Struts回顾：

1. Struts开发流程

(搭建一个strtus开发环境)

2. Struts 执行流程

3. Struts配置

-🡪 package

🡪 拦截器

🡪 全局视图

🡪 action

默认值

4. struts核心功能

a. 数据处理

b. 请求数据封装/类型转换

c. 文件上传与下载

d. 拦截器

e. 国际化

f. Ognl 表达式语言 / struts标签

g.数据效验

5. struts其他细节技术

数据回显、模型驱动、防止表单重复提交

最后，

综合练习！

# 1. Struts数据效验

表单数据的验证：

前台验证：主要是通过JS验证， 表达数据是否合法！

后台验证：通过后台java代码进行验证!

Struts也提供了数据效验的方式！

Struts数据效验， 通过拦截器完成：

<interceptor

name="validation" class="org.apache.struts2.interceptor.validation.AnnotationValidationInterceptor"/>

## 代码方式验证Action中所有的方法：

步骤：

1. 搭建struts环境

2. login.jsp

3.UserAction.java

|  |
| --- |
| Struts数据效验错误信息处理： |
| **public** **synchronized** **void** addFieldError(String fieldName, String errorMessage) {  // 1. 保存错误信息的map集合  **final** Map<String, List<String>> errors = internalGetFieldErrors();  // 2. Map的value值 (一个key，对应的多个值用list存储)  List<String> thisFieldErrors = errors.get(fieldName);  // 3. 如果错误信息的map中没有当前的key，就直接创建集合  **if** (thisFieldErrors == **null**) {  // 创建  thisFieldErrors = **new** ArrayList<String>();  errors.put(fieldName, thisFieldErrors);  }  // 4. 添加错误信息  thisFieldErrors.add(errorMessage);  } |
| Action实现： |
| /\*\*  \* 注意：如果要想用struts的数据效验功能，必须继承ActionSupport或实现相关接口  \* **@author** Jie.Yuan  \*  \*/  **public** **class** UserAction **extends** ActionSupport {    // 封装请求数据  **private** User user = **new** User();  **public** **void** setUser(User user) {  **this**.user = user;  }  **public** User getUser() {  **return** user;  }  // 重写数据验证的方法  @Override  **public** **void** validate() {  // 用户名非空  **if** (user.getUserName() == **null** || "".equals(user.getUserName())) {  // 保存错误信息  **super**.addFieldError("userName", "用户名必须填写！");  }  // 密码  **if** (user.getPwd() == **null** || "".equals(user.getPwd())) {  **super**.addFieldError("pwd", "密码必填");  }  }  …..  } |

## 代码方式验证Action中指定的方法：

写验证方法命名规则：

validate + 要验证的方法名

如：

public void validateRegister() {

只会验证当前action的register方法！

## XML方式验证Action中所有的方法：

总结代码方式验证：

繁琐，设计很多重复的验证逻辑！例如：非空验证、数值验证、email、日期等。

Struts对于常用的验证，进行了封装，即提供了验证器， 验证指定的常用业务逻辑！

**Struts提供的所有验证器：**

|  |
| --- |
| 路径： |
| xwork-core-2.3.4.1.jar/com.opensymphony.xwork2.validator.validators/default.xml |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE validators PUBLIC  "-//Apache Struts//XWork Validator Definition 1.0//EN"  "http://struts.apache.org/dtds/xwork-validator-definition-1.0.dtd">  <!-- START SNIPPET: validators-default -->  <validators>  <validator name=*"required"* class=*"com.opensymphony.xwork2.validator.validators.RequiredFieldValidator"*/>  <validator name=*"requiredstring"* class=*"com.opensymphony.xwork2.validator.validators.RequiredStringValidator"*/>  <validator name=*"int"* class=*"com.opensymphony.xwork2.validator.validators.IntRangeFieldValidator"*/>  <validator name=*"long"* class=*"com.opensymphony.xwork2.validator.validators.LongRangeFieldValidator"*/>  <validator name=*"short"* class=*"com.opensymphony.xwork2.validator.validators.ShortRangeFieldValidator"*/>  <validator name=*"double"* class=*"com.opensymphony.xwork2.validator.validators.DoubleRangeFieldValidator"*/>  <validator name=*"date"* class=*"com.opensymphony.xwork2.validator.validators.DateRangeFieldValidator"*/>  <validator name=*"expression"* class=*"com.opensymphony.xwork2.validator.validators.ExpressionValidator"*/>  <validator name=*"fieldexpression"* class=*"com.opensymphony.xwork2.validator.validators.FieldExpressionValidator"*/>  <validator name=*"email"* class=*"com.opensymphony.xwork2.validator.validators.EmailValidator"*/>  <validator name=*"url"* class=*"com.opensymphony.xwork2.validator.validators.URLValidator"*/>  <validator name=*"visitor"* class=*"com.opensymphony.xwork2.validator.validators.VisitorFieldValidator"*/>  <validator name=*"conversion"* class=*"com.opensymphony.xwork2.validator.validators.ConversionErrorFieldValidator"*/>  <validator name=*"stringlength"* class=*"com.opensymphony.xwork2.validator.validators.StringLengthFieldValidator"*/>  <validator name=*"regex"* class=*"com.opensymphony.xwork2.validator.validators.RegexFieldValidator"*/>  <validator name=*"conditionalvisitor"* class=*"com.opensymphony.xwork2.validator.validators.ConditionalVisitorFieldValidator"*/>  </validators>  <!-- END SNIPPET: validators-default --> |

