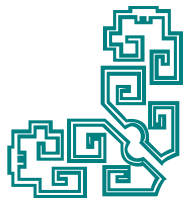
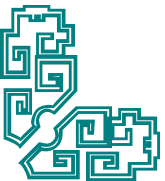
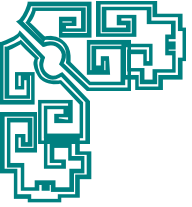




第3章 Struts 2应用

- ♥3.1 Struts 2概述
- ♥3.2 Struts 2基本应用及工作流程
- ♥3.3 Struts 2标签库应用
- ♥3.4 Struts 2拦截器
- ♥3.5 Struts 2国际化应用
- ♥3.6 Struts 2文件上传
- ♥3.7 Struts 2综合应用实例——添加学生信息



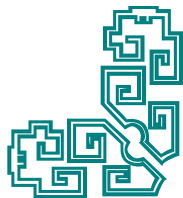
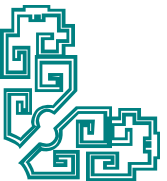


3.1 Struts 2概述

✧3.1.1 使用Struts2的动机

虽然Model2开发模式通过分离系统各部分模块的功能职责，克服了Model1的缺点，但是程序员在编写程序时需要严格遵守Servlet代码的编写规范；并且在实际开发中一旦暴露Servlet API就会大大增加编程的难度，增加开发的工作量。

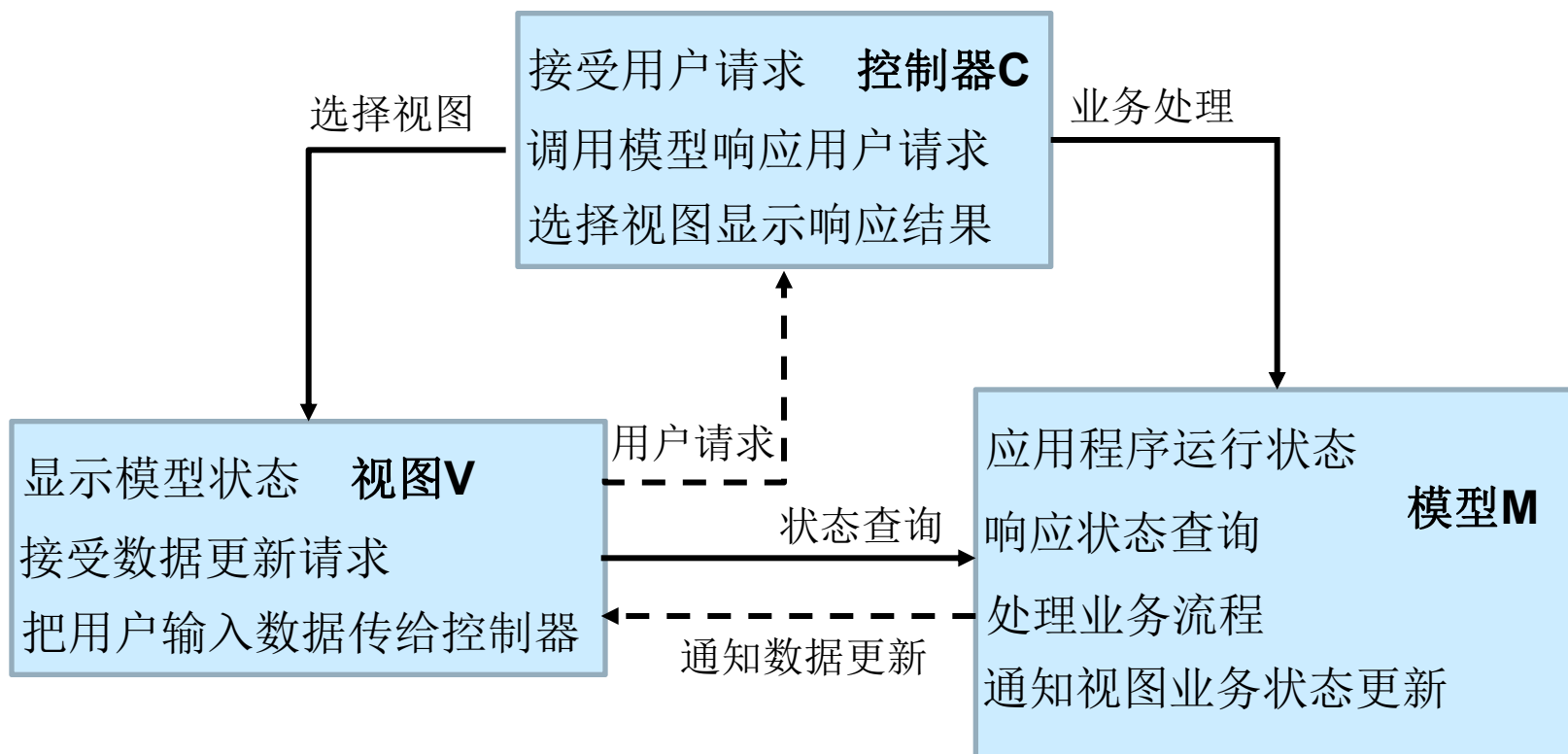
```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class HelloWorld extends HttpServlet{
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ...
    }
    protected void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        doPost(req, res);
    }
}
```

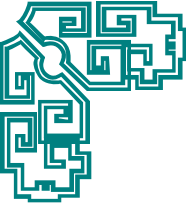


3.1 Struts 2概述

3.1.2 MVC简介

MVC是一种通用的Web软件设计模式，它强制性地把应用程序的数据处理、数据显示、流程控制分开，把应用程序分成三大基本模块：模型（Model）、视图（View）、控制器（Controller）。



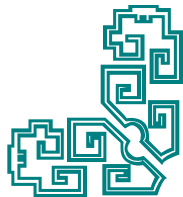
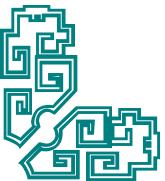


3.1 Struts 2概述

MVC有以下优点：

- ① 多个视图可以对应一个模型。按MVC设计模式，一个模型对应多个视图，可以减少代码的复制及代码的维护量，一旦模型发生改变，也易于维护。
- ② 模型返回的数据与显示逻辑分离。模型数据可以应用任何显示技术，例如，使用JSP页面、Velocity模板或者直接产生Excel文档等。
- ③ 应用被分隔为三层，降低了各层之间的耦合，提供了应用的可扩展性。
- ④ 控制层的概念也很有效，由于它把不同的模型和不同的视图组合在一起，完成不同的请求，因此控制层可以说是包含了用户请求权限的概念。
- ⑤ MVC更符合软件工程化管理的精神。不同的层各司其职，每一层的组件具有相同的特征，有利于通过工程化和工具化产生管理程序代码。

基本应用示例



3.1.3 Struts 2体系结构

◇ 1. Struts 2的基本流程

- ① Web浏览器请求一个资源。
- ② 过滤器Dispatcher查找请求，确定适当的Action。
- ③ 拦截器自动对请求应用通用功能，如验证和文件上传等操作。
- ④ Action的execute方法通常用来存储和（或）重新获得信息（通过数据库）。

⑤ 结果被返回到浏览器。可能是HTML、图片、PDF或其他。

其实，Struts 2框架的应用着重在控制上。简单的流程是：页面→控制器→页面。最重要的是控制器的取数据与处理后传数据的问题。Struts 2的体系结构还可以参考图3.1，更直观地展现出其流程。

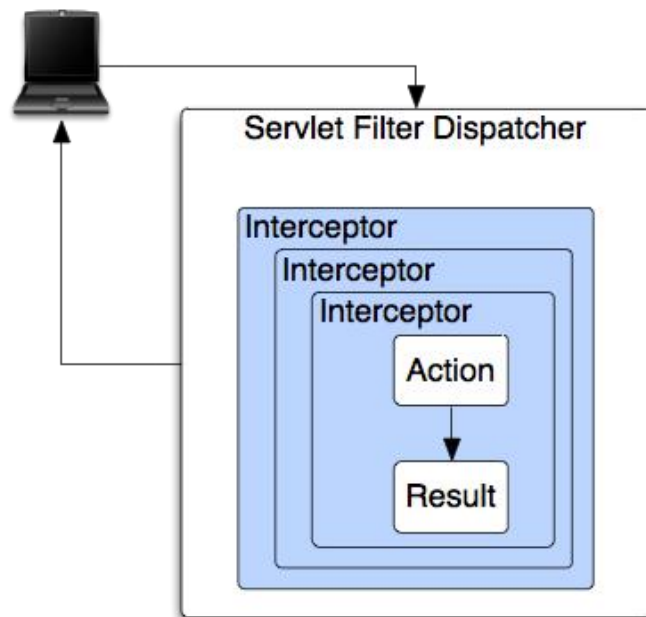


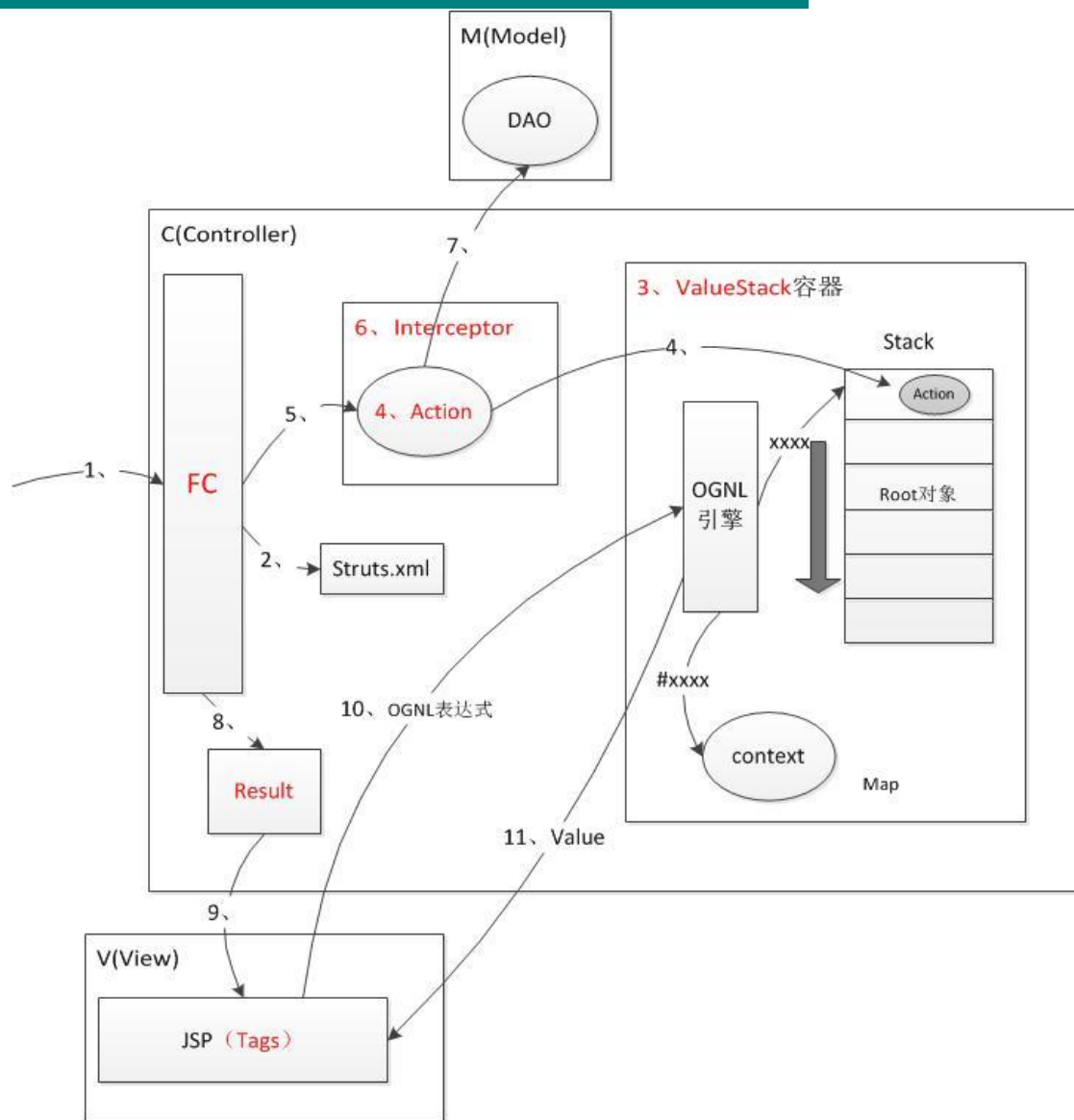
图3.1 Struts 2体系结构

3.1.3 Struts 2体系结构

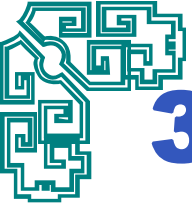
◇ 2. Struts 2的核心组件
包含前端控制器FC、
Action、ValueStack容器、
Result、Interceptor拦截器、
Tags标签6个核心组件。

其中，Result组件扮演
MVC模式下的V角色。

工作原理8句话：
请求提交控制器，
根据配置找Action，
创建VS栈容器，
实例Action放栈顶，
调用Action算输出，
历经层层拦截器，
根据方法返回值，
调用Result做输出。



Action



3.2 Struts 2基本应用及工作流程

✧ 3.2.1 简单Struts 2实例开发

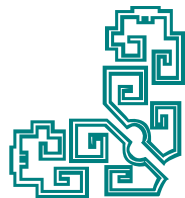
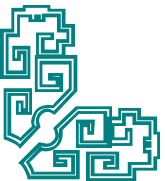
◆ 1. 下载Struts 2框架

MyEclipse 6没有对Struts 2的支持，所以需要用户自己下载Struts 2开发包。登录<http://struts.apache.org/>下载Struts 2完整版，本书使用的是Struts 2.0.14。将下载的Zip文件解压缩，它是一个典型的Web结构。打开其文件夹，里面包含以下4个文件：

- ① apps: 包含基于Struts 2的示例应用，是学习Struts 2非常有用的资料。
- ② docs: 包含Struts 2的相关文档，如Struts 2快速入门、Struts 2文档、API文档等内容。
- ③ lib: 包含Struts 2框架的核心类库，以及Struts 2的第三方插件类库。
- ④ src: 包含Struts 2框架的全部源代码。

◆ 2. 建立一个Web项目

打开MyEclipse，建立一个Web项目，命名为“Struts 2.0”。



3.2.1 简单Struts 2实例开发

3. 加载Struts 2基本类库

下面把这几个类库添加到项目中。

右击项目名，选择【Build Path】→【Configure Build Path】菜单项，出现如图3.2所示的对话框。单击【Add External JARs】按钮，进入下载的Struts 2目录的lib文件夹，选中上面的5个Jar包，单击【OK】按钮完成类库的添加。

