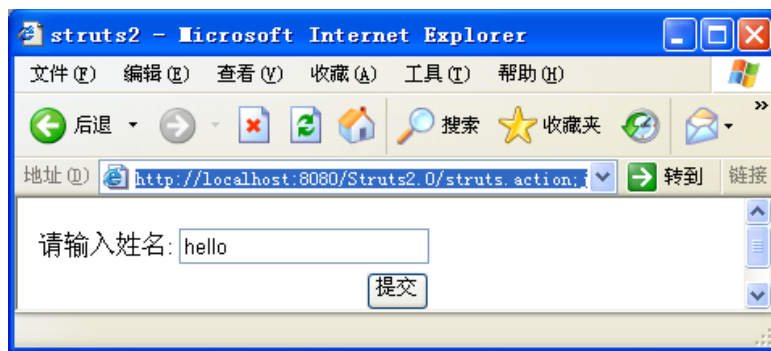


图 提交后返回当前页面

目录

- Struts 2体系架构
- Struts 2工作原理
- Struts 2组件功能分析
- Struts基本应用实例
- Action实现
- Struts配置文件
- 拦截器
- 输入校验
- 文件上传
- Struts综合应用实例

输入校验

在**struts 2**中，我们可以实现对**action**的**所有方法**进行校验或者对**action**的**指定方法**进行校验。

对于输入校验，**struts 2**通常提供了两种实现方法：

1. 基于手工编写代码验证。
2. 基于**XML**校验器配置方式。

手工代码编写实现action输入校验

通过在**action**中提供**validate()** 方法实现输入校验， **validate()**方法会校验**action**中所有方法。

当某个数据校验失败时，调用**addFieldError()**方法往系统的**fieldErrors**添加校验失败信息（为了使用**addFieldError()**方法，**action**可以继承**ActionSupport**）。

如果系统的**fieldErrors**包含失败信息，**struts**会将请求转发到名为**input**的**result**。

输入校验手工编写示例

```
➤ public void validate() {  
➤     if(this.mobile==null || "".equals(this.mobile.trim())) {  
➤         this.addFieldError("mobile", "手机号不能为空");  
➤     } else {  
➤         if(!Pattern.compile("^1[358]\\d{9}").matcher(this.mobile  
➤             .trim()).matches()) {  
➤             this.addFieldError("mobile", "手机号的格式不正确");  
➤         }  
➤     }  
➤ }
```

➤ 验证失败后，请求转发至input视图：

```
➤ <result name="input">/WEB-INF/page/addUser.jsp</result>
```

手工代码编写实现action指定方法输入校验

通过**validateXxx()**方法实现，**validateXxx()**只会校验action中方法名为**Xxx**的方法。其中**Xxx**的第一个字母要大写。

当某个数据校验失败时，调用**addFieldError()**方法往系统的**fieldErrors**添加校验失败信息（为了使用**addFieldError()**方法，**action**可以继承**ActionSupport**）。

如果系统的**fieldErrors**包含失败信息，**Struts**会将请求转发到名为**input**的**result**。

手工代码编写实现action指定方法输入校验

validateXxx()方法使用例子:

```
public String add() throws Exception{ return "success";}
```

```
public void validateAdd() {  
    if(username==null && "".equals(username.trim()))  
        this.addFieldError("username", "用户名不能为空");  
}
```

验证失败后, 请求转发至input视图:

```
<result name="input">/WEB-INF/page/addUser.jsp</result>
```


输入校验的执行流程

- 1、类型转换器对请求参数执行类型转换，并把转换后的值赋给**action**中的属性。
- 2、如果在执行类型转换的过程中出现异常，**conversionError**拦截器将异常信息添加到**fieldErrors**里。不管类型转换是否会出现异常，都会进入第3步。
- 3、系统先调用**action**中的**validateXxx()**方法，**Xxx**为方法名。
- 4、再调用**action**中的**validate()**方法。
- 5、经过上面4步，如果系统中的**fieldErrors**存在错误信息（即存放错误信息的集合的**size**大于0），系统自动将请求转发至名称为**input**视图。如果系统中的**fieldErrors**没有任何错误信息，系统将执行**action**中的处理方法。