如果写xml，从而定义验证规则：

1）XML文件名称语法： ActionClassName-validation.xml

注意：此xml需要与当期要验证的action在同一个目录：

举例：UserAction-validation.xml

2） 写XML

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE validators PUBLIC  "-//Apache Struts//XWork Validator 1.0.3//EN"  "http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">  <validators>    <!-- 验证的每一个字段用field表示 -->  <field name=*"user.userName"*>  <!-- 指定使用的验证器 -->  <field-validator type=*"requiredstring"*>  <!-- 验证失败的错误提示信息 -->  <message>用户名不能为空！</message>  </field-validator>  </field>    <!-- 验证pwd -->  <field name=*"user.pwd"*>    <!-- 非空 -->  <field-validator type=*"requiredstring"*>  <message>密码不能为空！</message>  </field-validator>    <!-- 长度 -->  <field-validator type=*"stringlength"*>  <param name=*"minLength"*>6</param>  <param name=*"maxLength"*>8</param>  <message>密码必须为6-8位！</message>  </field-validator>  </field>    <!-- 验证日期 -->  <field name=*"user.birth"*>  <field-validator type=*"date"*>  <message>日期格式不对！</message>  </field-validator>  </field>    <!-- 验证Email -->  <field name=*"user.email"*>  <field-validator type=*"email"*>  <message>邮箱格式错误！</message>  </field-validator>  </field>    </validators> |

## XML方式验证Action中指定的方法:

与上面xml验证方式大致相同： 验证xml文件内容不变；

文件命名：

语法：ActionClassName-ActionName-validation.xml

举例：UserAction-user\_register-validation.xml

验证UserAction中的register方法

## 验证总结

代码：

重写validate() , 验证action所有方法

Validate方法名(), 验证指定“方法名”的方法

Xml：

验证所有方法： ActionClassName-validation.xml

验证指定方法： ActionClassName-actionName-validation.xml

代码验证，

比较灵活，可以满足所有的需求.

比较繁琐，要写重复的验证判断逻辑!

适合： 表单字段较少的情况用！

XML验证：

通用，但不够灵活； 可以验证特定简单的业务。

适合： 验证表单字段较多，可以大大简化代码！

(配置文件过多)

## 验证错误处理

Struts在进行数据效验的时候，验证失败，会返回input视图，要求我们要在struts.xml中配置input视图对应的错误页面！

**配置：**

|  |
| --- |
| Struts.xml |
| <!-- 注册失败跳转到注册页面，显示失败信息 -->  <global-results>  <result name=*"input"*>/register.jsp</result>  </global-results> |

**Jsp显示错误**

* 方式1：显示所有错误

|  |
| --- |
| <%@taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!-- 显示的是struts在运行时期产生的所有错误 -->  <s:fielderror></s:fielderror> |

* 方式2：显示指定的错误

|  |
| --- |
| <!-- 修改struts标签默认的样式： 不让换行 -->  <style type=*"text/css"*>  ul{  display: *inline*;  }  ul li{  display: *inline*;  color: *red*;  }  </style> |
| 显示指定的错误： |
| <s:fielderror fieldName="user.userName"></s:fielderror> |

* 方式3: 修改标签定义的模板

找到fielderror标签定义的模板文件:

Struts-core.jar\template\simple\ fielderror.ftl

把修改后的fielderror.ftl文件，放到src/ template/ simple/ fielderror.ftl

这样标签显示的样式就修改了!

# 2. Struts 简单UI标签

|  |
| --- |
| <!-- 服务器标签 : 最终别解析为html标签-->  <s:form action=*"/user\_login"* method=*"post"* name=*"frmLogin"* id=*"frmLogin"* theme=*"simple"*>    用户名：<s:textfield name=*"user.name"*></s:textfield>  密码：<s:textfield name=*"user.pwd"*></s:textfield>  <s:submit value=*"登陆"*></s:submit>  </s:form> |

注意：

给form指定主题，form下所有的表单元素都应用此主题！

对于struts标签默认的主题样式：default.xml/struts.ui.theme=xhtml

可以通过常量修改, 改为简单主题：

<!-- 修改主题 (当前项目所有的标签都用此主题)-->

<constant

name=*"struts.ui.theme"* value=*"simple"*></constant>

# 3. Struts ognl表达式语言几个符号

# 获取非根元素值 、 动态都建map集合

$ 配置文件取值

% 提供一个ognl表达式运行环境

|  |
| --- |
| <body>  <br/>获取request域数据<br/>  <!-- property 标签是对象类型的标签，默认支持ognl表达式, 会从根元素去China名称对应的值 -->  <s:property value=*"China"*/> <br/>  <!-- 如果直接赋值，需要用单引号 -->  <s:property value=*"'China'"*/> <br/>  <s:property value=*"%{#request.cn}"*/> <br/>    <!-- 值类型的标签，value值默认就是值类型，不支持ognl表达式 -->  国家：<s:textfield name=*"txtCountry"* value=*"%{#request.cn}"*></s:textfield>  </body> |

