# 一、Filter作为控制器并搭建Struts2的基本环境

## 1.MVC模式

模型：封装应用程序的数据和业务逻辑 POJO

视图：实现应用程序的信息显示功能 JSP

控制器：接收来自用户的输入，调用模型层，响应对应的视图组件 Servlet，Filter

## 2. 使用 Filter 作为控制器的好处

使用一个过滤器来作为控制器, 可以方便地在应用程序里对**所有资源**(包括静态资源)进行控制访问，可以完全替代servlet

## 3.控制流程

**public** **void** doFilter() **throws** IOException, ServletException {

HttpServletRequest req = (HttpServletRequest) request;

//1. 获取 servletPath

String servletPath = req.getServletPath();

String path = **null**;

//2. 判断 servletPath, 若其等于 "/product-input.action", 则转发到

**if**("/product-input.action".equals(servletPath)){

path = "/WEB-INF/pages/input.jsp";

}

//3. 若其等于 "/product-save.action", 则

**if**("/product-save.action".equals(servletPath)){

//1). 获取请求参数

//2). 把请求信息封装为一个 Product 对象

//3). 执行保存操作

//4). 把 Product 对象保存到 request 中.

path = "/WEB-INF/pages/details.jsp";

}

//4.转发

**if**(path != **null**){

request.getRequestDispatcher(path).forward(request, response);

**return**;

}

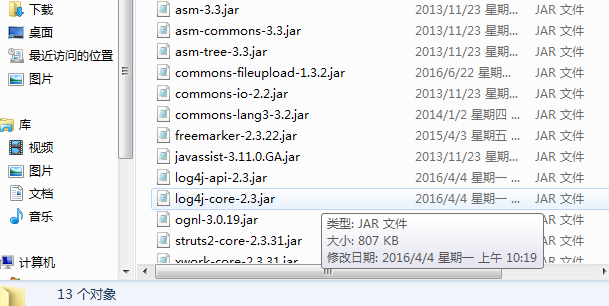
chain.doFilter(request, response);

}

## 4.搭建Struts2基本环境

### (1)加入 jar 包:

复制 struts-2.3.31\apps\struts2-blank\WEB-INF\lib 下的所有 jar 包到当前 web 应用的 lib 目录下.



### (2)在 web.xml 文件中配置 struts2:

复制 struts-2.3.31\apps\struts2-blank1\WEB-INF\web.xml 文件中的过滤器的配置到当前 web 应用的 web.xml 文件中

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

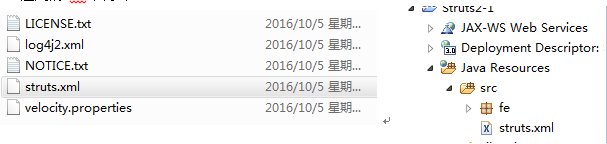
<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

### (3)在当前 web 应用的 classpath 下添加 struts2 的配置文件 struts.xml:

复制 struts-2.3.31\apps\struts2-blank\WEB-INF\classes 下的 struts.xml 文件到当前 web 应用的 src 目录下.



struts.xml:

# 二、Struts 2配置文件

## 1. Struts 2配置文件类型



用户只管web.xml和struts.xml

struts.properties参数配置文件：



## 2. web.xml系统配置文件：

<!-- 配置核心过滤器 -->

<!-- 定义Struts2的FilterDispathcer的Filter -->

<filter>

<filter-name>strutslesson</filter-name>

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

</filter-class>

</filter>

<!-- FilterDispatcher用来初始化struts2并且处理所有的WEB请求 -->

<filter-mapping>

<filter-name>strutslesson</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 配置Spring所需的监听器 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<!-- 配置欢迎页面 -->

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<!-- 配置Struts标签库或者其他标签库（Servlet2.3以下版本） -->

<jsp-config>

<taglib>

<!-- 配置标签库的URL -->

<taglib-uri><http://jsptags.com/tags/navigation/pager>

</taglib-uri>

<!-- 指定标签库定义文件的物理位置 -->

<taglib-location>/WEB-INF/jsp/pager-taglib.tld

</taglib-location>

</taglib>

</jsp-config>

## 3. struts.xml

### (1)功能

负责配置业务控制器Action

定义Action的处理结果和视图之间的映射关系

负责配置拦截器

<struts>

<!—1、包含的其他配置文件 -->

<!—include file=" " -->

<!– 2、定义struts常量-->

<constant name=*"常量名称 "* value=*"常量值 "* />

<!—3、定义bean -->

<bean name=*"bean名称"* class=*""* scope=*" "* static= *" "* option=*" "*/>

<!—4、定义包，package元素是Struts配置文件的核心 -->

<package name=*"struts-default"* extends=*""* namespace=*"名称空间"* abstract=*""* externalReferenceResolver>

<!—5、定义返回值 -->

<result-types>

<result-type name=*""* class=*""* default=*"true|false"*>

<param name=*"参数名"* >参数值</param>

</result-type>

</result-types>

<!—6、定义Action -->

<action name=*"用户请求名"* class=*"Action实现类"* method=*"调用Action类的方法"* >

<result name=*"Action定义的结果名"*>物理视图地址 </result>

</action>

<!—7、定义拦截器 -->

<interceptors>

<interceptor name=*""* class=*""*/>

<interceptor-stack name=*""*>

<interceptor-ref name=*""*/>

</interceptor-stack>

<default-interceptor-ref name=*"defaultStack"*/>

</interceptors>

</package>

</struts>

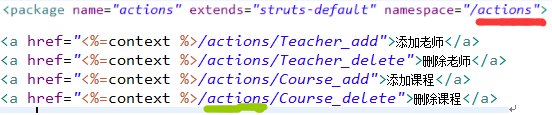
### (2)配置 package 元素



**1> namespace**

默认为""，可以接收所有路径的action<br />

namespace可以写为/，或者/xxx，或者/xxx/yyy，对应的action访问路径为/index.action，/xxx/index.action，或者/xxx/yyy/index.action.



### (3)配置 package -action 元素



### (4)模块包含—include

<include file=*"login.xml"* />

# 三、Action及相关

## 1.action、Action类和对象

### (1) action:

Struts.xml中

应用程序可以完成的每一个操作. 例如: 显示一个登陆表单; 把产品信息保存起来

### (2)Action类:

**1>Action类的规则:**

1. 属性的名字必须遵守与 JavaBeans 属性名相同的命名规则. 属性的类型可以是任意类型. 从字符串到非字符串(基本数据库类型)之间的数据转换可以自动发生
2. 必须有一个不带参的构造器
3. 至少有一个供 struts 在执行这个 action 时调用的方法
4. 同一个 Action 类可以包含多个 action 方法.
5. Struts2 会为每一个 HTTP 请求创建一个新的 Action 实例

**2>具体视图的返回**

可以由用户自己定义的Action来决定,具体的手段是根据返回的字符串找到对应的配置项，来决定视图的内容。

**3>具体Action的实现**

可以是一个普通的java类，里面有public String execute方法即可,或者实现Action接口,不过最常用的是从ActionSupport继承，好处在于可以直接使用Struts2封装好的方法

### (3) Action类:--Action对象

默认执行execute()方法

**public** **interface** Action {

**public** **abstract** String execute() **throws** Exception;

//执行成功，返回成功界面

**public** **static** **final** String ***SUCCESS*** = "success";

//执行成功，不需要返回成功界面

**public** **static** **final** String ***NONE*** = "none";

//执行失败，eg:信息验证失败

**public** **static** **final** String ***ERROR*** = "error";

//根据输入信息比如页码跳转

**public** **static** **final** String ***INPUT*** = "input";

//Action验证失败重新登录

**public** **static** **final** String ***LOGIN*** = "login";

}

**public** **class** IndexAction1 {

**public** String execute() {

**return** "success";

}

}

### (4) Action类:--ActionSupport对象

默认执行execute()方法

ActionSupport 是默认的 Action 类: 若某个 action 节点没有配置 class 属性, 则为待执行的 Action 类. 而 execute 方法即为要默认执行的 action 方法

<action name=*"testActionSupport"*>

<result>/testActionSupport.jsp</result>

</action>

**public** **class** IndexAction3 **extends** ActionSupport {

@Override

**public** String execute() {

**return** "success";

}

}

## 2.path路径

解决办法非常简单，统一使用绝对路径。

### (1)在jsp中用request.getContextRoot方式来拿到webapp的路径

<% String context = request.getContextPath(); %>

<a href=*"*<%=context %>*/actions/Studentadd"*>添加学生</a>

### (2)使用myeclipse经常用的，指定basePath

<%

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

%>

<base href=*"*<%=basePath%>*"* />

## 3. 动态方法调用DMI—ActionMethod

Action执行的时候并不一定要执行execute方法,可以在配置文件中配置Action的时候用 “method=” 来指定执行哪个方法，也可以在url地址中动态指定（动态方法调用DMI）（推荐）