数据校验举例

数据校验举例：
校验用户输入的登录信息

数据校验举例



输入校验项目开发文件

- login.jsp (登录界面)
- success.jsp (登录成功界面)
- LoginAction (业务处理action)
- struts.xml (Struts 2配置文件)
- web.xml (项目配置文件)

➤ login.jsp

- `<%@ page language="java" pageEncoding="UTF-8"%>`
- `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">`
- `<%@taglib prefix="s" uri="/struts-tags"%>`
- `<html>`
- `<head>`
- `<title>登录页面</title>`
- `</head>`
- `<body>`
- `<s:form action="login.action" method="post">`
- `<table width="60%" height="76" border="0">`
- `<!-- 各标签定义 -->`
- `<s:textfield name="username" label="用户名"/>`
- `<s:password name="password" label="密 码" />`
- `<s:submit value="登录" align="center"/>`
- `</table>`
- `</s:form>`
- `</body>`
- `</html>`

➤ **success.jsp**

- **<%@ page language="java" contentType="text/html; charset=UTF-8"%>**
- **<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">**
- **<html>**
- **<head>**
- **<title>登录成功</title>**
- **</head>**
- **<body>**
- **<!-- 取得request中用户名值 -->**
- **\${requestScope.user}, 欢迎您~~**
- **</body>**
- **</html>**

➤ LoginAction.java

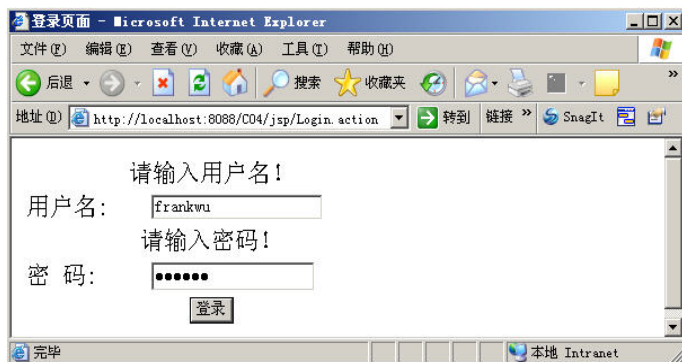
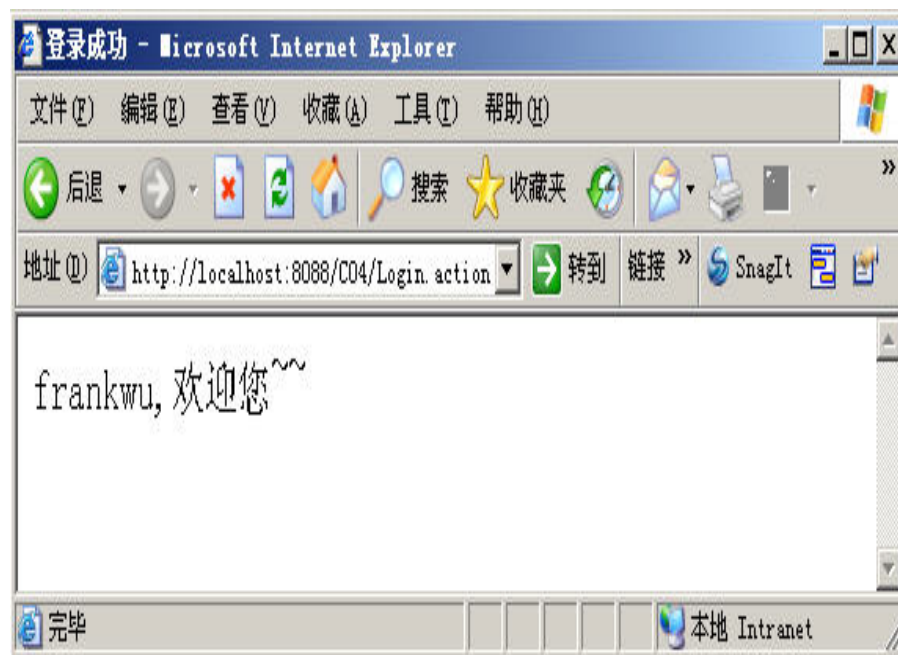
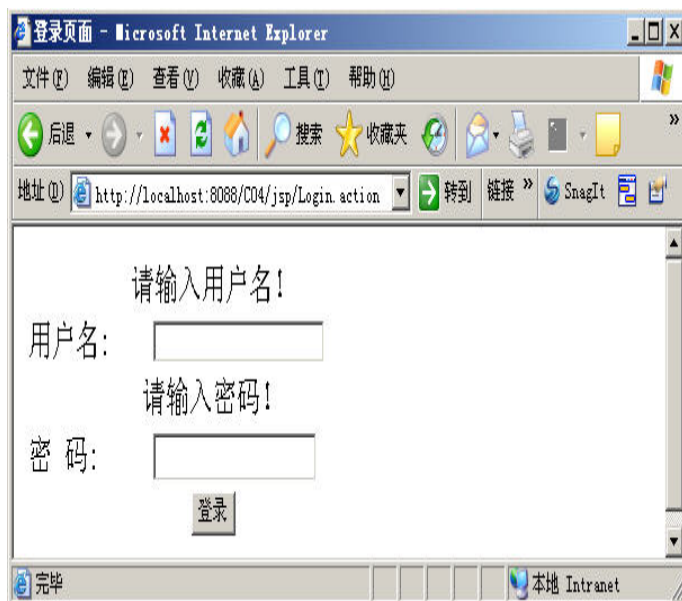
```
➤ public class LoginAction extends ActionSupport {
➤     private String username;
➤     private String password;
➤     public void validate() {
➤         if (getUsername() == null || getUsername().trim().equals("")) {
➤             addFieldError("username", "请输入用户名! ");
➤         }
➤         if (getPassword() == null || getPassword().trim().equals("")) {
➤             addFieldError("password", "请输入密码! ");
➤         }
➤     }

➤     public String execute() throws Exception {
➤         if(!username.equals("HelloWorld")) {
➤             Map request=(Map)ActionContext.getContext().get("request");
➤             request.put("user", getUsername());
➤             return SUCCESS;
➤         } else {
➤             return ERROR;
➤         }
➤     }
➤ }
```

