

目录

资源	2
Struts2Resource	2
基础	2
Struts2 运行环境和安装说明 Struts2Install	2
实例	4
创建 Struts2 工程	4
1.到 Struts2 官方网站下载包	4
2.找到 struts-2.1.8.1\apps\struts2-blank-2.1.8.1.war 解压	4
3.在 web.xml 中添加 Struts2 的过滤器(2.0 和 2.1 的过滤器不同)	4
4.拷贝基础包到工程	4
5.拷贝 struts.xml(struts2-blank-2.1.8.1\WEB-INF\src\java)到 src 目录下	5
6.写 Action 类(可以不继承任何类,也可以继承 ActionSupport)	5
7.写 HelloWorld.jsp 页面,访问 message 参数	6
8.访问 HelloAction,带 message 参数	6
Struts2 在 Action 中访问 request,session,application	6
只需用到 attribute	6
获得 HttpServletRequest	6
Struts2Action 中方法的访问	7
默认方法	7
访问 Action 中多个方法	8
Struts2 返回类型	8
Struts2_OGNL 表达式	11
Struts2 的 S 标签	12
Struts2 表单标签	15
Struts2 文件上传下载	16
文件上传	16
文件下载	17
Struts2 返回 JSON	19
Struts2 返回 XML	20
使用 response	20
Struts2 增删改查例子	20
Struts2 验证框架	24
1.创建 xml 文件名	25
2.创建 xml 内容	25
3.在 action 中验证	31
4.自定义验证类	32
Struts2 国际化	34
1.定义 properties 文件	34
2.在 jsp 中访问国际化资源文件	35
3.在 Action 中访问国际化资源文件,该 Action 继承了 ActionSupport 类	36

4.在输入验证框架访问国际化资源文件.....	36
5.动态改变语言	36
6.使用类来代替 properties 文件.....	37

资源

Struts2Resource

Struts2 资源

网站

Apache Struts 2 <http://struts.apache.org/2.x/index.html>

Opensymphony XWork <http://www.opensymphony.com/xwork>

Opensymphony WebWork <http://www.opensymphony.com/webwork>

基础

Struts2 运行环境和安装说明 Struts2Install

最低需要: Java5、Servlet2.4、JSP2.0、Tomcat5.5

支持: Java6、Servlet2.5、JSP2.0、Tomcat6.0 以及其他容器

源代码: <http://struts.apache.org/download.html>

struts-VERSION-all.zip 是完全版

struts-VERSION-lib.zip 是必要库版

包中 struts2-core-VERSION 是核心库

其他必备的库文件:

Apache Jakarta Common 项目:

ognl-VERSION.jar OGNI 引擎 freemarker-VERSION.jar FREEMARKER 模板引擎
xwork-VERSION.jar

其他插件:

struts2-xxx-plugin-VERSION.jar

分版本 2.1 2.0

Struts2AndStruts1 *Struts2 和 Struts1 对比*

Struts 2 = WebWork 2.2

对比:

- 1.配置文件: **Struts1** 配置文件放在 WEB-INF/struts-config.xml(可定制)目录下; **Struts2** 的配置文件要放在 WEB-INF/classes 目录下
- 2.控制器: **Struts1** 的控制器是一个 **ActionServlet** 类; **Struts2** 的控制器是一个过滤器。
- 3.动作表单: **Struts1** 的 HTML 表单对应一个 **ActionForm** 类的实例, 动作类可访问对应配置的 **ActionForm**, 操作 **ActionForm** 进行填充数据传输对象; **Struts2** 的 HTML 表单直接映射成 **POJO**, 动作类中可直接访问 **POJO**, 操作对 **POJO** 的验证。
- 4.动作类: **Struts1** 的动作类继承 **org.apache.struts.action.Action** 类; **Struts2** 的动作类可以是任何一个 **POJO**, 但是最好是继承 **ActionSupport** 类?。
- 5.显示对象: 在 JSP 中 **Struts2** 使用 **OGNL** 来显示各种对象模型, JSP 自带的 **JSTL** 和 **EL** 中 **EL** 常用来补充使用。
- 6.标签库: **Struts1** 常用 HTML 标签库、Bean 标签库和 Logic 标签库; **Struts2** 有通用标签库、表单标签库?。

为什么最好是继承 **ActionSupport** 类? **Struts2** 的标签具体有哪些?

ActionSupport 类的作用:

struts2 不要求我们自己设计的 *action* 类继承任何的 *struts* 基类或 *struts* 接口, 但是我们为了方便实现我们自己的 *action*, 大多数情况下都会继承 **com.opensymphony.xwork2.ActionSupport** 类, 并重写此类里的 **public String execute() throws Exception** 方法。因为此类中实现了很多的实用借口, 提供了很多默认方法, 这些默认方法包括国际化信息的方法、默认的处理用户请求的方法等, 这样可以大大的简化 *Action* 的开发。

Struts2 中通常直接使用 *Action* 来封装 *HTTP* 请求参数, 因此, *Action* 类里还应该包含与请求参数对应的属性, 并且为属性提供对应的 *getter* 和 *setter* 方法。

Struts2 的标签具体有哪些:

详见 <http://struts.apache.org/2.2.1.1/docs/tag-developers-guide.html>

实例

创建 Struts2 工程

Struts2Hello

1.到 [Struts2 官方网站](#)下载包

2.找到 struts-2.1.8.1\apps\struts2-blank-2.1.8.1.war 解压

3.在 web.xml 中添加 Struts2 的过滤器(2.0 和 2.1 的过滤器不同)

```
<!-- Struts2.1 -->
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- Struts2.0 -->
<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>
```

4.拷贝基础包到工程

struts2-core-2.1.8.1.jar --核心包

xwork-core-2.1.6.jar --xwork 所依赖的核心包, struts2 是在 xwork 基础上开发的

ognl-2.7.3.jar --ognl 表达式包

freemarker-2.3.15.jar --模板引擎包

commons-io-1.3.2.jar --处理 IO 包(可不用加)

commons-fileupload-1.2.1.jar --文件上传包

5. 拷贝 struts.xml(struts2-blank-2.1.8.1\WEB-INF\src\java)到 src 目录下

```
<struts>
    <!-- 关闭动态方法调用,(关闭 action 名 + 感叹号 + 方法名进行方法调用,如
login!checkLogin.action,调用 Action 名为 login 类中的 checkLogin 方法) -->
    <constant name="struts.enable.DynamicMethodInvocation" value="false" />
    <!-- 是否开发模式 -->
    <constant name="struts.devMode" value="false" />

    <!--
    导入其他配置文件
    <include file="helloWorld.xml"/>
    -->

    <!--
    name 标识(随便写)
    namespace 为路径前缀/:localhost:8888/StrutsPro/hello/helloWorld.action
    extends 继承参数,必须要有个.
    -->
    <package name="hello" namespace="/hello" extends="struts-default">
        <!-- name 为 Action 访问名称 -->
        <action name="helloWorld" class="com.struts2.hello.HelloAction">
            <!-- 无 name 默认 success,无 type 默认转发(type="dispatcher") -->
            <result>/helloWorld/HelloWorld.jsp</result>
        </action>

    </package>
</struts>
```