Jsp:

<a href=*"/user/userAdd"*>添加用户</a>

<a href=*"/user/user!add"*>添加用户</a>

前者会产生太多的action，所以不推荐使用

struts.xml:

<constant name=*"struts.devMode"* value=*"true"* />

<package name=*"user"* extends=*"struts-default"* namespace=*"/user"*>

<action name=*"user"* class=*"struts2.user.action.UserAction"* method=*"add"*>

<result>/user\_add\_success.jsp</result>

</action>

<action name=*"user"* class=*"struts2.user.action.UserAction"*>

<result>/user\_add\_success.jsp</result>

</action>

</package>

## 4.通配符映射

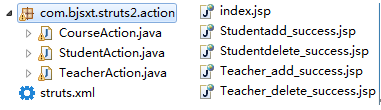
### (1).通配符映射规则

**被通配符匹配到的 URI 字符串的子串可以用 {1}, {2} 来引用**. {1} 匹配第一个子串, {2} 匹配第二个子串…**{0} 匹配整个 URI**

若找到多个匹配, **没有通配符的那个将胜出；**若 Struts 找到的带有通配符的匹配不止一个, **则按先后顺序进行匹配**

\* 可以匹配零个或多个字符, 但不包括 / 字符. 如果想把 / 字符包括在内, 需要使用 \*\*. 如果需要对某个字符进行转义, 需要使用 \.

### (2)示例代码



Jsp:

使用通配符，将配置量降到最低<br />

<a href=*"*<%=context %>*/actions/Studentadd"*>添加学生</a>

<a href=*"*<%=context %>*/actions/Studentdelete"*>删除学生</a>

<br />

不过，一定要遵守"约定优于配置"的原则

<br />

<a href=*"*<%=context %>*/actions/Teacher\_add"*>添加老师</a>

<a href=*"*<%=context %>*/actions/Teacher\_delete"*>删除老师</a>

<a href=*"*<%=context %>*/actions/Course\_add"*>添加课程</a>

<a href=*"*<%=context %>*/actions/Course\_delete"*>删除课程</a>

Struts.xml

核心：要通配符指明方法

<struts>

<constant name=*"struts.devMode"* value=*"true"* />

<!-- 带\*为同级，如果有多个action匹配上了，调用最前面那个 -->

<package name=*"actions"* extends=*"struts-default"* namespace=*"/actions"*>

<action name=*"Student\*"* class=*"com.bjsxt.struts2.action.StudentAction"* method=*"{1}"*>

<result>/Student{1}\_success.jsp</result>

</action>

<action name=*"\*\_\*"* class=*"com.bjsxt.struts2.action.{1}Action"* method=*"{2}"*>

<result>/{1}\_{2}\_success.jsp</result>

<!-- {0}\_success.jsp -->

</action>

</package>

</struts>

Java:

**public** **class** StudentAction **extends** ActionSupport {

**public** String add() {

**return** *SUCCESS*;

}

**public** String delete() {

**return** *SUCCESS*;

}

}

## 5.接受参数

### (1) 使用action属性接收参数

<a href=*"user/user!add?name=a&age=8"*>添加用户</a>

**public** **class** UserAction **extends** ActionSupport {

**private** String name;

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

**public** String add() {

System.*out*.println("name=" + name);

System.*out*.println("age=" + age);

**return** *SUCCESS*;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

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

**return** age;

}

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

**this**.age = age;

}

}

### (2) 使用Domain Model接收参数

**public** **class** UserAction **extends** ActionSupport {

**private** User user;

//private UserDTO userDTO;

**public** String add() {

System.*out*.println("name=" + user.getName());

System.*out*.println("age=" + user.getAge());

**return** *SUCCESS*;

}

**public** User getUser() {

**return** user;

}

**public** **void** setUser(User user) {

**this**.user = user;

}

}

### (3) 使用ModelDriven接收参数

**public** **class** UserAction **extends** ActionSupport **implements** ModelDriven<User>{

**private** User user = **new** User();

**public** String add() {

System.*out*.println("name=" + user.getName());

System.*out*.println("age=" + user.getAge());

**return** *SUCCESS*;

}

@Override

**public** User getModel() {

**return** user;

}

}

## 6.简单的数据验证

<a href=*"user/user!add?name=admin"* >添加用户</a>

**public** String add() {

**if**(name == **null** || !name.equals("admin")) {

**this**.addFieldError("name", "name is error");

**this**.addFieldError("name", "name is too long");

**return** *ERROR*;

}

**return** *SUCCESS*;

}

Struts.xml

<package name=*"user"* extends=*"struts-default"* namespace=*"/user"*>

<action name=*"user"* class=*"com.bjsxt.struts2.user.action.UserAction"*>

<result>/user\_add\_success.jsp</result>

<result name=*"error"*>/user\_add\_error.jsp</result>

</action>

</package>

## 7. 在 Action 中访问 WEB 资源:

### (一) 和 Servlet API 解耦的方式:

只能访问有限的 Servlet API 对象, 且只能访问其有限的方法(读取请求参数, 读写域对象的属性, 使 session 失效...).

**(1) ActionContext类**

Action 执行的上下文对象, 在 ActionContext 中保存了 Action 执行所需要的**所有对象**, 包括 parameters, request, session, application 等.

获取 ActionContext 对象，对应的 Map 对象, 并向其中添加一个属性，设置属性，获取属性

**public** **class** LoginAction1 **extends** ActionSupport {

**private** Map request;

**private** Map session;

**private** Map application;

**public** LoginAction1() {

request = (Map)ActionContext.*getContext*().get("request");

session = ActionContext.*getContext*().getSession();

application = ActionContext.*getContext*().getApplication();

}

**public** String execute() {

request.put("r1", "r1");

session.put("s1", "s1");

application.put("a1", "a1");

**return** *SUCCESS*;

}

}

**(2) XxxAware 接口—只用这种**

通过可以实现某些特定的接口, 让 Struts2 框架在运行时向 Action 实例注入 parameters, request, session 和 application 对应的 Map 对象:

**public** **class** LoginAction2 **extends** ActionSupport **implements** RequestAware,SessionAware, ApplicationAware {

**private** Map<String, Object> request;

**private** Map<String, Object> session;

**private** Map<String, Object> application;

//DI dependency injection

//IoC inverse of control

**public** String execute() {

request.put("r1", "r1");

session.put("s1", "s1");

application.put("a1", "a1");

**return** *SUCCESS*;

}

@Override

**public** **void** setRequest(Map<String, Object> request) {

**this**.request = request;

}

@Override

**public** **void** setSession(Map<String, Object> session) {

**this**.session = session;

}

@Override

**public** **void** setApplication(Map<String, Object> application) {

**this**.application = application;

}

}

### (二) 和 Servlet API 解耦的方式:

使用方法类似上面

可以访问更多的 Servlet API 对象, 且可以调用其原生的方法.

