[Struts2流程图 4](#_Toc254902771)

[搭建开发环境 5](#_Toc254902772)

[配置文件 6](#_Toc254902773)

[Action名称的搜索顺序 7](#_Toc254902774)

[struts.xml详解 7](#_Toc254902775)

[配置bean 7](#_Toc254902776)

[指定多个struts.xml 9](#_Toc254902777)

[配置常量 9](#_Toc254902778)

[配置package 9](#_Toc254902779)

[全局结果(同一包的的action公用) 9](#_Toc254902780)

[配置拦截器interceptor 9](#_Toc254902781)

[配置action 10](#_Toc254902782)

[配置结果视图result 11](#_Toc254902783)

[result的type取值 11](#_Toc254902784)

[result子标签name取值 11](#_Toc254902785)

[type=”chain”(服务器跳转, 从一个Action跳到另外一个Action) 11](#_Toc254902786)

[常用的常量介绍 12](#_Toc254902787)

[Action 13](#_Toc254902788)

[Action的7种配置 13](#_Toc254902789)

[DMI-动态方法调用:使用!(不推介) 13](#_Toc254902790)

[使用通配符映射action(推介使用) 14](#_Toc254902791)

[零配置/注解配置 15](#_Toc254902792)

[Action中接收表单数据的3种方法: 16](#_Toc254902793)

[Action里使用(request/session/application)的4种方法 18](#_Toc254902794)

[第一种: 原始类型,依赖struts 2 (非IOC) 18](#_Toc254902795)

[第二种: Map类型, 依赖struts2 (非IOC) 16 18](#_Toc254902796)

[第三种: Map类型, 不依赖struts 2 (IOC方法) (用得最多) 20](#_Toc254902797)

[第四种: 原始类型, 不依赖struts 2 (IOC) 20](#_Toc254902798)

[自定义类型转换器(converter) 21](#_Toc254902799)

[注册为局部类型转换器： 22](#_Toc254902800)

[注册为全局类型转换器： 22](#_Toc254902801)

[struts.xml向Action里的属性注入值 23](#_Toc254902802)

[文件上传 23](#_Toc254902803)

[多文件上传 24](#_Toc254902804)

[自定义拦截器 26](#_Toc254902805)

[execAndWait拦截器: 服务器忙时显示一个等待页面 28](#_Toc254902806)

[token拦截器: 防止表单重复提交 29](#_Toc254902807)

[自定义权限控制拦截器 29](#_Toc254902808)

[timer拦截器: 控制台打印一次request到response之间等待的时间 29](#_Toc254902809)

[校验 29](#_Toc254902810)

[校验----手工编写代码方式 29](#_Toc254902811)

[校验----基于XML配置方式 32](#_Toc254902812)

[国际化 38](#_Toc254902813)

[异常处理 41](#_Toc254902814)

[OGNL 41](#_Toc254902815)

[OGNL表达式语言 41](#_Toc254902816)

[OGNL能访问哪些东西: 45](#_Toc254902817)

[OGNL可以访问集合的伪属性(无get, set方法) 45](#_Toc254902818)

[ValueStack(值栈, 直接访问) 46](#_Toc254902819)

[OGNL中#的使用: 46](#_Toc254902820)

[OGNL中%的使用: 46](#_Toc254902821)

[OGNL中$的使用: 47](#_Toc254902822)

[查看值栈中的信息: 47](#_Toc254902823)

[选择取值栈中的对象 47](#_Toc254902824)

[压栈 47](#_Toc254902825)

[获得集合对象的子集(ONGL的选择, 投影功能, 不推介, 数据在查询出来就应该过滤好) 47](#_Toc254902826)

[(兰布达表达式, 用于递归算法)Lambda Expressions 48](#_Toc254902827)

[覆盖struts 2的主题样式 48](#_Toc254902828)

[SSH整合 48](#_Toc254902829)

[Struts2+Spring2.5+Hibernate3.3整合开发 48](#_Toc254902830)

[**Tag(标签库)** 50](#_Toc254902831)

[**Generic Tags** 50](#_Toc254902832)

[**Control Tags** 50](#_Toc254902833)

[**if /elseIf /else** 50](#_Toc254902834)

[**append** 50](#_Toc254902835)

[**generator** 50](#_Toc254902836)

[**iterator** 50](#_Toc254902837)

[**merge** 52](#_Toc254902838)