6. 写 Action 类(可以不继承任何类,也可以继承 ActionSupport)

```
//Action
public class HelloAction {

    private String message;

    // (默认执行方法)
    public String execute() {
        System.out.println("world:" + message);
        // SUCCESS 是 result 的默认值,即 result 中没有写 name,如
        <result>/example/HelloWorld.jsp</result>
        return "success";
    }
}
```

```

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}

```

可以继承 `Action` 接口,里面定义了一些返回的字符串常量(如 `SUCCESS,INPUT,ERROR...`),和 `execute` 方法. 或者继承 `ActionSupport` 类,`Action` 接口的默认实现.

7.写 HelloWorld.jsp 页面,访问 message 参数

```

<%@ taglib prefix="s" uri="/struts-tags" %>
<!-- s 标签访问 -->
<h2>Message:<s:property value="message"/></h2>

```

8.访问 HelloAction,带 message 参数

```

<a
href="${pageContext.request.contextPath}/hello/helloWorld.action?message=struts2">HelloWorld</a>

```

Struts2 在 Action 中访问 request,session,application

只需用到 attribute

```

//获取 request
ActionContext ctx = ActionContext.getContext();
ctx.put("request", "request");
//获取 Session
Map<String,Object> session = ctx.getSession();
session.put("session", "session");
//获取 ServletContext
Map<String,Object> application = ctx.getApplication();
application.put("application", "application");

```

获得 HttpServletRequest

1. 代码获得

```
HttpServletRequest request = ServletActionContext.getRequest();
HttpServletResponse response = ServletActionContext.getResponse();
HttpSession session = request.getSession();
//不要在 Action 构造器里调用,可能还没初始化好
ServletContext servletContext = ServletActionContext.getServletContext();
```

2. 接口获得

如果有多个 Action 需要用到 request,可以写个 Action 继承接口,使用 request 的 Action 继承再它.

```
/*继承 ServletRequestAware(获得 request),ServletResponseAware(获得 response),
ServletContextAware(获得 ServletContext),SessionAware(获得 Session)接口*/
public class FormAction implements ServletRequestAware,ServletResponseAware{

    private HttpServletRequest request;
    private HttpServletResponse response;

    //拦截器会把 request 注入到变量中
    public void setServletRequest(HttpServletRequest req) {
        this.request = req;
    }

    public void setServletResponse(HttpServletResponse res) {
        this.response = res;
    }
}
```

Struts2Action 中方法的访问

默认方法

```
//Struts2 中默认访问的是 Action 中 execute 方法
//url:http://127.0.0.1:8080/Struts2Pro/hello.action(访问 hello 指定的 Action
中的 execute 方法)
public String execute() {
    return "success";
}
```

访问 Action 中多个方法

1. 动态方法调用

只要在 url 中指定方法名称就可以了. 如:要访问 HelloAction 中的 add() 方法
url:<http://127.0.0.1:8080/Struts2Pro/hello!add.action> (加!和方法名字)

```
<!-- 在 struts.xml 中配置可以关闭动态方法(默认是激活,可能会让用户方法没有公开的方法,最好不用) -->
<constant name="struts.enable.DynamicMethodInvocation" value="false" />
```

2. 配置文件中多个方法

1. 配置多个方法

```
<!-- url:http://127.0.0.1:8080/Struts2Pro/userActionAdd.action 访问 FormAction 中的 add 方法 -->
<action name="userActionAdd" class="com.struts2.form.FormAction" method="add">
<!-- url:http://127.0.0.1:8080/Struts2Pro/userActionUpdate.action 访问 FormAction 中的 update 方法 -->
<action name="userActionUpdate" class="com.struts2.form.FormAction" method="update">
```

2. 使用通配符调用多个方法

```
<!--
name="*_*" class="com.struts2.form.{1}Action" method="{2}"
url:user_list.action 就是调用 userAction 中 list 方法.

name="*_*_*"
url:user_list_abd.action
method="{1}"->user(第一个*对应的字符串)
method="{2}"->list(第二个*对应的字符串)
method="{3}"->abd(第三个*对应的字符串)
-->
<action name="userAction_*" class="com.struts2.form.FormAction" method="{1}">
```

Struts2 返回类型

类型对应的类

Chain(chain)	构成一条动作链
Dispatcher(dispatcher)	默认类型,转发
FreeMarker(freemarker)	用于与 FreeMarker 的集成
HttpHeader(httpheader)	把 HTTP 标头发送回用户
Redirect(redirect)	重定向到另一个 URL
RedirectAction(redirectAction)	重定向到另一个 Action

<code>Stream(stream)</code>	把一个 <code>InputStream</code> 流发送给浏览器(下载用)
<code>Velocity(Velocity)</code>	用于与 <code>Velocity</code> 技术的集成
<code>XSLT(xslt)</code>	用于与 <code>XML/XSLT</code> 技术的集成
<code>PlainText(plaintext)</code>	发送普通文本,通常用来显示 <code>JSP</code> 页面的源代码

Chain

```
<!--
Chain 用途是构成一条动作链:前一个动作把控制权转交给后一个动作,而前一个动作的状态在后一个动作里仍保持
着.
动作链能不用就不用,有可能把一套连续动作弄成一团乱.
-->
<package name="package1" extends="struts-default">
  <action name="action1" class="...">
    <result type="chain">action2</result>
  </action>

  <action name="action2" class="...">
    <result type="chain">
      <param name="actionName">action3</param>
      <param name="namespace">/namespace2</param>
    </result>
  </action>
</package>

<package name="package2" namespace="/namespace2" extends="struts-default">
  <action name="action3" class="...">
    <result>/view.jsp</result>
  </action>
</package>
```

Dispatcher

```
<!-- 转发到 JSP,result 默认类型 -->
<result name="...">/view.jsp</result>

<!--或者-->
<result name="...">
  <param name="location">/view.jsp</param>
</result>
```