**(1)使用 ServletActionContext类**

ServletActionContext, ServletActionContext, ServletActionContext

**public** **class** LoginAction3 **extends** ActionSupport {

**private** HttpServletRequest request;

**private** HttpSession session;

**private** ServletContext application;

**public** LoginAction3() {

request = ServletActionContext.getRequest();

session = request.getSession();

application = session.getServletContext();

}

**public** String execute() {

request.setAttribute("r1", "r1");

session.setAttribute("s1", "s1");

application.setAttribute("a1", "a1");

**return** SUCCESS;

}

}

**(2)实现 ServletXxxAware 接口.**

ServletRequestAware , ServletContextAware , ServletResponseAware

**public** **class** LoginAction4 **extends** ActionSupport **implements** ServletRequestAware {

**private** HttpServletRequest request;

**private** HttpSession session;

**private** ServletContext application;

**public** String execute() {

request.setAttribute("r1", "r1");

session.setAttribute("s1", "s1");

application.setAttribute("a1", "a1");

**return** *SUCCESS*;

}

@Override

**public** **void** setServletRequest(HttpServletRequest request) {

**this**.request = request;

**this**.session = request.getSession();

**this**.application = session.getServletContext();

}

}

## 8.默认Action

<package name=*"default"* namespace=*"/"* extends=*"struts-default"*>

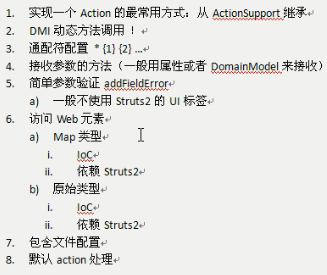
<default-action-ref name=*"index"*></default-action-ref>

<action name=*"index"*>

<result>/default.jsp</result>

</action>

</package>



# 四、结果集-- result元素

## 1.result



**根据参数转发不同页面**

<action name=*"tes"* class=*"com.atguigu.struts2.action.TestResultAction"*>

<result name=*"success"* type=*"dispatcher"*>/success.jsp</result>

<result name=*"login"* type=*"redirect"*>/login.jsp</result>

<result name=*"index"* type=*"redirectAction"*>

<param name=*"actionName"*>testAction</param>

<param name=*"namespace"*>/atguigu</param>

</result>

<result name=*"test"* type=*"chain"*>

<param name=*"actionName"*>testAction</param>

<param name=*"namespace"*>/atguigu</param>

</result>

</action>

<package name=*"resultTypes"* namespace=*"/r"* extends=*"struts-default"*>

<action name=*"r1"*>

<result type=*"dispatcher"*>/r1.jsp</result>

</action>

<action name=*"r2"*>

<result type=*"redirect"*>/r2.jsp</result>

</action>

<action name=*"r3"*>

<result type=*"chain"*>r1</result>

</action>

<action name=*"r4"*>

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

</action>

</package>

## 2. global-results全局结果集

全局结果集,定义了一个大家都可以使用的结果集。这里面的user和admin都可以共用global-results中的结果集。

<package name=*"user"* namespace=*"/user"* extends=*"struts-default"*>

<global-results>

<result name=*"mainpage"*>/main.jsp</result>

</global-results>

<action name=*"index"*>

<result>/index.jsp</result>

</action>

<action name=*"user"* class=*"com.bjsxt.struts2.user.action.UserAction"*>

<result>/user\_success.jsp</result>

<result name=*"error"*>/user\_error.jsp</result>

</action>

</package>

<package name=*"admin"* namespace=*"/admin"* extends=*"user"*>

<action name=*"admin"* class=*"com.bjsxt.struts2.user.action.AdminAction"*>

<result>/admin.jsp</result>

</action>

</package>

## 3. DynamicResult动态结果集

<package name=*"user"* namespace=*"/user"* extends=*"struts-default"*>

<action name=*"user"* class=*"com.bjsxt.struts2.user.action.UserAction"*>

<result>${r}</result>

</action>

</package>

@Override

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

**if**(type == 1) r="/user\_success.jsp";

**else** **if** (type == 2) r="/user\_error.jsp";

**return** "success";

}

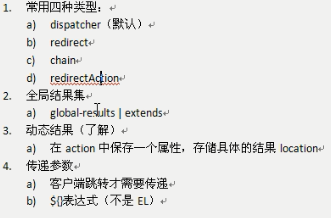
## 4.带参数的结果集

<action name=*"user"* class=*"com.bjsxt.struts2.user.action.UserAction"*>

<result type=*"redirect"*>/user\_success.jsp?t=${type}</result>

</action>

<li><a href=*"user/user?type=1"*>传参数</a></li>



# 五、OGNL

可以任意存取对象的属性或者调用对象的方法，能够遍历整个对象的结构图，实现对象属性类型的转换等功能。它使用相同的[表达式](http://baike.baidu.com/view/420676.htm)去存取对象的属性。

## 1. 值 栈

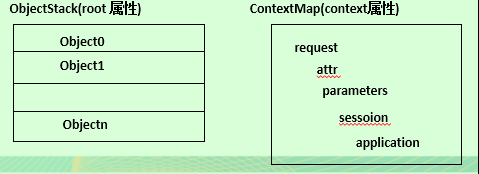
<s:debug></s:debug>

在 ValueStack 对象的内部有两个逻辑部分:

ObjectStack：Struts 把 Action 和相关对象压入 ObjectStack 中；

ContextMap: Struts 把各种各样的映射关系(一些 Map 类型的对象) 压入 ContextMap 中. 实际上就是对 ActionContext 的一个引用；Struts 会把下面这些映射压入 ContextMap 中





## 2.读取 ObjectStack 里的对象的属性

若想访问 Object Stack 里的某个对象的属性. 可以使用以下几种形式之一:

ObjectStack 里的对象可以通过一个从零开始的下标来引用. ObjectStack 里的栈顶对象可以用 [0] 来引用, 它下面的那个对象可以用 [1] 引用. 若希望返回栈顶对象的 message 属性值: [0].message 或 [0][“message”] 或 [0][‘message’]

若在指定的对象里没有找到指定的属性, 则到指定对象的下一个对象里继续搜索. 即 [n] 的含义是从第 n 个开始搜索, 而不是只搜索第 n 个对象

若从栈顶对象开始搜索, 则可以省略下标部分

ProductName: ^<s:property value=*"[0].productName"*/>

## 3.读取 Context Map 里的对象的属性

若想访问 ContextMap 里的某个对象的属性, 可以使用以下几种形式之一:

返回 session 中的 code 属性: #session.code

返回 request 中的 customer 属性的 name 属性值: #request.customer.name

返回域对象(按 request, session, application 的顺序)的 lastAccessDate 属性: #attr.lastAccessDate

## 4.调用字段和方法

可以利用 OGNL 调用

**任何一个 Java 类里的静态字段或方法**.

**被压入到 ValueStack 栈的对象上的公共字段和方法**.

默认情况下, Struts2 不允许调用任意 Java 类静态方法, 需要重新设置 **struts.ognl.allowStaticMethodAccess** 标记变量的值为 true.

调用静态字段或方法需要使用如下所示的语法:

**@fullyQualifiedClassName@fieldName**: @java.util.Calendar@DECEMBER

**@fullyQualifiedClassName@methodName(argumentList):** @app4.Util@now()

调用一个实例字段或方法的语法, 其中 object 是 Object Stack 栈里的某个对象的引用:

**.object.fieldName**: [0].datePattern

**.object.methodName(argumentList)**: [0].repeat(3, “Hello”);

<li>访问值栈中的action的普通属性: username = <s:property value=*"username"*/> </li>

<li>访问值栈中对象的普通属性(get set方法)：<s:property value=*"user.age"*/> | <s:property value=*"user['age']"*/> | <s:property value=*"user[\"age\"]"*/> | wrong: <%--<s:property value="user[age]"/>--%></li>

<li>访问值栈中对象的普通属性(get set方法): <s:property value=*"cat.friend.name"*/></li>

<li>访问值栈中对象的普通方法：<s:property value=*"password.length()"*/></li>

<li>访问值栈中对象的普通方法：<s:property value=*"cat.miaomiao()"* /></li>

<li>访问值栈中action的普通方法：<s:property value=*"m()"* /></li>

<hr />

<li>访问静态方法：<s:property value=*"@com.bjsxt.struts2.ognl.S@s()"*/></li>

<li>访问静态属性：<s:property value=*"@com.bjsxt.struts2.ognl.S@STR"*/></li>

<li>访问Math类的静态方法：<s:property value=*"@@max(2,3)"* /></li>

## 5. 访问数组类型的属性

有些属性将返回一个对象数组而不是单个对象, 可以像读取任何其他对象属性那样读取它们. 这种**数组型属性的各个元素以逗号分隔, 并且不带方括号**

可以使用**下标**访问数组中指定的元素: colors[0]

可以通过调用其 length 字段查出给定数组中有多少个元素: **colors.length**

## 6. 访问 List ，Set类型的属性

有些属性将返回的类型是 java.util.List, 可以像读取任何其他属性那样读取它们. 这种 List 的各个元素是字符串, **以逗号分隔, 并且带方括号**

可以使用下标访问 List 中指定的元素: colors[0]

可以通过调用其 size 方法或专用关键字 size 的方法查出给定List 的长度: **colors.size** 或 **colors.size()**

可以通过使用 isEmpty() 方法或专用关键字 isEmpty 来得知给定的 List 是不是空. **colors.isEmpty** 或 **colors.isEmpty()**

还可以使用 OGNL 表达式来**创建 List**, 创建一个 List 与声明一个 Java 数组是相同的: **{“Red”, “Black”, “Green”}**

<li>访问List:<s:property value=*"users"*/></li>

<li>访问List中某个元素:<s:property value=*"users[1]"*/></li>

<li>访问List中元素某个属性的集合:<s:property value=*"users.{age}"*/></li>

<li>访问List中元素某个属性的集合中的特定值:<s:property value=*"users.{age}[0]"*/> | <s:property value=*"users[0].age"*/></li>

<li>访问Set:<s:property value=*"dogs"*/></li>

<li>访问Set中某个元素:<s:property value=*"dogs[1]"*/></li>

## 7. 访问 Map 类型的属性

读取一个 Map 类型的属性将以如下所示的格式返回它所有的键值对:

若希望检索出某个 Map 的值, 需要使用如下格式: map[key]

可以使用 size 或 size() 得出某个给定的 Map 的键值对的个数

可以使用 isEmpty 或 isEmpty() 检查某给定 Map 是不是空.