[**sort** 52](#_Toc254902839)

[**subset** 52](#_Toc254902840)

[**Data Tags** 52](#_Toc254902841)

[**a** 52](#_Toc254902842)

[**action** 52](#_Toc254902843)

[**bean** 52](#_Toc254902844)

[**date** 52](#_Toc254902845)

[**debug** 52](#_Toc254902846)

[**i18n** 53](#_Toc254902847)

[**include** 53](#_Toc254902848)

[**param** 53](#_Toc254902849)

[**property** 53](#_Toc254902850)

[**push** 54](#_Toc254902851)

[**set** 54](#_Toc254902852)

[**text** 54](#_Toc254902853)

[**url** 54](#_Toc254902854)

[**UI Tags** 54](#_Toc254902855)

[**Form Tags** 54](#_Toc254902856)

[**checkbox** 55](#_Toc254902857)

[**checkboxlist** 55](#_Toc254902858)

[**combobox** 55](#_Toc254902859)

[**doubleselect** 56](#_Toc254902860)

[**head** 56](#_Toc254902861)

[**file** 56](#_Toc254902862)

[**form** 56](#_Toc254902863)

[**hidden** 56](#_Toc254902864)

[**label** 56](#_Toc254902865)

[**optiontransferselect** 56](#_Toc254902866)

[**optgroup** 56](#_Toc254902867)

[**password** 56](#_Toc254902868)

[**radio** 56](#_Toc254902869)

[**reset** 56](#_Toc254902870)

[**select** 56](#_Toc254902871)

[**submit** 57](#_Toc254902872)

[**textarea** 57](#_Toc254902873)

[**textfield** 57](#_Toc254902874)

[**token** 57](#_Toc254902875)

[**updownselect** 57](#_Toc254902876)

[**Non-Form UI Tags** 57](#_Toc254902877)

[**actionerror** 57](#_Toc254902878)

[**actionmessage** 57](#_Toc254902879)

[**component** 57](#_Toc254902880)

[**div** 57](#_Toc254902881)

[**fielderror** 57](#_Toc254902882)

[**Ajax Tags** 57](#_Toc254902883)

[**a** 57](#_Toc254902884)

[**autocompleter** 58](#_Toc254902885)

[**bind** 58](#_Toc254902886)

[**datetimepicker** 58](#_Toc254902887)

[**div** 58](#_Toc254902888)

[**head** 58](#_Toc254902889)

[**submit** 58](#_Toc254902890)

[**tabbedpanel** 58](#_Toc254902891)

[**textarea** 58](#_Toc254902892)

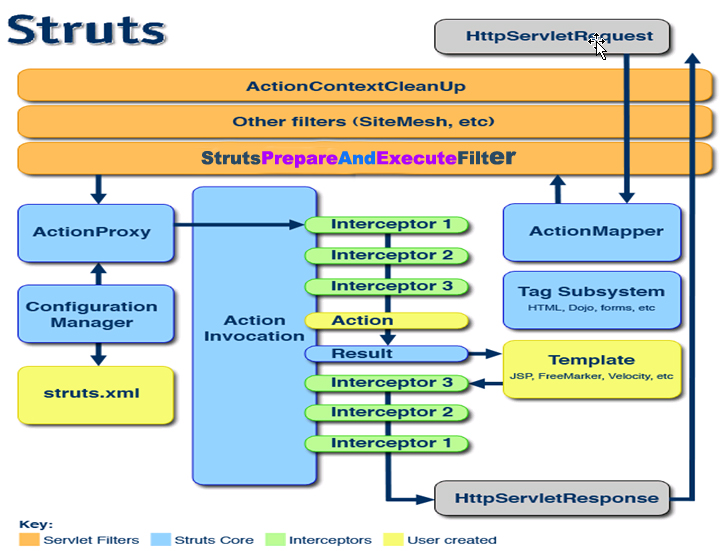
[**tree** 58](#_Toc254902893)

[**treenode** 58](#_Toc254902894)

[**模板相关** 59](#_Toc254902895)

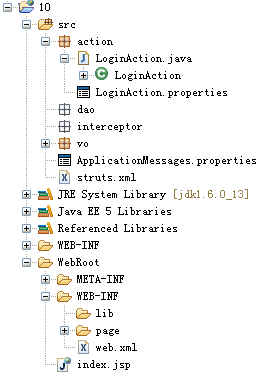
[**非表单UI** 61](#_Toc254902896)