➤ **struts.xml**

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


登录校验结果测试



基于XML配置方式实现输入校验

使用基于XML配置方式实现输入校验时，通过提供校验文件实现校验。校验文件和action类放在同一个包下，文件的取名格式为：**ActionClassName-validation.xml**，其中**ActionClassName**为action的简单类名，**-validation**为固定写法。

例如，Action类为cn.itcast.UserAction，那么该文件的取名应为：**UserAction-validation.xml**。

基于XML配置方式实现输入校验

XML校验文件的模版:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork
Validator 1.0.3//EN" "http://www.opensymphony.com/xwork/xwork-
validator-1.0.3.dtd">
<validators>
  <field name="校验字段名">
    <field-validator type="校验器">
      <param name="参数名">参数值</param>
      <message>提示错误信息值</message>
    </field-validator>
  </field>
</validators>
```

基于XML配置方式实现输入校验

- **<field>指定action中要校验的属性，<field-validator>指定校验器，指定的校验器是由系统提供的，系统提供了验证需求的校验器，定义可以在xwork-core-2.x.jar中com.opensymphony.xwork2.validator.validators下的default.xml中找到。**
- **根据校验器类型type不同，<param>和<message>指定不同的参数名和参数值、以及校验失败信息。**

struts 2提供的常用校验器

- **requiredstring** (必填字符串校验器)
- **required** (必填校验器)
- **stringlength** (字符串长度校验器)
- **regex** (正则表达式校验器)
- **int** (整数校验器)
- **email** (邮件地址校验器)
- **url** (网址校验器)
- **date** (日期校验器)

Struts 2常用校验器

(1) 必填字符串校验器

```
<validators>
  <!-- 需要校验的字段名 -->
  <field name="name">
    <!-- 验证字符串不能为空，即必填 -->
    <field-validator type="requiredstring">
      <!-- 去空格 -->
      <param name="trim">true</param>
      <!-- 错误提示信息 -->
      <message>姓名是必需的！ </message>
    </field-validator>
  </field>
</validators>
```

Struts 2常用校验器

(2) 必填校验器

该校验器的名字是required，也就是<field-validator>属性中的type=“required”，该校验器要求指定的字段必须有值，除了验证字符串以外，还可以验证其它字段。如果把上例改为必填校验器，其代码应为：