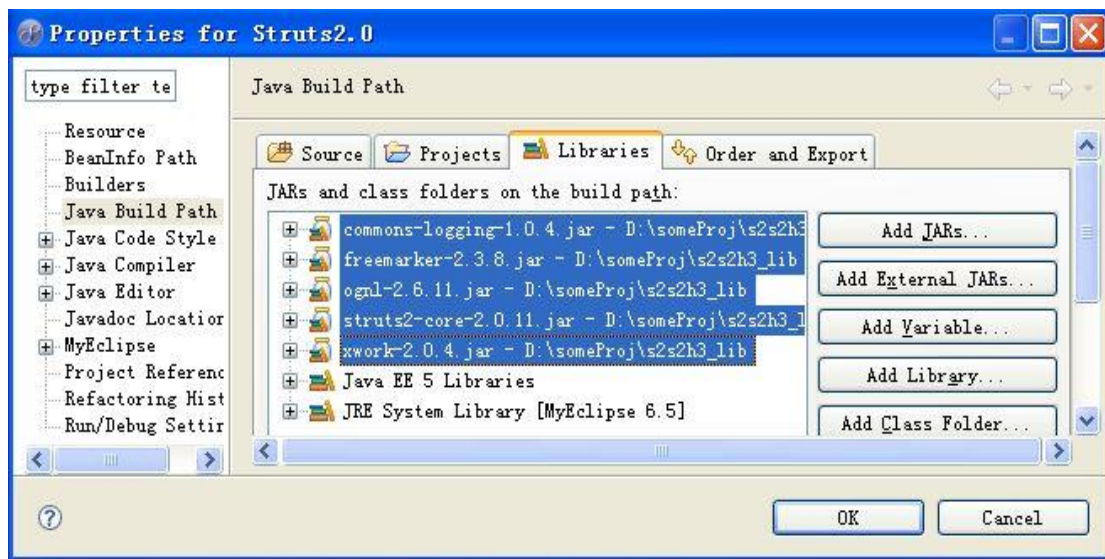


图3.2 添加Struts 2的Jar包



3.2.1 简单Struts 2实例开发

❖ 4. 修改web.xml文件

打开项目中的WebRoot/WEB-INF/web.xml文件，修改其代码如下：

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

```
<web-app version="2.5"
```

```
    xmlns="http://java.sun.com/xml/ns/javaee"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
```

```
    http://java.sun.com/xml/ns/JavaEE/web-app_2_5.xsd">
```

```
    <filter>
```

```
        <filter-name>struts 2</filter-name>
```

```
        <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-
```

```
class>
```

```
    </filter>
```

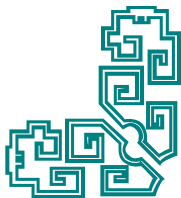
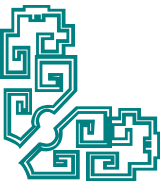
```
    <filter-mapping>
```

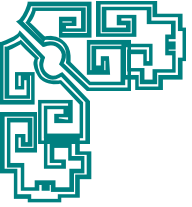
```
        <filter-name>struts 2</filter-name>
```

```
        <url-pattern>/*</url-pattern>
```

```
    </filter-mapping>
```

```
</web-app>
```



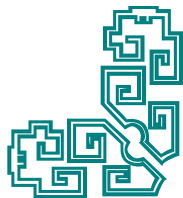
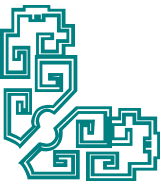


3.2.1 简单Struts 2实例开发

◇ 5. 创建hello.jsp

选择【WebRoot】→【new】→【jsp】菜单项，在File Name中输入文件名“hello.jsp”，修改后的代码如下：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
<head>
    <title>struts 2应用</title>
</head>
<body>
    <form action="struts.action" method="post">
        请输入姓名： <input type="text" name="name"/><br>
        <input type="submit" value="提交"/>
    </form>
</body>
</html>
```

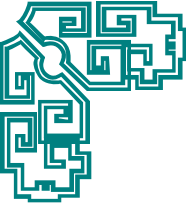


3.2.1 简单Struts 2实例开发

◇ 6. Action实现类

右击src文件夹，选择【new】→【Package】菜单项，在name框中输入包名“org.action”，右击该包，依此类推，建立class，命名为“StrutsAction”，修改后的代码如下：

```
package org.action;
import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class StrutsAction extends ActionSupport{
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name=name;
    }
    public String execute() throws Exception {
        if(!name.equals("HelloWorld")){
            Map request=(Map)ActionContext.getContext().get("request");
            request.put("name",getName());
            return "success";
        }else{
            return "error";
        }
    }
}
```

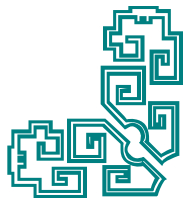
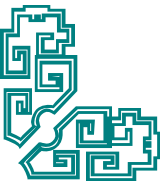


3.2.1 简单Struts 2实例开发

◇ 7. 创建并配置struts.xml文件编译后在webinfo下

任何一个Struts 2程序都不能缺少struts.xml文件，它是Struts 2运行的核心。右击src文件夹，选择【new】→【file】菜单项，在File name框中输入“struts.xml”，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">
        <action name="struts" class="org.action.StrutsAction">
            <result name="success">/welcome.jsp</result>
            <result name="error">/hello.jsp</result>
        </action>
    </package>
</struts>
```



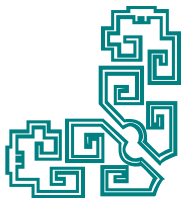
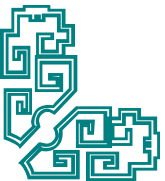


3.2.1 简单Struts 2实例开发

◇ 8. 创建welcome.jsp

创建welcome.jsp，其步骤和上面创建hello.jsp类似，这里不再重复叙述。其代码如下：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<html>
<head>
  <title>struts 2应用</title>
</head>
<body>
  hello <s:property value="#request.name"/>!
</body>
</html>
```

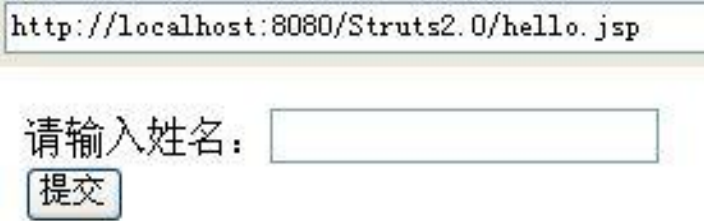


3.2.1 简单Struts 2实例开发

◆ 9. 部署和运行

启动Tomcat后，在浏览器中输入

“<http://localhost:8080/Struts2.0/hello.jsp>”，会看到如图3.3所示的界面。当在输入框中输入“张三”时，会出现如图3.4所示界面。如果输入“HelloWorld”，就会返回当前页面。

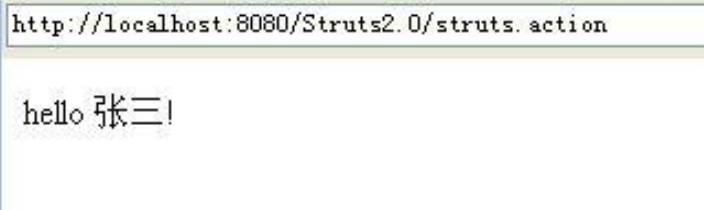


```
http://localhost:8080/Struts2.0/hello.jsp
```

请输入姓名：

提交

图3.3 运行界面



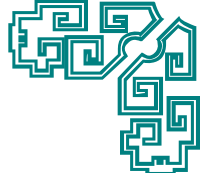
```
http://localhost:8080/Struts2.0/struts.action
```

hello 张三!

图3.4 运行成功界面

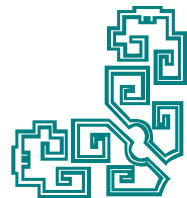
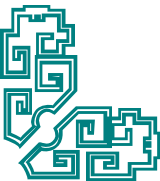


3.2.2 Struts 2 工作流程及各种文件详解



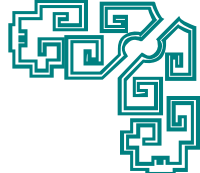
◆ 1. Struts 2 的工作流程

Struts 2 框架中的配置文件 `struts.xml` 会起映射作用，它会根据 “*” 来决定调用用户定义的哪个 `Action` 类。例如在项目 Struts 2.0 中，请求为 `struts.action`，前面 “*” 的部分是 “`struts`”，所以在 `struts.xml` 中有个 `Action` 类的 `name` 为 “`struts`”，这表示该请求与这个 `Action` 来匹配，就会调用该 `Action` 中 `class` 属性指定的 `Action` 类。但是在 Struts 2 中，用户定义的 `Action` 类并不是业务控制器，而是 `Action` 代理，其并没有和 `Servlet API` 耦合。所以 Struts 2 框架提供了一系列的拦截器，它负责将 `HttpServletRequest` 请求中的请求参数解析出来，传入到用户定义的 `Action` 类中。





3.2.2 Struts 2 工作流程及各种文件详解



◇ 2. Struts 2 中各种文件详解

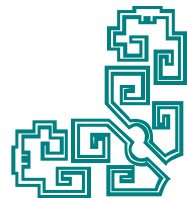
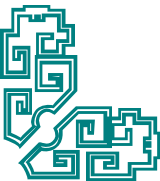
(1) web.xml 文件

后面的webapp标签中配置了下面这样一段：

```
...  
<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>  
...
```

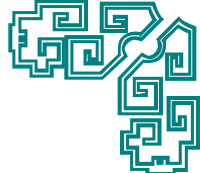
可以看出，里面配置了一个过滤器，那么就先来介绍过滤器的使用。

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





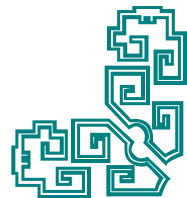
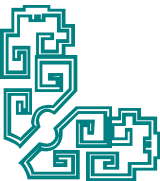
3.2.2 Struts 2 工作流程及各种文件详解



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

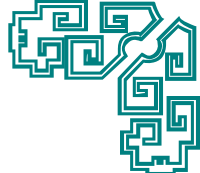
- **init(FilterConfig):** Servlet过滤器的初始化方法，Servlet容器创建Servlet过滤器实例后将调用这个方法。
- **doFilter(ServletRequest,ServletResponse,FilterChain):** 完成实际的过滤操作，当用户请求与过滤器关联的URL时，Servlet容器将先调用过滤器的doFilter方法，返回响应之前也会调用此方法。FilterChain参数用于访问过滤器链上的下一个过滤器。
- **destroy():** Servlet容器在销毁过滤器实例前调用该方法，这个方法可以释放Servlet过滤器占用的资源，过滤器类编写完成后，必须要在web.xml中进行配置，格式如下：

```
<filter>
    <!--自定义的名称-->
    <filter-name>过滤器名</filter-name>
    <!--自定义的过滤器类，注意，这里要在包下，要加包名-->
    <filter-class>过滤器对应类</filter-class>
    <init-param>
        <!--类中参数名称-->
        <param-name>参数名称</param-name>
        <!--对应参数的值-->
        <param-value>参数值</param-value>
    </init-param>
</filter>
```





3.2.2 Struts 2 工作流程及各种文件详解



过滤器的关联方式有3种：与一个URL关联、与一个URL目录下的所有资源关联、与一个Servlet关联。

与一个URL资源关联：

```
<filter-mapping>
```

```
    <!-- 这里与上面配置的名称要相同-->
```

```
    <filter-name>过滤器名</filter-name>
```

```
    <!-- 与该URL资源关联-->
```

```
    <url-pattern>xxx.jsp</url-pattern>
```

```
</filter-mapping>
```

与一个URL目录下的所有资源关联：

```
<filter-mapping>
```

```
    <filter-name>过滤器名</filter-name>
```

```
    <url-pattern>/*</url-pattern>
```

```
</filter-mapping>
```

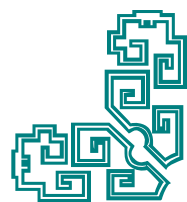
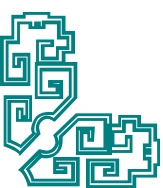
与一个Servlet关联：

```
<filter-mapping>
```

```
    <filter-name>过滤器名</filter-name>
```

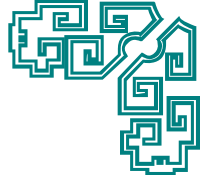
```
    <url-pattern>Servlet名称</url-pattern>
```

```
</filter-mapping>
```





3.2.2 Struts 2 工作流程及各种文件详解



(2) struts.xml 文件

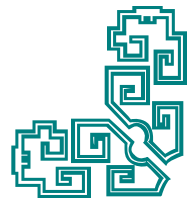
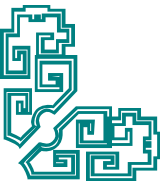
struts.xml 文件通常放在 Web 应用程序的 WEB-INF/classes 目录下，该目录下的 struts.xml 将被 Struts 2 框架自动加载。

struts.xml 文件是一个 XML 文件，文件前面是 XML 的头文件，然后是 <struts> 标签，位于 Struts 2 配置的最外层，其他标签都是包含在它里面的。

(3) package 元素

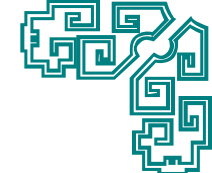
与 Java 中的包不同的是，Struts 2 中的包可以扩展另外的包，从而“继承”原有包的所有定义，并可以添加自己包的特有配置，以及修改原有包的部分配置。从这一点上看，Struts 2 中的包更像 Java 中的类。package 有以下几个常用属性：

- **name**: 该属性是必选的，指定包的名字，这个名字将作为引用该包的键。
- **extends**: 该属性是可选的，允许一个包继承一个或多个先前定义的包。
- **abstract**: 该属性是可选的，将其设置为 true，可以把一个包定义为抽象的。



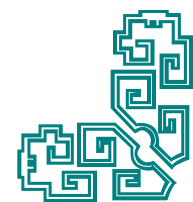
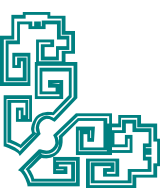


3.2.2 Struts 2 工作流程及各种文件详解



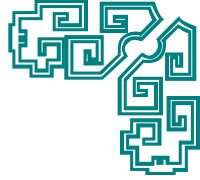
- **namespace**: 该属性是可选的，将保存的**action**配置为不同的名称空间。看下面这个例子：

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





3.2.2 Struts 2 工作流程及各种文件详解



(4) Action 元素

当一个请求匹配到某个 **Action** 名字时，框架就使用这个映射来确定如何处理请求。

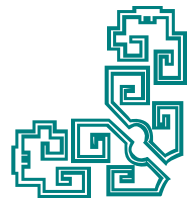
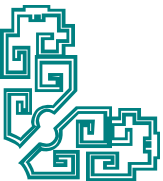
```
<action name="struts" class="org.action.StrutsAction">  
    <result name="success">/welcome.jsp</result>  
    <result name="error">/hello.jsp</result>  
</action>
```

如果一个请求要调用 **Action** 类中的其他方法，就需要在 **Action** 配置中加以配置。例如，如果在 **org.action.StrutsAction** 中有另外一个方法为：