HttpHeader

```
<!-- 把一个 HTTP 状态发送给浏览器 -->
<action name="CatchAll">
  <result type="httpheader">
    <param name="status">404</param>
  </result>
</action>
```

Redirect

```
<!--
重定向
参数:location:重定向的目的地
      parse: 表明是否把 location 参数的值视为一个 OGNL 表达式来解释,默认值为 true
-->
<action name="..." class="...">
  <result name="success" type="redirect">
    <!-- 内部资源 -->
    /jsp/Product.jsp
    <!-- Action 带动态参数(${userName}值为本 Action 中的 userName 属性值) -->
    UserAction.action?userName=${userName}
    <!--
      外部资源(如果需要使用&和+之类的特殊字符必须使用转义序列.如:&改成&amp;)
      http://www.google.com?user=1&site=4
      转成:http://www.google.com?user=1&amp;site=4
    -->
    http://www.google.com
  </result>
</action>
```

RedirectAction

```
<!--
重定向到一个 Action
参数:actionName:指定重定向 Action 的名字
      namespace: 指定重定向 Action 的命名空间(没有此参数,与本 action 同一个命名空间)
-->
<result type="redirectAction">UserAction</result>
<!-- 或者 -->
<result type="redirectAction">
  <param name="actionName">UserAction</param>
  <!-- 参数 -->
</result>
```

```
<param name="userId">xyz</param>
<param name="area">ga</param>
<result>
<!-- 生成 URL:UserAction.action?userId=xyz&area=ga -->
```

PlainText

```
<!-- 通常被用来发送 JSP 页面的源代码 -->
<action name="source_show" class="...">
  <result name="success" type="plaintext">/jsp/Menu.jsp</result>
</action>
```

Struts2_OGNL 表达式

Struts2 标签不支持 el 表达式,只能使用 OGNL.

1. OGNL 分为 Object Stack 和 Context Map.

把动作和相关对象压入 Object Stack.

把各种映射关系(一些 Map 类型的对象)压入 Context

Map. (parameters, request, session, application, attr)

OGNL 表达式加上一个前缀"#", 访问 Context Map.

没加访问 Object Stack.

2. OGNL 访问数组

```
//String[] colors = {"blue","green","red"};
```

```
colors[0]//访问第一个元素
```

```
colors.length//访问长度
```

3. OGNL 访问 List

```
countries[0]//访问第一个元素
```

```
countries.size//访问 countries.size()
```

```
countries.isEmpty//访问 countries.isEmpty()
```

创建 List

```
{"a","b","c"}//创建一个由 3 个 String 构成的 List
```

4. OGNL 访问 Map

```
cities["CA"]或者 cities['CA']//访问 cities 中 key 为 CA 的元素
```

```
cities.size//访问 cities.size()
```

```
cities.isEmpty//访问 cities.isEmpty()
```

创建 Map

```
{"CA": "S", "WA": "0", "UT": "ST"}
```

`{}`里面的表达式会被求值(`{#request.user.age>40}->true or false,{1+6}->7)`)

Struts2 的 S 标签

`<!-- 访问 request 必须要加# -->`

`<p>赋值, 取值</p>`

`<!-- 页面定义变量(本页赋值用 name 要加#取值), default 默认值.-->`

`<!-- 赋值字符串要加'', 不是去 Action 中找对应的变量.value="test"->在 Action 中找 test 变量 -->`

```
<s:set id="test1" value="'test1'" />
```

`<!-- #request.get('javax.servlet.forward.context_path') 必须经过 Action 转发到 jsp 才有值, 直接访问 jsp 无值 -->`

```

                                <s:set          id="msg"
value="#request.get('javax.servlet.forward.context_path')"/>
s:<s:property value="#msg" default="Struts2"/><br/>
el:${msg}<br/>

```

`<!-- 取 Action 中变量的值不用加# -->`

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

`<!-- Action 中的变量放在 request 中 -->`

```
--${requestScope.user}||<br/>
```

`<p>if 标签: If 标签用来控制基本的条件处理流程, 通常和 else 标签或者 elseif 标签连用。({}可加可不加)</p>`

`<!-- test 必须, 支持表达式(不支持 el 表达式) -->`

```
<s:if test="#request.user!=null">
```

```
    <div>user!=null</div>
```

```
    <s:if test="%{#request.user.age>40}">
```

```
        <div>中年人</div>
```

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

```
    <s:elseif test="%{#request.user.age>20}">
```

```
        <div>年轻人</div>
```

```
    </s:elseif>
```

```
    <s:else>
```

```
        <div>小孩</div>
```

```
    </s:else>
```

```

</s:if>
<s:else>
    <div>user==null</div>
</s:else>
<p><b>iterator 标签:</b> 对集合迭代 </p>
迭代 List<br />
<!-- 判断是否为空 -->
<s:if test="(userList!=null)&&(!userList.isEmpty())">
    <div>userList</div>

    <s:iterator value="#request.userList" id="u" status="st">
        index:<s:property value="#st.index+1" />&nbsp;
        是否第一个:<s:property value="#st.first" />&nbsp;
        是否最后一个:<s:property value="#st.last" />&nbsp;
        是否偶数:<s:property value="#st.even" />&nbsp;
        是否奇数:<s:property value="#st.odd" />&nbsp;
        <br />
        <s:property value="#u.name" />&nbsp;
        <s:property value="#u.age" />&nbsp;
        <!-- 格式化日期 -->
        <s:date name="#u.birthday" format="yyyy-MM-dd HH:mm:ss"
/>&nbsp;

        <br />
    </s:iterator>
</s:if>
<s:else>
    <div>nul</div>
</s:else>
迭代 Map<br />
<s:if test="(userMap!=null)&&(!userMap.isEmpty())">
    <div>userMap</div>

    <s:iterator value="#request.userMap" id="map" status="st">
        index:<s:property value="#st.index+1" />&nbsp;
        是否第一个:<s:property value="#st.first" />&nbsp;
        是否最后一个:<s:property value="#st.last" />&nbsp;
        是否偶数:<s:property value="#st.even" />&nbsp;
        是否奇数:<s:property value="#st.odd" />&nbsp;
        <br />
        <s:property value="#map.key" />&nbsp;
        <s:property value="#map.value.name" />&nbsp;
        <s:property value="#map.value.age" />&nbsp;
        <!-- 格式化日期 -->
        <s:date name="#map.value.birthday" format="yyyy-MM-dd