<li>访问Map:<s:property value=*"dogMap"*/></li>

<li>访问Map中某个元素:<s:property value=*"dogMap.dog101"*/> | <s:property value=*"dogMap['dog101']"*/> | <s:property value=*"dogMap[\"dog101\"]"*/></li>

<li>访问Map中所有的key:<s:property value=*"dogMap.keys"*/></li>

<li>访问Map中所有的value:<s:property value=*"dogMap.values"*/></li>

## 8.投影（过滤）

<li>投影(过滤)：<s:property value=*"users.{?#this.age==1}[0]"*/></li>

<li>投影：<s:property value=*"users.{^#this.age>1}.{age}"*/></li>

<li>投影：<s:property value=*"users.{$#this.age>1}.{age}"*/></li>

<li>投影：<s:property value=*"users.{$#this.age>1}.{age} == null"*/></li>

## 9.[]

<li>[]:<s:property value=*"[0].username"*/></li>

## 10. 使用 EL 访问值栈中对象的属性

<s:property value=“fieldName”> 也可以通过 JSP EL 来达到目的: ${fieldName}

原理:

包装 HttpServletRequest 对象后的 org.apache.struts2.dispatcher.StrutsRequestWrapper 对象传到页面上, 而这个类重写了 getAttribute() 方法.



# 六、标签

导入标签库：<%@ taglib prefix=*"s"* uri=*"/struts-tags"* %>

参看struts-2.3.31\docs\docs\index.jsp



访问属性

<a href=*"*<%=contextPath %>*/tags.action?username=u&password=p"*>tags</a>

## (一)通用标签

### 1. \*property 标签

property 标签用来输出一个值栈属性的值，**在许多情况下, JSP EL 可以提供更简洁的语法**



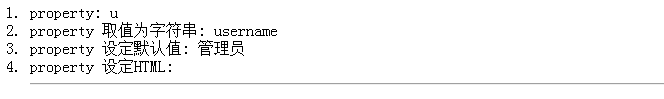
示例:

property: <s:property value=*"username"*/>

property 取值为字符串: <s:property value=*"'username'"*/>

property 设定默认值: <s:property value=*"admin"* default=*"管理员"*/>

property 设定HTML: <s:property value=*"'*<hr/>*'"* escape=*"false"*/>



### 2. \*set 标签



set 设定adminName值（默认为request 和 ActionContext）: <s:set var=*"adminName"* value=*"username"* />

set 从request取值: <s:property value=*"#request.adminName"* />

set 从ActionContext取值: <s:property value=*"#adminName"* />

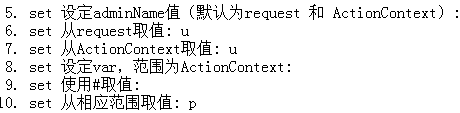
<%--set 设定范围: <s:set name="adminPassword" value="password" scope="page"/>

set 从相应范围取值: <%=pageContext.getAttribute("adminPassword") %>--%>

set 设定var，范围为ActionContext: <s:set var=*"adminPassword"* value=*"password"* scope=*"session"*/>

set 使用#取值: <s:property value=*"#adminPassword"*/>

set 从相应范围取值: <s:property value=*"#session.adminPassword"*/>



### 3. \*param 标签

param 标签用来把一个参数传递给包含着它的那个标签



无论在给出 value 值时有没有使用 %{}, Struts 都会对它进行 ognl 求值

如果想传递一个 String 类型的字符串作为参数值, 必须把它用单引号括起来.

可以把 value 属性的值写在开始标签和结束标签之间. 利用这种方式来传递一个 EL 表达式的值

### 4. \*bean 标签

bean 标签将创建一个 JavaBean, 并把它压入 ValueStack 值栈的 ContextMap 子栈. 这个标签的功能与 JSP 中的 useBean 动作元素很相似



bean 定义bean,并使用param来设定新的属性值:

<s:bean name=*"com.bjsxt.struts2.tags.Dog"* >

<s:param name=*"name"* value=*"'pp'"*></s:param>

<s:property value=*"name"*/>

</s:bean>

bean 查看debug情况:

<s:bean name=*"com.bjsxt.struts2.tags.Dog"* var=*"myDog"*>

<s:param name=*"name"* value=*"'oudy'"*></s:param>

</s:bean>

拿出值：<s:property value=*"#myDog.name"*/>



### 5. \*include 标签—了解

include 标签用来把一个 Servlet 或 JSP 页面的输出包含到当前页面里来.



include \_include1.html 包含静态英文文件

<s:include value=*"/\_include1.html"*></s:include>

include \_include2.html 包含静态中文文件

<s:include value=*"/\_include2.html"*></s:include>

include \_include1.html 包含静态英文文件，说明%用法

<s:set var=*"incPage"* value=*"%{'/\_include1.html'}"* />

<s:include value=*"%{#incPage}"*></s:include>

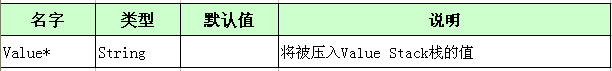


### 6. \*debug标签

### 7. \*url 标签



### 8. \*push 标签



### 8.i18n标签

### 9.text标签

### 10. \*date 标签

* date 标签用来对 Date 对象进行排版



* format 属性的值必须是 java.text.SimpleDateFormat 类里定义的日期/时间格式之一.

### 11. \*a 标签

### 12.action 标签

* action 标签用在页面上来执行一个 action.
* action 标签还会把当前 Action 对象压入 ValueStack 值栈的 ContextMap 子栈.



## (二)控制标签

### 1. \*if, else 和 elseif 标签



if elseif else:

age = <s:property value=*"#parameters.age[0]"* />

<s:set var=*"age"* value=*"#parameters.age[0]"* />

<s:if test=*"#age* < *0"*>wrong age!</s:if>

<s:elseif test=*"#parameters.age[0]* < *20"*>too young!</s:elseif>

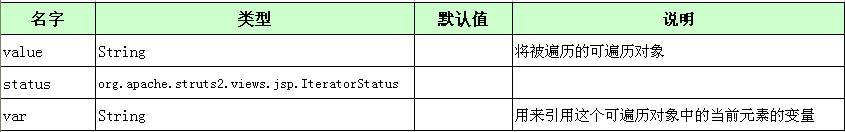
<s:else>yeah!</s:else>

<s:if test=*"#parameters.aaa == null"*>null</s:if>

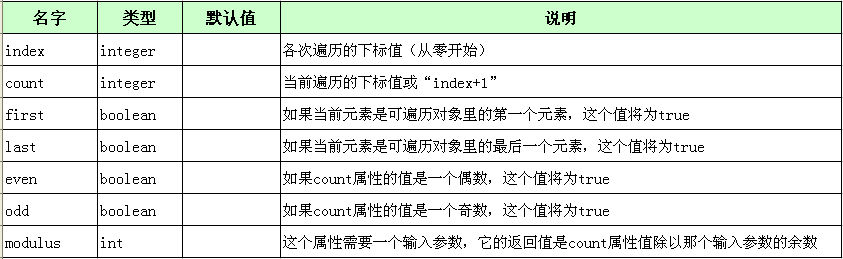


### 2. \*iterator 标签

iterator 标签用来遍历一个数组, Collection 或一个 Map, **并把这个可遍历对象里的每一个元素依次压入和弹出 ValueStack 栈**

****

iterator 标签的 status 属性的属性值



遍历集合：

<s:iterator value=*"{1, 2, 3}"* >

<s:property/> |

</s:iterator>

自定义变量：

<s:iterator value=*"{'aaa', 'bbb', 'ccc'}"* var=*"x"*>

<s:property value=*"#x.toUpperCase()"*/> |

</s:iterator>

使用status:

