**属性驱动接收请求参数和复杂数据封装**

1. **属性驱动接收请求参数：**（即指利用action接收表单提交过来的参数）

通过实现**ModelDriven**接口，进行表单提交过来的属性的自动设置。

范例：

**登录界面（由该界面进行参数的登录提交，提交名字和密码）：**

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>在此处插入标题</title>

</head>

<body>

<h3>方式一：Action作为表单模型，提供属性和属性对应的getter和setter方法</h3>

<form action = *"*${pageContext.request.contextPath }*/login3.action"* method = *"post"*>

用户名：<input type = *"text"* name = *"name"*></input><br></br> <!-- 这个name必须要跟UserInfo里面的name一致，

因为是由反射掉用的，如果不一致，那么就会找不到属性 -->

密码：<input type = *"password"* name = *"pwd"*></input><br></br>

<input type = *"submit"* value = *"Login"*></input>

</form>

</body>

</html>

**接收参数的action类（接收表单提交过来的参数，并自动为UserInfo类的属性赋值）：**

**package** wp.struts2\_01;

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

**import** com.opensymphony.xwork2.ModelDriven;

**public** **class** LoginAction **extends** ActionSupport **implements** ModelDriven<UserInfo>{

//必须手动初始化Javabean对象

**private** UserInfo userInfo = **new** UserInfo() ;

**public** UserInfo getUserInfo(){

**return** **this**.userInfo ;

}

**public** **void** setUserinfo(UserInfo userinfo) {

**this**.userInfo = userinfo;

}

@Override

**public** String execute() **throws** Exception {

System.***out***.println(userInfo.getName() + "---->" + userInfo.getPwd()) ;

System.***out***.println("userName:" + userInfo.getName()) ;

**return** ***NONE*** ;

}

/\*\*

\* 把页面中的数据设置到Javabean中

\* \*/

@Override

**public** UserInfo getModel() { //对调用getter和setter方法userinfo自动赋值

System.***out***.println( "用户信息对象：" + userInfo) ;

//System.out.println("userName:" + userInfo.getName()) ;

**return** userInfo ;

}

}

**UserInfo类：**

**package** wp.struts2\_01;

**public** **class** UserInfo {

**private** String name ; //此属性名称一定要跟提交过来的表单中的文本框属性name一致不然就不能找到文本框中的属性进行赋值操作。

**private** String pwd ;

**public** UserInfo() {

System.***out***.println("userInfo构造方法！");

}

**public** String getName() {

System.***out***.println("userInfo的getName()");

**return** **this**.name;

}

**public** **void** setName(String name) {

System.***out***.println("userInfo的setName()");

**this**.name = name;

}

**public** String getPwd() {

System.***out***.println("userInfo的getpwd()");

**return** **this**.pwd;

}

**public** **void** setPwd(String pwd) {

System.***out***.println("userInfo的setPwd()");

**this**.pwd = pwd;

}

}

**以上程序实现了对表单提交过来的文本框内容属性进行接收，并且将接收的属性利用ModelDriven<UserInfo>接口的public UserInfo getModel()方法自动为userInfo对象属性赋值。**

**先调用getModel()方法再调用execute()方法。**

**复杂数据封装：**

1. **封装到list集合：**

将product1.jsp表单提交的list集合数据，提交到productAction后进行输出

范例：

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>对list的封装</title>

</head>

<body>

<h3>方式一：Action作为表单模型，提供属性和属性对应的getter和setter方法</h3>

<form action = *"*${pageContext.request.contextPath }*/product1Action.action"* method = *"post"*>

商品名称：<input type = *"text"* name = *"list[0].name"*></input><br></br>

商品价格：<input type = *"text"* name = *"list[0].price"*></input><br></br>

商品名称：<input type = *"text"* name = *"list[1].name"*></input><br></br>

商品价格：<input type = *"text"* name = *"list[1].price"*></input><br></br>

商品名称：<input type = *"text"* name = *"list[2].name"*></input><br></br>

商品价格：<input type = *"text"* name = *"list[2].price"*></input><br></br>

<input type = *"submit"* value = *"批量导入"*></input>

</form>

</body>

</html>

**package** wp.struts2\_02;

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

**import** java.util.List; ;

**public** **class** ProductAction1 **extends** ActionSupport{

**private** List<Product> list ;

**public** List<Product> getList() {

System.***out***.println("productAction中的getList()");