```

```

HH:mm:ss" />&nbsp;

        <br/>
    </s:iterator>
</s:if>
<s:else>
    <div>null</div>
</s:else>

<p><b>append 标签:</b> 对多个集合进行合并 </p>
<br />

合并 List <br/>
<s:append id="newList">
    <s:param value="#request.userList"></s:param>
    <s:param value="#request.userList"></s:param>
</s:append>
<s:iterator value="#newList" id="u" status="st">
    index:<s:property value="#st.index+1"/>&nbsp;
    是否第一个:<s:property value="#st.first"/>&nbsp;
    是否最后一个:<s:property value="#st.last"/>&nbsp;
    是否偶数:<s:property value="#st.even"/>&nbsp;
    是否奇数:<s:property value="#st.odd"/>&nbsp;
    <br/>
    <s:property value="#u.name"/>&nbsp;
    <s:property value="#u.age"/>&nbsp;
    <!-- 格式化日期 -->
    <s:date name="#u.birthday" format="yyyy-MM-dd HH:mm:ss" />&nbsp;
    <br/>
</s:iterator>
<br />

List 和 Map 合并,结果为 Map <br/>
<s:append id="newListMap">
    <s:param value="#request.userList"></s:param>
    <s:param value="#request.userMap"></s:param>
</s:append>

<s:iterator value="#newListMap" id="map" status="st">
    index:<s:property value="#st.index+1"/>&nbsp;
    是否第一个:<s:property value="#st.first"/>&nbsp;
    是否最后一个:<s:property value="#st.last"/>&nbsp;
    是否偶数:<s:property value="#st.even"/>&nbsp;
    是否奇数:<s:property value="#st.odd"/>&nbsp;
    <br/>

```

```

        <s:property value="#map.key" />&nbsp;
        <s:property value="#map.value.name" />&nbsp;
        <s:property value="#map.value.age" />&nbsp;
        <!-- 格式化日期 -->
        <s:date name="#map.value.birthday" format="yyyy-MM-dd HH:mm:ss"
    />&nbsp;

    <br />
</s:iterator>

<!-- 显示栈里的参数信息 -->
<s:debug></s:debug>

```

Struts2 表单标签

```

<!--
    action="userAction!%{tip}" 动态方法 tip 为 Action 中的变量 submit 不加 method
-->
<s:form action="userAction" namespace="/form" method="POST">
    <!-- userBean.id 为 Action 中属性 userBean 中 id 的值 -->
    <s:hidden name="userBean.id" /></s:hidden>
    <s:textfield name="userBean.name" label="用户名" /></s:textfield>
    <s:textfield name="userBean.age" label="年龄" /></s:textfield>
    <s:textfield name="userBean.birthday" label="生日" >
        <!-- 格式化日期 -->
        <s:param name="value">
            <s:date name="userBean.birthday" format="yyyy-MM-dd" />
        </s:param>
    </s:textfield>
    <!-- 密码不能回填 -->
    <s:password name="userBean.password" label="密码" /></s:password>

    <s:textarea name="userBean.des" label="描述" cols="35" rows="8" /></s:textarea>

    <!-- 这种 checkbox 显示出来是一行只有一个 checkbox
    <s:iterator value="likeList">
        <s:checkbox name="userBean.likes" label="%{name}"
fieldValue="%{id}" /></s:checkbox>
    </s:iterator>
    -->
    <!-- 这种是一行多个 -->
    <s:checkboxlist list="likeList" name="userBean.likes" listKey="id" listValue="name"
label="爱好" /></s:checkboxlist>

```

```

        <!-- 下拉框
        <s:select list="likeList" listKey="id" listValue="name" headerKey="-1" headerValue="请选择爱好"></s:select>
        -->
        <!-- value="1" 设置默认,但是好像设置了不能回填 -->
        <s:radio list="#{'1':'男','2':'女'}" label="性别" name="userBean.sex"></s:radio>
        <!-- value 显示值,method 调用方法 -->
        <s:submit value="%{tip}" id="submitBut" method="%{tip}"></s:submit>
    </s:form>

```

Struts2 文件上传下载

文件上传

1.jsp 页面

```

<s:form action="fileAction" namespace="/file" method="POST"
enctype="multipart/form-data">
    <!-- name 为后台对应的参数名称 -->
    <s:file name="files" label="file1"></s:file>
    <s:file name="files" label="file2"></s:file>
    <s:file name="files" label="file3"></s:file>
    <s:submit value="提交" id="submitBut"></s:submit>
</s:form>

```

2.Action

```

//单个文件上传可以用 File files,String filesFileName,String filesContentType
//名称要与 jsp 中的 name 相同(三个变量都要生成 get,set)
private File[] files;
// 要以 File[]变量名开头
private String[] filesFileName;
// 要以 File[]变量名开头
private String[] filesContentType;

private ServletContext servletContext;

//Action 调用的上传文件方法
public String execute() {
    ServletContext servletContext =

```

```

ServletActionContext.getServletContext();
String dataDir = servletContext.getRealPath("/file/upload");
System.out.println(dataDir);
for (int i = 0; i < files.length; i++) {
    File saveFile = new File(dataDir, filesFileName[i]);
    files[i].renameTo(saveFile);
}
return "success";
}

```

3.配置上传文件临时文件夹(在 struts.xml 中配置)

```
<constant name="struts.multipart.saveDir" value="c:/temp"/>
```

文件下载

1. 下载的 url (到 Action)

```
<a href="${pageContext.request.contextPath}/file/fileAction!down.action">下载</a>
```

2. struts.xml 配置

```

<package name="file" namespace="/file" extends="struts-default">
    <action name="fileAction" class="com.struts2.file.FileAction">
        <!-- 下载文件配置 -->
        <!--type 为 stream 应用 StreamResult 处理-->
        <result name="down" type="stream">
            <!--
                不管实际类型，待下载文件 ContentType 统一指定为 application/octet-stream
                默认为 text/plain
            -->
            <param name="contentType">application/octet-stream</param>
            <!--
                默认就是 inputStream，它将会指示 StreamResult 通过 inputName 属性值的 getter 方法，
                比如这里就是 getInputStream() 来获取下载文件的内容，意味着你的 Action 要有这个方法
            -->
            <param name="inputName">inputStream</param>
            <!--
                默认为 inline(在线打开)，设置为 attachment 将会告诉浏览器下载该文件，filename 指定下载文件
                件保有存时的文件名，若未指定将会是以浏览的页面名作为文件名，如以 download.action 作为文件名，
            -->
        </result>
    </action>
</package>