<s:iterator value=*"{'aaa', 'bbb', 'ccc'}"* status=*"status"*>

<s:property/> |

遍历过的元素总数：<s:property value=*"#status.count"*/> |

遍历过的元素索引：<s:property value=*"#status.index"*/> |

当前是偶数？：<s:property value=*"#status.even"*/> |

当前是奇数？：<s:property value=*"#status.odd"*/> |

‘

S4’是第一个元素吗？：<s:property value=*"#status.first"*/> |

是最后一个元素吗？：<s:property value=*"#status.last"*/>

<br />

</s:iterator>

</li>

<li>

<s:iterator value=*"*#{1:'a', 2:'b', 3:'c'}*"* >

<s:property value=*"key"*/> | <s:property value=*"value"*/> <br />

</s:iterator>

</li>

<li>

<s:iterator value=*"*#{1:'a', 2:'b', 3:'c'}*"* var=*"x"*>

<s:property value=*"#x.key"*/> | <s:property value=*"#x.value"*/> <br />

</s:iterator>

</li>

<li>

<s:fielderror fieldName=*"fielderror.test"* theme=*"simple"*></s:fielderror>

</li>

在开始执行时, iterator 标签会先把 IteratorStatus 类的一个实例压入 ContextMap, 并在每次遍历循环时**更新它**. 可以将一个指向 IteratorStatus 对象的变量赋给 status 属性.

### 3. subset 标签

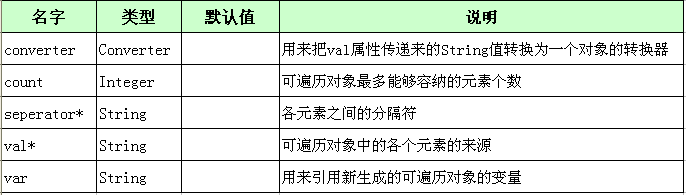
* subset 标签用来创建一个可遍历集合的子集.
* subset 标签通过 decider 属性来创建一个可遍历集合的子集.

### 4. append, merge 标签

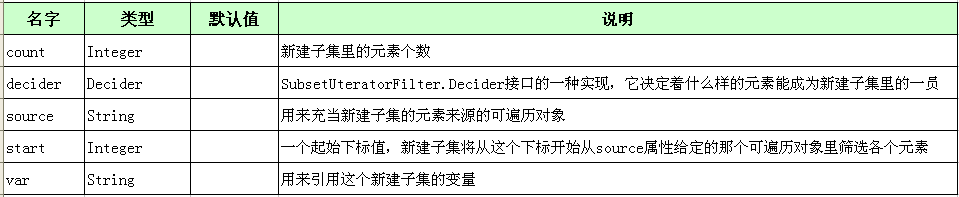
* append 标签用来合并可遍历对象.
* merge 标签用来交替合并可遍历对象.

### 5. generator 标签

* generator 标签用来生成一个可遍历对象并把它压入 ValueStack 栈.
* generator 标签结束标记将弹出遍历对象



* 如果在一个 generator 标签里给出了 converter 属性, 新生成的可遍历对象里的每一个元素都会传递到该属性所指定的方法进行必要的转换.



### 6. \*sort 标签



## (三)UI标签

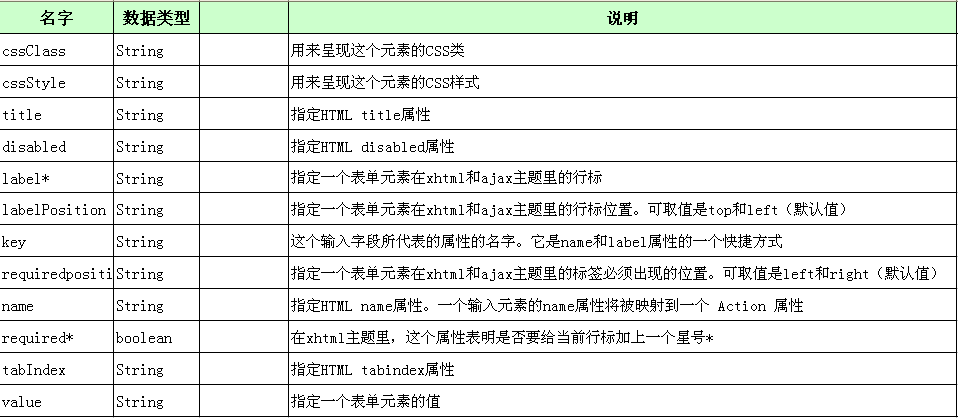
## (四)ajax标签

# 七、表单标签

## 1. 概述

* 表单标签将在 HTML 文档里被呈现为一个表单元素
* 使用表单标签的优点:
  + **表单回显**
  + 对页面进行布局和排版
* 标签的属性可以被赋值为一个静态的值或**一个 OGNL 表达式**. 如果在赋值时使用了一个 OGNL 表达式并把它用 **%{}** 括起来, 这个表达式将会被求值.

## 2. 表单标签的共同属性



* \* 该属性只在没有使用 simple 主题时才可以使用.

## 3. form 标签

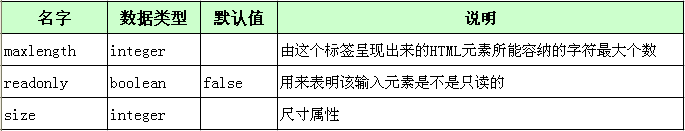
* form 标签用来呈现 HTML 语言中的表单元素



* 默认情况下, form 标签将被呈现为一个表格形式的 HTML 表单. 嵌套在 form 标签里的输入字段将被呈现为一个表格行. 每个表格行由两个字段组成, 一个对应着行标, 一个对应着输入元素. 提交按钮将被呈现为一个横跨两列单元格的行

## 4. textfield, password, hidden 标签

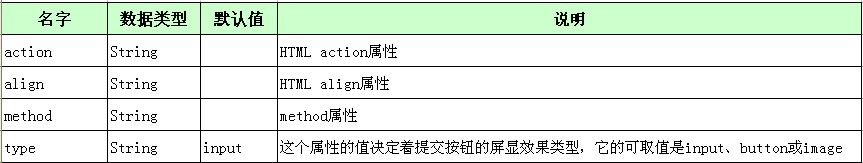
* textfield 标签将被呈现为一个输入文本字段, password 标签将被呈现为一个口令字段, hidden 标签将被呈现为一个不可见字段.



* password 标签扩展自 textfield 标签, 多了一个 showPassword 属性. 该属性时布尔型. 默认值为 false, 它决定着在表单回显时是否显示输入的密码.

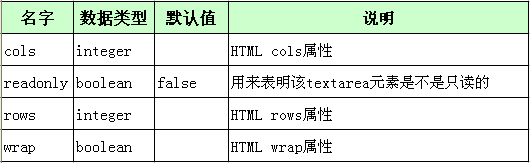
## 5. submit 标签

* submit 标签将呈现为一个提交按钮. 根据其 type 属性的值. 这个标签可以提供 3 种呈现效果:
  + input: <input type=“submim” …/>
  + button: <input type=“button” …/>
  + image: <input type=“image” />



## 6. textarea 标签

* textarea 标签将呈现为一个 HTML 文本域元素

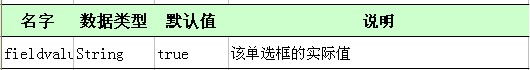


## 7. \*checkbox 标签

* checkbox 标签将呈现为**一个** HTML 复选框元素. 该复选框元素通常用于提交一个布尔值
* 当包含着一个复选框的表单被提交时, 如果某个复选框被选中了, 它的值将为 **true**, 这个复选框在 HTTP 请求里增加一个请求参数.但**如果该复选框未被选中, 在请求中就不会增加一个请求**参数.

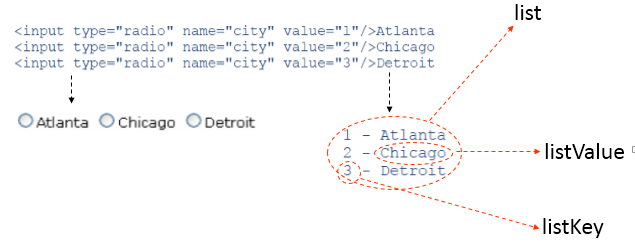
checkbox 标签解决了这个局限性, 它采取的办法是为单个复选框元素创建一个配对的不可见字段

* checkbox 标签有一个 **fieldValue** 属性, 该属性指定的值将在用户提交表单时作为被选中的单选框的实际值发送到服务器. 如果没有使用 fieldValue 属性, 单选框的值将为 true 或 false.