```
public String find() throws Exception{return SUCCESS;}
```

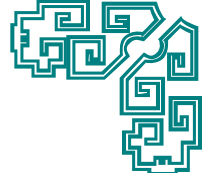
那么如果想要调用这个方法，就必须在 **Action** 中配置 **method** 属性，其配置方法为：

```
<!-- name 值是用来和请求匹配的 -->  
<action name="find" class="org.action.StrutsAction" method="find">  
    <result name="success">/welcome.jsp</result>  
    <result name="error">/hello.jsp</result>  
</action>
```





3.2.2 Struts 2 工作流程及各种文件详解



(5) result 元素

一个result代表一个可能的输出。当Action类中的方法执行完成时，返回一个字符串类型的结果代码，框架根据这个结果代码选择对应的result，向用户输出。

```
<result name ="逻辑视图名" type ="视图结果类型"/>
```

```
    <param name ="参数名">参数值</param>
```

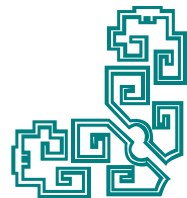
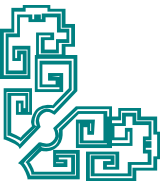
```
</result>
```

param中的name属性有两个值：

- location：指定逻辑视图。
- parse：是否允许在实际视图名中使用OGNL表达式，参数默认为true。

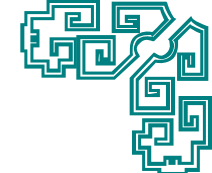
result中的name属性有如下值：

- success：表示请求处理成功，该值也是默认值。
- error：表示请求处理失败。
- none：表示请求处理完成后不跳转到任何页面。
- input：表示输入时如果验证失败应该跳转到什么地方（关于验证后面会介绍）。
- login：表示登录失败后跳转的目标。



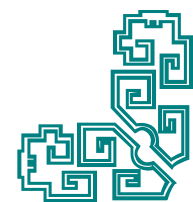
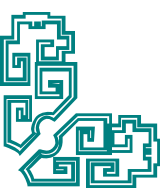


3.2.2 Struts 2 工作流程及各种文件详解



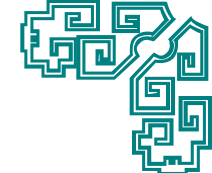
type（非默认类型）属性支持的结果类型有以下几种：

- chain：用来处理Action链。
- chart：用来整合JFreeChart的结果类型。
- dispatcher：用来转向页面，通常处理JSP，该类型也为默认类型。
- freemarker：处理FreeMarker模板。
- httpheader：控制特殊HTTP行为的结果类型。
- jasper：用于JasperReports整合的结果类型。
- jsf：JSF整合的结果类型。
- redirect：重定向到一个URL。
- redirect-action：重定向到一个Action。
- stream：向浏览器发送InputStream对象，通常用来处理文件下载，还可用于返回AJAX数据。
- tiles：与Tiles整合的结果类型。
- velocity：处理Velocity模板。
- xslt：处理XML/XLST模板。
- plaintext：显示原始文件内容，如文件源代码。



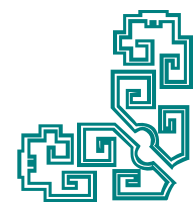
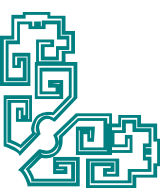


3.2.2 Struts 2 工作流程及各种文件详解



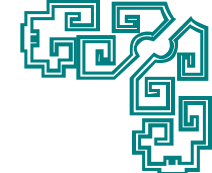
redirect-action类型用于当一个**Action**处理结束后，直接将请求重定向到另一个**Action**。如下列配置：

```
...  
<action name="struts" class="org.action.StrutsAction" >  
    <result name="success">/welcome.jsp</result>  
    <result name="error">/hello.jsp</result>  
</action>  
<action name="login" class="org.action.StrutsAction">  
    <result name="success" type="redirect-action">struts</result>  
</action>  
...
```





3.2.2 Struts 2 工作流程及各种文件详解



(6) ActionSupport 类

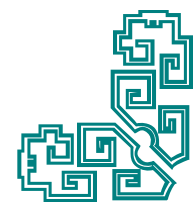
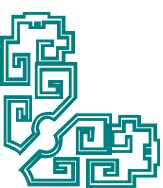
ActionSupport 类为 Action 提供了一些默认实现，主要包括预定义常量、从资源文件中读取文本资源、接收验证错误信息和验证的默认实现。

下面是 ActionSupport 类所实现的接口：

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

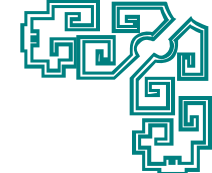
Action 接口同样位于 `com.opensymphony.xwork2` 包，定义了一些常量和一个 `execute()` 方法。

```
public interface Action {  
    public static final String SUCCESS="success";  
    public static final String NONE="none";  
    public static final String ERROR="error";  
    public static final String INPUT="input";  
    public static final String LOGIN="login";  
    public String execute() throws Exception;  
  
}
```





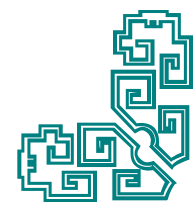
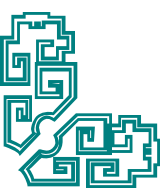
3.2.2 Struts 2 工作流程及各种文件详解



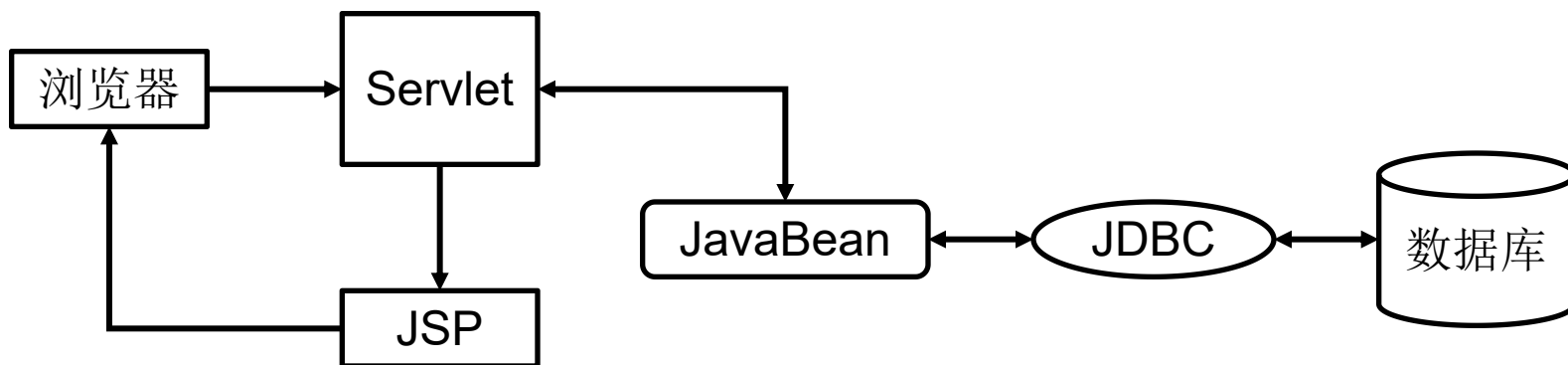
由于3.2.1节的例子中继承了ActionSupport类，所以可以看出，在execute的返回值中，其代码可以改为：

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

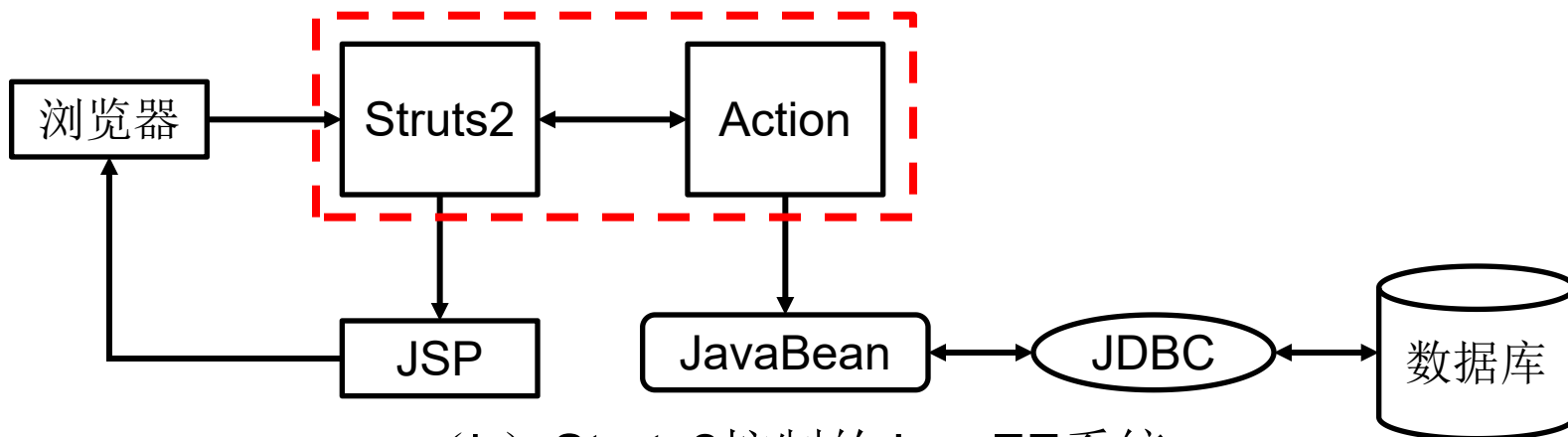
整体工作流程



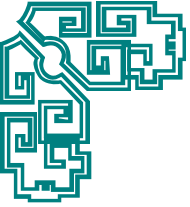
3.2.3 Action的实现方式



(a) Servlet控制的JavaEE系统



(b) Struts2控制的JavaEE系统



3.2.3 Action的实现方式

1. POJO实现方式

“Plain Ordinary Java Object”（POJO），简单普通的java对象。主要用来指代那些没有遵循特定的java对象模型、约定或者框架的对象。

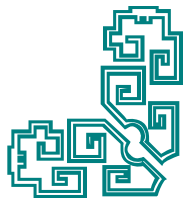
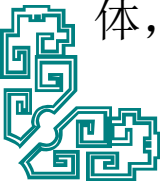
POJO的内在含义是指那些：

- （a）有一些private的参数作为对象的属性，然后针对每一个参数定义get和set方法访问的接口。
- （b）没有从任何类继承、也没有实现任何接口，更没有被其它框架侵入的java对象。

JavaBean 是一种JAVA语言写成的可重用组件。JavaBean符合一定规范编写的Java类，不是一种技术，而是一种规范。大家针对这种规范，总结了很多开发技巧、工具函数。符合这种规范的类，可以被其它的程序员或者框架使用。

与JavaBean的区别：

- （a）POJO其实是比javabean更纯净的简单类或接口。POJO严格地遵守简单对象的概念，而一些JavaBean中往往会封装一些简单逻辑。
- （b）POJO主要用于数据的临时传递，它只能装载数据，作为数据存储的载体，而不具有业务逻辑处理的能力。

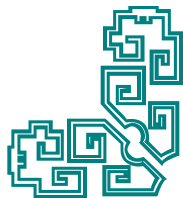
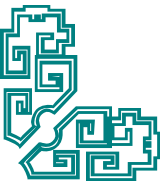


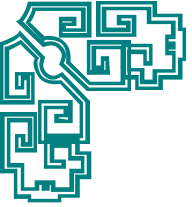


3.2.3 Action的实现方式

POJO示例

```
public class User {  
    private String userName;  
    private String password;  
  
    public String getUserName() { ... }  
  
    public void setUserName(String userName) { ... }  
  
    public String getPassword() { ... }  
  
    public void setPassword(String password) { ... }  
}
```

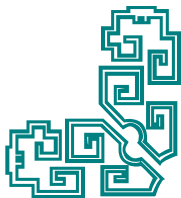
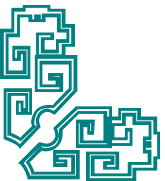


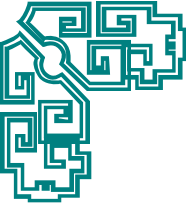


3.2.3 Action的实现方式

POJO实现Action示例

```
public class User {  
    private String userName;  
    private String password;  
  
    public String getUserName() { ... }  
  
    public void setUserName(String userName) { ... }  
  
    public String getPassword() { ... }  
  
    public void setPassword(String password) { ... }  
  
    /*调用业务逻辑方法*/  
    public String execute() { ... }  
}
```





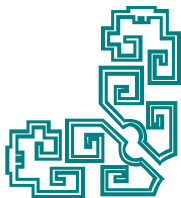
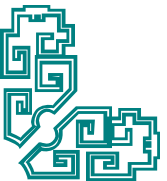
3.2.3 Action的实现方式

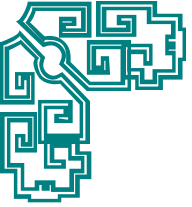
2. 实现Action接口

为了让Action类更规范，是不同的开发人员编写的execute()方法返回的字符串风格时一致的，Struts2提供一个Action接口，该接口定义了处理类应该实现的通用规范。

标准Action接口的代码如下：