```

这里使用的是动态文件名, `${fileName}`, 它将通过 Action 的 `getFileName()` 获得文件名

```

-->
<param name="contentDisposition">attachment;filename="${fileName}"</param>
<!-- 输出时缓冲区的大小 -->
<param name="bufferSize">4096</param>
</result>
</action>
</package>

```

3.Action

```

//Action 调用的下载文件方法
public String down() {
    return "down";
}

//获得下载文件的内容, 可以直接读入一个物理文件或从数据库中获取内容
public InputStream getInputStream() throws Exception {
    String dir = servletContext.getRealPath("/file/upload");
    File file = new File(dir, "icon.png");
    if (file.exists()) {
        //下载文件
        return new FileInputStream(file);

        //和 Servlet 中不一样, 这里我们不需对输出的中文转码为 ISO8859-1
        //将内容(Struts2 文件下载测试)直接写入文件, 下载的文件名必须是文本(txt)类型
        //return new ByteArrayInputStream("Struts2 文件下载测试".getBytes());
    }
    return null;
}

// 对于配置中的 ${fileName}, 获得下载保存时的文件名
public String getFileName() {
    String fileName = "图标.png";
    try {
        // 中文文件名也是需要转码为 ISO8859-1, 否则乱码
        return new String(fileName.getBytes(), "ISO8859-1");
    } catch (UnsupportedEncodingException e) {
        return "icon.png";
    }
}

```

Struts2 返回 JSON

1. 导入 jsonplugin 包

Struts2.16: 导入 jsonplugin-0.34.jar 包([下载包](#))和 commons-logging-1.0.4.jar(Struts2 lib 下有)

Struts2.18 导入 struts2-json-plugin-2.1.8.1.jar(Struts2 lib 下有)

2. struts.xml 中 package 中 extends="json-default"

```
<package name="json" namespace="/json" extends="json-default">
```

3. result 中 type="json"

```
<!-- 封装所以的 get 开头的方法 -->
```

```
<result type="json" name="user">
</result>
```

```
<!-- 只包含 user.id 属性 -->
```

```
<result type="json" name="user">
  <param name="includeProperties">
    user\.id
  </param>
</result>
```

```
<!-- 不包含 user 属性 -->
```

```
<result type="json" name="list">
  <param name="excludeProperties">
    user
  </param>
</result>
```

```
<!-- 根对象只包含 user -->
```

```
<result type="json">
  <param name="root">
    user
  </param>
</result>
```

```
<!-- "root"对象中父类的 field(属性)不会(会?) 默认存放到 JSON 数据中, 如果不想这样做,
需要在配置时指定 ignoreHierarchy 为 false: -->
```

```
<result type="json">
  <param name="ignoreHierarchy">false</param>
</result>
```

4.避免使用 **get** 开头的 **action** 方法

在属性 **get** 方法上面加

@JSON(name="newName") json 中的名称

@JSON(serialize=false) 属性不被加入 json

@JSON(format="yyyy-MM-dd") 格式化日期

5.在 **action** 中赋值,返回对应的 **result** 字符串

Struts2 返回 XML

使用 **response**

```
//直接调用 responseOut 方法
public void responseOut() throws Exception {
    StringBuilder sb = new StringBuilder();
    sb.append("<?xml version=\"1.0\" encoding=\"UTF-8\" ?>");
    sb.append("<users>");
    sb.append("    <user id=\"50\">");
    sb.append("        <userName>abc</userName>");
    sb.append("    </user>");
    sb.append("</users>");

    //得到 response
    HttpServletResponse response = ServletActionContext.getResponse();
    //设置编码
    response.setCharacterEncoding("UTF-8");
    response.setContentType("text/xml;charset=utf-8");
    response.setHeader("Cache-Control", "no-cache");
    PrintWriter out = response.getWriter();
    out.write(sb.toString());
    out.flush();
    out.close();
}
```

Struts2 增删改查例子

UserBean

//要添加 set,get

```
public class UserBean {
    private Long id;
```

```
private String name;
private String password;
private int age;
private Date birthday;
private Long sex;
private Set<Long> likes;
private String des;//描述
}
```

```
public class LikeBean {
    private Long id;
    private String name;
}
```

FormAction

```
public class FormAction{
    //属性要生成 get,set
    //标记(添加,修改)
    private String tip;
    private UserBean userBean;
    private List<LikeBean> likeList;
    private List<UserBean> userList;
    //到列表页面
    public String list(){
        userList = new ArrayList<UserBean>();
        userList.add(new UserBean("abc",19,new Date()));
        userList.add(new UserBean("def",70,new Date()));
        userList.add(new UserBean("ghi",40,new Date()));
        return "list";
    }
    //到添加页面
    public String toAdd(){
        likeList = new ArrayList<LikeBean>();
        likeList.add(new LikeBean(new Long(1),"上网"));
        likeList.add(new LikeBean(new Long(2),"读书"));
        likeList.add(new LikeBean(new Long(3),"游戏"));

        tip = "add";
        return "info";
    }
    //添加方法,Struts2 会自动把对应的值赋值给 userBean
    public String add(){
        System.out.println(userBean);
        return "toList";
    }
}
```

//到更新页面,取出值,设置标记

```
public String toUpdate(){
    likeList = new ArrayList<LikeBean>();
    likeList.add(new LikeBean(new Long(1),"上网"));
    likeList.add(new LikeBean(new Long(2),"读书"));
    likeList.add(new LikeBean(new Long(3),"游戏"));

    tip = "update";
    System.out.println("update name:"+userBean.getName());
    userBean = new UserBean("update",100,new Date());
    userBean.setId(new Long(101));
    userBean.setSex(new Long(2));
    userBean.setDes("Struts2 用户");
    Set<Long> likes = new HashSet<Long>();
    likes.add(new Long(2));
    likes.add(new Long(3));
    userBean.setLikes(likes);
    return "info";
}

//更新方法
public String update(){
    System.out.println(userBean);
    return "toList";
}

//删除方法
public String delete(){
    System.out.println("del name:"+userBean.getName());
    return "toList";
}
}
```

struts.xml 配置

```
<struts>

<package name="form" namespace="/form" extends="struts-default">
    <!-- 使用通配符调用多个方法,method 为 * 对应的字符串 -->
    <action name="userAction_*" class="com.struts2.form.FormAction" method="{1}">
        <!-- 方法返回的字符串对应 name,再转发到页面 -->
        <result name="list">/form/list.jsp</result>
        <result name="info">/form/info.jsp</result>
        <!-- 重定向 Action,可以有 Action 名字和空间,参数 -->
        <result name="toList" type="redirectAction">
            <!-- 空间参数,不写代表和当前 Action 同一个空间 -->
            <param name="namespace">form</param>
        </result>
    </action>
</package>
```

```
<!-- 添加标签 -->  
<%@ taglib prefix="s" uri="/struts-tags" %>  
  
    <div><a href="${pageContext.request.contextPath}/form/userAction_toAdd.action">添加</a></div>  
  
    <div>  
  
        <!-- 判断用户是否为空 -->  
  
        <s:if test="(userList!=null)&&(!userList.isEmpty())">  
  
            <div>用户列表</div>  
  
            <!-- 遍历用户 -->  
  
            <s:iterator value="#request.userList" id="u" status="st">  
  
                <s:property value="#st.index+1"/>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
                <br/>  
            </s:iterator>  
  
            </s:if>  
  
            <s:else>  
  
                <div>无用户</div>  
  
            </s:else>  
  
        </div>
```