**return** list;

}

**public** **void** setList(List<Product> list) {

System.***out***.println("productAction中的setList()");

**this**.list = list;

}

@Override

**public** String execute() **throws** Exception {

**for**(Product product : list){

System.***out***.println(product);

}

**return** ***NONE***;

}

}

**package** wp.struts2\_02;

**public** **class** Product {

**private** String name ;

**private** Double price ;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Double getPrice() {

**return** price;

}

**public** **void** setPrice(Double price) {

**this**.price = price;

}

@Override

**public** String toString() {

**return** "Product [name=" + name + ", price=" + price + "]";

}

}

1. **封装到map：**

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>对map的封装</title>

</head>

<body>

<h3>封装到map集合</h3>

<form action = *"*${pageContext.request.contextPath }*/product2*

*Action2.action"* method = *"post"*>

商品名称：<input type = *"text"* name = *"map['one'].name"*></input><br></br>

商品价格：<input type = *"text"* name = *"map['one'].price"*></input><br></br>

商品名称：<input type = *"text"* name = *"map['two'].name"*></input><br></br>

商品价格：<input type = *"text"* name = *"map['two'].price"*></input><br></br>

商品名称：<input type = *"text"* name = *"map['three'].name"*></input><br></br>

商品价格：<input type = *"text"* name = *"map['three'].price"*></input><br></br>

<input type = *"submit"* value = *"批量导入"*></input>

</form>

</body>

</html>

**package** wp.struts2\_02;

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

**import** java.util.List;

**import** java.util.Map; ;

**public** **class** ProductAction2 **extends** ActionSupport{

**private** Map<String, Product> map ;

**public** Map<String, Product> getMap() {

**return** map;

}

**public** **void** setMap(Map<String, Product> map) {

**this**.map = map;

}

@Override

**public** String execute() **throws** Exception {

**for**(String key : map.keySet()){

Product product = map.get(key) ;

System.***out***.println(product);

}

**return** ***NONE***;

}

}

在以上程序中，与list不同的是：

商品名称：<input type = *"text"* name = *"map['one'].name"*></input><br></br>

商品价格：<input type = *"text"* name = *"map['one'].price"*></input><br></br>

**Name = “map[‘one’].name”而不是name = “list[0].name”**

**请求参数的类型转换机制：**

在struts2之中，struts框架会自动实现类型的转换。比如：页面提交的数据会根据action类之中的属性类型进行自动转换。

范例：请求参数的自动转换

在regist.jsp页面中提交上来的用户名，年龄，爱好等不同类型的数据到RegistAction中进行自动转换为对应的数据类型。、

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>用户注册测试</title>

</head>

<body>

<h2>用户注册测试---内部类型转换测试</h2>

<form action=*"*${pageContext.request.contextPath}*/log"* method = *"post"*>

username:<input type = *"text"* name = *"username"*></input><br></br>

age:<input type = *"text"* name = *"age"*/><br></br>

<h3>日期格式默认使用yyyy-MM-dd</h3>

birthday:<input type = *"text"* name = *"birthday"* /><br></br>

<input type = *"checkbox"* name = *"hobby"* value = *"音乐"*></input>

<input type = *"checkbox"* name = *"hobby"* value = *"跑步"*/>

<input type = *"checkbox"* name = *"hobby"* value = *"跳舞"*/>

<input type = *"checkbox"* name = *"hobby"* value = *"篮球"*/>

<input type = *"submit"* value = *"提交"*/>

</form>

</body>

</html>

**package** wp.struts2\_03;

**import** java.util.Arrays;

**import** java.util.Date;

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

**import** sun.management.resources.agent;

**public** **class** RegistAction **extends** ActionSupport{

**private** String username ;

**private** **int** age ;

**private** Date birthday ;

**private** String[] hobby ;

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** Date getBirthday() {

**return** birthday;

}

**public** **void** setBirthday(Date birthday) {

**this**.birthday = birthday;

}

**public** String[] getHobby() {

**return** hobby;

}

**public** **void** setHobby(String[] hobby) {

**this**.hobby = hobby;

}

@Override

**public** String toString() {

**return** "RegistAction [username=" + username + ", age=" + age + ", birthday=" + birthday + ", hobby="

+ Arrays.toString(hobby) + "]";