```
<validators>
  <!-- 需要校验的字段名 -->
  <field name="name">
    <!-- 验证字符串必填 -->
    <field-validator type="required">
      <!-- 错误提示信息 -->
      <message>姓名是必需的! </message>
    </field-validator>
  </field>
</validators>
```

Struts 2常用校验器

(3) 整数校验器

该校验器的名字是int，该校验器要求字段的整数值必须在指定范围内，故其有min和max参数。如果有个age输入框，要求其必须是整数，且输入值必须在18与100之间，该校验器的配置应该为：

```
<validators>
  <!-- 需要校验的字段名 -->
  <field name="age">
    <field-validator type="int">
      <!-- 年龄最小值 -->
      <param name="min">18</param>
      <!-- 年龄最大值 -->
      <param name="max">100</param>
      <!-- 错误提示信息 -->
      <message>年龄必须在18至100之间</message>
    </field-validator>
  </field>
</validators>
```


Struts 2常用校验器

(4) 日期校验器

该校验器的名字是date，该校验器要求字段的日期值必须在指定范围内，故其有min和max参数。其配置格式如下：

```
<validators>
  <field name="date">
    <field-validator type="date">
      <!-- 日期最小值-->
      <param name="min">1980-01-01</param>
      <!-- 日期最大值-->
      <param name="max">2009-12-31</param>
      <!-- 错误提示信息-->
      <message>日期必须在1980-01-01至2009-12-31之间</message>
    </field-validator>
  </field>
</validators>
```

Struts 2常用校验器

(5) 邮件地址校验器

该校验器的名称是email，该校验器要求字段的字符如果非空，就必须是合法的邮件地址。如下面的代码：

```
<validators>
  <!-- 需要校验的字段的字段名 -->
  <field name="email">
    <field-validator type="email">
      <message>必须输入有效的电子邮件地址 </message>
    </field-validator>
  </field>
</validators>
```

Struts 2常用校验器

(6) 网址校验器

该校验器的名称是url，该校验器要求字段的字符如果非空，就必须是合法的URL地址。如下面的代码：

```
<validators>
  <!-- 需要校验的字段名 -->
  <field name="url">
    <field-validator type="url">
      <message>必须输入有效的网址 </message>
    </field-validator>
  </field>
</validators>
```

Struts 2常用校验器

(7) 字符串长度校验器

该校验器的名称是stringlength，该校验器要求字段的长度必须在指定的范围内，一般用于密码输入框。如下面的代码：