```
public interface Action {  
    public static final String SUCCESS="success";  
    public static final String NONE="none";  
    public static final String ERROR="error";  
    public static final String INPUT="input";  
    public static final String LOGIN="login";  
  
    public String execute() throws Exception;  
}
```





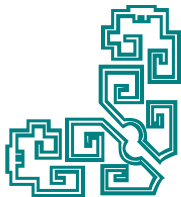
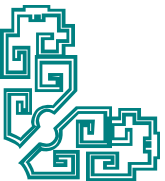
3.2.3 Action的实现方式

3. 继承ActionSupport类

Struts2框架为Action接口提供了一个实现类ActionSupport，该类提供了许多默认方法，例如：默认处理用户请求的方法、数据校验方法、获取国际化信息的方法等。编写Action类时继承ActionSupport类会大大简化Action的开发，从而节省时间、提高效率。

下面是ActionSupport类所实现的接口：

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





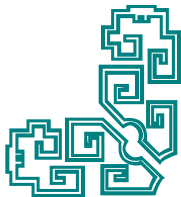
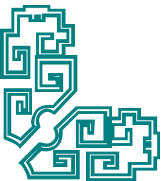
3.2.3 Action的实现方式

补充：访问**ActionContext**

在Struts2中，**Action**不再和任何**Servlet API**耦合，但有些时候Action类不访问Servlet API 是不能实现业务逻辑的。

例如：跟踪HTTP Session的状态，此时Action就需要访问Servlet API 中的HttpSession。

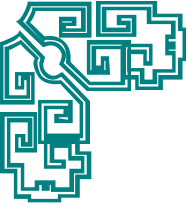
Struts2中Action可以通过ActionContext类来访问Servlet API， ActionContext 提供了读写Servlet API 中的HttpServletRequest、HttpSession和ServletContext 中数据的方法。



3.2.3 Action的实现方式

ActionContext类的常用方法：

方法	功能描述
Object get(Object key)	获取属性值，与HttpServletRequest的getAttribute方法类似
Map getApplication()	返回一个Map对象，模拟Web应用的ServletContext对象
static ActionContext getContext()	获取系统的ActionContext对象
Map getParameters()	获取所有的请求参数，与HttpServletRequest的getParameterMap方法类似
Map getSession()	模拟HttpSession实例
void setApplication(Map application)	将Map中的键值对转换成application的属性名和属性值
void setSession(Map session)	将Map中的键值对转换成session的属性名和属性值



3.2.3 Action的实现方式

补充：动态方法调用

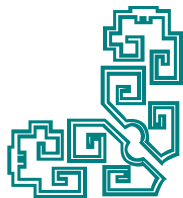
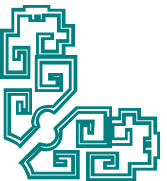
Struts2框架允许一个Action中包含多个控制处理逻辑，请求一个Action中的不同处理逻辑方法的方式成为DMI（Dynamic Method Invocation，动态方法调用），其请求格式如下：

`actionName!methodName.action`

其中，`actionName`是Action的名字，即`struts.xml`中Action的`name`属性值；`methodName`是Action实现类中处理逻辑的方法名。

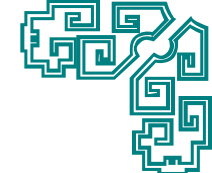
思考：

`http://localhost:8080/Action/MutiFuncAction!exe2.action?name=f`
上述请求中各部分的含义？





3.2.4 Struts 2数据验证及验证框架的应用

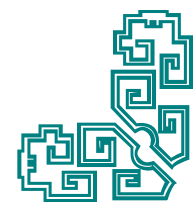
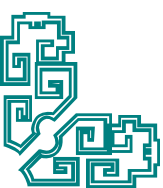


◆ 1. 数据校验

在3.2.1节的例子中，即使用户输入空的name，服务器也会处理用户请求。但如果是注册操作，空的用户名和密码就可能引起异常。

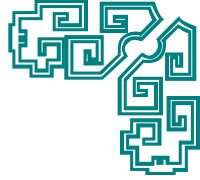
下面来看数据校验的实现，可以把3.2.1节的例子中的Action类添加如下方法：

```
...
public void validate() {
    //如果姓名为空，则把错误信息添加到Action类的fieldErrors
    if(this.getName()==null||this.getName().trim().equals("")){
        addFieldError("name","姓名是必须的！");    //把错误信息
保存起来
    }
}
...
```



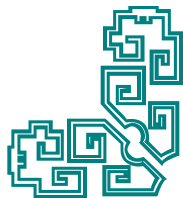
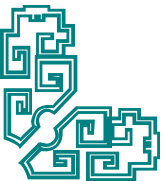


3.2.4 Struts 2数据验证及验证框架的应用



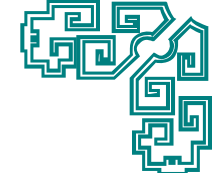
如果执行该方法之后，**Action**类的**fieldErrors**中已经包含了数据校验错误信息，将把请求转发到**input**逻辑视图处，所以要在**Action**配置中加入以下代码：

```
...  
<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>  
</action>  
...
```





3.2.4 Struts 2数据验证及验证框架的应用

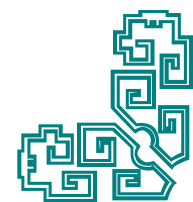
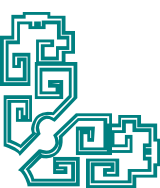


把JSP页面改写一下（标签的具体应用会在3.3节具体讲解）：

```
<%@ page language="java" pageEncoding="utf-8"%>
<!-- 导入标签开发能力 -->
<%@ taglib uri="/struts-tags" prefix="s" %>
<html>
<head>
    <title>struts 2</title>
</head>
<body>
    <s:form action="struts" method="post">
        <s:textfield name="name" label="请输入姓名"></s:textfield>
        <s:submit value="提交"></s:submit>
    </s:form>
</body>
</html>
```

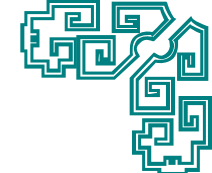
也可以通过<s:fielderror>标签获取错误提示信息，如：

```
<s:fielderror>
    <s:param>name</s:param>
</s:fielderror>
```





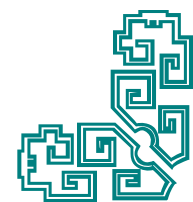
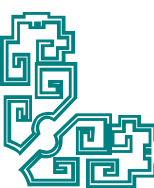
3.2.4 Struts 2数据验证及验证框架的应用



修改之后，部署运行。不输入任何姓名直接提交，将会看到如图3.5所示的界面。

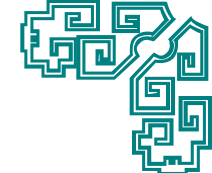
The screenshot shows a web browser window with the address bar containing the URL `http://localhost:8080/Struts2.0/struts.action`. The main content area displays a validation error message in Chinese: "姓名是必须的!" (Name is required!). Below the message, there is a text input field with the placeholder text "请输入姓名:" (Please enter name:). To the right of the input field is a button labeled "提交" (Submit).

图3.5 校验结果





3.2.4 Struts 2数据验证及验证框架的应用

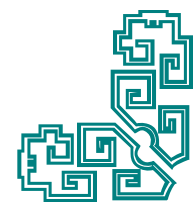
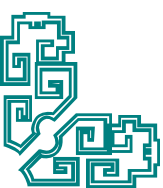


◇ 2. Struts 2验证框架的应用

()方法上面的校验是通过重写`validate`实现的，如果页面有很多输入框，就要做很多判断，而这些判断的语句基本相同。

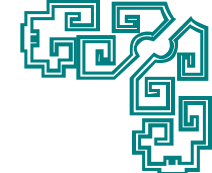
为此，**Struts 2**提供了校验框架，只需要增加一个校验配置文件，就可以完成对数据的校验。**Struts 2**提供了大量的数据校验器，包括表单域校验器和非表单域校验器两种。

校验配置文件的命名要遵循如下规则：**ActionName-validation.xml**。其中，**ActionName**是需要校验的用户自定义**Action**类名，且该配置文件与**Action**类的文件位于同一路径下。注意：如果一个**Action**类中有2个甚至多个方法，对应的在**struts.xml**文件中有多个**Action**的配置与之匹配，这是如果想对一个方法进行验证，则命名应该为**ActionName-name-validation.xml**，这里的**name**是**struts.xml**文件中**Action**属性里面的**name**。





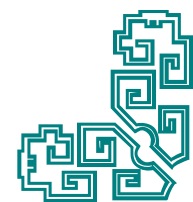
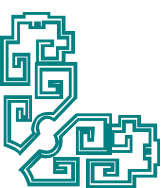
3.2.4 Struts 2数据验证及验证框架的应用



(1) 必填字符串校验器

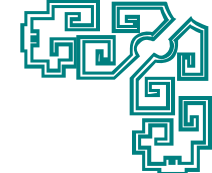
其校验规则定义文件如下：

```
<?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>
<!-- 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>
```





3.2.4 Struts 2数据验证及验证框架的应用

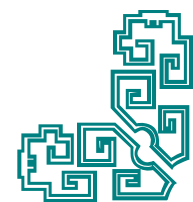
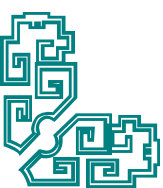


下面具体介绍其他校验框架的应用。

（2）必填校验器

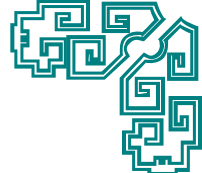
该校验器的名字是**required**，也就是<field-validator>属性中的**type="required"**，该校验器要求指定的字段必须有值，与必填字符串校验器最大的区别就是可以有空字符串。如果把上例改为必填校验器，其代码应为：

```
<?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>
<!-- 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>
```





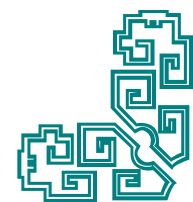
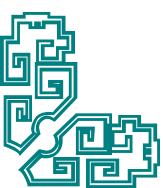
3.2.4 Struts 2数据验证及验证框架的应用



（3）整数校验器

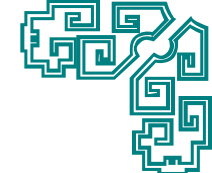
该校验器的名字是int，该校验器要求字段的整数值必须在指定范围内，故其有min和max参数。如果有个age输入框，要求其必须是整数，且输入值必须在18与100之间，该校验器的配置应该为：

```
<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>
```





3.2.4 Struts 2数据验证及验证框架的应用



(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>
```

```
<!-- 需要校验的字段名 -->
```

```
<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 -->
```

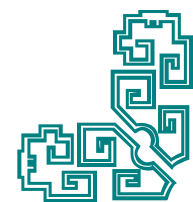
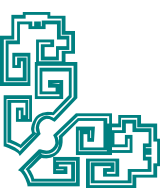
```
<message>日期必须在1980-01-01至2009-12-31之间
```

```
</message>
```

```
</field-validator>
```

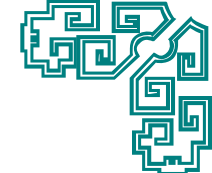
```
</field>
```

```
</validators>
```





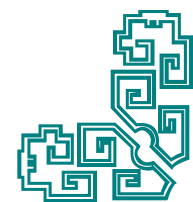
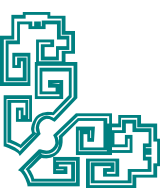
3.2.4 Struts 2数据验证及验证框架的应用



(5) 邮件地址校验器

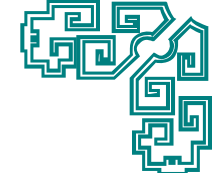
该校验器的名称是**email**，该校验器要求字段的字符如果非空，就必须是合法的邮件地址。如下面的代码：

```
<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>
```





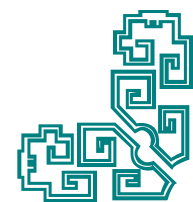
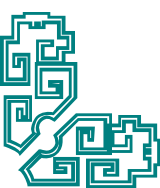
3.2.4 Struts 2数据验证及验证框架的应用



(6) 网址校验器

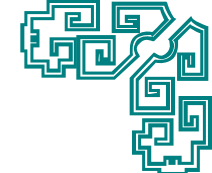
该校验器的名称是url，该校验器要求字段的字符如果非空，就必须是合法的URL地址。如下面的代码：

```
<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>
```





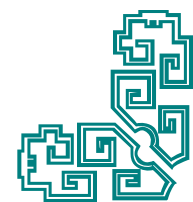
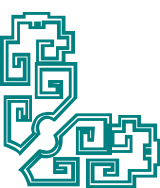
3.2.4 Struts 2数据验证及验证框架的应用



(7) 字符串长度校验器

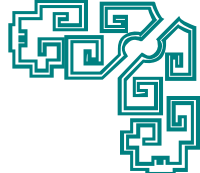
该校验器的名称是**stringlength**，该校验器要求字段的长度必须在指定的范围内，一般用于密码输入框。如下面的代码：

```
<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>
```





3.2.4 Struts 2数据验证及验证框架的应用



（8）正则表达式校验器

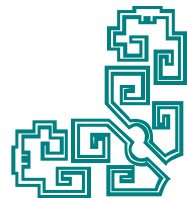
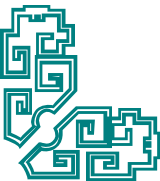
该校验器的名称是**regex**，它检查被校验字段是否匹配一个正则表达式。如下面的代码：

```
<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>
```

正则表达式：

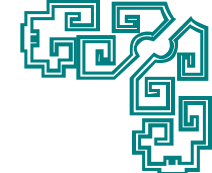
\d{6}：6位数字

\w{4,16}：4-16位的字母或数字





3.2.4 Struts 2数据验证及验证框架的应用



补充：字段校验器与非字段校验器

字段校验器配置格式：

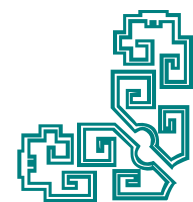
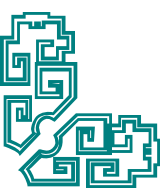
```
<field name="被校验的字段">  
  <field-validator type="校验器名">  
    <param name="参数名">参数值</param>  
    <message>校验失败后的提示信息</message>  
  </field-validator>  
  <!-- 如果校验字段满足多个规则，下面可以配置多个校验器-->  
</field>
```

非字段校验器配置格式：

```
<validator type="校验器名">  
  <param name="fieldName">需要被校验的字段</param>  
  <param name="参数名">参数值</param>  
  <message>校验失败后的提示信息</message>  
</validator>
```

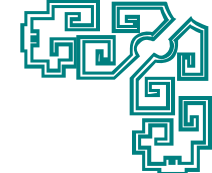
非字段校验：先指定校验器：由谁来校验，来校验谁！

字段校验器：先指定校验的属性：我来校验谁，由谁来校验！





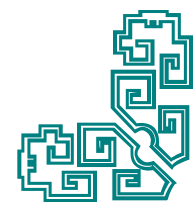
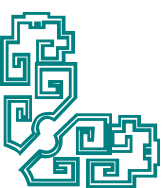
3.2.4 Struts 2数据验证及验证框架的应用



expression校验器和**fieldexpression**校验器:

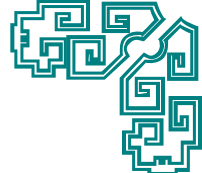
```
<validator type="expression">
  <param name="expression">![CDATA[(pass == rpass)]]</param>
  <message>校验失败后的提示信息</message>
</validator>
```

```
<field name="pass">
  <field-validator type="fieldexpression">
    <!--指定逻辑表达式-->
    <param name="expression">![CDATA[(pass == rpass)]]</param>
    <!--指定校验失败的提示信息-->
    <message>两个密码必须相同！</message>
  </field-validator>
</field>
```





3.2.4 Struts 2数据验证及验证框架的应用



校验器的短路：

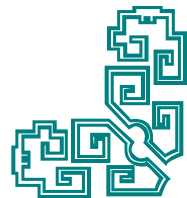
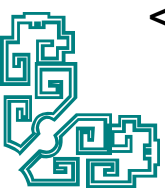
若对一个字段使用多个验证器，默认情况下会执行所有的验证。若希望前面的验证器验证没有通过，后面的就不再验证，可以使用短路验证。

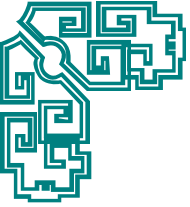
(1) 对同一个字段内的多个验证器，如果一个短路验证器验证失败，其他验证器不会继续校验；

(2) 验证程序配置文件中<validator .../> 元素和 <field-validator .../> 元素可以指定一个可选的 **short-circuit** 属性，该属性指定该验证器是否是短验证器，默认值为 **false**

```
<field name="name">
  <field-validator type="requiredstring" short-circuit="true">
    <message>你需要输入一个URL。</message>
  </field-validator>

  <field-validator type="url" short-circuit="true">
    <message>URL错误。</message>
  </field-validator>
</field>
```