}

@Override

**public** String execute() **throws** Exception {

System.out.println("这是RegistAction的execute方法");

System.out.println(**this**.username);

System.out.println(**this**.age);

**this**.toString() ;

**return** **super**.execute();

}

}

**Result结果集的使用：**

**结果集：可以把它看成是响应，通过配置方式实现页面路径跳转。**

1. 局部结果集

局部结果是指一个action对应的一个结果集。

范例：

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>局部结果集</title>

</head>

<body>

<h1>这是局部结果集跳转到的页面local\_result.jsp</h1>

</body>

</html>

**package** wp.struts2\_03;

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

**public** **class** LocalResult **extends** ActionSupport{

@Override

**public** String execute() **throws** Exception {

System.***out***.println("local\_result");

**return** ***SUCCESS***;

}

}

**2.全局结果集**

全局结果指的是多个action都可以跳转到某一个结果集中。

范例：全局结果集

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>全局结果集</title>

</head>

<body>

<h1>这是全局结果集跳转到的页面globle\_result.jsp</h1>

</body>

</html>

**package** wp.struts2\_03;

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

**public** **class** GlobalResult **extends** ActionSupport{

@Override

**public** String execute() **throws** Exception {

System.***out***.println("globel\_result");

**return** ***SUCCESS***;

}

}

以上两个程序中，第一个局部结果集是一个action对应了一个跳转页面，第二个程序中是可以多个action对应一个跳转页面，当执行action后就执行跳转。当action节点中不配置result节点时会直接去找全局结果集进行跳转。在<result name = “success”></result>中，name节点可以不用配置，默认就是success。

**注意：同名的局部结果集会覆盖全局结果集。先走局部，如果局部不存在，再去走全局，如果全局也不存在，则会报错。**

**结果集类型：**

**<result>...jsp</result>:默认是进行的服务器端的跳转。**

**Chain：从一个action跳转到另一个action<result type = “chain”></result>**

**Dispacter:从一个action跳转到一个jsp页面。**

**Redirect:重定向跳转。客户端跳转。由action---->jsp页面。**

**redirectAction:重定向另一个action由action------>action**

范例：使用chain跳转的配置文件

<!-- 配置结果集类型 -->

<action name=*"r1"* class = *"wp.struts2\_04.ResultType\_01"*>

<!-- 默认就是服务器端跳转，所以写不写dispatcher都一样 -->

<result **type = *"chain"***>r2</result>

</action>

<!-- 使用chain跳转由r1跳转到r2 -->

<action name = *"r2"* class = *""*>

<result name = *"success"*>/Conversion/result\_type2.jsp</result>

</action>

以上就是由r1.action跳转到r2.action。然后跳转到result\_type2.jsp页面。

范例：使用redirect跳转

<!-- 使用redirect跳转 -->

<action name = *"r3"* class = *""*>

<result name = *"success"* type = ***"redirect"***>/Conversion/result\_type2.jsp</result>

</action>

范例：使用redirectAction跳转

<!-- 使用redirectAction重定向到action -->

<action name = *"r4"* class = *""*>

<result type = ***"redirectAction"*** name = *"success"*>r5</result>

</action>

<action name = *"r5"*>

<result name = *"success"*>/Conversion/result\_type1.jsp</result>

</action>

**Struts2的文件下载：**

**Struts2的参数校验方式：**

1. **手动全局校验参数（对action中的所有方法进行校验）**

一个登录页面（.jsp页面）中传递过来登录信息。对登录信息进行手动校验。若登录名或则登录密码为空则通过<result name = “input”>节点中配置的name = “input”进行跳转到提示错误信息页面。