## 8. list, listKey 和 listValue 属性

* list, listKey, listValue 这 3 个属性对 **radio, select, checkboxlist** 等标签非常重要



* 可以把一个 String, 一个数组, 一个 Enumeration, Iterator, Map 或 Collection 赋给 list 属性.

## 9. radio 标签

* radio 标签将呈现为一组单选按钮, 单选按钮的个数与程序员通过该标签的 list 属性提供的选项的个数相同.
* 一般地, **使用 radio 标签实现 “多选一”, 对于 “真/假” 则该使用 checkbox 标签.**



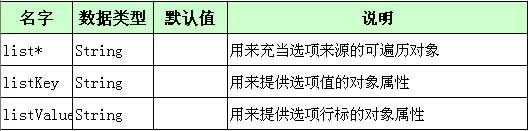
## 10. select 标签

* select 标签将呈现一个 select 元素.



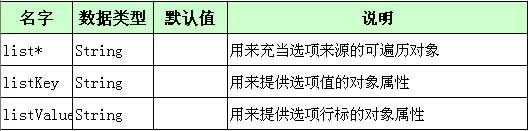
## 11. optiongroup 标签

* optiongroup 标签对 select 元素所提供的选项进行分组. 每个选项有它自己的来源.



## 12. checkboxlist 标签

* checkboxlist 标签将呈现一组多选框.



* **checkbox 标签被映射到一个字符串数组或是一个基本类型的数组. 若它提供的多选框一个也没有被选中, 相应的属性将被赋值为一个空数组而不是空值**.

## 13、主题

* 主题: 为了让所有的 UI 标签能够产生同样的视觉效果而归集到一起的一组模板. 即**风格相近的模板被打包为一个主题**
  + **simple**: 把 UI 标签翻译成最简单的 HTML 对应元素, 而且会忽视行标属性
  + **xhtml**: xhtml 是**默认的主题**. 这个主题的模板通过使用一个布局表格提供了一种自动化的排版机制.
  + **css**\_xhtml: 这个主题里的模板与 xhtml 主题里的模板很相似, 但它们将使用 css 来进行布局和排版
  + ajax: 这个主题里的模板以 xhtml 主题里德模板为基础, 但增加了一些 Ajax 功能.
* 修改主题:
  + 通过 UI 标签的 theme 属性
  + 在一个表单里, 若没有给出某个 UI 标签的 theme 属性, 它将使用这个表单的主题
  + 在 page, request, session 或 application 中添加一个 theme 属性
  + 修改 struts.properties 文件中的 struts.ui.theme 属性.

## 9. 异常处理: exception-mapping 元素

exception-mapping 元素: 配置当前 action 的**声明式异常处理**

exception-mapping 元素中有 2 个属性

exception: 指定需要捕获的的异常类型。异常的全类名

result: 指定一个响应结果, 该结果将在捕获到指定异常时被执行, 既可以来自当前 action 的声明, 也可以来自 **global-results** 声明.

可以通过 **global-exception-mappings** 元素为应用程序提供一个全局性的异常捕获映射. 但在 global-exception-mappings 元素下声明的任何 exception-mapping 元素只能引用在 **global-results** 元素下声明的某个 result 元素

声明式异常处理机制由 ExceptionMappingInterceptor 拦截器负责处理, 当某个 exception-mapping 元素声明的异常被捕获到时, ExceptionMappingInterceptor 拦截器就会向 ValueStack 中添加**两个对象**:

**exception**: 表示被捕获异常的 Exception 对象

**exceptionStack**: 包含着被捕获异常的栈

可以在视图上通过 <s:property> 标签显示异常消息

# 七、增删查改

## 1.Params 拦截器

* **Parameters 拦截器将把表单字段映射到 ValueStack 栈的栈顶对象的各个属性中**. 如果某个字段在模型里没有匹配的属性, Param 拦截器将尝试 ValueStack 栈中的下一个对象

## 2.ModelDriven 拦截器

* 当用户触发 add 请求时, ModelDriven 拦截器将调用 EmployeeAction 对象的 getModel() 方法, 并把返回的模型(Employee实例)压入到 ValueStack 栈.
* 接下来 Parameters 拦截器将把表单字段映射到 ValueStack 栈的栈顶对象的各个属性中. **因为此时 ValueStack 栈的栈顶元素是刚被压入的模型(Employee)对象, 所以该模型将被填充**. 如果某个字段在模型里没有匹配的属性, Param 拦截器将尝试 ValueStack 栈中的下一个对象

## 3. Preparable 拦截器

### (1)

* Struts 2.0 中的 modelDriven 拦截器负责把 Action 类以外的一个对象压入到值栈栈顶
* 而 prepare 拦截器负责准备为 getModel() 方法准备 model

### (2)PrepareInterceptor拦截器用方法

* 若 Action 实现 Preparable 接口，则 Action 方法需实现 prepare() 方法
* PrepareInterceptor 拦截器将调用 prepare() 方法，**prepareActionMethodName**()方法 或 **prepareDoActionMethodName** ()方法
* PrepareInterceptor 拦截器根据 **firstCallPrepareDo** 属性决定获取 prepareActionMethodName 、prepareDoActionMethodName的顺序。默认情况下先获取 prepareActionMethodName (), 如果没有该方法，就寻找prepare*Do*ActionMethodName()。如果找到对应的方法就调用该方法
* PrepareInterceptor 拦截器会根据 **alwaysInvokePrepare** 属性决定是否执行prepare()方法

### (3) 使用 paramsPrepareParamsStack 拦截器栈

* paramsPrepareParamsStack 从字面上理解来说， 这个stack的拦截器调用的顺序为：首先 params，然后 prepare，接下来 modelDriven，最后再 params
* Struts 2.0的设计上要求 modelDriven 在 params 之前调用，而业务中prepare要负责准备model，准备model又需要参数，这就需要在 prepare之前运行params拦截器设置相关参数，这个也就是创建paramsPrepareParamsStack的原因。
* 流程如下：
  + 1. params拦截器首先给action中的相关参数赋值，如id
  + 2. prepare拦截器执行prepare方法，prepare方法中会根据参数，如id，去调用业务逻辑，设置model对象
  + 3. modelDriven拦截器将model对象压入value stack，这里的model对象就是在prepare中创建的
  + 4. params拦截器再将参数赋值给model对象
  + 5. action的业务逻辑执行

# 八、类型转换

## 1. 概述

从一个 HTML 表单到一个 Action 对象, 类型转换是从字符串到非字符串.

HTTP 没有 “类型” 的概念. 每一项表单输入只可能是一个字符串或一个字符串数组. 在服务器端, 必须把 String 转换为特定的数据类型

在 struts2 中, 把请求参数映射到 action 属性的工作由 Parameters 拦截器负责, 它是默认的 defaultStack 拦截器中的一员. Parameters 拦截器可以自动完成字符串和基本数据类型之间转换.

## 2. 类型转换错误

### (1)如果类型转换失败:

* + **若 Action 类没有实现 ValidationAware 接口：** Struts 在遇到类型转换错误时仍会继续调用其 Action 方法, 就好像什么都没发生一样.
  + **若 Action 类实现 ValidationAware 接口：**Struts 在遇到类型转换错误时将不会继续调用其 Action 方法: Struts 将检查相关 action 元素的声明是否包含着一个 name=input 的 result. 如果有, Struts 将把控制权转交给那个 result 元素; 若没有 input 结果, Struts 将抛出一个异常

### (2) 类型转换错误消息的定制

* 作为默认的 default 拦截器的一员, **ConversionError 拦截器**负责添加与类型转换有关的出错消息(前提: **Action 类必须实现了 ValidationAware 接口**)和保存各请求参数的原始值.
* 若字段标签使用的不是 simple 主题, 则非法输入字段将导致一条有着以下格式的出错消息:
* 覆盖默认的出错消息
  + **在对应的 Action 类所在的包中新建 ActionClassName.properties 文件, ClassName 即为包含着输入字段的 Action 类的类名**
  + **在属性文件中添加如下键值对:**
* 定制出错消息的样式:
  + 每一条出错消息都被打包在一个 HTML span 元素里, 可以通过覆盖其行标为 errorMessage 的那个 css 样式来改变出错消息的格式.
* 显示错误消息: 如果是 simple 主题, 可以通过 <s:fielderror fieldName=“filedname”></s:fielderror> 标签显示错误消息

