[MVC分层 2](#_Toc258795163)

[视图层使用的JavaBean 2](#_Toc258795164)

[配置文件 2](#_Toc258795165)

[web.xml文件 2](#_Toc258795166)

[<servlet>元素 3](#_Toc258795167)

[<welcome-file-list>元素 3](#_Toc258795168)

[<error-page>元素 4](#_Toc258795169)

[struts-config.xml文件 5](#_Toc258795170)

[<data-sources>元素 5](#_Toc258795171)

[<form-beans>元素 8](#_Toc258795172)

[<global-exceptions>元素 9](#_Toc258795173)

[<global-forwards>元素 9](#_Toc258795174)

[<action-mappings>元素 10](#_Toc258795175)

[<controller>元素 12](#_Toc258795176)

[<message-resources>元素 12](#_Toc258795177)

[<plug-in>元素 13](#_Toc258795178)

[表单验证 和 业务逻辑验证 13](#_Toc258795179)

[ActionForm 的 validate方法 14](#_Toc258795180)

[动态验证框架 15](#_Toc258795181)

[Action里的业务逻辑验证 19](#_Toc258795182)

[ActionForm 20](#_Toc258795183)

[ActionForm的scope=session(分步收集) 21](#_Toc258795184)

[动态ActionForm 24](#_Toc258795185)

[动态验证框架 25](#_Toc258795186)

[Action 27](#_Toc258795187)

[Struts内置的Action 27](#_Toc258795188)

[ForwardAction 27](#_Toc258795189)

[IncludeAction 29](#_Toc258795190)

[DispatchAction 30](#_Toc258795191)

[MappingDispatchAction 30](#_Toc258795192)

[LookupDispatchAction 32](#_Toc258795193)

[SwitchAction 33](#_Toc258795194)

[Token解决重复提交 34](#_Toc258795195)

[实用类 34](#_Toc258795196)

[ActionForward的使用 35](#_Toc258795197)

[ActionMapping 37](#_Toc258795198)

[表单回显数据 37](#_Toc258795199)

[Action 38](#_Toc258795200)

[ActionErrors 39](#_Toc258795201)

[Struts标签 39](#_Toc258795202)

[Bean 39](#_Toc258795203)

[HTML 40](#_Toc258795204)

[Logic 41](#_Toc258795205)

[Tiles 41](#_Toc258795206)

[国际化 41](#_Toc258795207)

[Struts 1中的文件上传 45](#_Toc258795208)

[类型转换器 46](#_Toc258795209)

[数组在页面之间的传递 51](#_Toc258795210)

[异常 51](#_Toc258795211)

[----------------------未分类--------------------- 52](#_Toc258795212)

# MVC分层

|  |  |
| --- | --- |
| View(视图层) | JSP, 自定义标签库, XML, HTML, JS, CSS, 消息资源, ActionForm |
| Controller(控制层) | XML配置文件, Servlet |
| Model(模型层) | JavaBean, EJB, Java实用类 |

## 视图层使用的JavaBean

|  |
| --- |
| 1. 必须提供不带参数的构造方法;  2. 为Bean的所有属性提供公共类型的get/set方法;  3. 对于boolean类型的属性, 如果存在isXXX()方法, 那么该方法返回boolean类型的属性值;  4. 对于数组类型的属性, 应该提供getXXX(int index)和setXXX(int index, Property ElementValue)方法, 用来读取或设置数组中的元素. |

# 配置文件

## web.xml文件

|  |
| --- |
|  |

### <servlet>元素

|  |
| --- |
| <servlet>  <servlet-name>action</servlet-name>  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>  <init-param>  <param-name>config</param-name>  <param-value>/WEB-INF/struts-config.xml</param-value>  </init-param>  <init-param>  <param-name>config/moduleA</param-name>  <param-value>/WEB-INF/moduleA/struts-config.xml</param-value>  </init-param>  <init-param>  <param-name>debug</param-name> <!-- 对Servlet日志输出的等级; 可去除 -->  <param-value>2</param-value>  </init-param>  <init-param>  <param-name>detail</param-name> <!-- Appach第三方组件日志输出的等级; 可去除-->  <param-value>2</param-value>  </init-param>  <load-on-startup>2</load-on-startup> <!--tomcat启动就执行ActionService方法初始化: Struts.cfg.xml的读取, 国际化资源文件的加载, -->  </servlet> |

### <welcome-file-list>元素

|  |
| --- |
| <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list> |
| **扩展:**  由于中<welcome-file-list>元素中不能配置Servlett映射, 因此不能把Struts的Action作为欢迎文件. 可以采用一种变通的方法来实现中欢迎文件中调用Struts 的 Action:  1. 在Struts配置文件中, 为被调用的Action创建一个全局(global)的转发项, 例如:  <global-forwards>  <forward name="welcome" path="welcome.do" />  </global-forwards>  2. 在index.jsp中加入如下代码:  <body>  <logic:forward name="welcome"/>  </body> |

### <error-page>元素

|  |
| --- |
| <error-page>**<!-- 配置404错误页面 -->**  <error-code>404</error-code>  <location>/404.jsp</location>  </error-page>  <error-page>**<!-- 配置505错误页面 -->**  <error-code>404</error-code>  <location>/505.jsp</location>  </error-page>  <error-page>**<!-- 配置ServletException异常页面 -->**  <exception-type>java.servlet.ServletException</exception-type>  <location>/common/system.error.jsp</location>  </error-page>  <error-page>**<!-- 配置IOException异常页面 -->**  <exception-type>java.io.IOException</exception-type>  <location>/common/system.ioerror.jsp</location>  </error-page> |

## struts-config.xml文件

|  |
| --- |
| "struts-config" must match "(display-name?,description?,data-sources?,form-beans?,global-exceptions?,global-forwards?,action-mappings?,controller?,message-resources\*,plug-in\*)". |

### <data-sources>元素

|  |
| --- |
| **<!--配置数据源 -->**  <data-sources>  <data-source type="org.apached.commons.dbcp.BasicDataSource" key="A"> <!--可以声明多个数据源, 则需要为每一个数据源分配唯一的key值, 然后在Action中通过以下方式获取特定的数据源: dataSourceA = getDataSource(request, "A"); -->  <set-property value="JDBC driver" property="driverClass"/>  <set-property value="url" property="jdbc:mysql://localhost:3306/test"/>  <set-property value="user" property="root"/>  <set-property value="password" property="root"/>  <set-property value="maxcount" property="2"/><!-- 同时打开的最大连接数 , 默认为2-->  <set-property value="mincount" property="1"/><!-- 同时打开的最小连接数, 默认为1 -->  </data-source>  </data-sources>  <data-sources>  <data-source type="org.apached.commons.dbcp.BasicDataSource" key="B">  <set-property value="JDBC driver" property="driverClass"/>  <set-property value="url" property="jdbc:mysql://localhost:3306/test"/>  <set-property value="user" property="root"/>  <set-property value="password" property="root"/>  <set-property value="maxcount" property="2"/><!-- 同时打开的最大连接数 , 默认为2-->  <set-property value="mincount" property="1"/><!-- 同时打开的最小连接数, 默认为1 -->  </data-source>  </data-sources> |
| 比较流行的数据源   |  |  |  | | --- | --- | --- | | 名称 | 供应商 | URL | | Poolmank | 开放源代码软件 |  | | Expresso | Jcorporate |  | | JDBC Pool | 开放源代码软件 |  | | DBCP | Jakarta |  | |
| 在Action中使用数据源访问数据库   |  | | --- | | **public** ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)  **throws** Exception {  javax.sql.DataSource dataSource;  java.sql.Connection myConnection = **null**;    **try**{  dataSource = getDataSource(request);  myConnection = dataSource.getConnection();    //这里写操作数据库的代码      }**catch**(SQLException sqle){  getServlet().log("Connection.process", sqle);  }**finally**{  **try**{  myConnection.close();  }**catch**(SQLException e){  getServlet().log("connection.close", e);  }  }    **return** mapping.findForward("error");  } | |