```
<validators>
  <!-- 需要校验的字段名 -->
  <field name="password">
    <field-validator type="stringlength">
      <!-- 长度最小值 -->
      <param name="minLength">6</param>
      <!-- 长度最大值 -->
      <param name="maxLength">20</param>
      <!-- 错误提示信息 -->
      <message>密码长度必须在6到20之间</message>
    </field-validator>
  </field>
</validators>
```

Struts 2常用校验器

(8) 正则表达式校验器

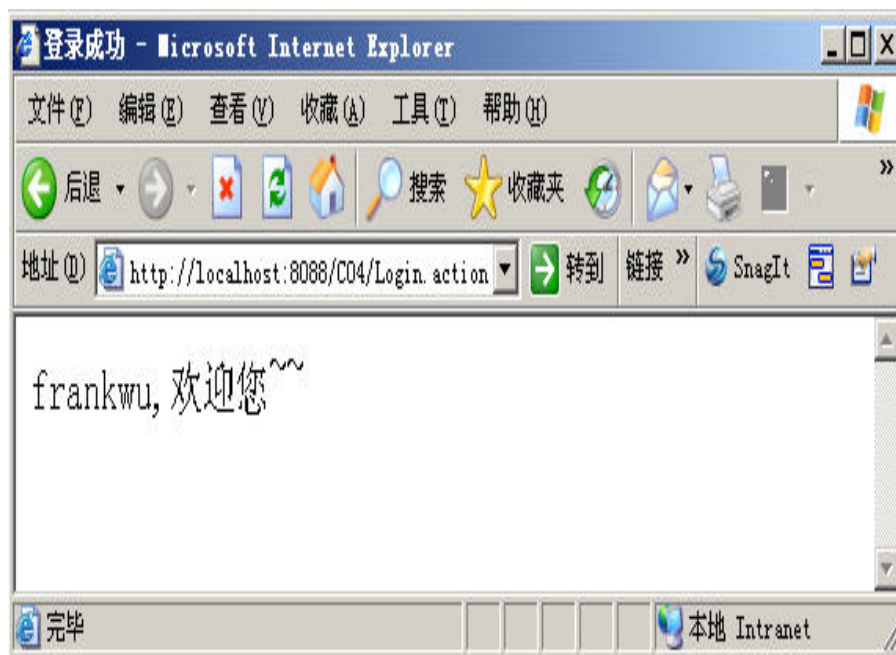
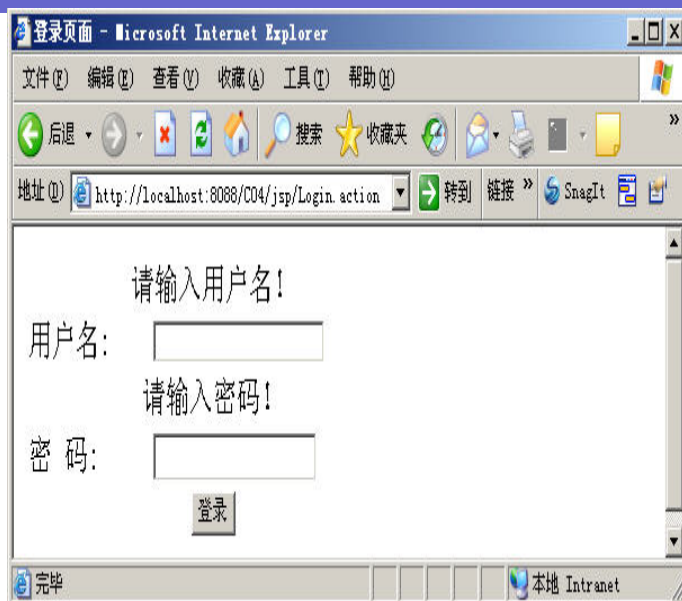
该校验器的名称是regex，它检查被校验字段是否匹配一个正则表达式。如下面的代码：

```
<validators>
  <field name="xh">
    <field-validator type="regex">
      <param name="expression"><![CDATA[(\d{6})]]></param>
      <message>学号必须是6位的数字</message>
    </field-validator>
  </field>
</validators>
```

数据校验举例（XML校验）

数据校验举例：
重新校验用户登录（XML验证）

登录校验结果测试 (XML校验)



目录

- Struts 2体系架构
- Struts 2工作原理
- Struts 2组件功能分析
- Struts基本应用实例
- Action实现
- Struts配置文件
- 拦截器
- 输入校验
- 文件上传
- Struts综合应用实例

Struts 2文件上传

✧ 上传单个文件

下面举例实现文件的上传并说明需要注意的步骤。该例中把要上传的文件放在指定的文件夹下（D:/upload），所以需要提前在D盘下建立upload文件夹。

依然根据原始的步骤来开发该实例。

◇ 1. 建立项目

打开MyEclipse，建立一个Web项目，命名为“StrutsUploadSingle”。

◇ 2. 加载Struts 2的基本类库

右键选择项目，进入MyEclipse，选择Add Struts Capabilities。

◇ 3. 修改web.xml

加载Struts 2核心类库后，MyEclipse已帮助我们修改web.xml，增加了获取用户请求的过滤器。

上传单个文件

◆ 4. 修改index.jsp

在创建项目的时候，在项目的WebRoot下会自动生成一个index.jsp文件，可应用该文件，修改其中内容，也可以自己建立JSP文件。这里用该index.jsp文件，修改其中内容即可。代码实现为：

```
<%@ page language="java" pageEncoding="utf-8"%>
<%@ taglib uri="/struts-tags" prefix="s"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>文件上传</title>
</head>
<body>
    <s:form action="upload.action" method="post" enctype="multipart/form-data">
        <s:file name="upload" label="上传的文件"></s:file>
        <s:submit value="上传"></s:submit>
    </s:form>
</body>
</html>
```

上传单个文件

◇ 5. Action类