## 3.定制类型转换器

* 自定义类型转换器必须实现 ongl.TypeConverter 接口或对这个接口的某种具体实现做扩展

扩展 StrutsTypeConverter 类

### (1)配置自定义的类型转换器

* 在应用程序里使用一个自定义的类型转换器之前, 必须先对它进行配置. 这种配置**既可以基于字段, 也可以基于类型**
* 基于字段配置: 可以**为某个 Model（该 Model 类也可能是 Action） 的各个属性分别配置一个自定义的转换器**.
  + 1. 创建一个属性文件: ModelClassName-conversion.properties, 该文件需和相对应的 Model 类放在同一个目录下
  + 2. 编辑属性文件:
  + 基于类型配置:
  + 在 WEB-INF/classes/ 目录下创建 xwork-conversion.properties 文件.
  + 在 xwork-conversion.properties 文件里把每一个**需要进行类型转换的类**与一个类型转换器关联起来
* 实现自定义的时间类型转换器: 时间 pattern 需要以 web 应用的初始化参数配置在 web.xml 中
* 若类型转换失败，给出自定义的信息。

## 4. 类型转换与复杂属性配合使用

### (1) form 标签的 name 属性可以被映射到一个属性的属性.

### (2) Struts 还允许填充 Collection 里的对象, 这常见于需要快速录入批量数据的场合

# 九、消息处理与国际化

## 1. 概述

* 在程序设计领域, 把在无需改写源代码即可让开发出来的应用程序能够支持多种语言和数据格式的技术称为国际化.
* 与国际化对应的是本地化, 指让一个具备国际化支持的应用程序支持某个特定的地区
* Struts2 国际化是建立在 Java 国际化基础上的：
  + 为不同国家/语言提供对应的消息资源文件
  + Struts2 框架会根据请求中包含的

Locale 加载对应的资源文件

* + 通过程序代码取得该资源文件中

指定 key 对应的消息

## 2. 配置国际化资源文件

* Action 范围资源文件：在Action类文件所在的路径建立名为**ActionName\_language\_country.properties** 的文件
* 包范围资源文件：在包的根路径下建立文件名为**package\_language\_country.properties** 的属性文件，一旦建立，处于该包下的所有 Action 都可以访问该资源文件。注意：包范围资源文件的 baseName 就是package，不是Action所在的包名。
* **全局资源文件**
  + 命名方式: **basename\_language\_country.properties**
  + struts.xml  
    <constant name=*"struts.i18n.encoding"* value=*"GBK"* /> <!-- 解决中文编码 -->
  + struts.properties  
    struts.custom.i18n.resources=baseName

临时指定资源文件：**<s:i18n.../>** 标签的 name 属性指定临时的国际化资源文件

## 3. 加载资源文件的顺序

* 假设我们在某个 ChildAction 中调用了getText("username")：
  + (1)加载和 ChildAction 的类文件在同一个包下的系列资源文件 **ChildAction.properties**
  + (2)加载 ChildAction 实现的接口 IChild，且和 IChildn 在同一个包下 **IChild.properties** 系列资源文件。
  + (3)加载 ChildAction 父类 Parent，且和 Parent 在同一个包下的 baseName 为 **Parent.properties** 系列资源文件。
  + (4) 若 ChildAction 实现 ModelDriven 接口，则对于getModel()方法返回的model 对象，重新执行第(1)步操作。
  + (5) 查找当前包下 **package.properties** 系列资源文件。
  + (6) 沿着当前包上溯，直到最顶层包来查找 **package.properties** 的系列资源文件。
  + (7) 查找 **struts.custom.i18n.resources** 常量指定 baseName 的系列资源文件。
  + (8) 直接输出该key的字符串值。

## 4. 访问国际化消息

* JSP 页面访问国际化消息：
  + 不带占位符：
    - <s:text name="key"/>
    - 表单元素的 label 属性：**可替换为 key 或使用 getText() 方法，并对其进行强制 OGNL 解析**
  + 带占位符：
    - 在 <s:text.../> 标签中**使用多个 <s:param.../> 标签来填充消息中的占位符**。
    - Struts2 直接在国际化消息资源文件中通过 “**${}**” 使用表达式，该表达式将从**值栈中获取对应的属性值**
* Action 访问国际化消息：
  + 若 Action 类继承了 ActionSupport ，则可调用 TextProvider 接口的 getText 方法。

## 5. 利用超链接实现动态加载国际化资源文件

* Struts2 使用 **i18n 拦截器** 处理国际化，并且将其注册在默认的拦截器中
* i18n拦截器在执行Action方法前，自动查找请求中一个名为request\_locale 的参数。如果该参数存在，拦截器就将其作为参数，转换成Locale对象，并将其设为用户默认的Locale(代表国家/语言环境)。并把其设置为 session 的 WW\_TRANS\_I18N\_LOCALE 属性
* 若 request 没有名为request\_locale 的参数，则 i18n 拦截器会从 Session 中获取 WW\_TRANS\_I18N\_LOCALE 的属性值，若该值不为空，则将该属性值设置为浏览者的默认Locale
* 若 session 中的 WW\_TRANS\_I18N\_LOCALE 的属性值为空，则从 ActionContext 中获取 Locale 对象。

# 十、Struts2 运行流程分析

# 十一、输入验证

## 1. 概述

* 一个健壮的 web 应用程序必须确保用户输入是合法、有效的.
* Struts2 的输入验证
  + **基于 XWork Validation Framework 的声明式验证**：Struts2 提供了一些基于 XWork Validation Framework 的内建验证程序. 使用这些验证程序不需要编程, 只要在一个 XML 文件里对验证程序应该如何工作作出声明就可以了. 需要声明的内容包括:
    - **哪些字段需要进行验证**
    - **使用什么验证规则**
    - **在验证失败时应该把什么样的出错消息发送到浏览器端**
  + 编程验证：通过编写代码来验证用户输入

## 2. 声明式验证

* 声明式验证程序可以分为两类:
  + **字段验证**: 判断某个字段属性的输入是否有效
  + **非字段验证**: 不只针对某个字段，而是针对多个字段的输入值之间的逻辑关系进行校验。例如：对再次输入密码的判断。
* 使用一个声明式验证程序需要 3 个步骤:
  + 1. **确定哪些 Action 字段需要验证**
  + 2. **编写一个验证程序配置文件**. 它的文件名必须是以下两种格式之一:
    - **若一个 Action 类的多个 action 使用同样的验证规则: ActionClassName-validation.xml**
    - **若一个 Action 类的多个 action 使用不同的验证规则**: **ActionClass-alias-validation.xml**, 例如 UserAction-**User\_create**-validation.xml
  + 3. **确定验证失败时的响应页面**: 在 struts.xml 文件中定义一个 <result name=“input”> 的元素.

## 3. Struts2 内建的验证规则

* conversion validator：转换验证器
* date validator：日期验证器
* double validator：浮点验证器
* email validator：email 验证器
* expression validator：表达式验证器
* fieldexpression validator：字段表达式验证器
* int validator：整型验证器
* regex validator：正则表达式验证器
* required validator：非空验证器
* requiredstring validator：非空字符串验证器
* stringlength validator：字符串长度验证器
* url validator：url 格式验证器
* visitor validator：复合属性验证器

## 4. Struts2 内建的验证程序

* required: 确保某给定字段的值不是空值 null “”
* requiredstring: 确保某给定字段的值既不是空值 null, **也不是空白**.
  + trim 参数. 默认为 true, 表示 struts 在验证该字段值之前先剔除前后空格.
* stringlength: 验证一个非空的字段值是不是有足够的长度.
  + minLength: 相关字段的最小长度. 若没有给出这个参数, 该字段将没有最小长度限制
  + maxLength:相关字段的最大长度. 若没有给出这个参数, 该字段将没有最大长度限制
  + trim: 在验证之前是否去除前后空格
* date: 确保某给定日期字段的值落在一个给定的范围内
  + max:相关字段的最大值. 若没给出这个参数, 该字段将没有最大值限制
  + min:相关字段的最小值. 若没给出这个参数, 该字段将没有最小值限制
* email: 检查给定 String 值是否是一个合法的 email
* url: 检查给定 String 值是否是一个合法的 url
* regex: 检查某给定字段的值是否与一个给定的正则表达式模式相匹配.
  + expresssion\*: 用来匹配的正则表达式
  + caseSensitive: 是否区分字母的大小写. 默认为 true
  + trim: 是否去除前后空格. 默认为 true
* int: 检查给定整数字段值是否在某一个范围内
  + min: 相关字段的最小值. 若没给出这个参数, 该字段将没有最小值限制
  + max: 相关字段的最大值. 若没给出这个参数, 该字段将没有最大值限制
* conversion: 检查对给定 Action 属性进行的类型转换是否会导致一个转换错误. 该验证程序还可以在默认的类型转换消息的基础上添加一条自定义的消息
* expression 和 fieldexpression: 用来验证给定字段是否满足一个 OGNL 表达式.
  + 前者是一个非字段验证程序, 后者是一个字段验证程序.
  + 前者在验证失败时将生成一个 action 错误, 而后者在验证失败时会生成一个字段错误
  + expression\*: 用来进行验证的 OGNL 表达式