范例：

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

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

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>手动校验参数</title>

</head>

<body>

<h3>登录，请求数据校验---代码手动校验</h3>

<!-- <s:fielderror/>是一个struts标签，用于显示错误信息的标签 -->

<s:fielderror/>

<form action = *"*${pageContext.request.contextPath }*/user\_login.action"*>

用户名：<input type = *"text"* name = *"username"*></input><br></br>

密码：<input type = *"password"* name = *"pwd"*></input><br></br>

<input type = *"submit"* value = *"提交"*></input>

</form>

</body>

</html>

**package** wp.demo\_01;

**import** org.apache.commons.lang3.StringUtils;

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

**public** **class** LoginAction **extends** ActionSupport{

**private** String username ;

**private** String pwd ;

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPwd() {

**return** pwd;

}

**public** **void** setPwd(String pwd) {

**this**.pwd = pwd;

}

//校验 ActionSupport提供了一个校验功能

@Override

**public** **void** validate() { **//此方法自动执行**

//编写校验的逻辑代码，程序运行时会自动调用validate方法进行校验

**if**(**StringUtils.*isBlank*(username)**){**//用户名手动校验**

//添加提示信息

//addFieldError会把错误信息放入一个集合，在页面上通过<s:fielderror>标签获取集合中的内容

//在页面上显示错误信息

addFieldError("usernameError", "用户名不能为空......");

}

**if**(**StringUtils.*isBlank*(pwd)**){ **//密码手动校验**

addFieldError("passwordError", "用户密码不能为空");

}

}

**public** String login() **throws** Exception {//方法名称由execute改变为login

System.***out***.println(username + "----->" + pwd);

**return** ***NONE***;

}

/\*\*

\* **input:校验出错后会自动跳转到input对应的视图。在struts.xml中配置input**

\* \*/

}

配置文件：

<action name = *"user\_\*"* class = *"wp.demo\_01.LoginAction"* method = *"{1}"*>

**<!-- name="input"表示校验出错时自动跳转 -->**

<result name = *"input"*>/demo\_01/login.jsp</result>

</action>

在以上程序中，通过login.jsp页面提交用户名和用户登录密码到LoginActin中。**通过自动调用validate()方法进行对用户名和密码是否为空进行手动校验，**如果登录用户名为空则通过addFieldError("usernameError", "用户名不能为空......");将错误信息添加到一个集合中，在验证失败的跳转页面中通过**<s:fielderror>标签**进行错误信息的显示。

在配置文件中：因为将LoginAction.java中的execute()方法名称改成了Login()方法。所以配置文件中使用了通配符的方式**name = “user\_\*” method = “{1}”。在result节点中name = “input”是配置当校验失败时要跳转的页面信息。**

**2.手动局部校验参数：（校验action中指定业务方法---校验一个方法。只会对指定的方法进行校验）**

范例：对login方法进行校验

**格式：validateXxx();表示要校验Xxx方法**

**package** wp.demo\_01;

**import** org.apache.commons.lang3.StringUtils;

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

**public** **class** LoginAction2 **extends** ActionSupport{ //局部校验

**private** String username ;

**private** String pwd ;

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPwd() {

**return** pwd;

}

**public** **void** setPwd(String pwd) {

**this**.pwd = pwd;

}

//校验 ActionSupport提供了一个校验功能

//只校验login方法

**public** **void** validateLogin() { //局部校验。即只对指定的方法进行校验

//编写校验的逻辑代码，程序运行时会自动调用validate方法进行校验

**if**(StringUtils.*isBlank*(username)){

//添加提示信息

//addFieldError会把错误信息放入一个集合，在页面上通过<s:fielderror>标签获取集合中的内容

//在页面上显示错误信息

addFieldError("usernameError", "用户名不能为空......");

}

**if**(StringUtils.*isBlank*(pwd)){

addFieldError("passwordError", "用户密码不能为空");

}

}

**public** String find() **throws** Exception {

System.***out***.println(username + "----->" + pwd);

**return** ***NONE***;

}

**public** String login() **throws** Exception {

System.***out***.println(username);

**return** ***NONE***;

}

/\*\*

\* input:校验出错后会自动跳转到input对应的视图。在struts.xml中配置input

\* \*/

}

在以上程序之中，当在地址栏输入user\_find()方法时，不会对find方法进行校验。如果在地址栏输入user\_login表示只对login()方法进行校验。

***一般都使用局部校验，很少使用全局校验。这种手动校验不适用与表单元素非常多的情况。造成代码量很大，且不容易维护（代码耦合性太强）。在企业中一般推荐使用xml文件来配置表单校验。执行xml配置校验的要求：Action必须继承ActionSupport类（为了实现validateable接口）***