### <form-beans>元素

|  |
| --- |
| 普通ActinForm Bean:  <form-beans>  <form-bean name="loginForm"  type="com.bjsxt.struts.LoginActionForm"  className="指定form-bean元素对应的配置类, 默认值为org.apache.struts.config.FormBeanConfig. 如果在这里设置自定义的类, 该类必须继承FormBeanConfig类"  />  </form-beans> |
| 如果是动态ActinForm Bean:  <form-beans>  <form-bean name="loginForm" type="org.apache.struts.action.DynaActionForm">  <form-property name="username" type="java.lang.String" size="20" className="指定formproperty元素对应的配置类, 默认值为org.apache.struts.config.FormPropertyConfig"/>  <form-property name="age" type="java.lang.Integer" initial="以字符串的形式设置初始值, 如果没有设置该属性, 则整数类型的表单字段默认值为0, 对象类型表单字段的默认值为null", boolean类型的默认值为false />  </form-bean>  </form-beans> |

### <global-exceptions>元素

|  |
| --- |
| <global-exceptions>  <exception  className=*"指定exception元素对应的配置类, 默认值为org.apache.struts.config.ExceptionConfig"*  handler=*"指定异常处理类, 默认值为org.apache,struts.action.ExceptionHandler"*  key=*"指定在Resource Bundle中描述该异常的消息key"*  path=*"指定当异常发生时的转发路径"*  scope=*"指定ActionMessages实例的存放范围, 可选值包括request和session, 默认为request"*  type=*"指定exception元素对应的配置类, 默认为org.apache.struts.action.Exception"*  bundle=*"指定Resource Bundle"*  id=*"还不明确"*  >  </exception>  </global-exceptions> |

### <global-forwards>元素

|  |
| --- |
| <global-forwards>  <forward  className="和forward元素对应的配置类, 默认值为org.apache.struts.action.ActionForward"  contextRelative="true或false. 默认为true, 表示当path属性以 / 开头时, 给出的是相对于当前上下文的URL."  name="转发路径的逻辑名"  path="指定转发路径的URL. 必须以 / 开头. 当contextRelative=false时, 表示URL路径相对于当前应用; 当contextRelative=true时, 表示URL路径相对于当前上下文"  redirect="true或false, true表示重定向, false表示转发, 默认为false"  />  </global-forwards> |

### <action-mappings>元素

|  |
| --- |
| <!-- 一个Action对应一个ActionConfig对象 -->  <action-mappings>    <action  attribute="设置和Action关联的ActionForm Bean在request 或 session范围内的属性key"  className="和Action元素对应的配置元素, 默认值为org.apache.struts.action.ActionMapping"  forward="指定转发URL路径"  include="指定包含的URL路径"  input="指定当表单验证失败时, 将把请求转发的目标URL, 一般设置为输入表单的url"  name="指定和该Action关联的ActionForm Bean的名字. 该名字必须在form-bean元素中定义过"  path="指定访问Action的路径, 必须以/开头, 且不写后面的 .do"  parameter="指定ACtion的配置参数. 中Action中可以调用ActionMapping的getParameter()方法来读取该配置参数."  roles="指定允许调用该Action的安全角色. 多个角色之间以逗号隔开. 中处理请求时, RequestProcessor会根据该配置项来决定用户是否有调用该Action的权限."  scope="指定ActionForm Bean的存在范围, 可选request和session, 默认为session."  type="指定Action类的完整类名"  unknown="默认为false, 如果设置为true, 表示可以处理用户发出的所有无效Action URL, 一个配置文件里只有一个Action的unknown属性为true"  validate="默认true, 指定是否要调用ActionForm Bean的validate()方法"  >  <forward  name="success"  path="/bean-write.jsp"  contextRelative="true"  className=""  id=""  module=""  redirect="true"  />  </action> |
| 实例1: **JSP页面在/WEB-INF/下面**  <action path="/RegUserUI"  type="org.apache.struts.actions.ForwardAction"  parameter="/WEB-INF/user/RegUser.jsp">  </action> |

### <controller>元素

|  |
| --- |
| **<!-- 文件上传 -->**  <controller  bufferSize="指定上载文件的输入缓冲的大小, 默认值为4096"  className="指定和controller元素对应的匹配类, 默认值为 org.apache.struts.config.ControllerConfig"  contentType="指定相应结果的内容类型和字符编码, 默认值为text/html, 如果在Action和JSP网页中也设置了内容类型和字符编码, 将会覆盖该设置"  locale="默认false, 指定是否把Locale对象保存到当前用户的Session中."  processorClass="指定负责处理请求的java类的完整类名, 默认值为org.apache.struts.action.RequestProcessor, 如果把此项设置成自定义类, 那么这个自定义类必须继承org.apache.struts.action.RequestProcessor"  maxFileSize="//文件默认大小为250M, 可以配置单位: K, M, G"  nocache="默认为false, 如果设置成true, 则在相应结果总将加入特定的头参数, Pragma, Cache-Control和Expires, 防止页面被存储中客户浏览器的缓存中. 每次都清理浏览器缓存"  tempDir="上传文件的临时目录 "  /> |

### <message-resources>元素

|  |
| --- |
| **<!-- 配置国际化资源文件 -->**  <message-resources  className="和message-resources对应的配置类, 默认值为org.apache.struts.config.MessageResourcessConfig"  factory="指定消息资源的工厂类, 默认值为org.apache.struts.util.PropertyMessageResourcesFactory类"  key="指定Resource Bundle存放中ServletContext对象中时采用的属性key, 默认值为有Globals.MESSAGES\_KEY定义的字符串常量. 只允许有一个Resource Bundle采用默认的属性key"  id="我现在还不晓得啥子意思"  null="默认是true. 指定MessageResources类如何处理未知的消息key. 如果此项为true, 将返回空字符串, 如果此项为false, 将返回类似???global.label.missing???的字符串."  parameter="例如:res.MessageResources, res为包名, MessageResources是默认资源文件的文件名"  /> |

### <plug-in>元素

|  |
| --- |
| <plug-in className="org.apache.struts.validator.ValidatorPlugIn">  <set-property property="pathnames" value="/WEB-INF/validator-rules.xml, /WEB-INF/validation.xml" />  </plug-in> |

# 表单验证 和 业务逻辑验证

|  |
| --- |
| **备注:**  Struts早期的版本中使用ActionError类来表示错误消息; ActionError类是ActionMessage的子类.  Struts 1.2将废弃ActionError, 统一采用ActionMessage类来表示正常或错误消息. |
| **JSP中通过<html:errors>标签**  <h1><bean:message key="user.title"/></h1> **<!-- 直接去资源文件中的内容, 用**<bean:message>**标签, 这就是国际化 -->**  <hr/>  **<!-- 显示错误提示信息, 第一种: 用**<html:messages> 和 <bean:write>**标签 -->**  <font color="red">  <html:messages id="msg" property="error1">  <bean:write name="msg"/>  </html:messages>  </font>  <font color="blue">  <html:messages id="msg" property="error2">  <bean:write name="msg"/>  </html:messages>  </font>  <hr/>  <font color="red">  **<!-- 显示错误提示信息, 第二种: 用**<html:errors>**标签-->**  <html:errors property="request中保存的ActionMessage的别名"/> //从request范围内读取ActionMessages对象, 再从中读取ActionMessage对象, 把它包含的错误信息显示中页面上.  </font> |