# 4. Struts中常用的几个技术

## 数据回显

数据回显，必须要用struts标签！

|  |
| --- |
| Action中： |
| // 进入修改页面  **public** String viewUpdate() {  // 模拟一个对象(先获取一个id，再根据id调用service查询，把查到的结果保存到域)  User userInfo = **new** User();  userInfo.setUserName("Jack");  userInfo.setEmail("yuanjie@itcast.cn");    ActionContext ac = ActionContext.*getContext*();  // Map<String,Object> request = (Map<String, Object>) ac.get("request");  // request.put("userInfo", userInfo);    /\*\*\*\*\*\*\*\*\*\*\*\*\* 数据回显\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // 获取值栈  ValueStack vs = ac.getValueStack();  vs.pop();// 移除栈顶元素  vs.push(userInfo); // 入栈      // 进入修改页面  **return** "viewUpdate";  } |
| JSP页面： |
| <body>  <%@taglib uri=*"/struts-tags"* prefix=*"s"* %>      <br/>  <!-- 在页面文本框内，显示要修改记录的数据 -->    <!-- 手动通过value设置显示的值  <s:form action="#">    用户名： <s:textfield name="user.userName" value="%{#request.userInfo.userName}"></s:textfield> <br/>    邮箱： <s:textfield name="user.email" value="%{#request.userInfo.email}"></s:textfield> <br/>  </s:form>  -->    <!-- 数据回显技术：s:textfield会自动查找根元素数据(Ognl表达式语言取值) -->  <s:form action=*"#"*>    用户名： <s:textfield name=*"userName"*></s:textfield> <br/>    邮箱： <s:textfield name=*"email"*></s:textfield> <br/>  </s:form>    <s:debug></s:debug>  </body> |

## 模型驱动

Struts运行时候，会执行默认的拦截器栈，其中有一个拦截器，模型驱动拦截器：

<interceptor

name="modelDriven" class="com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor"/>

|  |
| --- |
| @Override  **public** String intercept(ActionInvocation invocation) **throws** Exception {  Object action = invocation.getAction();  **if** (action **instanceof** ModelDriven) {  ModelDriven modelDriven = (ModelDriven) action;  ValueStack stack = invocation.getStack();  Object model = modelDriven.getModel();  **if** (model != **null**) {  stack.push(model);  }  **if** (refreshModelBeforeResult) {  invocation.addPreResultListener(**new** RefreshModelBeforeResult(modelDriven, model));  }  }  **return** invocation.invoke();  } |

### 引入

|  |
| --- |
| prams拦截器，可以把请求数据自动填充的action的属性中  举例1：  JSP  <input type=text name=userName />  <input type=text name=pwd />  Action  class UserAction{  // 封装请求数据  private String userName;  private String pwd;  //.. set方法  }  举例2：  JSP  <input type=text name=user.userName />  <input type=text name=user.pwd />  Action  class UserAction{  // 封装请求数据  private User user;  ..  }  举例3：(模型驱动)  JSP  <input type=text name=userName />  <input type=text name=pwd />  Action  class UserAction{  // 封装请求数据  private User user;  ..  } |

### 步骤及原理

步骤：

1. 实现ModelDriver接口

2. 实现接口方法： 接口方法返回的就是要封装的对象

3. 对象一定要实例化。

分析：

|  |
| --- |
| /\*\*  \* 1. 数据回显  \* 2. 模型驱动  \* **@author** Jie.Yuan  \*  \*/  **public** **class** UserAction **extends** ActionSupport **implements** ModelDriven<User> {    // 封装请求数据  **private** User user = **new** User();  **public** **void** setUser(User user) {  **this**.user = user;  }  **public** User getUser() {  **return** user;  }    // 实现模型驱动接口方法  @Override  **public** User getModel() {  **return** user;  }      **public** String add() {  // 测试： 使用了模型驱动，是否数据正常？ Ok  System.*out*.println(user);  **return** "success";  } |

## 防止表单重复提交

Struts提供了防止表单重复提交拦截器：

<interceptor name="token"

class="org.apache.struts2.interceptor.TokenInterceptor"/>

综合案例：

1. 建库、建表

|  |
| --- |
| -- 删除数据库  DROP DATABASE hib\_demo;  -- 创建数据库  CREATE DATABASE hib\_demo DEFAULT CHARACTER SET utf8;  -- 建表  CREATE TABLE employee (  id INT PRIMARY KEY AUTO\_INCREMENT,  empName VARCHAR(20),  workDate DATE -- 入职时间  ) |

2. 搭建环境

搭建struts环境

组件：c3p0/dbutils/驱动