## 5. 短路验证器

* <validator …/> 元素和 <field-validator …/> 元素可以指定一个可选的 **short-circuit 属性**，该属性**指定该验证器是否是短验证器，默认值为 false**。
* 对同一个字段内的多个验证器，如果一个短路验证器验证失败，其他验证器不会继续校验

## 6. 字段验证 vs 非字段验证

* 字段验证字段优先，可以为一个字段配置多个验证规则
* 非字段验证验证规则优先
* 大部分验证规则支持两种验证器，但个别的验证规则只能使用非字段验证，例如表达式验证。

## 7. 自定义验证器

* 自定义验证器必须实现 Validator 接口.
* ValidatorSupport 和 FieldValidatorSupport 实现了 Validator 接口
  + 若需要普通的验证程序, 可以继承 ValidatorSupport 类
  + 若需要字段验证程序, 可以继承 FieldValidatorSupport 类
  + 若验证程序需要接受一个输入参数, 需要为这个参数增加一个相应的属性
* 注册验证程序: 自定义验证器需要在类路径里的某个 validators.xml 文件里注册: 验证框架首先在根目录下找validators.xml文件,没找到validators.xml文件, 验证框架将调用默认的验证设置,即default.xml里面的配置信息.
* 自定义一个 18 位身份证验证器
  + 编写验证器类
  + 在 validators.xml 文件中进行注册
  + 在验证配置文件中使用

## 8. 编程验证

* Struts2 提供了一个 Validateable 接口, 可以使 Action 类实现这个接口以提供编程验证功能.
* ActionSupport 类已经实现了 Validateable 接口

# 十二、文件的上传下载

## 1. 表单的准备

* 要想使用 HTML 表单上传一个或多个文件
  + 须把 HTML 表单的 enctype 属性设置为 **multipart/form-data**
  + 须把 HTML 表单的method 属性设置为 **post**
  + 需添加 <input type=“file”> 字段.

## 2. Struts 对文件上传的支持

* 在 Struts 应用程序里, **FileUpload 拦截器**和 Jakarta Commons FileUpload 组件可以完成文件的上传.
* 步骤:
  + 1. 在 Jsp 页面的文件上传表单里使用 file 标签. 如果需要一次上传多个文件, 就必须使用多个 file 标签, 但它们的名字必须是相同的
  + 2. 在 Action 中新添加 3 个和文件上传相关的属性. 这 3 个属性的名字必须是以下格式
    - [File Name] : File -被上传的文件。例如：data
    - [File Name]ContentType : String -上传文件的文件类型。例如：dataContentType
    - [File Name]FileName : String -上传文件的文件名。例如：dataFileName
  + 如果上上传多个文件, 可以使用 List

## 3. 配置 FileUpload 拦截器

* FileUpload 拦截器有 3 个属性可以设置.
  + maximumSize: 上传单个文件的最大长度(以字节为单位), 默认值为 2 MB
  + allowedTypes: 允许上传文件的类型, 各类型之间以逗号分隔
  + allowedExtensions: 允许上传文件扩展名, 各扩展名之间以逗号分隔
  + 可以在 struts.xml 文件中覆盖这 3 个属性
  + Commons FileUpload 组件默认接受上传文件总的最大值为 2M， 可以通过在 struts 配置文件中配置常量的方式修改
* 与文件上传有关的出错消息在 struts-messages.properties 文件里预定义. 可以在文件上传 Action 相对应的资源文件中重新定义错误消息

## 4. 文件下载概述

* 在某些应用程序里, 可能需要动态地把一个文件发送到用户的浏览器中, 而这个文件的名字和存放位置在编程时是无法预知的

## 5. Stream 结果类型

* Struts 专门为文件下载提供了一种 Stream 结果类型. 在使用一个 Stream 结果时, 不必准备一个 JSP 页面.
* Stream 结果类型可以设置如下参数:
  + contentType：被下载的文件的 MIME 类型。默认值为 text/plain
  + contentLength：被下载的文件的大小，以字节为单位
  + contentDisposition： 可以设置下载文件名的ContentDispositon 响应头，默认值为 inline，通常设置为如下格式： *attachment;filename="document.pdf"*.
  + inputName：Action 中提供的文件的输入流。默认值为 inputStream
  + bufferSize：文件下载时缓冲区的大小。默认值为 1024
  + allowCaching ：文件下载时是否允许使用缓存。默认值为 true
  + contentCharSet：文件下载时的字符编码。
* **Stream 结果类型的参数可以在 Action 以属性的方式覆盖**

# 十三、防止表单重复提交

## 1.概述

* 表单的重复提交：
  + 若刷新表单页面, 再提交表单不算重复提交.
  + 在不刷新表单页面的前提下:
    - 多次点击提交按钮
    - 已经提交成功, 按 "回退" 之后, 再点击 "提交按钮".
    - 在控制器响应页面的形式为转发情况下，若已经提交成功, 然后点击 "刷新(F5)“
* 重复提交的缺点:
  + 加重了服务器的负担
  + 可能导致错误操作.

## 2. 标记管理

* Struts 提供的 **token 标签**可以用来生成一个独一无二的标记. 这个标签**必须嵌套在 form 标签的内部使用**, **它将在表单里插入一个隐藏字段并把标记值（隐藏域的字段的值）保存在HttpSession 对象里**.
* Token 标签必须与 Token 或 TokenSession 拦截器配合使用, 这两个拦截器都能对标记进行处理.
* Token 拦截器在遇到重复提交情况时, 会返回 invalid.token 结果并加上一个 Action 错误. 这种错误默认的消息是: The form has already been processed or no token was supplied, please try again.
* **TokenSession 拦截器采取的做法只是阻断后续的提交, 用户将看到同样的响应，但实际上并没有重复提交**

# 十四、自定义拦截器

## 1. Struts2 拦截器

* 拦截器（Interceptor）是 Struts 2 的核心组成部分。
* Struts2 很多功能都是构建在拦截器基础之上的，例如文件的上传和下载、国际化、数据类型转换和数据校验等等。
* Struts2 拦截器在访问某个 Action 方法之前或之后实施拦截
* Struts2 拦截器是可插拔的, 拦截器是 AOP（面向切面编程） 的一种实现．
* 拦截器栈(Interceptor Stack): 将拦截器按一定的顺序联结成一条链. 在访问被拦截的方法时, Struts2 拦截器链中的拦截器就会按其之前定义的顺序被依次调用

## 2. Struts2 自带的拦截器(1)



## 3. Interceptor 接口

* 每个拦截器都是实现了 接口的 Java 类:
  + init: 该方法将在拦截器被创建后立即被调用, 它在拦截器的生命周期内只被调用一次. 可以在该方法中对相关资源进行必要的初始化
  + interecept: 每拦截一个请求, 该方法就会被调用一次.
  + destroy: 该方法将在拦截器被销毁之前被调用, 它在拦截器的生命周期内也只被调用一次.
* Struts 会依次调用为某个 Action 而注册的每一个拦截器的 interecept 方法.
* 每次调用 interecept 方法时, Struts 会传递一个 **ActionInvocation** 接口的实例.
* ActionInvocation: 代表一个给定 Action 的执行状态, 拦截器可以从该类的对象里获得与该 Action 相关联的 Action 对象和 Result 对象. 在完成拦截器自己的任务之后, 拦截器将调用 ActionInvocation 对象的 invoke 方法前进到 Action 处理流程的下一个环节.
* **AbstractInterceptor** 类实现了 Interceptor 接口. 并为 init, destroy 提供了一个空白的实现

4.定义自定义拦截器的步骤

* + 自定义拦截器
  + 在 struts.xml 文件中配置自定义的拦截器