3.3 Struts 2标签库应用

✧ 3.3.1 Struts 2的OGNL表达式

在介绍标签库前，有必要先来学习Struts 2的OGNL表达式。

◆ 1. OGNL表达式

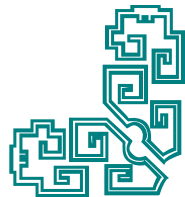
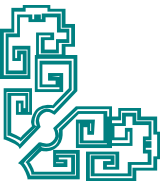
OGNL表达式是Struts 2框架的特点之一。

OGNL是Object-Graph Navigation Language的缩写，全称为对象图导航语言，是一种功能强大的表达式语言，它通过简单一致的语法，可以任意存取对象的属性或者调用对象的方法，能够遍历整个对象的结构图，实现对象属性类型的转换等功能。

OGNL表达式允许我们通过一个字符串，来访问JAVA对象，从而避免了在JSP上嵌套着写JAVA代码

标准的OGNL会设定一个根对象（root对象）。假设使用标准OGNL表达式来求值（不是Struts 2 OGNL），如果OGNL上下文有两个对象foo对象和bar对象，同时foo对象被设置为根对象（root），则利用下面的OGNL表达式求值。

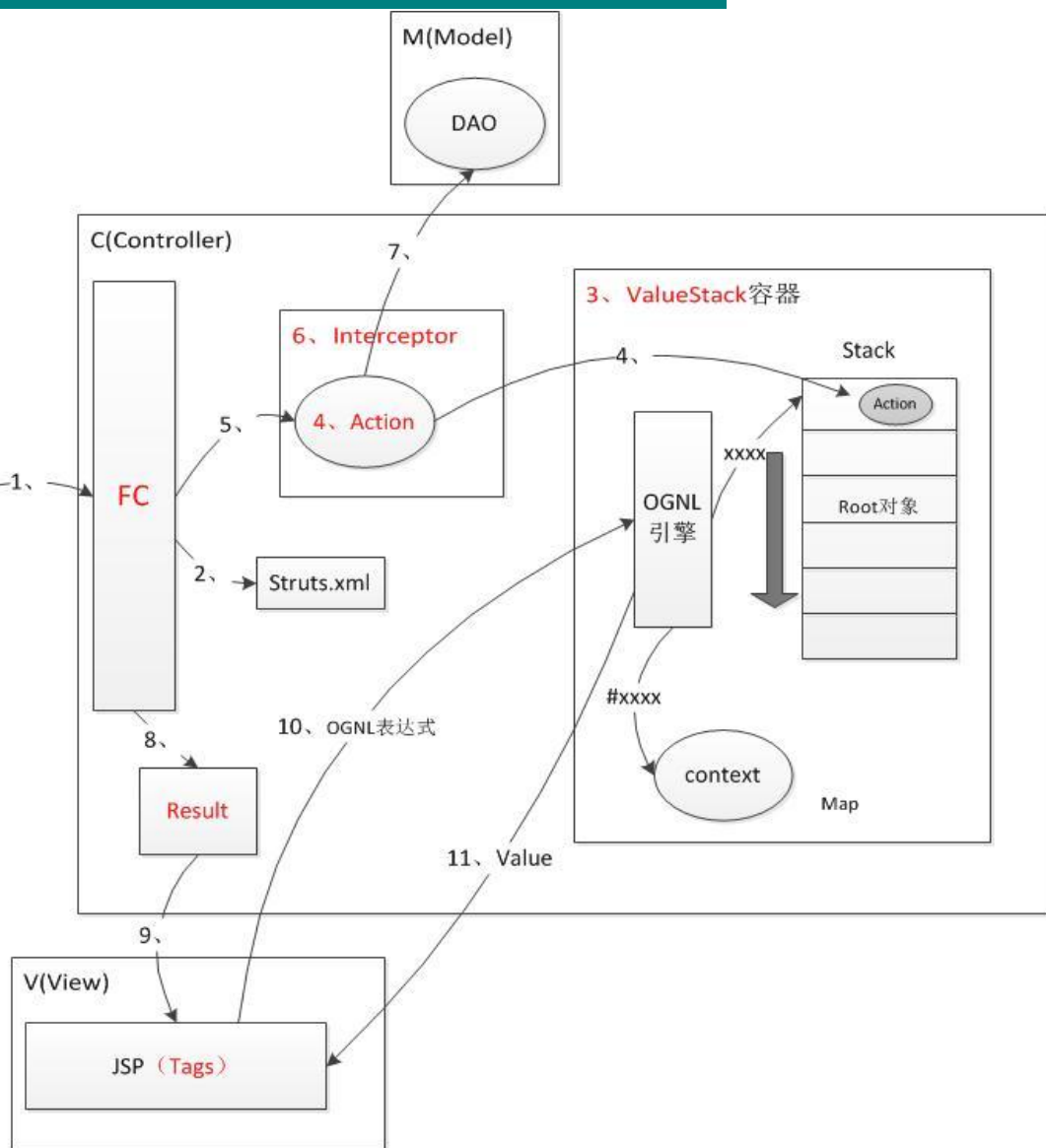
#foo.blah	// 返回foo.getBlah()
#bar.blah	// 返回bar.getBlah() 非根对象加#
blah	// 返回foo.getBlah(), 因为foo为根对象



3.3.1 Struts 2的OGNL表达式

Struts2中并不是直接将OGNL组件搬过来使用，而是使用一个对象容器**ValueStack**对这个组件进行了改造及封装，因此我们在Struts2中是通过ValueStack来使用OGNL表达式。并且ValueStack中封装了Action的数据，上下文等数据，这些数据我们都可以通过OGNL表达式来获得。

ValueStack中封装了OGNL引擎、context对象、Stack对象库，**result**，**值栈**，**前端控制器**，**action**，



3.3.1 Struts 2的OGNL表达式

在Struts 2框架中，值栈（Value Stack）就是OGNL的根对象。假设值栈中存在两个对象实例Man和Animal，这两个对象实例都有一个name属性，Animal有一个species属性，Man有一个salary属性。假设Animal在值栈的顶部，Man在Animal后面，如图3.6所示。

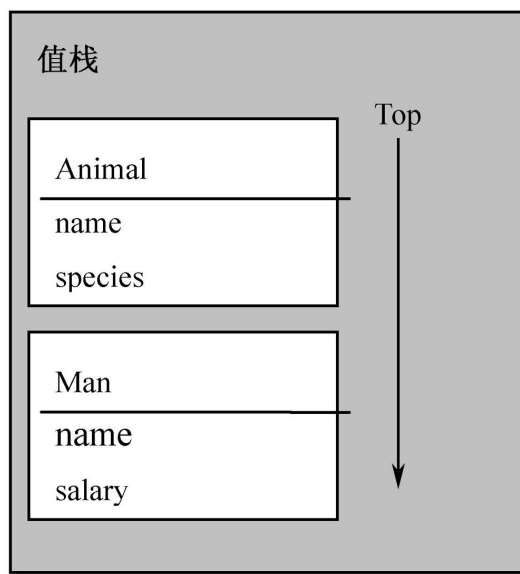
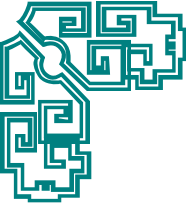


图3.6 一个包含了Animal和Man的值栈



3.3.1 Struts 2的OGNL表达式

下面的代码片段能更好地理解OGNL表达式。

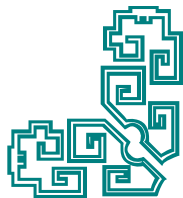
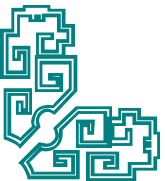
```
species           // 调用animal.getSpecies()  
salary           // 调用man.getSalary()  
name             // 调用animal.getName(), 因为Animal位于值栈的顶部
```

最后一行实例代码返回的是`animal.getName()`返回值，即返回了`Animal`的`name`属性，因为`Animal`是值栈的顶部元素，OGNL将从顶部元素搜索，所以会返回`Animal`的`name`属性值。如果要获得`Man`的`name`值，则需要如下代码：

```
man.name
```

Struts 2允许在值栈中使用索引，实例代码如下：

```
[0].name         // 调用animal.getName()  
[1].name         // 调用man.getName()
```



3.3.1 Struts 2的OGNL表达式

Struts 2中的OGNL Context是ActionContext，如图3.7所示。

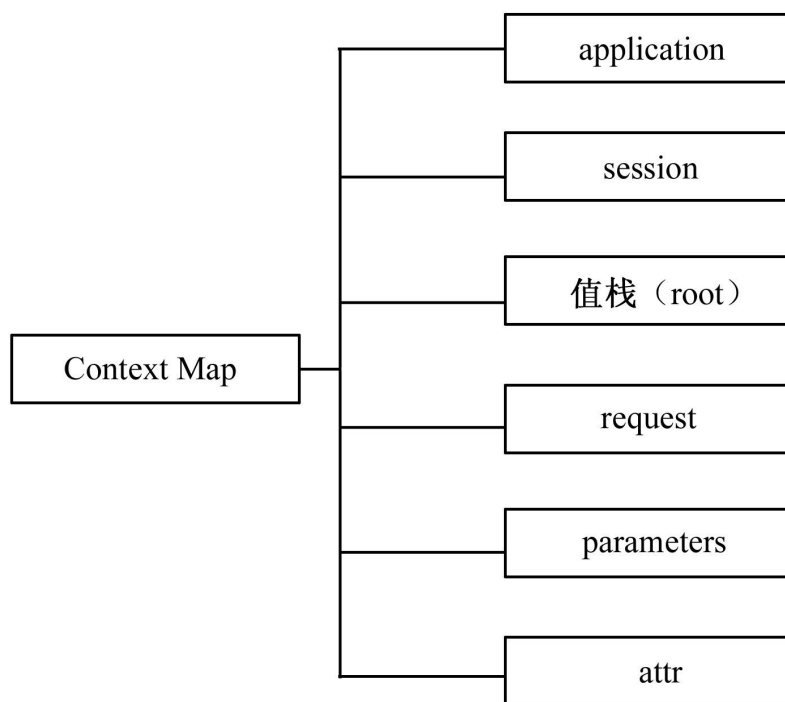
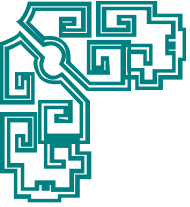


图3.7 Struts 2的OGNL Context结构示意图



3.3.1 Struts 2的OGNL表达式

由于值栈是Struts 2中OGNL的根对象。如果用户需要访问值栈中的对象，则可以通过如下代码访问值栈中的属性：

`${foo}` // 获得值栈中的foo属性

如果访问其他Context中的对象，由于不是根对象，在访问时需要加#前缀。

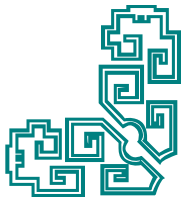
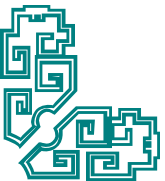
- application对象：用来访问ServletContext，如#application.userName或者#application ["userName"]，相当于调用Servlet的getAttribute("userName")。
- session对象：用来访问HttpSession，如#session.userName或者#session["userName"]，相当于调用session.getAttribute("userName")。
- request对象：用来访问HttpServletRequest属性的Map，如#request.userName或者#request["userName"]，相当于调用request.getAttribute("userName")。如在3.2.1节中StrutsAction类中代码：

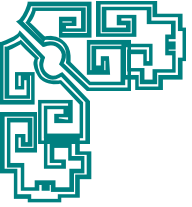
```
Map request=(Map)ActionContext.getContext().get("request");  
request.put("name",getName());
```

这就是先得到request对象，然后把值放进去，在该例的success.jsp中有：

```
<s:property value="#request.name"/>    //OGNL表达式
```

其中#request.name相当于调用了request.getAttribute("name")。





3.3.1 Struts 2的OGNL表达式

◇ 2. OGNL集合操作

使用如下代码直接生成一个List对象：

`{e1, e2, e3...}`

下面的代码可以直接生成一个Map对象：

`#{key: value1, key2: value2, ...}`

对于集合类型，OGNL表达式可以使用in和not in两个元素符号。其中，in表达式用来判断某个元素是否在指定的集合对象中；not in判断某个元素是否不在指定的集合对象中，代码如下所示：

```
<s: if test="foo' in {'foo', 'bar'}">
```

```
... //“ foo in {foo, bar}”
```

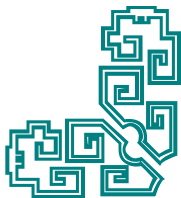
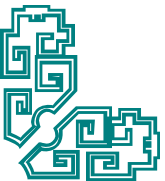
```
</s: if>
```

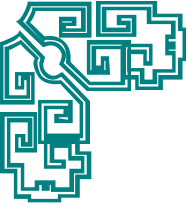
或

```
<s: if test="foo' not in {'foo', 'bar'}">
```

```
...
```

```
</s: if>
```





3.3.1 Struts 2的OGNL表达式

除了in和not in之外，OGNL还允许使用某个规则获得集合对象的子集，常用的有以下3个相关操作符。

?: 获得所有符合逻辑的元素。

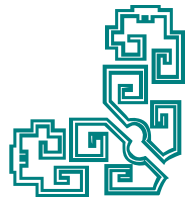
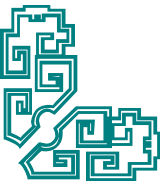
^: 获得符合逻辑的第一个元素。

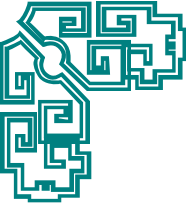
\$: 获得符合逻辑的最后一个元素。

如下面的代码：

`Person .relatives.{?# this.gender=='male'}`//this是relatives集合

该代码可以获得Person的所有性别为male的relatives集合。

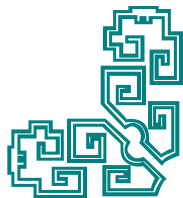
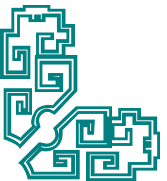


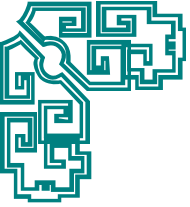


3.3.2 数据标签

数据标签属于非UI标签，主要用于提供各种数据访问相关的功能，数据标签主要包括以下几个。

- **property**：用于输出某个值。
- **set**：用于设置一个新变量。
- **param**：用于设置参数，通常用于**bean**标签和**action**标签的子标签。
- **bean**：用于创建一个JavaBean实例。
- **action**：用于在JSP页面直接调用一个Action。
- **date**：用于格式化输出一个日期。
- **debug**：用于在页面上生成一个调试链接，当单击该链接时，可以看到当前值栈和Stack Context中的内容。
- **il8n**：用于指定国际化资源文件的baseName。
- **include**：用于在JSP页面中包含其他的JSP或Servlet资源。
- **push**：用于将某个值放入值栈的栈顶。
- **text**：用于输出国际化（国际化内容会在后面讲解）。
- **url**：用于生成一个URL地址。