**XML配置规则校验：**

1. 全局校验（对action中的所有方法生效）

**范例：校验用户名不能为空**

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

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

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>xml校验参数</title>

</head>

<body>

<h3>登录，请求数据校验---代码XML全局校验</h3>

<!-- <s:fielderror/>是一个struts标签，用于显示错误信息的标签 -->

<s:fielderror/>

<form action = *"*${pageContext.request.contextPath }*/mylogin.action"*>

用户名：<input type = *"text"* name = *"username"*></input><br></br>

密码：<input type = *"password"* name = *"pwd"*></input><br></br>

<input type = *"submit"* value = *"提交"*></input>

</form>

</body>

</html>

**package** wp.demo\_02;

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

**public** **class** MyLoginAction **extends** ActionSupport{

**private** String username ;

**private** String pwd ;

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPwd() {

**return** pwd;

}

**public** **void** setPwd(String pwd) {

**this**.pwd = pwd;

}

**public** String execute() **throws** Exception {

System.***out***.println(username + "---->" + pwd);

**return** ***SUCCESS*** ;

};

}

Struts.xml配置文件：

<!-- 配置myLOGIN.action -->

<action name = *"mylogin"* class = *"wp.demo\_02.MyLoginAction"*>

<result>/demo\_02/success.jsp</result>

<!-- 验证出错跳转视图 -->

<result name = *"****input****"*>/demo\_02/login.jsp</result>

</action>

**在以上配置文件中：<result>/demo\_02/success.jsp</result> 表示当校验成功时跳转到的页面。<result name = *"input"*>/demo\_02/login.jsp</result> 表示当校验失败时跳转的页面。**

MyLoginAction-validation.xml配置文件：

<?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>

<!-- 表单中要验证的input框name属性 -->

<field name=*"username"*>

<!-- 验证类型 -->

<field-validator type=*"requiredstring"*>

<!-- 显示校验出错信息 -->

<message>用户名不能为空！</message>

</field-validator>

</field>

<field name=*"pwd"*>

<field-validator type=*"requiredstring"*>

<message>密码不能为空！</message>

</field-validator>

</field>

</validators>

**在以上配置文件中：**

<!DOCTYPE validators PUBLIC

"-//Apache Struts//XWork Validator 1.0.3//EN"

"**http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd**">

**配置信息是在Library-->web应用程序--->xwork-validator-1.0.3.dtd中找到的。**

<field name=*"username"*>**表示表单中input框name属性**

<field-validator type=*"requiredstring"*>**表示验证类型。为必填字符串.**

**要想让该配置文件有自动提示：在联网情况下会有自动提示，在不联网下就没有。但是如果想要在不联网的情况下进行自动提示就需要进行配置。**

**配置步骤：**

**窗口--->首选项--->在收索框内输入XML Catalog--->找到XML Catalog--->选择用户指定的条目--->添加--->(在位置：点击文件系统找到：D:\eclipse\struts-2.3.34\src\xwork-core\src\main\resources--->key type选择URI--->键（K）填入http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd)。然后点击完成。**

<message>密码不能为空！</message>**表示验证出错时的提示信息**

注意：**该配置文件的命名：action的名称-validation.xml 比如：MyLoginAcrtion-validation.xml.且该配置文件必须与action放在同一个类的目录下面。**

**在一个包的一个类下面新建一个MyLoginAcrtion-validation.xml文件步骤：**

**右键该类-->新建-->其他-->XML-->XML文件名--->完成**

验证类型：

在Libraries下的web应用程序\xwork-core-2.3.34.jar下的com.opensymphony.xwork2.validator.validators中的default.xml文件中找到