```
private File upload;
private String uploadFileName;

public String execute() throws Exception {
    if(upload!=null) {
        InputStream is=new FileInputStream(getUpload());
        OutputStream os=new FileOutputStream("d:\\upload\\"+uploadFileName);
        byte buffer[]=new byte[1024];
        int count=0;
        while((count=is.read(buffer))>0) {
            os.write(buffer,0,count);
        }
        os.close();
        is.close();
        return SUCCESS;
    }
    return ERROR;
}
```

上传单个文件

◇ 6. struts.xml文件

struts.xml配置代码如下：

```
<?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="default" extends="struts-default">
    <action name="upload" class="action.UploadAction">
      <result name="success">/success.jsp</result>
    </action>
  </package>
</struts>
```

上传单个文件

◇ 7. 建立 **success.jsp**

上传成功后，跳转到成功页面，代码如下：

```
<%@ page language="java" pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>成功页面</title>
</head>
<body>
    恭喜你！ 上传成功
</body>
</html>
```

上传单个文件

◆ 8. 部署运行

部署项目，在浏览器中输入<http://localhost:8080/StrutsUploadSingle/index.jsp>，出现如图（左）所示的界面，选择要上传的文件，单击【上传】按钮，会跳转到如图（右）所示的界面。打开D盘，在upload文件夹下就可以看到已上传的文件。



图 运行界面



图 成功界面

多文件上传

➤ 修改index.jsp:

```
<%@ page language="java" pageEncoding="utf-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>文件上传</title>
</head>
<body>
  <s:form action="upload.action" method="post" enctype="multipart/form-data">
    <!-- 这里上传三个文件,这里可以是任意多个-->
    <s:file name="upload" label="上传的文件一"></s:file>
    <s:file name="upload" label="上传的文件二"></s:file>
    <s:file name="upload" label="上传的文件三"></s:file>
    <s:submit value="上传"></s:submit>
  </s:form>
</body>
</html>
```

多文件上传

➤ 修改Action类 (UploadAction.java) :

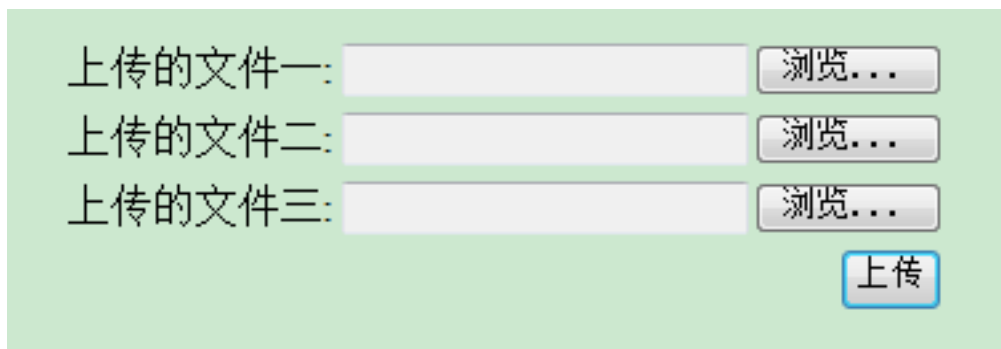
```
private List<File> upload; //上传的文件内容, 由于是多个, 用List集合  
private List<String> uploadFileName; //文件名
```