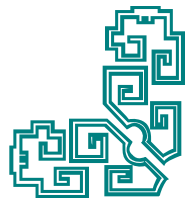
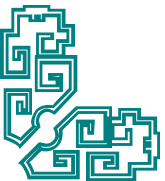


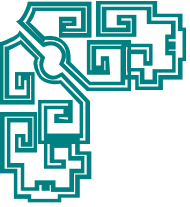


3.3.2 数据标签

◆ 1. <s:property>标签 /<@taglib prefix='s'

- **property**标签的作用是输出指定值。**property**标签输出**value**属性指定的值。如果没有指定的**value**属性，则默认输出值栈栈顶的值。该标签有如下几个属性：
 - **default**: 该属性是可选的，如果需要输出的属性值为null，则显示**default**属性指定的值。
 - **escape**: 该属性是可选的，指定是否**escape** HTML代码。
 - **value**: 该属性是可选的，指定需要输出的属性值，如果没有指定该属性，则默认输出值栈栈顶的值。该属性也是最常用的，如前面用到的：
`<s:property value="#request.name"/>`
 - **id**: 该属性是可选的，指定该元素的标志。





3.3.2 数据标签

◇ 2. <s:set>标签

该标签有如下几个属性：

- **name**：该属性是必选的，重新生成新变量的名字。
- **scope**：该属性是可选的，指定新变量的存放范围。
- **id**：该属性是可选的，指定该元素的引用id。

下面是一个简单例子，展示了property标签访问存储于session中的user对象的多个字段：

```
<s:property value="#session['user'].username"/>
```

```
<s:property value="#session['user'].age"/>
```

```
<s:property value="#session['user'].address"/>
```

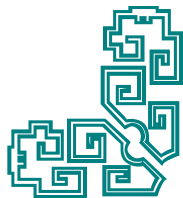
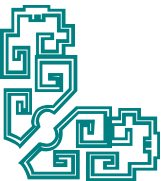
使用set标签使得代码易于阅读：

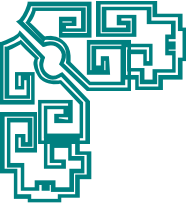
```
<s:set name="user" value="#session['user'] " />
```

```
<s:property value="#user.username"/>
```

```
<s:property value="#user.age" />
```

```
<s:property value="#user.address" />
```





3.3.2 数据标签

◇ 3. <s:param>标签

param标签主要用于为其他标签提供参数，该标签有如下几个属性：

- name: 该属性是可选的，指定需要设置参数的参数名。
- value: 该属性是可选的，指定需要设置参数的参数值。
- id: 该属性是可选的，指定引用该元素的id。

例如，要为name为fruit的参数赋值：

```
<s:param name="fruit">apple</s:param>
```

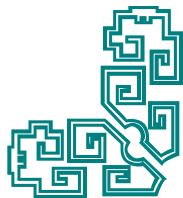
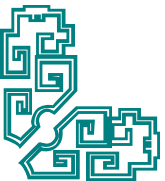
或者

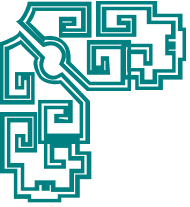
```
<s:param name="fruit" value="apple" />
```

注意：上面是将apple对象赋给fruit参数。

如果想指定fruit参数的值为apple字符串，则应该这样写：

```
<s:param name="fruit" value="apple" />
```





3.3.2 数据标签

◇ 4. <s:bean>标签

该标签有如下几个属性：

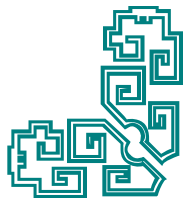
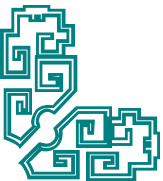
- **name**：该属性是必选的，用来指定要实例化的JavaBean的实现类。
- **id**：该属性是可选的，如果指定了该属性，则该JavaBean实例会被放入Stack Context中，从而允许直接通过id属性来访问该JavaBean实例。下面是一个简单的例子：

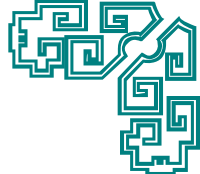
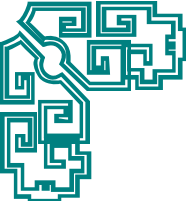
有一个Student类，该类中有name属性，并有其getter和setter方法：

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

然后在JSP文件的body中加入下面的代码：

```
<s:bean name="Student">  
    <s:param name="name" value="zhangsan"/>  
    <s:property value="name"/>  
</s:bean>
```





3.3.2 数据标签

在项目中导入Struts 2的5个重要Jar包，再把Student类放在项目的src文件夹下，`<s:bean>`标签内容放在一个JSP文件的body体内，再修改web.xml文件，就可以部署运行该项目，会得到如图3.8所示的界面。



图3.8 bean标签实例界面

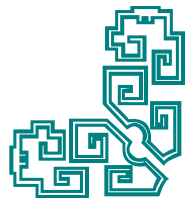
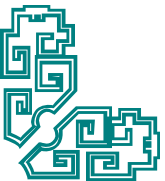
如果把bean标签的内容改为：

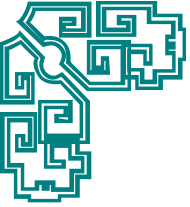
```
<s:bean name="Student" id="s" >
```

```
    <s:param name="name" value="zhangsan"/>
```

```
</s:bean>
```

```
<s:property value="#s.name"/>
```



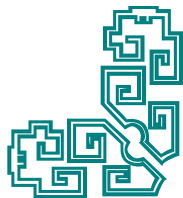
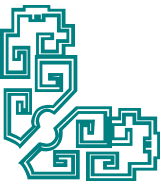


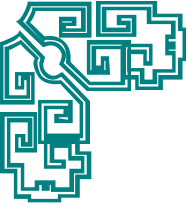
3.3.2 数据标签

◆ 5. <s:action>标签

使用action标签可以允许在JSP页面中直接调用Action。该标签有以下几个属性：

- **id**：该属性是可选的，该属性将会作为该Action的引用标志id。
- **name**：该属性是必选的，指定该标签调用哪个Action。
- **namespace**：该属性是可选的，指定该标签调用的Action所在的namespace。
- **executeResult**：该属性是可选的，指定是否要将Action的处理结果页面包含到本页面。如果值为true，就是包含，false就是不包含，默认为false。
- **ignoreContextParam**：该属性是可选的，指定该页面中的请求参数是否需要传入调用的Action。如果值为false，将本页面的请求参数传入被调用的Action。如为true，不将本页面的请求参数传入到被调用的Action。





3.3.2 数据标签

◇ 6. <s:date>标签

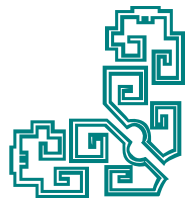
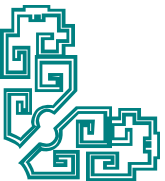
date标签主要用于格式化输出一个日期。该标签有如下属性：

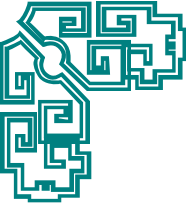
- **format**：该属性是可选的，如果指定了该属性，将根据该属性指定的格式来格式化日期。
- **nice**：该属性是可选的，该属性的取值只能是**true**或**false**，用于指定是否输出指定日期和当前时刻之间的时差。默认为**false**，即不输出时差。
- **name**：属性是必选的，指定要格式化的日期值。
- **id**：属性是可选的，指定引用该元素的id值。
- **nice**属性为**true**时，一般不指定**format**属性。因为**nice**为**true**时，会输出当前时刻与指定日期的时差，不会输出指定日期。当没有指定**format**，也没有指定**nice="true"**时，系统会采用默认格式输出。其用法为：

`<s:date name="指定日期取值" format="日期格式"/><!-- 按指定日期格式输出 -->`

`<s:date name="指定日期取值" nice="true"/><!-- 输出时间差 -->`

`<s:date name="指定日期取值"/><!-- 默认格式输出 -->`





3.3.2 数据标签

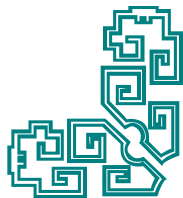
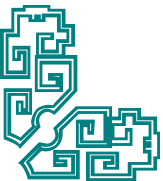
◇ 7. <s:include>标签

include标签用于将一个JSP页面或一个Servlet包含到本页面中。该标签有如下属性：

- **value**：该属性是必选的，指定需要被包含的JSP页面或Servlet。
- **id**：该属性是可选的，指定该标签的id引用。

用法如下：

```
<s:include value="JSP或Servlet文件" id="自定义名称"/>
```

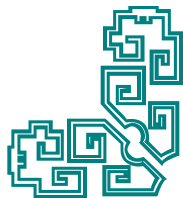
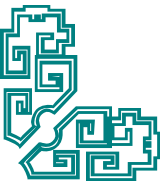


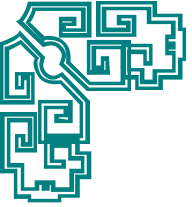


3.3.3 控制标签

控制标签有以下几个：

- **if**：用于控制选择输出的标签。
- **elseif**：用于控制选择输出的标签，必须和**if**标签结合使用。
- **else**：用户控制选择输出的标签，必须和**if**标签结合使用。
- **append**：用于将多个集合拼接成一个新的集合。
- **generator**：用于将一个字符串按指定的分隔符分隔成多个字符串，临时生成的多个子字符串可以使用**iterator**标签来迭代输出。
- **iterator**：用于将集合迭代输出。
- **merge**：用于将多个集合拼接成一个新的集合，但与**append**的拼接方式不同。
- **sort**：用于对集合进行排序。
- **subset**：用于截取集合的部分元素，形成新的子集合。



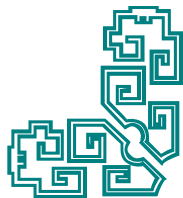
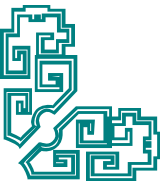


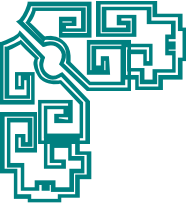
3.3.3 控制标签

❖ 1. **<s:if>/<s:elseif>/<s:else>**标签

这3个标签可以组合使用，但只有if标签可以单独使用，而elseif和else标签必须与if标签结合使用。if标签可以与多个elseif标签结合使用，但只能与一个else标签使用。其用法格式如下：

```
<s:if test="表达式">
    标签体
</s:if>
<s:elseif test="表达式">
    标签体
</s:elseif>
<!-- 允许出现多次elseif标签-->
...
<s:else>
    标签体
</s:else>
```





3.3.3 控制标签

◆ 2. <s:iterator迭代器>标签//遍历

该标签主要用于对集合进行迭代，这里的集合包含List、Set，也可以对Map类型的对象进行迭代输出。该标签的属性如下：

- **value**：该属性是可选的，指定被迭代的集合，被迭代的集合通常都由OGNL表达式指定。如果没有指定该属性，则使用值栈栈顶的集合。
- **id**：该属性是可选的，指定集合元素的id。
- **status**：该属性是可选的，指定迭代时的IteratorStatus实例，通过该实例可判断当前迭代元素的属性。如果指定该属性，其实例包含如下几个方法：

int getCount()：返回当前迭代了几个元素。

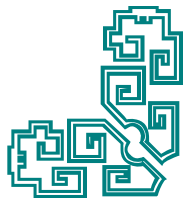
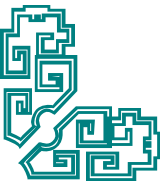
int getIndex()：返回当前被迭代元素的索引。

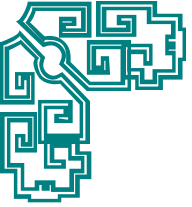
boolean isEven：返回当前被迭代元素的索引元素是否是偶数。

boolean isOdd：返回当前被迭代元素的索引元素是否是奇数。

boolean isFirst：返回当前被迭代元素是否是第一个元素。

boolean isLast：返回当前被迭代元素是否是最后一个元素。

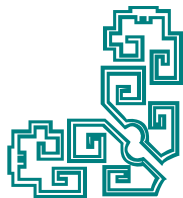
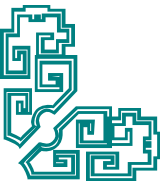




3.3.3 控制标签

应用举例：

```
<%@ page language="java" pageEncoding="utf-8"%>
<%@taglib uri="/struts-tags" prefix="s" %>
<html>
<head>
  <title>控制标签</title>
</head>
<body>
  <table border="1" width="200">
    <s:iterator value="{ 'apple','orange','pear','banana' }" id="fruit"
status="st">
      <tr <s:if test="#st.even">style="background-
color:silver"</s:if>>
        <td><s:property value="fruit"/></td>
      </tr>
    </s:iterator>
  </table>
</body>
</html>
```



3.3.3 控制标签

通过添加Struts 2必须的Jar包，再建立上面JSP文件，修改web.xml后，就可以部署运行，运行结果如图3.9所示。



图3.9 iterator标签实例运行结果

3.3.3 控制标签

◇ 3. <s:append>标签：多个集合拼接

应用举例，可以把上例的JSP文件进行修改，其代码为：

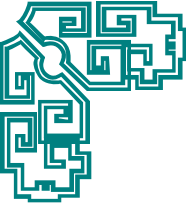
```
<%@ page language="java" pageEncoding="utf-8"%>
<%@taglib uri="/struts-tags" prefix="s" %>
<html>
<head>
  <title>控制标签</title>
</head>
<body>
  <s:append id="newList">
    <s:param value="{ 'apple','orange','pear','banana' }"/>
    <s:param value="{ 'chinese','english','french' }"/>
  </s:append>
  <table border="1" width="200">
    <s:iterator value="#newList" id="fruit" status="st">
      <tr <s:if test="#st.even">style="background-color:silver"</s:if>>
        <td><s:property value="fruit"/></td>
      </tr>
    </s:iterator>
  </table>
</body>
</html>
```

3.3.3 控制标签

部署运行，运行结果如图3.10所示。



图3.10 append标签实例运行界面



3.3.3 控制标签

◆ 4. <s:merge>标签

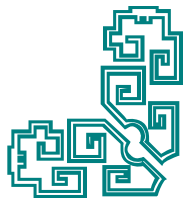
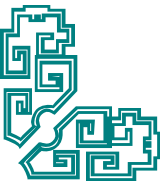
假设有2个集合，第一个集合包含3个元素，第二个集合包含2个元素，分别用append标签和merge标签方式进行拼接，它们产生新集合的方式有所区别。下面分别列出：

用append方式拼接，新集合元素顺序为：

- 第1个集合中的第1个元素
- 第1个集合中的第2个元素
- 第1个集合中的第3个元素
- 第2个集合中的第1个元素
- 第2个集合中的第2个元素

用merge方式拼接，新集合元素顺序为：

- 第1个集合中的第1个元素
- 第2个集合中的第1个元素
- 第1个集合中的第2个元素
- 第2个集合中的第2个元素
- 第1个集合中的第3个元素





3.3.4 表单标签

大部分的表单标签和HTML表单元素是一一对应的关系，如下面的代码片段：

```
<s:form action="login.action" method="post"/>
```