<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>

***Requiredstring：必填字符串校验器。***

***Stringlength：要验证的字符串长度范围。***

2.局部校验（对action中的某个方法生效）

**配置文件命名方式：类名-Action的访问路径-validation.xml eg:MyRegistAction-myAction-validation.xml**

范例：

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

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

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>XML校验参数</title>

</head>

<body>

<h3>注册，请求数据校验---代码XML局部校验</h3>

<!-- <s:fielderror/>是一个struts标签，用于显示错误信息的标签 -->

<s:fielderror/>

<form action = *"*${pageContext.request.contextPath }*/myregist.action"*>

用户名：<input type = *"text"* name = *"username"*>（非空，且长度为3-10位）</input><br></br>

密码：<input type = *"password"* name = *"pwd"*>（必须，且长度为6-12位）</input><br></br>

确认密码：<input type = *"password"* name = *"repwd"*>(必须和密码一致)</input> <br></br>

年龄：<input type = *"text"* name = *"age"*>(年龄在18-80之间)</input> <br></br>

电话：<input type = *"text"* name = *"phone"*>(手机号规则，必须是11位)</input> <br></br>

邮箱：<input type = *"text"* name = *"email"*>(邮箱格式)</input> <br></br>

<input type = *"submit"* value = *"注册"*></input>

</form>

</body>

</html>

**package** wp.demo\_02;

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

**public** **class** MyRegistAction **extends** ActionSupport{

**private** String username ;

**private** String pwd ;

**private** String repwd ;

**private** **int** age ;

**private** String phone ;

**private** String email ;

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPwd() {

**return** pwd;

}

**public** **void** setPwd(String pwd) {

**this**.pwd = pwd;

}

**public** String getRepwd() {

**return** repwd;

}

**public** **void** setRepwd(String repwd) {

**this**.repwd = repwd;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

@Override

**public** String toString() {

**return** "MyRegistAction [username=" + username + ", pwd=" + pwd + ", repwd=" + repwd + ", age=" + age

+ ", phone=" + phone + ", email=" + email + "]";

}

**public** String regist() **throws** Exception {

System.***out***.println(toString());

**return** ***NONE***;

}

}

**Struts.xml配置文件：**

<!-- 注册action，局部校验 -->

<action name = *"myregist"* class = *"wp.demo\_02.MyRegistAction"* method = *"regist"*>

<result name = *"input"*>/demo\_02/myregist.jsp</result>

</action>

**在该配置文件中，method = “regist”是因为在MyRegistAction.java类中将execute()方法名称改变成了regist。Name = “input”表示验证出错时跳转的界面。**

**MyRegistAction-myregist-validation.xml配置文件**：

<?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>

<!-- 用户名：<input type = "text" name = "username">（非空，且长度为3-10位）</input><br></br> -->

<field name=*"username"*>

<field-validator type=*"requiredstring"*>

<message>用户名不能为空</message>

</field-validator>

<field-validator type=*"stringlength"*>

<param name=*"minlength"*>3</param>

<param name=*"maxlength"*>10</param>

<message>密码长度为3-10位</message>

</field-validator>

</field>

<!-- 密码：<input type = "password" name = "pwd">（必须，且长度为6-12位）</input><br></br> -->

<field name=*"pwd"*>

<field-validator type=*"requriedstring"*>

<message>密码不能为空！</message>

</field-validator>

<field-validator type=*"stringlength"*>

<param name=*"minlength"*>6</param>

<param name=*"maxlength"*>12</param>

<message>密码的长度是6-12位</message>

</field-validator>

</field>

<!-- 确认密码：<input type = "password" name = "repwd">(必须和密码一致)</input> <br></br> -->

<field name=*"pwd"*>

<field-validator type=*"fieldexpression"*>

<param name=*"expression"*><![CDATA[pwd==repwd]]></param><!-- 用于屏蔽特殊字符 -->

<message>两次输入的密码必须一致</message>

</field-validator>

</field>

<!-- 年龄：<input type = "text" name = "age">(年龄在18-80之间)</input> <br></br> -->

<field name=*"age"*>

<field-validator type=*"int"*>

<param name = *"min"*>18</param>

<param name = *"max"*>80</param>

<message>年龄必须在18-80之间</message>

</field-validator>

</field>

<!-- 电话：<input type = "text" name = "phone">(手机号规则，必须是11位)</input> <br></br> -->

<field name=*"phone"*>

<field-validator type=*"regex"*>

<param name=*"regex"*><![CDATA[^\d{11}$]]></param>

<message>手机号必须11位</message>

</field-validator>

</field>

<!-- 邮箱：<input type = "text" name = "email">(邮箱格式)</input> <br></br> -->

<field name=*"email"*>

<field-validator type=*"email"*>

<message>请填写正确的邮箱格式</message>

</field-validator>

</field>

</validators>

**在以上配置文件中，一共用了requiredstring，email，regex,int,fieldexpression.等几种验证类型。**

提示：

**被校验的方法首字母要大写。**

**全局校验一定会执行，如果出现局部校验，先执行局部校验，再执行全局校验。**