```
public String execute() throws Exception {  
    if(upload!=null) {  
        for (int i=0; i < upload.size(); i++) { //遍历, 对每个文件进行读/写操作  
            InputStream is=new FileInputStream(upload.get(i));  
            OutputStream os= new FileOutputStream("d:\\upload\\"+getUploadFileName().get(i));  
            byte buffer[]=new byte[1024];  
            int count=0;  
            while((count=is.read(buffer))>0) {  
                os.write(buffer,0,count);  
            }  
            os.close();  
            is.close();  
        }  
        return SUCCESS;  
    }  
    return ERROR;  
}
```


多文件上传

◆ 部署运行

部署项目，在浏览器中输入<http://localhost:8080/StrutsUploadMultiple/index.jsp>，出现如图所示的界面。打开D盘，在upload文件夹下就可以看到已上传的多个文件。



The screenshot shows a web form with a light green background. It contains three rows of file upload controls. Each row consists of a text input field and a '浏览...' (Browse...) button. The labels for the rows are '上传的文件一:' (Upload file 1:), '上传的文件二:' (Upload file 2:), and '上传的文件三:' (Upload file 3:). Below these three rows is a single '上传' (Upload) button.

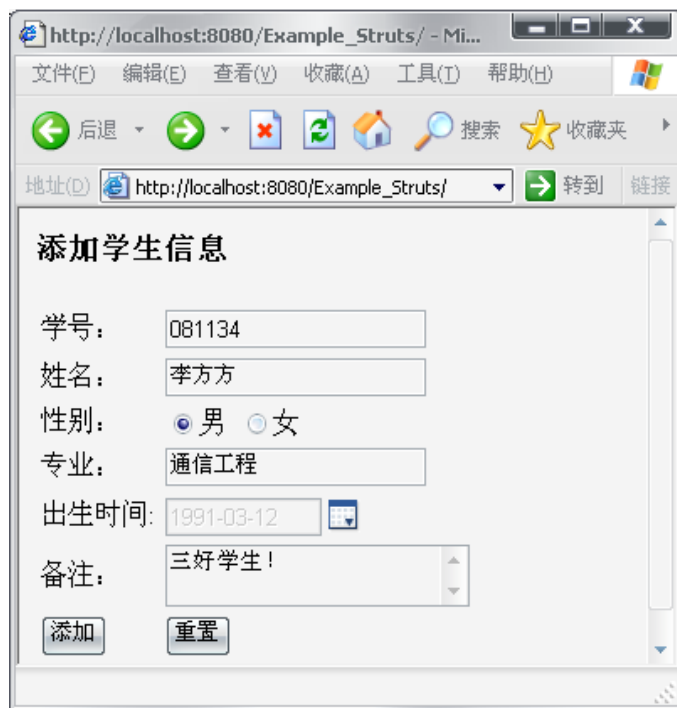
图 运行界面

目录

- Struts 2体系架构
- Struts 2工作原理
- Struts 2组件功能分析
- Struts基本应用实例
- Action实现
- Struts配置文件
- 拦截器
- 输入校验
- 文件上传
- **Struts综合应用实例**

Struts 2综合应用实例——添加学生信息

在该实例中，通过构建一个添加学生信息项目，来综合应用Struts 2的知识点，包括标签、Action和Struts 2配置等。首先来看添加学生信息的界面，如下图所示。



The screenshot shows a web browser window with the address bar displaying 'http://localhost:8080/Example_Struts/'. The browser's menu bar includes '文件(F)', '编辑(E)', '查看(V)', '收藏(A)', '工具(T)', and '帮助(H)'. The toolbar contains icons for '后退', '前进', '刷新', '主页', '搜索', and '收藏夹'. The address bar shows '地址(D) http://localhost:8080/Example_Struts/' and a '转到' button. The main content area is titled '添加学生信息' and contains the following form fields:

- 学号: 081134
- 姓名: 李方方
- 性别: ☒ 男 ☐ 女
- 专业: 通信工程
- 出生时间: 1991-03-12
- 备注: 三好学生!

At the bottom of the form are two buttons: '添加' and '重置'.

图 添加学生信息界面

Struts 2综合应用实例——添加学生信息

◆ 1. 建立数据库和学生数据表

首先建立数据库xscj，建立学生表，表名xsb，表结构字段与学生信息的提交界面对应。

◆ 2. 建立Web项目

打开MyEclipse，建立一个Web项目，命名为“Struts_Student”。

◆ 3. 加载Struts 2的基本类库

右键选择项目，进入MyEclipse，选择Add Struts Capabilities。

◆ 4. 修改web.xml

加载Struts 2核心类库后，MyEclipse已帮助我们修改web.xml，增加了获取用户请求的过滤器。

◆ 5. 建立stu.jsp文件

在项目的WebRoot文件夹下建立stu.jsp文件，显示学生注册信息界面。

Struts 2综合应用实例——添加学生信息

◆ 6. 建立表对应的JavaBean和DBConn类

在src文件夹下新建包“org.model”，该包下建class文件，命名为“Xsb”。该类中有6个字段，分别为xh、xm、xb、zy、cssj和bz，并生成它们的getter和setter方法，代码如下：