23 / 38 整理者: Matt Zhao Email: mingliangzhao@126.com

```

<!-- 添加标签 -->
<%@ taglib prefix="s" uri="/struts-tags" %>

    <!-- 显示标记(添加/修改) -->
    <s:property value="tip"/>

    <!-- action="userAction_ %{tip}" tip 为 Action 中的标记变量 submit 不加 method -->
    <s:form action="userAction" namespace="/form" method="POST">

        <!-- userBean.id 为 Action 中属性 userBean 中 id 的值 -->
        <s:hidden name="userBean.id"></s:hidden>

        <s:textfield name="userBean.name" label="用户名" ></s:textfield>
        <s:textfield name="userBean.age" label="年龄" ></s:textfield>
        <s:textfield name="userBean.birthday" label="生日" >

            <!-- 格式化日期 -->
            <s:param name="value"><s:date name="userBean.birthday" format="yyyy-MM-dd"
/></s:param>

        </s:textfield>

        <!-- 密码不能回填 -->
        <s:password name="userBean.password" label="密码" ></s:password>
        <s:textarea name="userBean.des" label="描述" cols="35" rows="8"></s:textarea>

        <!-- 这种 checkbox 显示出来是一行只有一个 checkbox
        <s:iterator value="likeList">

            <s:checkbox name="userBean.likes" label="%{name}" fieldValue="%{id}"></s:checkbox>

        </s:iterator>

        -->

        <!-- 这种是一行多个 -->
        <s:checkboxlist list="likeList" name="userBean.likes" listKey="id" listValue="name" label="爱好
"></s:checkboxlist>

        <!-- 下拉框
        <s:select list="likeList" listKey="id" listValue="name" headerKey="-1" headerValue="请选择爱好
"></s:select>

        -->

        <!-- value="1" 设置默认,但是好像设置了不能回填 -->
        <s:radio list="#{'1':'男','2':'女'}" label="性别" name="userBean.sex"></s:radio>

        <s:submit value="%{tip}" id="submitBut" method="%{tip}"></s:submit>

    </s:form>

```

Struts2 验证框架

Action 配置中一定要设置 input 返回页面

添加验证只要创建验证的 xml 文件

Action 配置中一定要设置 input 返回页面

添加验证只要创建验证的 xml 文件

1.创建 xml 文件名

验证 Action 中全部方法

在 Action 同包下,创建:Action 类名-validation.xml

如:ValidateAction 创建 ValidateAction-validation.xml

验证 Action 中单个方法

```
<!-- 每个方法单独配置一个 Action -->
<!-- 在 Action 同包下,创建:Action 类名-action 方法配置名称-validation.xml -->
<action name="validateAdd" class="com.struts2.validator.ValidateAction" method="add">
<!-- 要创建 ValidateAction-validateAdd-validation.xml -->

<!-- 使用通配符配置 -->
<!-- 在 Action 同包下,创建:Action 类名-action 方法对应的名称-validation.xml -->
<action name="validate_*" class="com.struts2.validator.ValidateAction" method="{1}">
<!-- 要创建 ValidateAction-validate_add-validation.xml,validate_add 为访问这个 action 方法的路径 -->
```

注意事项

注意:

- 1.要验证的方法不能叫 input.
- 2.这样配置在 form 表单中要在<s:form action="validate_add">中 action 写好名称,不能写 action="validate_ ",然后<s:submit value="提交"method="add" />这样会找不到对应的配置文件,跳过验证.
- 3.如果验证出错,返回 input 页面时,那些存在 ValueStack 中的值会丢失,可以将 Action 实现 Preparable 接口,然后 prepare()方法里初始化添加页面需要的值.
- 4.如果使用 Preparable 接口,必须在 action 配置中添加<interceptor-ref name="paramsPrepareParamsStack" />.这样 prepare()才能得到 form 提交的参数.

2.创建 xml 内容

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0.2//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">

<validators>

<!-- 要验证的字段名称 -->
```

```

    <!-- 要验证 Action 中 UserBean 的 id 字段,name="userBean.id"(userBean 为 Action 中的变量名) -->
    <field name="mail">
        <!-- type 要验证的类型,short-circuit(默认 false),true 含义,如果当前验证失败了,下面的验证就不执行了.如
        requiredstring 失败了,email 就不验证了. -->
        <!-- field-validator 下面可以有多个 param 元素,但是最多只能有一个 message -->
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message>Please enter a mail</message>
        </field-validator>
        <field-validator type="email">
            <message>
                Invalid MAIL
            </message>
        </field-validator>
    </field>
</validators>

```

Struts 内建验证程序(type 的值)

required

保证字段的值不是空值 null.空字符串不是空值 null.