# Struts2流程图



|  |
| --- |
| 1. HTTP请求发出后, 如果在Context中有对象, 则先进入过滤器ActionContextCleanUp清理Context对象;  2. 然后判断如果项目中使用了SiteMesh这样的框架, 那先通过这样的过滤器;  3. 接下来, 通过StrutsPrepareAndExecuteFilter(转发过滤器, 它是Struts 2框架的核心控制器，它负责 拦截 由<url-pattern>/\*</url-pattern>指定的所有用户请求，),然后过滤用户请求, 默认情况下，如果用户请求的路径不带后缀或者路径以.action结尾，该请求将被转入Struts 2框架处理，否则该请求将略过Struts 2框架的处理。)  4.如果是一个Action请求, 则转发到ActionMapper(Action管理器), ActionMapper再一次调用StrutsPrepareAndExecuteFilter(转发过滤器)进入一个ActionProxy(Action代理工厂, 类似于Struts1里的ActionServlet);  5.ActionProxy调用Configuration Manager(配置管理器, 这个配置管理器里面包含struts.xml配置信息)寻找相应URL控制,  6.通过ActionInvocation找到Action, 如果找到的Action有拦截器, 则穿过拦截器, 最后才到达Action,  7.(Action是一个具有操作能力的POJO, 可接收页面传过来的数据, 和控制绑定在一起,可调用相关的业务逻辑, 返回一个字符串类型的结果),  8.根据Action结果, 结合struts.xml中定义的result导航到相应的目标视图,  9.如果使用了Template(模板, 比如表现层技术:FreeMarker/ Velocity), 则通过模板组合成一个新的结果-->穿出拦截器栈-->产生响应结果(HttpServletResponse) --> —穿出过滤器, 返回给用户. |

# 搭建开发环境



|  |
| --- |
| 第1步:导入jar包  struts2-core-2.x.x.jar :Struts 2框架的核心类库  xwork-2.x.x.jar :XWork类库，Struts 2在其上构建  ognl-2.6.x.jar :对象图导航语言（Object Graph Navigation Language），struts2框架通过其读写对象的属性  freemarker-2.3.x.jar :Struts 2的UI标签的模板使用FreeMarker编写  commons-logging-1.1.x.jar :ASF出品的日志包，Struts 2框架使用这个日志包来支持Log4J和JDK 1.4+的日志记录。  commons-fileupload-1.2.1.jar (可选包)文件上传组件，2.1.6版本后需要加入此文件 |
| 第2步:在web.xml中以过滤器的方式注册Struts2  <!-- 配置编码格式为:中文(需放在Struts2配置前面) -->  <filter>  <filter-name>struts-cleanup</filter-name>  <filter-class>org.apache.struts2.dispatcher.ActionContextCleanUp</filter-class>  </filter>    <filter-mapping>  <filter-name>struts-cleanup</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!-- 配置Struts2 -->  <filter>  <filter-name>struts2</filter-name>  <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>  <!-- 自从Struts 2.1.3以后，下面的FilterDispatcher (2.0版)已经标注为过时  <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class> -->  </filter>  <filter-mapping>  <filter-name>struts2</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |
| 第3步: 在src下创建一个struts.xml  <?xml version=*"1.0" encoding="UTF-8" ?>*  <!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"  "http://struts.apache.org/dtds/struts-2.0.dtd">  <struts>  </struts> |

# 配置文件

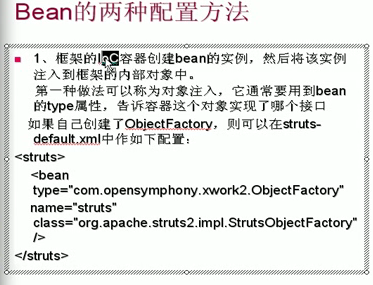
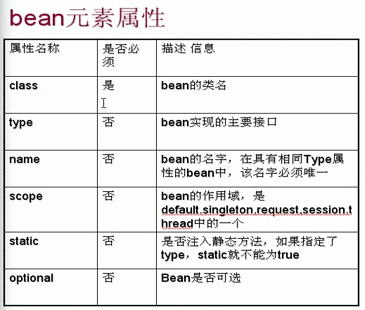
|  |  |  |
| --- | --- | --- |
| 加载的顺序:  先  |  |  V  后 | struts-default.xml | 默认配置文件(只读) |
| struts-plugin.xml | 插件配置文件 |
| struts.xml | 主配置文件 |
| struts.properties | 属性文件 |
| web.xml |  |
|  | 前3个xml文件的格式是一样的,因为他们使用同一个DTD文件.配置常量:<constant name=”” value=””>针对整个项目用的.  在web.xml中加入初始化参数<init-param>  如果在多个文件中配置了同一个常量，则后一个文件中配置的常量值会覆盖前面文件中配置的常量值. | |