对应着：

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

```
<s:textfield name="username" label="用户名" />
```

对应着：

用户名：

```
<s:password name="password" label="密码"/>
```

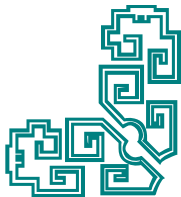
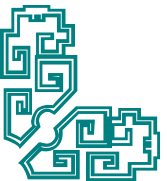
对应着：

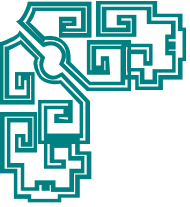
密码：

还有下面这种情况，如果有这样一个JavaBean类，类名为“User”，该类中有两个属性：一个是username；另一个是password，并分别生成它们的getter和setter方法，在JSP页面的表单中可以这样为表单元素命名：

```
<s:textfield name="user.username" label="用户名" />
```

```
<s:password name="user.password" label="密码"/>
```





3.3.4 表单标签

◆ 1. <s:checkboxlist>标签

该标签需要指定一个list属性。用法举例：

```
<s:checkboxlist label="请选择你喜欢的水果" list="{ 'apple','oranger','pear','banana' }"
name="fruit">
</s:checkboxlist>
```

或者为：

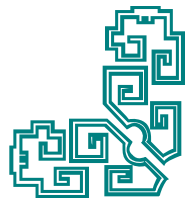
```
<s:checkboxlist label="请选择你喜欢的水果"
list="#{1:'apple',2:'oranger',3:'pear',4:'banana'}" name="fruit">
</s:checkboxlist>
```

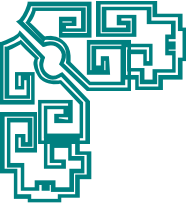
这两种方式的区别：前一种根据name取值时取的是选中字符串的值；后一种在页面上显示的是value的值，而根据name取值时取的却是对应的key，这里就是1、2、3或4。

◆ 2. <s:combobox>标签

combobox标签生成一个单行文本框和下拉列表框的组合。两个表单元素只能对应一个请求参数，只有单行文本框里的值才包含请求参数，下拉列表框只是用于辅助输入，并没有name属性，故不会产生请求参数。用法举例：

```
<s:combobox label="请选择你喜欢的水果" list="{ 'apple','oranger','pear','banana' }"
name="fruit">
</s:combobox>
```





3.3.4 表单标签

◇ 3. <s:datetimepicker>标签

`datetimepicker`标签用于生成一个日期、时间下拉列表框。当使用该日期、时间列表框选择某个日期、时间时，系统会自动将选中日期、时间输出指定文本框中。用法举例：

```
<s:form action="" method="">  
    <s:datetimepicker name="date" label="请选择日期"></s:datetimepicker>  
</s:form>
```

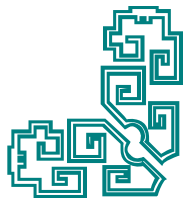
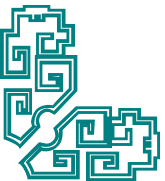
◇ 4. <s:select>标签

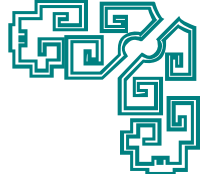
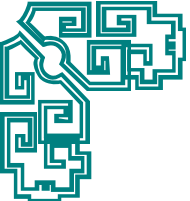
`select`标签用于生成一个下拉列表框，通过为该元素指定`list`属性的值，来生成下拉列表框的选项。用法举例：

```
<s:select list="{ 'apple','oranger','pear','banana' }"  
label="请选择你喜欢的水果"></s:select>
```

或者为：

```
<s:select list="fruit" list="#{1:'apple',2:'oranger',3:'pear',4:'banana'}"  
listKey="key" listValue="value"></s:select>
```





3.3.4 表单标签

◇ 5. <s:radio>标签

radio标签的用法与checkboxlist用法很相似，唯一的区别就是checkboxlist生成的是复选框，而radio生成的是单选框。用法举例：

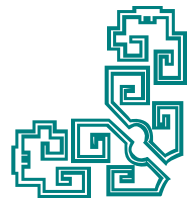
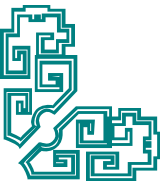
```
<s:radio label="性别" list="{ '男','女' }" name="sex"></s:radio>
```

或者为：

```
<s:radio label="性别" list="#{ 1:'男',0:'女' }" name="sex">  
</s:radio>
```

◇ 6. <s:head>标签

head标签主要用于生成HTML页面的head部分。在介绍<s:datetimepicker>标签时说过，要在head中加入该标签，主要原因是<s:datetimepicker>标签中有一个日历小控件，其中包含了JavaScript代码，所以要在head部分加入该标签。

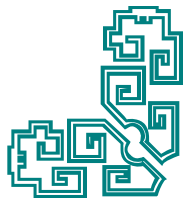
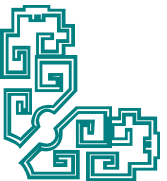




3.3.5 非表单标签

非表单标签主要用于在页面中生成一些非表单的可视化元素。这些标签不经常用到，下面大致介绍一下这些标签：

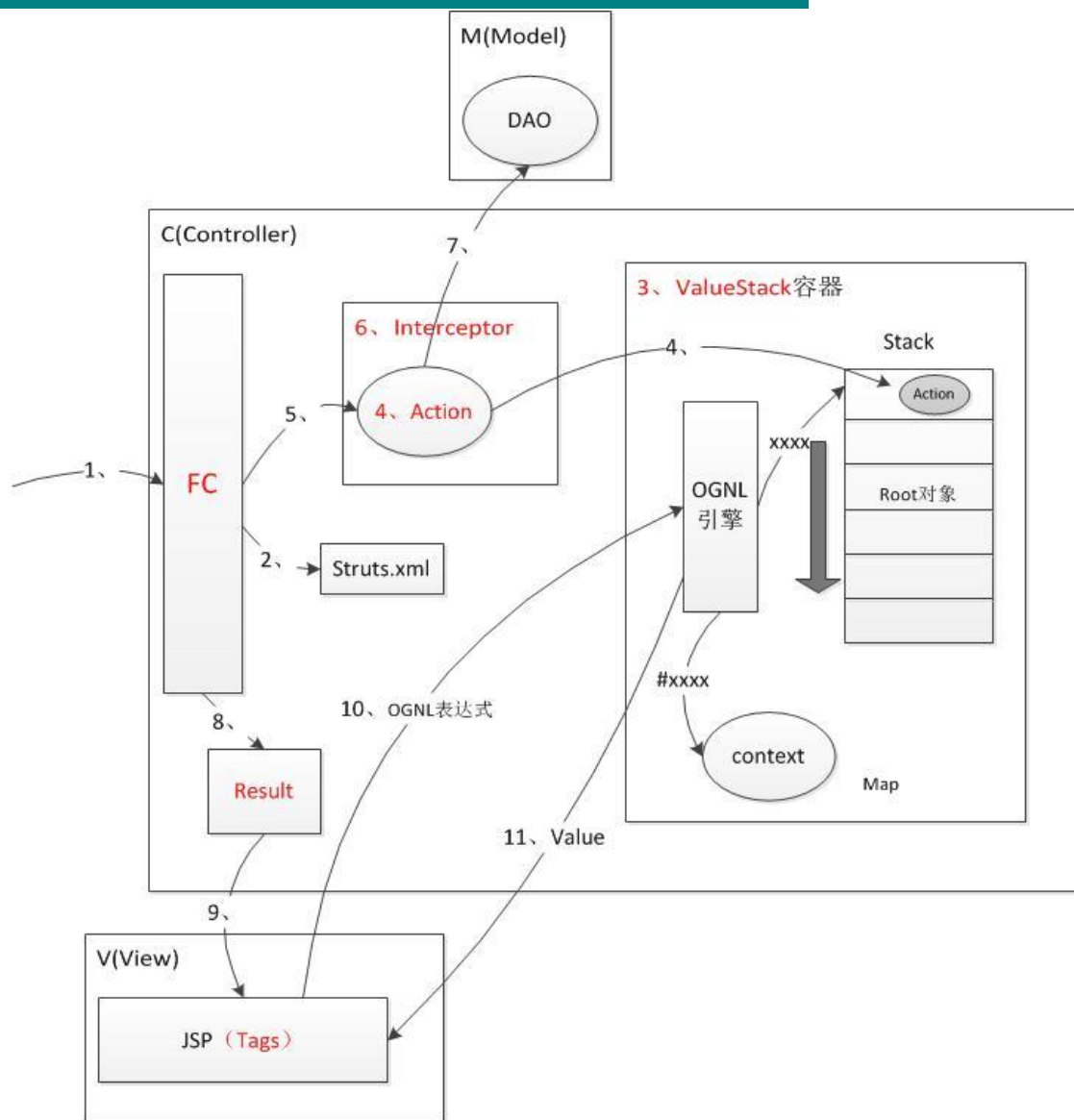
- **a**：生成超链接。
- **actionerror**：输出Action实例的getActionMessage()方法返回的消息。
- **component**：生成一个自定义组件。
- **div**：生成一个div片段。
- **fielderror**：输出表单域的类型转换错误、校验错误提示。
- **tablePanel**：生成HTML页面的Tab页。
- **tree**：生成一个树形结构。
- **treenode**：生成树形结构的节点。

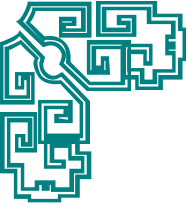


3.4 Struts 2拦截器

拦截器是Struts的核心所在，当核心控制器FilterDispatcher拦截到用户请求时，大量拦截器会对用户请求进行处理，然后才调用用户自定义的Action类中的方法处理请求。

Struts2内建的大量拦截器都是以name-class对的形式配置在struts-default.xml文件中的，其中name是拦截器的名字，class指定该拦截器的实现类。以前的例子中，在配置struts.xml文件时，都继承了struts-default包，这样就可以应用里面定义的拦截器。

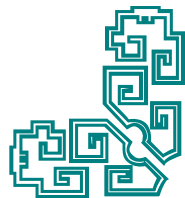
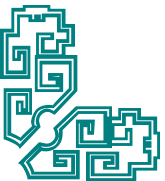




3.4 Struts 2拦截器

过滤器与拦截器的区别：

- ①拦截器是基于**Java**的反射机制的，而过滤器是基于函数回调。
- ②拦截器不依赖于**servlet**容器，过滤器依赖于**servlet**容器。
- ③拦截器只能对**action**请求起作用，而过滤器则可以对几乎所有的请求起作用。
- ④拦截器可以访问**action**上下文、值栈里的对象，而过滤器不能访问。
- ⑤在**action**的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次。
- ⑥拦截器可以获取**IOC**容器中的各个**bean**，而过滤器就不行，这点很重要，在拦截器里注入一个**service**，可以调用业务逻辑。



3.4 Struts 2拦截器

✧ 3.4.1 拦截器配置

定义拦截器使用<interceptor.../>元素。其格式为：

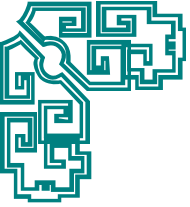
```
<interceptor name="拦截器名" class="拦截器实现类"></interceptor>
```

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

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>
```

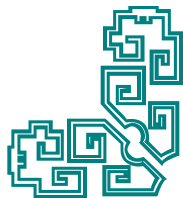
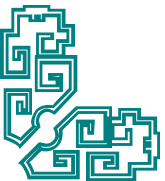


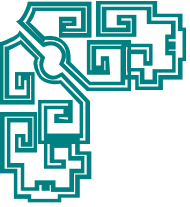
3.4.1 拦截器

下面是默认拦截器的配置方法：

```
<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>  
  <default-interceptor-ref name="拦截器名或拦截器栈名"></default-interceptor-ref>  
</package>
```

包里定义的。。。都要应用这个拦截器栈处理





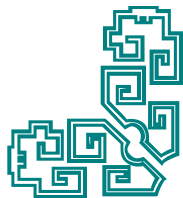
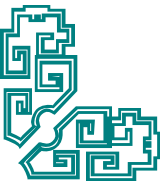
3.4.2 拦截器实现类

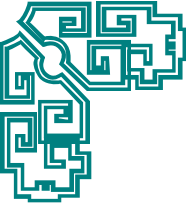
Struts 2提供了一些接口或类供程序员自定义拦截器。如Struts 2提供了com.opensymphony.xwork2.interceptor.Interceptor接口，程序员只要实现该接口就可完成拦截器实现类。该接口的代码如下：

```
import java.io.Serializable;
import com.opensymphony.xwork2.ActionInvocation;
public interface Interceptor extends Serializable{
    void init();
    String intercept(ActionInvocation invocation) throws Exception;
    void destroy();
}
```

该接口中有三个方法：

- **init()**：该方法在拦截器被实例化之后、拦截器执行之前调用。
- **intercept(ActionInvocation invocation)**：该方法用于实现拦截的动作。
- **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。

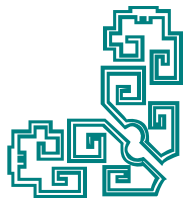
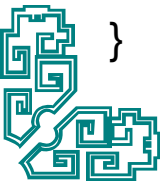


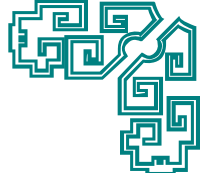
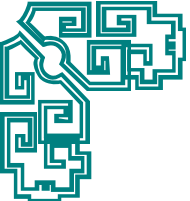


3.4.3 实例应用自定义拦截器

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", 返回错误页面
        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();
    }
}
```

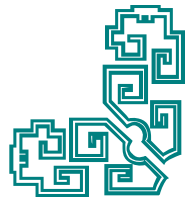
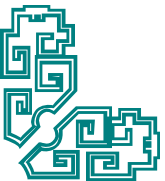