```

<field name="userName">
    <field-validator type="required">
        <message>Please enter a user name</message>
    </field-validator>
</field>

```

requiredstring

保证字段不是空值 null,也不是空白(empty).

param:trim(boolean) ->true->去除前后空格

```

<field name="userName">
    <field-validator type="requiredstring">
        <param name="trim">true</param>
        <message>Please enter a user name</message>
    </field-validator>
</field>
<field name="password">
    <field-validator type="requiredstring">
        <param name="trim">>false</param>

```

```
<message>Please enter a password</message>
</field-validator>
</field>
```

int

验证字段值是否可以转换为一个整数.

param: min(int);max(int)

```
<field name="yeaar">
  <field-validator type="int">
    <param name="min">1999</param>
    <param name="max">2010</param>
    <message>year: 1999-2010</message>
  </field-validator>
</field>
```

date

验证给定日期字段的值是否在一个给定的范围内.

param:max(date);min(date)

```
<field name="borthday">
  <field-validator type="int">
    <!-- 格式取决于当前地理时区 -->
    <param name="min">1999-01-01</param>
    <param name="max">2010-01-01</param>
    <message>birthday:1999-2010</message>
  </field-validator>
</field>
```

email

给定的 String 值是否是一个电子邮件地址

```
<field name="email">
  <field-validator type="email">
    <message>Invalid email</message>
  </field-validator>
</field>
```

url

给定的 String 值是否是一个合法的 URL(要有前缀)

```
<field name="url">
  <field-validator type="url">
    <message>Invalid URL</message>
  </field-validator>
</field>
```

expression,fieldexpression

验证给定字段是否满足一个 OGNL 表达式.

区别:expression 不是一个字段验证程序,失败时将生成一个动作错误.(JSP 中调用 <s:actionerror/>才显示出错信息)

fieldexpression 是一个字段验证程序,失败时将抛出一个字段错误.(对字段验证)

param:expression(String)OGNL 表达式

expression:

```
public class ExpressionTestAction {
    //属性生成 get,set
    private int min;
    private int max;
}

<validator type="expression">
  <param name="expression">
    max > min
  </param>
  <message>
    Maximum temperature must be greater than Minimum temperature
  </message>
</validator>

<!-- jsp -->
<s:actionerror/>
```

fieldexpression:

```
public class FieldExpressionTestAction {
    //属性生成 get,set
    private int min;
    private int max;
}
```

```

<!-- 对字段验证 -->
<field name="max">
  <field-validator type="fieldexpression">
    <param name="expression">
      max > min
    </param>
    <message>
      Maximum temperature must be greater than Minimum temperature
    </message>
  </field-validator>
</field>

```

visitor

把同一个验证程序配置文件用于多个动作(对一个 Bean 写验证文件,每个使用的 Action 只要引用)

```

//UserBean
public class UserBean {
  //属性 get,set
  private String name;
  private int age;
}

//UserBean-validation.xml(和 UserBean 放在同一个包中)
<field name="name">
  <field-validator type="requiredstring">
    <message>用户名必须</message>
  </field-validator>
</field>
<field name="age">
  <field-validator type="int">
    <param name="min">18</param>
    <param name="max">99</param>
    <message>Age must be between 18 and 99</message>
  </field-validator>
</field>

//Action 的 validation.xml
<!-- userBean 变量名 -->
<field name="userBean">
  <field-validator type="visitor">
    <!-- message 会和 UserBean 验证中的 message 一起显示 -->
    <message>用户: </message>
  </field-validator>
</field>

```

```

    </field-validator>
</field>

```

如果另一个 Action 对 UserBean 使用另一个标准的验证,可以创建新的验证文件

```

//UserBean-specific-validation.xml
<!-- 和之前的验证不同 -->
<field name="age">
    <field-validator type="int">
        <param name="min">30</param>
        <param name="max">50</param>
        <message>Age must be between 30 and 50</message>
    </field-validator>
</field>
//另一个 Action 的 validation.xml
<field name="userBean">
    <field-validator type="visitor">
        <!-- xml 中扩展的名字,执行 UserBean-specific-validation.xml 的验证 -->
        <param name="context">specific</param>
        <message>用户 1: </message>
    </field-validator>
</field>

```

conversion

检查对某个属性进行类型转换是否会导致一个转换错误

```

<field name="age">
    <field-validator type="conversion">
        <message>
            An age must be an integer.
        </message>
    </field-validator>
</field>

```

stringlength

验证一个非空的字段值是不是足够的长度

param:minLength(int);maxLength(int);trim(boolean)

```

<field name="password">
    <field-validator type="requiredstring">
        <param name="minLength">6</param>
        <param name="maxLength">14</param>
    </field-validator>
</field>

```

```

        <message>length:6-14</message>
    </field-validator>
</field>

```

regex

给定的值是否与一个给定的正则表达式匹配

param:expression(String)正则表达式;caseSensitive(boolean)是否区别大小写,默认为true;trim(boolean)是否去除前后空格

```

<field name="phone">
    <field-validator type="regex">
        <param name="expression">
            <![CDATA[\d\d\d\d\d\d\d\d\d\d]]>
        </param>
        <message>
            Invalid phone number or invalid format
        </message>
    </field-validator>
</field>

```

3.在 action 中验证

利用 Validateable 接口实现验证,实现 void validate()方法.

ActionSupport 类已经实现了这个接口

```

//继承 ActionSupport
public class User extends ActionSupport {
    //属性 get,set
    private String userName;
    private String password;
    private static List<String> userNames = new ArrayList<String>();
    static {
        userNames.add("harry");
        userNames.add("sally");
    }
    //验证方法
    public void validate() {
        if (userNames.contains(userName)) {
            //添加出错信息
            addFieldError("userName", "" + userName + " has been taken.");
        }
    }
}

```

```

    }
}

```

4. 自定义验证类

要创建一个普通的验证程序(非字段验证程序),扩展 `ValidatorSupport` 类.验证失败要从 `validate` 方法调用 `addActionError` 方法.

要创建一个字段验证程序,扩展 `FieldValidatorSupport` 类.验证失败要从 `validate` 方法调用 `addFieldError` 方法.

如果要能接受参数,要在类中定义一个相应的属性,并生成 `get,set`.

编写类

```

public class StrongPasswordValidator extends FieldValidatorSupport {
    //属性
    private int minLength = -1;
    public void setMinLength(int minLength) {
        this.minLength = minLength;
    }
    public int getMinLength() {
        return minLength;
    }
    //验证方法
    public void validate(Object object) throws ValidationException {
        String fieldName = getFieldName();
        String value = (String) getFieldValue(fieldName, object);
        if (value == null || value.length() <= 0) {
            // use a required validator for these
            return;
        }
        if ((minLength > -1) && (value.length() < minLength)) {
            addFieldError(fieldName, object);
        } else if (!isPasswordStrong(value)) {
            addFieldError(fieldName, object);
        }
    }

    private static final String GROUP_1 = "abcdefghijklmnopqrstuvwxyz";
    private static final String GROUP_2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private static final String GROUP_3 = "0123456789";
    protected boolean isPasswordStrong(String password) {
        boolean ok1 = false;
        boolean ok2 = false;
    }
}

```