## ActionForm 的 validate方法

|  |
| --- |
| **public** ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {  ActionErrors errors = **new** ActionErrors();  **if**((username == **null**) || username.length()<1){  errors.add("username", **new** ActionMessage("请输入用户名!"));  }  **return** errors;  }  如果返回的ActionErrors对象为null, 或者是不包含任何ActionMessage的对象, 就表示没有错误. |
| 说明:  第1步: ActionForm 的 validate ()方法返回一个ActionErrors对象, 里面包含一个ActionMessage对象, 这个ActionMessage对象中封装了错误消息, 消息key为"username",  若ActionErrors中不包含任何ActionMessage对象,则不执行以下两步.  第2步: ActionServlet把ActionForm的validate()方法返回的ActionErrors对象保存中request范围内, 然后通过<action>元素的input属性, 把客户端请求转发给login.jsp页面.  第3步: 页面通过<html:errors>标签从request范围内读取ActionErrors对象, 再从ActionErrors对象中读取ActionMessage对象, 把它包含的错误消息显示中网页上. |
| request>ActionErrors>ActionMessage>"username" |

## 动态验证框架

|  |
| --- |
| DynaValidatorForm支持中动态ActionForm中使用Validator框架;  ValidatorForm支持在标准ActionForm中使用Validator框架.  以上两个的配置方法都一样: 必须的两个jar文件: jakarta-oro.jar commons-validator.jar 和一个必须的配置文件: validator-rules.xml  如果验证失败, 就会创建包含错误消息的ActionMessage对象, 并把该对象添加到ActionMessages集合对象中. |
| 中struts-config.xml中配置插件  <plug-in className="org.apache.struts.validator.ValidatorPlugIn">  <set-property property="pathnames" value="/WEB-INF/validator-rules.xml, /WEB-INF/validation.xml" />  </plug-in> |
| 需要手动配置的验证文件: validation.xml  <?xml version="1.0" encoding="utf-8" ?>  <!DOCTYPE form-validation PUBLIC  "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.1.3//EN"  "http://jakarta.apache.org/commons/dtds/validator\_1\_1\_3.dtd">  <form-validation>  <global> <!-- global元素用于定义全局常量, 在本文件所有formset中都能使用 -->  <constant>  <constant-name>phone</constant-name> <!-- 在formset中可以通过 ${phome} 来访问 -->  <constant-value>^\d{8}\d\*$</constant-value>  </constant>  <constant>  <constant-name>zip</constant-name>  <constant-value>^\d{6}\d\*$</constant-value>  </constant>  </global>  <formset>  <constant> <!-- 定义局部常量, 只在本formset中都能使用 -->  <constant-name>phone</constant-name> <!-- 在formset中可以通过 ${phome} 来访问 -->  <constant-value>^\d{8}\d\*$</constant-value>  </constant>  <form name="loginForm"> <!-- ActionForm的名字 -->  <field property="username" depends="required"> <!-- ActionForm里的属性 -->  <arg key="prompt.username"/>  </field>  <field property="password" depends="required,mask">  <msg name="指定验证规则的名字" resource="默认为true: 表示使用来自Resource Bundle的消息; 如果为false, 表示直接中key属性中设置消息文本." key="参考前面的resource属性"/> <!-- 消息文本 -->  <arg0 key="" name="" resource=""/> <!-- arg0 到 arg3 元素用于替换符合消息中的参数, 与msg属性用户大致相同 -->  <arg key="prompt.password"/>  <var>  <var-name>mask</var-name>  <var-value>^[0-9a-zA-Z]\*$</var-value>  </var>  </field>  </form>  </formset>  </form-validation> |
| GenericValidator类的验证方法:   |  |  | | --- | --- | | isBlankOrNull | 字段是否为null,或长度是否为0 | | isByte | 字段是否可以被转换为有效的byte类型的数据 | | isCreditCard | 字段是否为有效的信用卡号 | | isDate | 字段是否为有效的日期 | | isDouble | 字段是否是可以被转换为有效的double类型的数据. | | isEmail | 字段是否为有效的E-Mail地址 | | isFloat | 字段是否可以被转换为有效的float类型的数据 | | isInRange | 字段是否介于最小值和最大值之间 | | isInt | 字段是否可以被转换为有效的int类型的数据 | | isLong | 字段是否可以被转换为有效的long类型的数据 | | isShort | 字段是否可以被转换为有效的short类型的数据. | | matchRegexp | 字段是否和正规表达式匹配 | | maxLength | 字段是否小于或等于最大值 | | minLength | 字段是否大于或等于最小值 |   中Struts中定义了专门用于验证表单字段的org.apache.struts.validator.FieldChecks类, 它提供了和GenericValidator类似的方法:   |  |  | | --- | --- | | validateByte |  | | validateCreditCard |  | | validateDate |  | | validateDouble |  | | validateEmail |  | | validateFloat |  | | validateInteger |  | | validateLong |  | | validateMask | 验证字段值是否和一个给定的Jakarta正规表达式(Regular Expression, 简称RegExp)匹配, Jakarta RegExp表达式用于定义字符模式, 例如"^[a-zA-Z]\*$"表示字符串必须包含字母; "^\d{5}\d#$"表示字符串必须是五位数字. | | validateMinLength |  | | validateMaxLength |  | | validateRange |  | | validateRequired |  | | validateShort |  | |
|  |

## Action里的业务逻辑验证

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | ActionMessages messages = **new** ActionMessages();//创建ActionMessages对象  String badUserName = "Monster"; //badUserName是用于传递给key对应value中的参数  //ActionMessage message = new ActionMessage("user.login.success", new Object[]{username});  ActionMessage message1 = **new** ActionMessage("hello.dont.talk.to.monster", badUserName);//根据key找到对应的value, badUserName是value的参数值 | | messages.add("username", message1);//把message1添加到messages中, username只是用于前台提取内容使用的别名.  //把消息放入集合里, 这里是普通消息. 所以用saveMessages  **this**.saveMessages (request, messages);//把消息放入集合里, 这里是普通消息, 所以用saveMessages, 保存messages, 用于前台提取. | | messages.add("error1", message);  //把消息放入集合里, 这里是错误消息. 所以用saveErrors  **this**.saveErrors(request, messages); | |
| execute()方法把ActionMessage对象保存在ActionMessages对象中, 再把ActionMessages对象存放中request范围内, 最后返回一个ActionForward对象, 该对象包含的请求转发路径为<action>元素的input属性指定的login.jsp |
| request>ActionMessages> ActionMessage >"username" |

# ActionForm

|  |
| --- |
| ActionForm的生命周期:  1. 控制器接收到请求;  2. 从scope范围内查找ActionForm实例, 如果该实例不存在, 就自动创建一个新的实例;  3. 调用ActionForm的reset()方法;  4. 把ActionForm实例保存在scope指定的范围内;  5. 把用户输入的表单数据组装到ActionForm中;  6. 如果<action>的validate属性为true, 则调用ActionForm的validate()方法.  7.1 存在验证错误: 把请求转发给<action>的input属性指定的web组件, ActionForm实例依然保存在scope指定的范围内.  7.2 无验证错误: 调用Action的excute()方法, 把ActionForm实例传递给execute()方法;  8. 把请求转发给其他web组件, ActionForm实例依然保存中scope范围内. |