3.4.3 实例应用自定义拦截器

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>
```



3.4.3 实例应用自定义拦截器

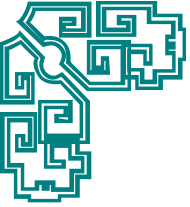
经过这样简单的配置后，重新部署项目，在运行界面输入“hello”，也会经过拦截返回到当前页面，如图3.11、图3.12所示。



图3.11 运行界面



图3.12 提交后返回当前页面

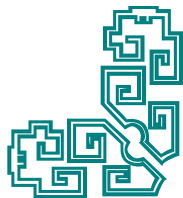
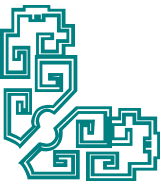


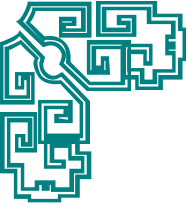
3.5 Struts 2国际化应用

下面以登录界面为例，讲解国际化应用内容。

- ◆ **1. 建立项目**
打开MyEclipse，建立一个Web项目，命名为“Test”。
- ◆ **2. 加载Struts 2的基本类库**
该步骤与3.2.1节中的第3步步骤相同，这里不再赘述。
- ◆ **3. 修改web.xml**
其内容见3.2.1节第4步。

门户系统：各国人都要访问——国际化应用





3.5 Struts 2国际化应用

◇ 4. 建立资源文件

需要在项目的src文件夹下建立一个名为“struts.properties”的文件。只需要在该文件中编写下面代码：

struts.custom.il8n.resources=资源文件名

该例中资源文件名为“messageResource”，故**struts.properties**应为：

struts.custom.il8n.resources= messageResource

下面来建立两个资源文件，分别为中文和英文。其中，英文文件名为
messageResource_en_US.properties，中文文件名为
messageResource_zh_CN.properties

username=Username

password=Password

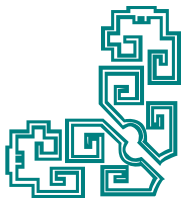
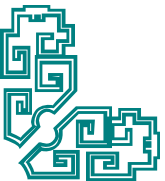
login=login

可以看出，它们分别是一个key-value对。然后再提供下面的文件：

username=登录名

password=口令

login=登录



3.5 Struts 2国际化应用

将上面文件以“messageResource_temp.properties”为文件名保存在项目的WEB-INF/classes文件夹下，因为该文件包含了非西欧字符，所以必须用native2ascii命令来处理。选择【开始】→【运行】菜单项，输入“cmd”，看到如图3.13所示的界面。

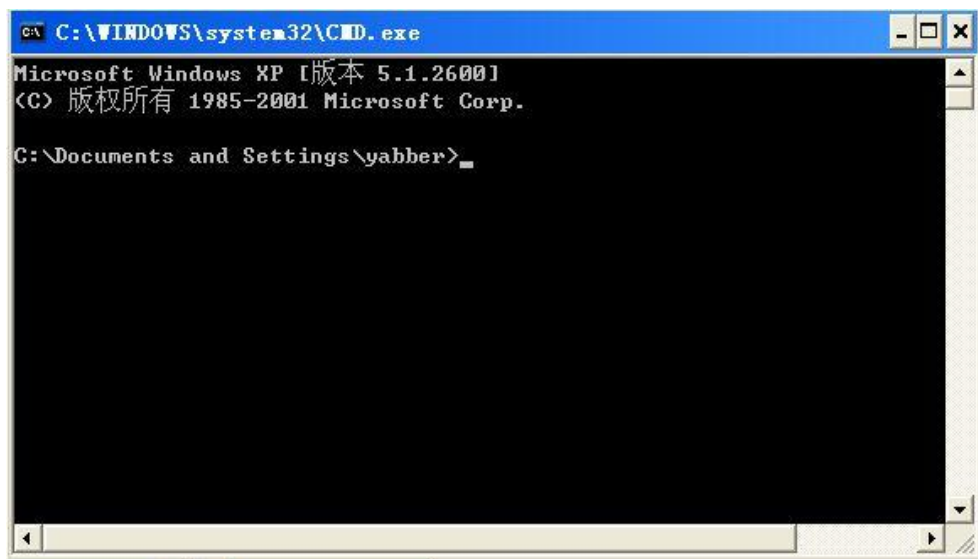


图3.13 输入cmd后的界面

3.5 Struts 2国际化应用

然后输入找到项目的class路径下，如“D:\lyb\workspace\Test\WebRoot\WEB-INF\classes”，得到如图3.14所示的界面。在命令行输入“native2ascii messageResource_temp.properties messageResource_zh_CN.properties”。这样就会在class路径下产生messageResource_zh_CN.properties文件。

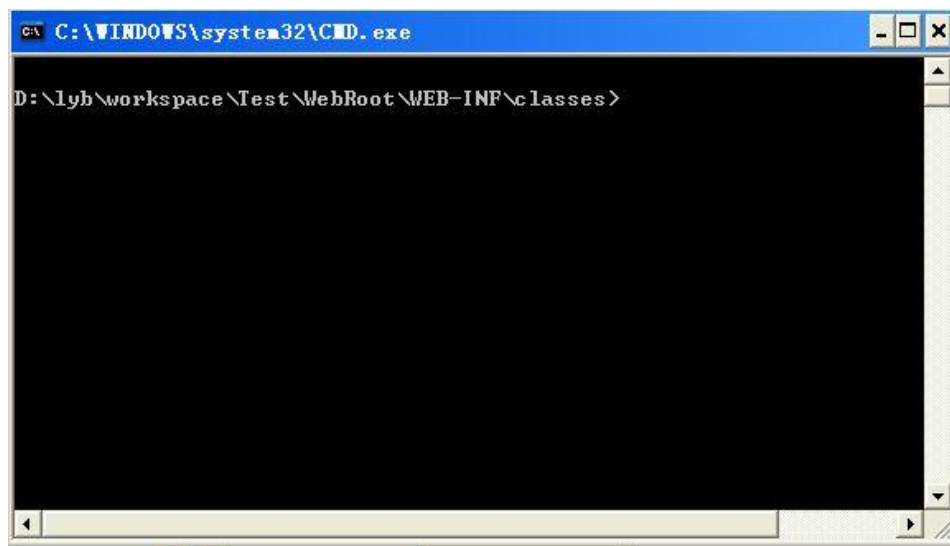
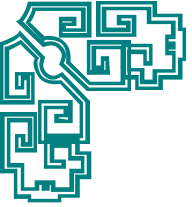


图3.14 找到项目的class路径

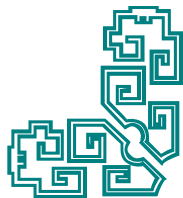
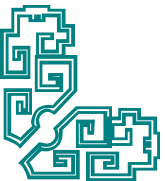


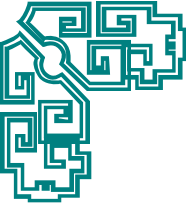
3.5 Struts 2国际化应用

◇ 5. 建立login.jsp文件

Struts 2访问国际化消息主要有以下三种方式：

- ① 在JSP页面中输出国际化消息，可以使用Struts 2的<s:text.../>标签，该标签可以指定name属性，该属性指定国际化资源文件中的key。
- ② 在Action中访问国际化消息，可以使用ActionSupport类的getText()方法，该方法可以接收一个参数，该参数指定了国际化资源文件中的key。
- ③ 在表单元素的label属性里输出国际化信息，可以为该表单标签指定一个key属性，该属性指定了国际化资源文件中的key。

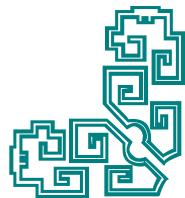
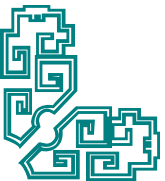




3.5 Struts 2国际化应用

下面是login.jsp文件代码:

```
<%@ page language="java" pageEncoding="utf-8"%>
<%@ taglib uri="/struts-tags" prefix="s"%>
<html>
<head></head>
<body>
    <s:i18n name="messageResource">
        <s:form action="login" method="post">
            <s:textfield name="username" key="username" size=
"20"></s:textfield>
            <s:password name="password" key="password" size=
"21"></s:password>
            <s:submit value="%{getText('login')}" />
        </s:form>
    </s:i18n>
</body>
</html>
```



3.5 Struts 2国际化应用

◆ 6. 部署运行

部署运行项目，右击“IE浏览器”，选择【属性】→【语言】菜单项，修改浏览器应用语言，当中文在最上方时表示当前为中文环境，而相应的英文在最上面时表示当前为英文环境。可以发现中文环境时登录界面如图3.15所示，英文环境时，登录界面如图3.16所示。

如果不能加载中英文资源，可在struts.properties文件中添加
`struts.locale=locale`



图3.15 中文环境时登录界面

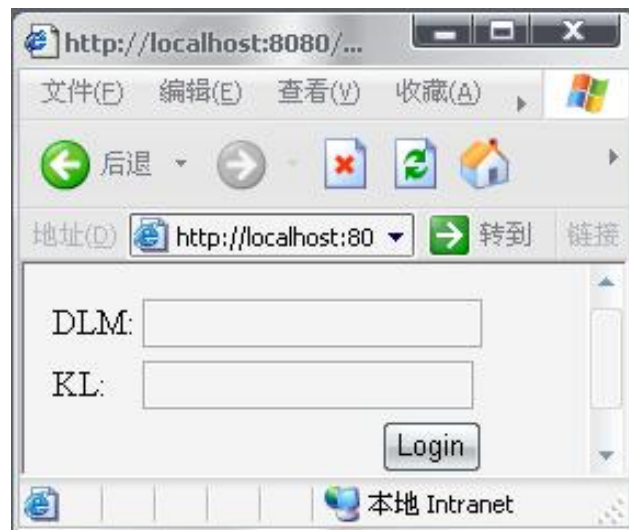


图3.16 英文环境时登录界面



3.6 Struts 2文件上传

✧ 3.6.1 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

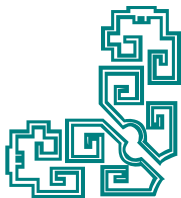
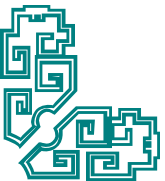
打开MyEclipse，建立一个Web项目，命名为“StrutsUpload”。

◆ 2. 加载Struts 2的基本类库

注意这里要把上面提到的upload及io两个Jar包也添加进来。

◆ 3. 修改web.xml

其内容见3.2.1节的第4步。



3.6.1 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，读者可以应用该文件，修改其中内容，也可以自己建立JSP文件，这里就用该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>
```

3.6.1 上传单个文件

◇ 5. Action类

在src文件夹下建立action包，在该包下建立自定义Action类UploadAction。该类的实现代码。

◇ 6. struts.xml文件

struts.xml是Struts 2应用中必不可少的一个文件，它是从页面通向Action类的桥梁，配置了该文件后，JSP文件的请求才能顺利地找到要处理请求的Action类。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>
```

3.6.1 上传单个文件

◇ 7. 建立 **success.jsp**

上传成功后，跳转到成功页面。代码非常简单：

```
<%@ 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>
```

3.6.1 上传单个文件

◇ 8. 部署运行

部署项目，启动Tomcat，在浏览器中输入
<http://localhost:8080/StrutsUpload/index.jsp>，出现如图3.17所示的界面，选择要上传的文件，单击【上传】按钮，就会跳转到如图3.18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3.17 运行界面



图3.18 成功界面

3.6.2 多文件上传

下面是在单个文件上传示例的基础上修改，来介绍多文件上传。



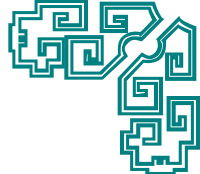
修改index.jsp:

```
<%@ 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>
```

页面完成以后，就可以修改对应的Action。[代码](#)修改。



3.7 Struts 2综合应用实例——添加学生信息

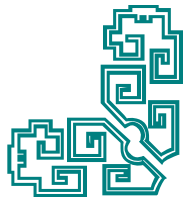
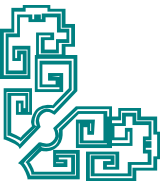


在该实例中，通过构建一个添加学生信息项目，来综合应用Struts 2的知识点，包括标签、Struts 2配置等。首先来看看添加学生信息的界面，如图3.19所示。

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

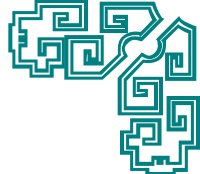
- 学号 (Student ID): Text input field with value "081134".
- 姓名 (Name): Text input field with value "李方方".
- 性别 (Gender): Radio buttons for "男" (Male) and "女" (Female). "男" is selected.
- 专业 (Major): Text input field with value "通信工程".
- 出生时间 (Birth Date): Text input field with value "1991-03-12" and a calendar icon.
- 备注 (Remarks): Text area with value "三好学生!".
- Buttons: "添加" (Add) and "重置" (Reset).

图3.19 添加学生信息界面





3.7 Struts 2综合应用实例——添加学生信息



◆ 1. 建立数据库

首先建立数据库XSCJ，建立学生表，表结构见附录A表XSB，该例中去掉了ZXF字段、ZP字段，关于照片上传、显示的内容会在后面的例子中详细讲解。

◆ 2. 建立Web项目

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

◆ 3. 加载Struts 2的基本类库

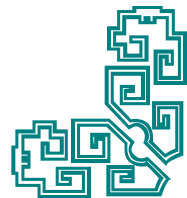
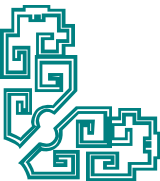
该步骤与3.2.1节的第3步步骤相同，这里不再赘述。

◆ 4. 修改web.xml

其内容见3.2.1节的第4步。

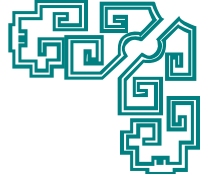
◆ 5. 建立stu.jsp文件

在项目的WebRoot文件夹下建立stu.jsp文件，[代码](#)。





3.7 Struts 2综合应用实例——添加学生信息

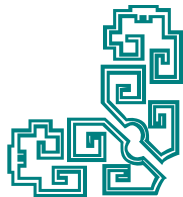
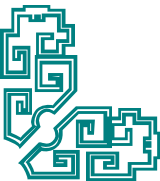


◆ 6. 建立表对应的JavaBean和DBConn类

在src文件夹下新建包“org.model”，在该包下建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：

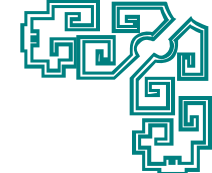
```
package org.model;
import java.sql.Date;
public class Xsb {
    private String xh;
    private String xm;
    private byte xb;
    private String zy;
    private Date cssj;
    private String bz;
    // 生成它们的getter和setter方法
}
```

注意，cssj为java.sql.Date类型。在src文件夹下建立包org.work，在该包下建立class文件，命名为“DBConn”，该类负责和数据库连接，[代码](#)。





3.7 Struts 2综合应用实例——添加学生信息



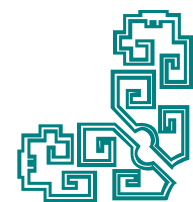
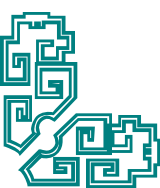
◇ 7. 建立Action类SaveAction

SaveAction.java代码。

◇ 8. 创建并配置struts.xml文件

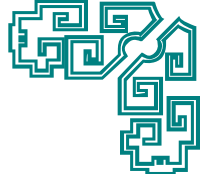
在src文件夹下建立该文件，代码如下：

```
<?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="save" class="org.action.SaveAction">
            <result name="success">/success.jsp</result>
            <result name="error">/stu.jsp</result>
        </action>
    </package>
</struts>
```





3.7 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>
```

◇ 10. 部署运行

部署后，启动Tomcat，在浏览器中输入

“http://localhost:8080/Example_Struts/stu.jsp”，可以看到如图3.19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。