## Action名称的搜索顺序

|  |
| --- |
| 1．获得请求路径的URI，例如url是：http://server/struts2/path1/path2/path3/test.action  2．首先寻找namespace为/path1/path2/path3的package，如果存在这个package，则在这个package中寻找名字为test的action，如果不存在这个package则转步骤3；  3．寻找namespace为/path1/path2的package，如果存在这个package，则在这个package中寻找名字为test的action，如果不存在这个package，则转步骤4；  4．寻找namespace为/path1的package，如果存在这个package，则在这个package中寻找名字为test的action，如果仍然不存在这个package，就去默认的namaspace（默认的命名空间为空字符串"" ）的package下面去找名字为test的action，如果还是找不到，页面提示找不到action。  同一个package的action不能同名  不同的package的action可以同名  package的namespace 必须以/开头  antion的name 不能以/开头 |

## struts.xml详解

### 配置bean



### 指定多个struts.xml

|  |
| --- |
| <struts>  <include file="struts-user.xml"/>  <include file="struts-order.xml"/>  </struts> |

### 配置常量

|  |
| --- |
| <constant name=”” value=”” /> |

### 配置package

|  |
| --- |
| <package name="itcast" namespace="/test" extends="struts-default">  1>如果没有为action指定class，默认是ActionSupport。  2>如果没有为action指定method，默认执行action中的execute() 方法。  3>abstract: 声明为抽象包,里面不能有action  4>package必须要配置name="products" extends="struts-default"  5>namespace默认等于””; 如果配置,必须以/开头, 不能以/结尾 |

#### 全局结果(同一包的的action公用)

|  |
| --- |
| <global-results>  <result name="message">/message.jsp</result>  </global-results> |

#### 配置拦截器interceptor

|  |
| --- |
| <interceptors> //自定义拦截器 或 拦截器栈  <interceptor name=*"PermissionInterceptor"* class=*"cn.cstp.org.interceptor.PermissionInterceptor"* /> //自定义一个拦截器  <interceptor-stack name=*"myInterceptorStack"*> //自定义拦截器栈, 封装多个拦截器, action中就可以写一条语句调用多个拦截器  <interceptor-ref name=*"defaultStack"* /> //引入默认拦截器栈  <interceptor-ref name=*"PermissionInterceptor "* /> //引入自定义拦截器  </interceptor-stack>  </interceptors>  action中需要引入拦截器或拦截器栈, 默认引入defaultStack, 如果手动引入了拦截诶其, 这不会再自动引入defaultStack了 |

#### 配置action

|  |
| --- |
| 普通action  <action name="helloworld" class="cn.itcast.action.HelloWorldAction" method="execute" >  <interceptor-ref name=*"myInterceptorStack"* /> //引入 拦截器 或 拦截器栈(绑定默认的拦截器)  </action>   1. name不能以/开头 2. method默认等于"execute" |
| 转发action(没有调用方法, 直接跳转到目标页面)  <action name=*"zhuanfa"*>  <result type=*"redirect"*>/WEB-INF/page/error.jsp</result> <!-- type=*"redirect" 客户端跳转 -->*  </action> |
| 访问不存在的action,其实就是一个默认action 加 转发action  <default-action-ref name=*"error"* />  <action name=*"error"*>  <result>/error.jsp</result>  </action>  最低优先级action, 作为最后一个action  <action name=*"\*"*>  <result>/error.jsp</result>  </action> |
| 访问不存在的资源, (不是action).  需要在web.xml中配置  <error-page>  <error-code>404</error-code>  <location>/error.jsp</location>  </error-page> |

##### 配置结果视图result

|  |
| --- |
| <result name="success">/WEB-INF/page/hello.jsp</result>  1>如果没有指定result的name属性，默认值为success。 |