## ActionForm的scope=session(分步收集)

|  |
| --- |
| **第一种方法:**  多个Action共同使用一个ActionForm,  不能使用ActionForm中的reset()方法来清理.  需要在ActionForm自己定义一个myReset()方法, 清理表单数据写在里面, 在进入第一个页面的Action中调用ActionForm.myReset() |
| **第二种方法:** (更好, 但要比上一种复杂一丁点):  原理:  1. ActionForm中必须要提供和HTML表单的隐藏page字段, 对应的page属性.  2. 在reset()方法中, 只能把和当前正在处理的比捣蛋相关的属性回复称默认值.  3. 中validate()方法中, 仅对和当前表单相关的属性进行校验.  示例代码:  **public** **class** CopyOfLoginActionForm **extends** ActionForm {  **private** String username;//第一页的表单内容  **private** **int** age; //第二页的表单内容  **private** **int** page;//标记第一页提交还是第二页提交  //这里省略了所有属性的getXXX(), setXXX()    @Override  **public** **void** reset(ActionMapping mapping, HttpServletRequest request) {  **int** numPage = 0;  **try**{  numPage = **new** Integer(request.getParameter("page")).intValue();  }**catch**(Exception e){  }    **if**(numPage == 1){  username = **null**;  }  **if**(numPage == 2){  age = 0;  }  page = 0;  }  @Override  **public** ActionErrors validate(ActionMapping mapping,  HttpServletRequest request) {  ActionErrors errors = **new** ActionErrors();  **int** numPage = 0;  **try**{  numPage = **new** Integer(page).intValue();  }**catch**(Exception e){}    **if**(numPage == 1){  **if**((username == **null**) || username.length()<1){  errors.add("username", **new** ActionMessage("请输入用户名!"));  }  }  **if**(numPage == 2){  **if**(age == 0){  errors.add("age", **new** ActionMessage("请输入年龄!"));  }  }  **return** errors;  }    } |

## 动态ActionForm

|  |
| --- |
| 优点: 减少ActionForm的数量, DynaActionForm继承自ActionForm, 具有它的所有功能  动态ActionForm提供了initalize()方法, 默认值在<form-property>的initial属性中配置;  提供了一个空的reset()方法, 不执行任何重置操作. |
| struts-cfg.xml中的配置:  <form-bean name="dynaForm" type="org.apache.struts.action.DynaActionForm"> <!-- type是框架提供的, 不用修改 -->  <form-property name="username" type="java.lang.String"/> <!-- 属性列表, 属性名要与表单中的相同, 类型要写全类名 -->  <form-property name="age" type="java.lang.Integer" size=”数组长度” initial="初始化值"/>  </form-bean>   |  | | --- | | 类型 | | java.lang.BigDecimal | | java.lang.BigInteger | | java.lang.Boolean | | java.lang.Byte | | java.lang.Character | | java.lang.Class | | java.lang.double | | java.lang.Float | | java.lang.Integer | | java.lang.Long | | java.lang.Short | | java.lang.String | | java.sql.Date | | java.sql.Time | | java.sql.Timestamp | |
| 上面已经总结了: [动态验证框架](#_动态验证框架) |
| DynaActionForm把所有的属性保存中一个Map类对象中, 并提供了下面的用于访问所有属性的通用方法:  public Object get(String name)  public void set(String name, Object value);  例如在action中使用动态ActionForm中的数据  DynaActionForm daf = (DynaActionForm)form;  String username = (String)daf.get("username"); |
| JSP中使用动态ActionForm中的数据:  ${**dynaForm**.**map**.属性名}  // **dynaForm**: 是ActionForm的别名  // **map:** 是DynaActionForm中的属性, 这个map保存了所有属性键值对. 写法样式固定. |

## 动态验证框架

|  |
| --- |
| 1. 拷贝国际化资源文件到src下面;  2. ActionForm必须使用org.apache.struts.validator.DynaValidatorForm; (DynaValidatorForm继承自DynaActionForm)  <form-beans>  <form-bean name="loginForm" type="org.apache.struts.validator.DynaValidatorForm">  <form-property name="username" type="java.lang.String"/>  <form-property name="password" type="java.lang.String"/>  </form-bean>  </form-beans>  3. 在struts-config.xml中注册国际化资源文件basename,  <message-resources parameter="MessageResources" />  4. 以插件形式注册验证文件.  <plug-in className="org.apache.struts.validator.ValidatorPlugIn">  <set-property property="pathnames" value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>  </plug-in> |
| 5. 配置validation.xml  <?xml version="1.0" encoding="ISO-8859-1" ?>  <!DOCTYPE form-validation PUBLIC  "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.1.3//EN"  "http://jakarta.apache.org/commons/dtds/validator\_1\_1\_3.dtd">  <form-validation>  <formset>  <form name="loginForm">  <field property="username" depends="required">  <arg key="prompt.username"/>  </field>  <field property="password" depends="required,mask">  <arg key="prompt.password"/>  <var>  <var-name>mask</var-name>  <var-value>^[0-9a-zA-Z]\*$</var-value>  </var>  </field>  </form>  </formset>  </form-validation> |
| 7. JSP中使用<bean:message>标签读取国际化消息文本. |

# Action

## Struts内置的Action

### ForwardAction

|  |
| --- |
| ForwardAction专门用于转发请求, 不执行任何其他业务操作.  <action  path="指定访问Action的路径, 必须以/开头, 且不写后面的 .do"  parameter="指定ACtion的配置参数. 中Action中可以调用ActionMapping的getParameter()方法来读取该配置参数."  type=" type="org.apache.struts.actions.ForwardAction" **//这个不能修改**  scope="指定ActionForm Bean的存在范围, 可选request和session, 默认为session."  name="指定和该Action关联的ActionForm Bean的名字. 该名字必须在form-bean元素中定义过"  validate="默认true, 指定是否要调用ActionForm Bean的validate()方法"  input="指定当表单验证失败时, 将把请求转发的目标URL, 一般设置为输入表单的url"  />  ActionServlet把请求转发给ForwardAction, ForwardAction再把请求转发给action元素中的parameter属性指向的web组件. |
| 此外, 还可以使用action元素的forward属性来实现请求转发:  <action  path="指定访问Action的路径, 必须以/开头, 且不写后面的 .do"  forward="指定转发URL路径"  scope="指定ActionForm Bean的存在范围, 可选request和session, 默认为session."  name="指定和该Action关联的ActionForm Bean的名字. 该名字必须在form-bean元素中定义过"  validate="默认true, 指定是否要调用ActionForm Bean的validate()方法"  input="指定当表单验证失败时, 将把请求转发的目标URL, 一般设置为输入表单的url"  >  ActionServlet把请求转发给ForwardAction, ForwardAction再把请求转发给action元素中的forward属性指向的web组件. |

### IncludeAction

|  |
| --- |
| 中JSP中, 尽管可以直接通过<include>指令包含另一个web组件, 但是struts框架提倡先把请求转发给控制器, 再由控制器来负责包含其他web组件.  web组件可以通过IncludeAction类来包含另一个web组件. |
| <action  path="指定访问Action的路径, 必须以/开头, 且不写后面的 .do"  parameter="sampleInclude.jsp"  **//指定需要包含的web组件**  type="org.apache.struts.actions.IncludeAction" **//这个不能修改**  name="指定和该Action关联的ActionForm Bean的名字. 该名字必须在form-bean元素中定义过"  input="指定当表单验证失败时, 将把请求转发的目标URL, 一般设置为输入表单的url"  scope="指定ActionForm Bean的存在范围, 可选request和session, 默认为session."  /> |
| <action  path="指定访问Action的路径, 必须以/开头, 且不写后面的 .do"  include="sampleInclude.jsp"  **//指定需要包含的web组件**  name="指定和该Action关联的ActionForm Bean的名字. 该名字必须在form-bean元素中定义过"  input="指定当表单验证失败时, 将把请求转发的目标URL, 一般设置为输入表单的url"  scope="指定ActionForm Bean的存在范围, 可选request和session, 默认为session."  /> |

### DispatchAction

|  |
| --- |
| 同一个Action类中完成一组相关的业务操作, 不必覆盖excute()方法.  <action  path="aaaaaaaaaaaa"  parameter="method"  **//跟JSP中的方法参数名一样.**  **........**  />  JSP中请求为: http://swu.edu.cn/aaaaaaaaaaaa.do?method=login&username=youxuan&password=xiaoguo  DispatchAction子类将调用login这个方法. |