```
boolean ok3 = false;
int length = password.length();
for (int i = 0; i < length; i++) {
    if (ok1 && ok2 && ok3) {
        break;
    }
    String character = password.substring(i, i + 1);
    System.out.println("character:" + character);
    if (GROUP_1.contains(character)) {
        ok1 = true;
        continue;
    }
    if (GROUP_2.contains(character)) {
        ok2 = true;
        continue;
    }
    if (GROUP_3.contains(character)) {
        ok3 = true;
    }
}
return (ok1 && ok2 && ok3);
}
```

注册 xml

在 src 下创建 validators.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator Config
1.0/EN" "http://www.opensymphony.com/xwork/xwork-validator-config-1.0.dtd">

<validators>

    <!-- 名称(type 对应值),类路径 -->
    <validator name="strongpassword" class="com.validator.StrongPasswordValidator"/>

</validators>
```

使用验证

```
<field name="password">
    <field-validator type="strongpassword">
        <param name="minLength">8</param>
    </field-validator>
</field>
```

```

<message>
    Password must be at least 8 characters long
    and contains at least one lower case character,
    one upper case character, and a digit.
</message>
</field-validator>
</field>

```

Struts2 国际化

1. 定义 properties 文件

资源文件查找顺序

JAVA 国际化

如果系统同时存在资源文件、类文件，系统将以类文件为主，而不会调用资源文件。

对于简体中文的 **Locale**, **ResourceBundle** 搜索资源的顺序是：

- (1) `baseName_zh_CN.class`
- (2) `baseName_zh_CN.properties`
- (3) `baseName_zh.class`
- (4) `baseName_zh.properties`
- (5) `baseName.class`
- (6) `baseName.properties`

/ Struts2 找不到对应语言的配置文件时, 会先找系统语言对应的配置文件, 再找不到才使用 `baseName.properties` */*

国际化分为三类：全局的，包级别的，类级别的

全局的：

1. 在 `struts.xml` 中的 `<constant name="struts.custom.i18n.resources" value="message"></constant>` 指定 `baseName`

国际化文件名为：`baseName_语言名_国家名.properties`

(如：`message_zh_CN.properties`)

2. 全局的国际化资源文件放在 `src` 下面

包级别的：

1. 包级别的国际化资源文件放在该包下面

2. 命名规则为：`package_语言名_国家`

名.`properties` (如：`package_zh_CN.properties`)

其中 `package` 不变, 不是指的是包名, 每个包的国际化文件命名都这样

类级别的:

1. 与该类放在同一目录中
2. 命名规则为: 类名_语言名_国家名.properties

(如: RegisterAction_zh_CN.properties)

若同一 key 在上面三个国际化文件中都为 value 值则: (优先级) 类级别的>包级别的>全局的

2. 在 jsp 中访问国际化资源文件

```
<!-- 输出国际化 -->
<!-- name 为国际化文件中的 key -->
<s:text name="hello">
    <!-- 若该国际化文件的 value 中有{0}则可用下面的标签把参数传进去 -->
    <s:param>mengya</s:param>
    <s:param><s:property value="firstName"/></s:param>
</s:text>
```

```
<s:property value="%{getText('customer.contact')}" />
```

传参数

```
<s:property value="%{getText('customer.contact','sh')}" />
```

定义变量, 加 var 属性

```
<s:text name="greetings" var="msg" />
<s:property value="#msg" />
```

<!-- 指定特定的国际化文件, name 为全局国际化文件的 baseName -->

```
<s:il8n name="temp">
    <!-- 下面的<s:text>标签与上面的用法一样 -->
    <s:text name="hello">
        <s:param>mengya</s:param>
    </s:text>
</s:il8n>
```

表单国际化:

1. theme 不能为 simple 如: <s:form action="register"> (默认的 theme 不是 simple)
2. 使用 key 如: <s:textfield name="username" key=""

```
id="usernameId"></s:textfield>
```

```
3. 或者使用 getText 方法 <s:textfield name="username"
label="%{getText('customer.name')}" />
```

3.在 Action 中访问国际化资源文件,该 Action 继承了 ActionSupport 类

```

this.getText("username.invalid")
若该 key 对应的 value 需要参数则:
this.getText("username.invalid",new String[]{username})
或
List list = new ArrayList();
list.add(username);
this.getText("username.invalid",list)
如:
if (null == username || username.length() < 6 || username.length() > 10)
{
    List list = new ArrayList();
    list.add(username);
    this.addActionError(this.getText("username.invalid",new
String[]{username}));
}
该国际化资源文件中的 key 和 value 为:
username.invalid = \u7528\u6237\u540d "{0}"
\u586b\u5199\u4e0d\u6b63\u786e

```

4.在输入验证框架访问国际化资源文件

```

<!-- 使用<message key="..."></message> -->
<field-validator type="requiredstring">
    <param name="trim">true</param>
    <message key="username.invalid"></message>
</field-validator>

```

5.动态改变语言

只有在 url 传入 request_locale=en_US 就可以指定为 en_US 的语言

1.如果配置文件是 en_US 结尾,传入 en,en_US 都可以找到

2.如果配置文件是 en 结尾,只能传入 en,传入 en_US 会找不到,从而找默认的配置文(en 的范围>en_US,使用英语的国家不止美国一个)

```

<a href="{pageContext.request.contextPath}/i18n/i18nAction?request_locale=en_US">English</a>

```

6.使用类来代替 properties 文件

```
//继承 ListResourceBundle 实现 getContents 方法
public class MyCustomResourceBundle extends ListResourceBundle {
    public Object[][] getContents() {
        if (Calendar.getInstance().get(Calendar.HOUR_OF_DAY) < 12) {
            return contents1;
        } else {
            return contents2;
        }
    }
    static final Object[][] contents1 = {
        { "greetings", "中午好 {0}" },
        { "farewell", "再见 " } };

    static final Object[][] contents2 = {
        { "greetings", "你好 {0}" },
        { "farewell", "再见 " } };
}

//en 对应的字符串
public class MyCustomResourceBundle_en extends ListResourceBundle {
    public Object[][] getContents() {
        if (Calendar.getInstance().get(Calendar.HOUR_OF_DAY) < 12) {
            return contents1;
        } else {
            return contents2;
        }
    }
    static final Object[][] contents1 = {
        { "greetings", "Good morning {0}" },
        { "farewell", "Good bye" } };

    static final Object[][] contents2 = {
        { "greetings", "Hello {0}" },
        { "farewell", "Good bye" } };
}

//jsp 中使用
<s:i18n name="com.resourcebundle.MyCustomResourceBundle">
    <s:text name="greetings">
        <s:param>Jon</s:param>
    </s:text>.
```

```
<s:text name="farewell"/>  
</s:i18n>
```