```
package org.model;
import java.sql.Date;
public class Xsb {
    private String xh;
    private String xm;
    private byte xb;
    private String zy;
    private Date cssj;
    private String bz;
    // 生成它们的getter和setter方法
}
```

在src文件夹下建立包org.work，在该包下建立class文件，命名为“DBConn”，该类负责数据库链接和学生JavaBean对象的保存。

Struts 2综合应用实例——添加学生信息

◇ 7. 建立Action类SaveAction.java

SaveAction用于实例化一个学生JavaBean对象，并利用DBConn实例调用Save方法，保存JavaBean对象至数据库中。

◇ 8. 创建并配置struts.xml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="default" namespace="/" extends="struts-default">
        <action name="save" class="org.action.SaveAction">
            <result name="success">/success.jsp</result>
            <result name="error">/stu.jsp</result>
        </action>
    </package>
</struts>
```

Struts 2综合应用实例——添加学生信息

◇ 9. 创建success.jsp页面

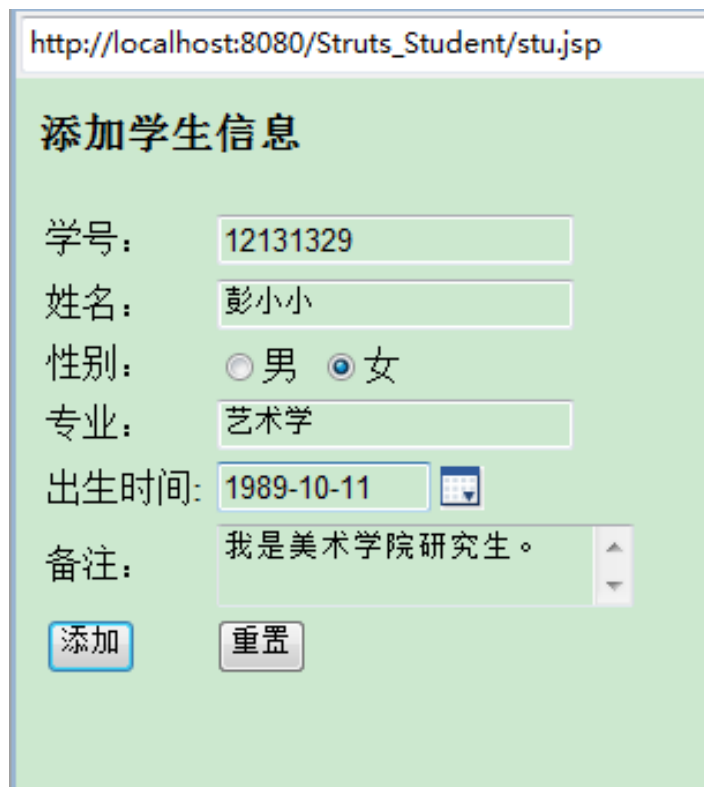
在WebRoot文件夹下创建success.jsp文件，代码如下：

```
<%@ page language="java" pageEncoding="utf-8"%>
<html>
<head>
</head>
<body>
    恭喜你，添加成功！
</body>
</html>
```

Struts 2综合应用实例——添加学生信息

◆ 10. 部署运行

部署后，在浏览器中输入http://localhost:8080/Struts_Student/stu.jsp，可以看到如下图所示界面。输入要添加的学生信息后，单击【添加】按钮，如果添加成功就会跳转到success.jsp页面。



The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/Struts_Student/stu.jsp`. The page title is "添加学生信息" (Add Student Information). The form contains the following fields and controls:

- 学号: (Student ID) Text input field containing "12131329".
- 姓名: (Name) Text input field containing "彭小小".
- 性别: (Gender) Radio buttons for "男" (Male) and "女" (Female). The "女" option is selected.
- 专业: (Major) Text input field containing "艺术学".
- 出生时间: (Date of Birth) Text input field containing "1989-10-11" with a calendar icon to its right.
- 备注: (Remarks) Text area containing "我是美术学院研究生。".
- Buttons: "添加" (Add) and "重置" (Reset).