### MappingDispatchAction

|  |  |
| --- | --- |
| jsp页面 | <a href="${pageContext.request.contextPath }/creatUser.do">creatUser</a><br />  <a href="${pageContext.request.contextPath }/retrieveUser.do">retrieveUser</a><br />  <a href="${pageContext.request.contextPath }/updateUser.do">updateUser</a><br />  <a href="${pageContext.request.contextPath }/deleteUser.do">deleteUser</a><br /> |
| 配置文件 | <action path="/creatUser"  type="com.yourcompany.struts.action.UserMappingDispatchAction"  parameter="creatUser" />  <action path="/retrieveUser"  type="com.yourcompany.struts.action.UserMappingDispatchAction"  parameter="retrieveUser" />  <action path="/updateUser"  type="com.yourcompany.struts.action.UserMappingDispatchAction"  parameter="updateUser" />  <action path="/deleteUser"  type="com.yourcompany.struts.action.UserMappingDispatchAction"  parameter="deleteUser" /> |
| Action | package com.yourcompany.struts.action;  import java.io.PrintWriter;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import org.apache.struts.action.ActionForm;  import org.apache.struts.action.ActionForward;  import org.apache.struts.action.ActionMapping;  import org.apache.struts.actions.MappingDispatchAction;  public class UserMappingDispatchAction extends MappingDispatchAction {  public ActionForward creatUser(ActionMapping mapping, ActionForm form,  HttpServletRequest request, HttpServletResponse response)  throws Exception {  PrintWriter out = response.getWriter();  out.print("creatUser");  return null;  }  public ActionForward retrieveUser(ActionMapping mapping, ActionForm form,  HttpServletRequest request, HttpServletResponse response)  throws Exception {  PrintWriter out = response.getWriter();  out.print("retrieveUser");  return null;  }    public ActionForward updateUser(ActionMapping mapping, ActionForm form,  HttpServletRequest request, HttpServletResponse response)  throws Exception {  PrintWriter out = response.getWriter();  out.print("updateUser");  return null;  }    public ActionForward deleteUser(ActionMapping mapping, ActionForm form,  HttpServletRequest request, HttpServletResponse response)  throws Exception {  PrintWriter out = response.getWriter();  out.print("deleteUser");  return null;  }    } |

### LookupDispatchAction

|  |
| --- |
| LookupDispatchAction是DispatchAction的子类,  中LookupDispatchAction类中也可以定义多个业务方法.  通常LookupDispatchAction主要应用于在一个表单中有多个提交按钮, 而这些按钮又有一个共同的名字的场合, 这些按钮的名字和具体的ActionMapping的parameter属性值相对应. |
| JSP中的form  <html:submit property="aaaaaa">  <bean:message key="button.saveorder"/>  </html:submit>  <html:submit property="aaaaaa">  <bean:message key="button.checkout"/>  </html:submit>  **资源文件里的key<-->value**  button.saveorder=Save Order  button.checkout=Check Out  **生成以下代码**  <input type="submit" name="aaaaaa" value="Save Order" />  <input type="submit" name="aaaaaa" value="Check Out" /> |
| LookupDispatchAction子类中实现getKeyMethodMap()方法  protected Map getKeyMethodMap(){  Map map = new HashMap();  map.put("button.checkout", "checkout"); //map的key必须跟资源文件的key相同, value必须跟本类中的方法名相同.  map.put("button.saveorder", "saveorder");  return map;  }  本类中还有两个跟excute()方法签名相同的方法: saveorder() 和 checkout() |
| <action  path="aaaaaaaaaaaa" **// parameter属性值必须跟JSP中property值相同,**  parameter="aaaaaa"  **........**  /> |

### SwitchAction

|  |
| --- |
| SwitchAction用于子应用模块之间的切换. |
| <action  path="/action/toModule" **// parameter属性值必须跟JSP中property值相同,**  type="org.apache.struts.actions.SwitchAction"  **........**  /> |
| 请求SwitchAction的URL需要提供两个请求参数:  1. prefix: 指定子应用模块的前缀, 以 / 开头, 默认之应用模块的前缀为空字符串"" .  2. page: 指定被请求ewb组件的URL, 只需指定相对于被切换后的子应用模块的相对路径.  例如: 如果要从默认模块切换到ModuleB, 并把请求转发给ModuleB的/index.do 则URL是:  http://locaohost/action/toModule?prefix=/moduleB&page=/index.do  此时访问SwitchAction的URI为/action/toModule |

# Token解决重复提交

|  |
| --- |
| token 又名 同步令牌 |

# 实用类

|  |
| --- |
| Struts提供了一组实用类, 由于都是线程安全的, 所以可以被所有的Struts应用共享.  他们的方法都是静态的.  1. RequestUtils类  2. TagUtils类  3. ModuleUtils类  4. Globals类 |

## ActionForward的使用

|  |
| --- |
| **struts-config.xml文件不允许动态修改**  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.apache.struts.action.Action;  **import** org.apache.struts.action.ActionForm;  **import** org.apache.struts.action.ActionForward;  **import** org.apache.struts.action.ActionMapping;  **public** **class** MustLoginAction **extends** Action {  @Override  **public** ActionForward execute(ActionMapping mapping, ActionForm form,  HttpServletRequest request, HttpServletResponse response)  **throws** Exception {    **if** (request.getSession().getAttribute("user") == **null**) {  //ActionForward af = mapping.findForward("login");  //af.setRedirect(**false**); //尝试修改struts-config.xml抛出异常; struts-config.xml文件不能修改  //**return** af;  response.sendRedirect(request.getContextPath() + “/login.jsp”);  return null; //自己做了重定向, 返回一个null.  }  **return** mapping.findForward("success");  }  } |
| **动态ActionForward，可以运行期动态调用不同的目标页面**  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.apache.struts.action.Action;  **import** org.apache.struts.action.ActionForm;  **import** org.apache.struts.action.ActionForward;  **import** org.apache.struts.action.ActionMapping;  **public** **class** DynaActionForwardTestAction **extends** Action {  @Override  **public** ActionForward execute(ActionMapping mapping, ActionForm form,  HttpServletRequest request, HttpServletResponse response)  **throws** Exception {    String page = request.getParameter("page");  // 原始写法  // ActionForward af = null;  // if ("1".equals(page)) {  // af = mapping.findForward("page1");  // }else if ("2".equals(page)) {  // af = mapping.findForward("page2");  // }  // return af;    ActionForward af = **new** ActionForward();  af.setPath("/page" + page + ".jsp?name=Tom");  **return** af;  }  } |

## ActionMapping

|  |
| --- |
| <action path="/login1"  forward="/login.jsp" /> //**转发Action** |
| <action path="/testunknown"  unknown="true" //**错误.do请求路径, 是一种容错处理. 只能有一个Action配置**unknown="true"  forward="/testunknown.jsp" /> |

## 表单回显数据

|  |
| --- |
| **用EL表达式**  <form action="login.do" method="post">  用户：<input type="text" name="username" value="${loginForm.username }"><br>  密码：<input type="password" name="password" value="${loginForm.password }"><br>  <input type="submit" value="登录">  </form> |
| **用Struts的HTML标签**  <%@ taglib prefix="html" uri="http://struts.apache.org/tags-html"%>  <html:form action="login2.do" method="post">  用户：<html:text property="username"/><br>  密码：<html:password property="password"/><br>  <html:submit value="登录"/>  </html:form> |

## Action

|  |
| --- |
| Action的作用:  1. 接收表单数据, 并校验;  2. 调用业务逻辑方法;  3. 检测并处理异常(手动处理异常, 声明式异常);  4. 根据业务逻辑进行转向操作 |
| public class LoginAction extends Action {  @Override  public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws Exception {  LoginActionForm laf = (LoginActionForm) form;  String username = laf.getUsername();  String password = laf.getPassword();  ......  request.setAttribute("errorinfo", errorInfo);  return mapping.findForward("error");  }  } |

## ActionErrors

|  |
| --- |
| ActionErrors是对错误信息的包装, 一旦执行action或者form.validate中出现异常, 即可产生一个ActionError并最终加入到ActionErrors. 在Form验证的过程中, 如果有Error发生, 则会将页面重新导向至输入页, 并提示错误.  如果ActionForm的validate()方法返回null 或返回一个不包含ActionMessage的ActionErrors对象, 就表示表单验证成功. |

# Struts标签

|  |  |
| --- | --- |
| 配置 | 配置：在struts-config.xml文件中加入  <message-resources parameter="MessageResources" />  \* 拷贝MessageResources.properties文件到src下 |
| HTML | <%@ taglib prefix="html" uri="http://struts.apache.org/tags-html"%> |
| Bean | <%@ taglib prefix="bean" uri="http://struts.apache.org/tags-bean"%> |
| Logic | <%@ taglib prefix="logic" uri="http://struts.apache.org/tags-logic"%> |

## Bean

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | bundle |  | | filter | true/false (是否过滤HTML标签) | | format | yyyy-MM-dd HH:mm:ss  ###,###.#### (格式化数字, 小数位数不够尾部不补0)  ###,###.000 (格式化数字, 小数位数不够尾部就补0) | | formatKey |  | | ignore |  | | locale |  | | \*name | scope中的变量名 | | property | 取得对象的属性  支持点导航: property =”user.name” | | scope | 设置取数据的范围 |   <bean:message key="资源文件中的key" arg0="给key对应值传参数"/> 用于输出本地化的文本内容, 区别于 <html:errors property="request中保存的ActionMessage的别名"/>  <bean:write name=”scope中的Bean的名称, 指向指定对象” property=”前面对象的属性” filter=”过滤吗: true/false” format=”yyyy-MM-dd HH:mm:ss”> |

## HTML

|  |
| --- |
| 备注:  html标签会自动url重写,自动回显表单数据. |
| 自动生成的<html:base/>这个东西最好删掉, 因为我们一般都用绝对路径, 绝对路径不容易出错. |
| <html:errors property="request中保存的ActionMessage的别名"/> //从request范围内读取ActionMessages对象, 再从中读取ActionMessage对象, 把它包含的错误信息显示中页面上.  <html:submit property="submit" value="提交"/>  <html:reset property="reset" value="重置"/> |
| <html:form action="/login" >**区别于**<form action="${pageContext.request.contextPath}/login.do" method="post"> |
| <html:link forward="#">asdf</html:link> <!-- forward属性和struts-cfg.xml中的<global-forwards>元素的<forward>子元素匹配. --> |
| <a href='<html:rewrite action="/VerifyUserName"/>'>用户名是否存在</a><br/>  struts配置文件中有/VerifyUserName这个action, 这个<html:rewrite>能够自动生成项目名和后面的.do |
| <html:text property="username" />  <html:submit/>  <html:reset/> |

## Logic

|  |  |
| --- | --- |
| <logic:empty name=”scope中的变量”>和<logic:notEmpty>  <logic:notEmpty name=”scope中的变量”>和<logic: notEmpty > | 是否为空 |
| <logic:present name=”scope中的变量的别名” property=”前面对象的属性” >和<logic:notPresent> | 是否存在 |
| <logic:iterate name=" scope中的变量" id="临时存储数据的变量"></logic:iterate> | 迭代  只能迭代  Collection(ArrayList, Vector)  Enumeration  Iterator  Map(HashMap, Hashtable, TreeMap) |
| <logic:equal name="project" property="verifyState" value="1">等待审核中 </logic:equal> | name: 对象 property: 对象的属性 |

## Tiles

|  |  |
| --- | --- |
| <tiles:insert> |  |

## nested

|  |  |
| --- | --- |
| <nested:nest> |  |
| <nested:writeNesting> |  |
| <nested:root> |  |

Validator

|  |
| --- |
| ActionErrors errors = new ActionErrors();  ActionMessage message = new ActionMessage(“error”);  errors.add(“error”, message);  return errors; |

# 国际化

|  |
| --- |
| 拷贝资源文件(MessageResources.properties)到src下面,  然后在Struts-cfg.xml中添加下列语句:  <message-resources parameter="res.MessageResources" />  res.MessageResources.properties文件路径: |
| **如何在后台代码中 读取资源文件 的内容**  MessageResources message = getResources(request);  String title = message.getMessage("hello.jsp.title"); |
| **如何在JSP中 读取资源文件 的内容, 读取 错误提示信息**  <body>  <h1><bean:message key="user.title"/></h1> **<!-- 直接去资源文件中的内容, 用**<bean:message>**标签, 这就是国际化 -->**  <hr/>  **<!-- 显示错误提示信息, 用**<html:messages> 和 <bean:write>**标签 -->**  <font color="red">  <html:messages id="msg" property="error1">  <bean:write name="msg"/>  </html:messages>  </font>  <font color="blue">  <html:messages id="msg" property="error2">  <bean:write name="msg"/>  </html:messages>  </font>  <hr/>  <font color="red">  <html:errors prefix="error1" /> **<!-- 显示错误提示信息, 用**<html:errors>**标签-->**  </font>  <html:img pageKey="hello.jsp.page.stutsimage" altKey="hello.jsp.page.struts" /> **<!-- 图片国际化 -->**  <form action="login.do" method="post">  <bean:message key="user.username"/>：<input type="text" name="username"><br>  <bean:message key="user.password"/>：<input type="password" name="password"><br>  <html:submit property="submit"> **<!-- 按钮国际化 -->**  <bean:message key="user.button.login"/>  </html:submit>  </form>  </body> |
| **选择语言的Action代码:**  **package** com.bjsxt.struts;  **import** java.util.Locale;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.apache.struts.action.Action;  **import** org.apache.struts.action.ActionForm;  **import** org.apache.struts.action.ActionForward;  **import** org.apache.struts.action.ActionMapping;  **public** **class** ChangeLanguageAction **extends** Action {  @Override  **public** ActionForward execute(ActionMapping mapping, ActionForm form,  HttpServletRequest request, HttpServletResponse response)  **throws** Exception {  String lang = request.getParameter("lang");//lang=zh代表中文, lang=en代表英文,    Locale currentLocale = Locale.getDefault();  **if** ("zh".equals(lang)) {  currentLocale = **new** Locale("zh", "CN");  }**else** **if**("en".equals(lang)) {  currentLocale = **new** Locale("en", "US");  }  //request.getSession().setAttribute(Globals.LOCALE\_KEY, currentLocale);    **this**.setLocale(request, currentLocale);  **return** mapping.findForward("index");  }  } |

# Struts 1中的文件上传

|  |
| --- |
| JSP中的上传表单:  <form action="upload.do" method="post" enctype="multipart/form-data">  <input type=”file” name=”myfile” /><br />  <input type=”submit” /><br />  </form>  JSP中输出文件属性:  文件名: ${uploadForm.myfile.fileName} <!-- 红色部分是固定写法 --> |
| ActionForm中:  //必须采用FormFile  **private** FormFile myfile; |
| Action中:  UploadActionForm uaf = (UploadActionForm)form;  System.out.println("title" + uaf.getTitle());  FormFile myFile = uaf.getMyfile(); //得到文件  **if** (myFile != **null**) {  System.out.println("fileName=" + myFile.getFileName()); //读取文件名  FileOutputStream fos = **new** FileOutputStream("c:\\" + myFile.getFileName()); //创建一个文件输出流  fos.write(myFile.getFileData()); //写文件  fos.flush(); 清空缓冲区, 写到磁盘上  fos.close();  } |
| struts-cfg.xml中配置上传文件的: 大小门限  <controller  bufferSize="指定上载文件的输入缓冲的大小, 默认值为4096"  className="指定和controller元素对应的匹配类, 默认值为 org.apache.struts.config.ControllerConfig"  contentType="指定相应结果的内容类型和字符编码, 默认值为text/html, 如果在Action和JSP网页中也设置了内容类型和字符编码, 将会覆盖该设置"  locale="默认false, 指定是否把Locale对象保存到当前用户的Session中."  processorClass="指定负责处理请求的java类的完整类名, 默认值为org.apache.struts.action.RequestProcessor, 如果把此项设置成自定义类, 那么这个自定义类必须继承org.apache.struts.action.RequestProcessor"  maxFileSize="//文件默认大小为250M, 可以配置单位: K, M, G"  nocache="默认为false, 如果设置成true, 则在相应结果总将加入特定的头参数, Pragma, Cache-Control和Expires, 防止页面被存储中客户浏览器的缓存中. 每次都清理浏览器缓存"  tempDir="上传文件的临时目录 "  /> |

## 类型转换器

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 默认类型转换, 在ActionServlet中注册:   |  |  | | --- | --- | | java.sql.Date sqlDate | jsp表单中格式必须是: YYYY-MM-DD | | java.util.Date utilDate | 默认情况下struts无法转换 | | boolean | yes, y, 1, on, true 不区分大小写都会转换成True类型，而且忽略大小写，其他情况转换成false | | BigDecimal |  | | BigInteger |  | | Byte |  | | Character |  | | Double |  | | Float |  | | Integer |  | | Long |  | | Short |  |   自定义类型转换:  \* 自定义转换器的实现步骤  第一步: 实现converter接口，重写convert方法  第二步: 将实现的conerter注册，通常情况采用servlet注册需要注意标签的配置,<load-on-startup>10</load-on-startup>(也可以采用struts plugin注册)   |  | | --- | | **自定义转换器类:**  **import** java.text.ParseException;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  **import** org.apache.commons.beanutils.Converter;  /\*\*  \* java.util.Date类型转换器  \* **@author** Administrator  \*  \*/  **public** **class** UtilDateConverter **implements** Converter {  //转换代码写在convert里面.  **public** Object convert(Class type, Object value) { //value: jsp传入的值, type: 传入目标类型, 返回转换后的对象.  System.out.println("UtilDateConverter.value=" + value);  **if** (value == **null**) {  **return** value;  }  **if** (value **instanceof** Date) {  **return** value;  }  Date d = **null**;  **if** (value **instanceof** String) {  SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd");  **try** {  d = sdf.parse((String)value);  } **catch** (ParseException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  **return** d;  }  } | | **注册方法(1)**(不推介): 自己写一个类MyActionServlet, 继承ActionServlet, 覆写initOther(),  用supper. initOther()执行一次, 然后注册: ConvertUtils.register(**new** UtilDateConverter(), Date.**class**);  如此以来, web.xml中就不能配置ActionServlet了, 要配置MyActionServlet这个类 | | **注册方法(2.1)**(常用): 采用一个Servlet, 配置成TomCat启动时执行.  **import** java.util.Date;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** org.apache.commons.beanutils.ConvertUtils;  /\*\*  \* 注册java.util.date转换器  \* **@author** Administrator  \*  \*/  **public** **class** UtilDateConverterInitWithServlet **extends** HttpServlet {  @Override  **public** **void** init() **throws** ServletException {  System.out.println("UtilDateConverterInitWithServlet.init()");  ConvertUtils.register(**new** UtilDateConverter(), Date.**class**);  }  }  **web.xml中配置:**  <servlet>  <servlet-name>UtilDateConverterInitWithServlet</servlet-name>  <servlet-class>com.bjsxt.struts.UtilDateConverterInitWithServlet</servlet-class>  <load-on-startup>3</load-on-startup> <!-- 只要大于0就行了 -->  </servlet> | | **注册方法(2.2)**(常用): struts-config.xml中配置插件的形式:  **在插件初始化方法中注册转换器.**  **import** java.util.Date;  **import** javax.servlet.ServletException;  **import** org.apache.commons.beanutils.ConvertUtils;  **import** org.apache.struts.action.ActionServlet;  **import** org.apache.struts.action.PlugIn;  **import** org.apache.struts.config.ModuleConfig;  **public** **class** UtilDateConverterInitWithPlugin **implements** PlugIn {  **public** **void** destroy() {  }  **public** **void** init(ActionServlet servlet, ModuleConfig config)  **throws** ServletException {  System.out.println("UtilDateConverterInitWithPlugin.init()");  ConvertUtils.register(**new** UtilDateConverter(), Date.**class**);  }  }  **struts-config.xml中配置插件的形式:**  <plug-in className="com.bjsxt.struts.UtilDateConverterInitWithPlugin"/> | |

## 数组在页面之间的传递

|  |
| --- |
|  |

# 异常

|  |
| --- |
| 声明式异常(自动处理的异常), 捕获Action抛出的异常.  <global-exceptions>  <exception key="user.not.found" type="com.bjsxt.struts.UserNotFoundException" path="/login\_error.jsp"/>  <exception key="user.password.error" type="com.bjsxt.struts.PasswordErrorException" path="/login\_error.jsp"/>  </global-exceptions>  如果配置了input 和path, 优先转向path指向的页面. |
| 编程式异常:  截获异常  创建相应的异常消息  传递异常消息  转向相应的页面处理异常 |

# ----------------------未分类---------------------

设计思想:

|  |
| --- |
| ActionForm(数据传输对象DTO)=VO(值对象) |

Struts 1 和 Struts 2 的ActionForm中使用复杂对象（javaBean）的区别：

|  |  |
| --- | --- |
| Struts 1 | 定义javaBean变量时需要new一个对象 |
| Struts 2 | 定义javaBean变量时不需要new一个对象 |

访问路径的三种写法:

|  |  |  |
| --- | --- | --- |
| <a href=”/strutsdemo/upload.do”>超链接</a> | 太差 |  |
| <a href=”${pageContext.request.contextPath}/upload.do”>超链接</a> | 一般 |  |
| <html:link action=”/upload”>超链接</html:link> | 最好 | 自带url重写 |

web变量:

|  |
| --- |
| 放在那4个作用域(page,request,session,application)中的属性. |

日期处理

|  |
| --- |
| import java.text.FieldPosition;  import java.text.SimpleDateFormat;  import java.util.Calendar;  import java.util.Date;  public class UtialDate {  public static void main(String[] args) {  f3();  }  /\*\*  \* 创建Date类型对象  \* 创建Calendar对象  \*/  private static void f0(){    //创建Date对象，默认的日期时间为系统当前时间。  Date currentDate = new Date();    //创建Calendar对象，默认的日期时间为系统当前时间。  Calendar currentCalen = Calendar.getInstance();    //将Calendar对象-->Date对象  Date da = currentCalen.getTime();    //将Date对象-->Calendar对象  Calendar currentD2C = Calendar.getInstance();  currentD2C.setTime(new Date());  }    /\*\*  \* java日期时间对象-->指定格式化字符串输出  \*/  private static void f1(){  String dateString;  SimpleDateFormat sdf;  SimpleDateFormat sdfH;  SimpleDateFormat sdfS;    String dateFormat = "yyyy年MM月dd日";  String dateFormatH = "yyyy年MM月dd日 HH时mm分ss秒";  String dateFormatS = "yyyy年MM月dd日 HH时mm分ss秒 S毫秒";    Date currentDate = new Date();  sdf = new SimpleDateFormat(dateFormat);  dateString = sdf.format(currentDate);  System.out.println(dateString);    sdfH = new SimpleDateFormat(dateFormatH);  dateString = sdfH.format(currentDate);  System.out.println(dateString);    sdfS = new SimpleDateFormat(dateFormatS);  dateString = sdfS.format(currentDate);  System.out.println(dateString);    }    /\*\*  \* 获取日期时间变量中的某个部分的值  \*/  private static void f2(){  Date currentDate;  Calendar calendar;  int year;  int month;  int day;  int hour12;  int hour24;  int min;  int sec;  int msec;    //创建一个Date对象  currentDate = new Date();  //创建一个Calendar对象  calendar = Calendar.getInstance();  //将Date对象转化为Calendar对象  calendar.setTime(currentDate);    year = calendar.get(Calendar.YEAR);  month = calendar.get(Calendar.MONTH)+1;  day = calendar.get(Calendar.DAY\_OF\_MONTH);  hour12 = calendar.get(Calendar.HOUR);  hour24 = calendar.get(Calendar.HOUR\_OF\_DAY);  min = calendar.get(Calendar.MINUTE);  sec = calendar.get(Calendar.SECOND);  msec = calendar.get(Calendar.MILLISECOND);    System.out.println("year=" + year);  System.out.println("month=" + month);  System.out.println("day=" + day);  System.out.println("hour12=" + hour12);  System.out.println("hour24=" + hour24);  System.out.println("min=" + min);  System.out.println("sec=" + sec);  System.out.println("msec=" + msec);  }    /\*\*  \* 日期时间格式的字符串-->日期时间类型的变量  \*/  private static void f3(){    Date date;  String dateString = "2009-01-01 12:33:12";//被转换的字符串  String dateFormat = "yyyy-MM-dd HH:mm:ss";//目标格式    //创建simpleDateFormat对象  SimpleDateFormat sdFormat = new SimpleDateFormat(dateFormat);    try{  date = sdFormat.parse(dateString);  System.out.println(date);  }catch(Exception e){  e.printStackTrace();  }  }    // y-年，四个y表示四位数的年，小写  // M-月，两个M时如果是单数月会补0，一定要大写  // d-日，两个d时如果是单数天会补0，小写  // H-小时，两个H时单数补0，一定要大写  // m-分钟，两个m时单数补0，一定要小写，主要和月份区分开来  // s-秒，两个s时单数补0，小写  // S-毫秒，一定能够要大写    /\*\*  \* 日期之间的 求差的计算  \*  \*/  public static void diffDate(){      Date currentDate;//当前系统日期  Date firstDate; //用于比较的日期  Calendar currentCalendar;  Calendar firstCalendar;    //创建日期对象  currentDate = new Date();  firstDate = getFormatString2Date("2009-05-29", "yyyy-MM-dd");    //将Date对象转化成Calendar对象以便于计算  currentCalendar = Calendar.getInstance();  firstCalendar = Calendar.getInstance();    currentCalendar.setTime(currentDate);  firstCalendar.setTime(firstDate);    long diffmills = currentCalendar.getTimeInMillis() - firstCalendar.getTimeInMillis();  long diffm = diffmills/1000;  long diffs = diffmills/(1000\*60);  long diffh = diffmills/(1000\*60\*60);  long diffd = diffmills/(1000\*60\*60\*24);    System.out.println(diffmills + " 毫秒");  System.out.println(diffm + " 秒");  System.out.println(diffs + " 分钟");  System.out.println(diffh + " 小时");  System.out.println(diffd + " 天");    }          /\*\*  \* 某个日期 加上 某个时间单元 后的日期  \*  \*/  public static void addDate(){    Date currentDate;//当前系统日期  Calendar currentCalendar;    //创建日期对象  currentDate = new Date();  //将Date对象转化成Calendar对象以便于计算  currentCalendar = Calendar.getInstance();    //currentDate加上100毫秒后到日期  currentCalendar.setTime(currentDate);  currentCalendar.add(Calendar.MILLISECOND, 100);  Date addDateMills = currentCalendar.getTime();    //currentDate加上50秒后到日期  currentCalendar.setTime(currentDate);  currentCalendar.add(Calendar.SECOND, 50);  Date addDateS = currentCalendar.getTime();    //currentDate加上20分钟后到日期  currentCalendar.setTime(currentDate);  currentCalendar.add(Calendar.MINUTE, 20);  Date addDateM = currentCalendar.getTime();    //currentDate加上一个小时后到日期  currentCalendar.setTime(currentDate);  currentCalendar.add(Calendar.HOUR\_OF\_DAY, 1);  Date addDateH = currentCalendar.getTime();    //currentDate加上一天后的日期  currentCalendar.setTime(currentDate);  currentCalendar.add(Calendar.DAY\_OF\_YEAR, 1);  Date addDateDay = currentCalendar.getTime();    //currentDate加上一周后的日期  currentCalendar.setTime(currentDate);  currentCalendar.add(Calendar.WEEK\_OF\_YEAR, 1);  Date addDateWeek = currentCalendar.getTime();    //currentDate加上一个月后的日期  currentCalendar.setTime(currentDate);  currentCalendar.add(Calendar.MONTH, 1);  Date addDateMonth = currentCalendar.getTime();      String currentDateString = getDate2FormatString(currentDate,"yyyy-MM-dd HH:mm:ss S");    System.out.println("日期："+currentDateString + " 加上" + " 100 毫秒 后的日期为 -> "+getDate2FormatString(addDateMills,"yyyy-MM-dd HH:mm:ss S"));  System.out.println("日期："+currentDateString + " 加上" + " 50 秒 后的日期为 -> "+getDate2FormatString(addDateS,"yyyy-MM-dd HH:mm:ss S"));  System.out.println("日期："+currentDateString + " 加上" + " 20 分钟 后的日期为 -> "+getDate2FormatString(addDateM,"yyyy-MM-dd HH:mm:ss S"));  System.out.println("日期："+currentDateString + " 加上" + " 1 小时 后的日期为 -> "+getDate2FormatString(addDateH,"yyyy-MM-dd HH:mm:ss S"));  System.out.println("日期："+currentDateString + " 加上" + " 1 天 后的日期为 -> "+getDate2FormatString(addDateDay,"yyyy-MM-dd HH:mm:ss S"));  System.out.println("日期："+currentDateString + " 加上" + " 1 周 后的日期为 -> "+getDate2FormatString(addDateWeek,"yyyy-MM-dd HH:mm:ss S"));  System.out.println("日期："+currentDateString + " 加上" + " 1 月 后的日期为 -> "+getDate2FormatString(addDateMonth,"yyyy-MM-dd HH:mm:ss S"));    }      /\*\*  \* 格式化 日期时间信息  \* @param userDate 日期对象  \* @param formatString 格式haunted字符串  \* @return  \*/  public static String getDate2FormatString(Date userDate, String formatString) {    StringBuffer dateformatstring = new StringBuffer();    // check formatString whether null  if ((null == formatString) && (formatString.length() < 0))  System.out.println("Function getDate2FormatString error : not find date format");  // check userDate whether  if (null == userDate)  return null;  SimpleDateFormat dateFormat = new SimpleDateFormat(formatString);  dateFormat.format(userDate, dateformatstring, new FieldPosition(0));    return dateformatstring.toString();  }          /\*\*  \* 将 日期字符串 转化为 日期对象  \* @param str 日期字符串  \* @param dtfm 格式字符串  \* @return  \*/  public static Date getFormatString2Date(String str, String dtfm) {  return new Date();  }    /\*\*  \* 获取日期时间型对象的属性，比如日期属于所在年份的第几天、日期是星期几等信息  \*  \*/  public static void dateProperty(){    //当前系统日期  Date currentDate;  Calendar currentCalendar;    //创建日期对象  currentDate = new Date();  //将Date对象转化成Calendar对象以便于计算  currentCalendar = Calendar.getInstance();  currentCalendar.setTime(currentDate);    //--------------------------------------------------------------  //取得当前日期是星期几  //  //week的值是一个数字为1-7 其中 1代表星期天，依次类推7代表星期六  //根据西方的习惯，一周时从星期天开始的。  //---------------------------------------------------------------  int week = currentCalendar.get(Calendar.DAY\_OF\_WEEK);  //取得当前日期是当年的第几天  int daycount = currentCalendar.get(Calendar.DAY\_OF\_YEAR);    System.out.println(week);  System.out.println(daycount);    }  } |