###### result的type取值

|  |  |
| --- | --- |
| redirect | 客户端 跳转到一个url, 不能是一个action的名称 |
| dispacher(默认) | 服务器端跳转到一个url, 不能是一个action的名称 |
| redirectAction | 客户端 , 从一个Action跳转到另一个Action, 值是一个action的名称 |
| chain | 服务器端, 从一个Action跳转到另一个Action, 值是一个action的名称 |
| stream | 向浏览器发送InputStream对象, 通常用来处理文件下载,还可以用于返回AJAX数据 |
| velocity | 处理Velocity模板(前台表现层) |
| freemaker | 处FreeMarker模板(前台表现层) |
| httpheader | 控制特殊HTTP行为的结果类型(hppt文件头) |
| xslt | 处理XML/XLST模板 |
| plainText | 输出原始文件内容, 例如显示文件源代码(一般情况下浏览器不给解析出来, 你就查看源代码呗) |
| tiles | tiles布局用 |

###### result子标签name取值

|  |  |
| --- | --- |
| locaction(默认) | 目标页面 |
| actionName | 跳转到的action名字 |
| charSet | 字符编码(type=plainText时使用) |
| namespace | 另一个包的namespace名称 |

###### type=”chain”(服务器跳转, 从一个Action跳到另外一个Action)

|  |
| --- |
| <package name=*"test"* extends=*"struts-default"* namespace=*""*>  <action name=*"test"* class=*""*>  <result name=*"success"* type=*"chain"*>  <param name=*"actionName"*>test1</param> <!-- name默认等于location(一个实际资源) -->  </result>  </action>    <action name=*"test1"*extends=*"struts-default"* namespace=*""*>  <result name=*"success"*>/test1\_suc.jsp</result>  </action>  </package> |

|  |  |
| --- | --- |
| ${param.name} | <%  request.getParameter("name");  %>  //getAttribute();取pageContext, request, session, application里的参数  //getParameter();取get或post传来的参数 |

定制的rul:

http://域名/项目名/package的namespase/action的name

生成的rul:

http://域名/项目名/result的值

## 常用的常量介绍

|  |  |
| --- | --- |
| struts.i18n.encoding | 指定默认编码集,作用于HttpServletRequest的setCharacterEncoding方法 和freemarker 、velocity的输出 默认值:ISO8859-1(英文) |
| struts.action.extension | 该属性指定需要Struts 2处理的请求后缀，该属性的默认值是action，即所有匹配\*.action的请求都由Struts2处理。如果用户需要指定多个请求后缀，则多个后缀之间以英文逗号（,）隔开。 |
| struts.serve.static.browserCache | 设置浏览器是否缓存静态内容,默认值为true(生产环境下使用),开发阶段最好关闭 |
| struts.ui.theme | 默认的视图主题 |
| struts.enable.DynamicMethodInvocation | 该属性设置Struts 2是否支持动态方法调用，该属性的默认值是true。如果需要关闭动态方法调用，则可设置该属性为false。 |
|  |  |
| struts.multipart.maxSize | 上传文件的大小限制 单位: |
| struts.multipart.saveDir | 设置存储上传文件的目录文件夹 |
|  |  |
| struts.objectFactory.spring.autoWire | 是否自动绑定Spring |
| struts.objectFactory | 与spring集成时，指定由spring负责action对象的创建 |
|  |  |
| struts.configuration.xml.reload | 当struts的配置文件修改后,系统是否自动重新加载该文件,默认值为false(开发阶段使用),开发阶段最好打开 |
| struts.configuration.file | struts自动加载的配置文件列表 |
| struts.devMode | 是否为struts开发模式 |
| struts.devMode | (开发阶段使用),这样可以打印出更详细的错误信息 |
| struts.ui.theme | simple(不要生成主题相关代码) 默认的视图主题为xhtml |

# Action

## Action的7种配置

|  |  |
| --- | --- |
| Action基本配置 | Action内方法名称 == 属性method的值(默认为execute) |
| method属性 | Action内方法名称 比 属性method的值多个do (eg: doBreak, doMain) 好处在与避开java关键字) |
| DMI-动态方法调用(不推介) | struts.xml里面的action标签不配置method属性; 访问方式: action的名字+!+方法名 |
| Forword Action | 空action (只做转发) (设计原则: jsp之间不能相互跳转, 必须经过action跳转) |
| 默认Action | 访问不存在的action,其实就是一个默认的Forword action的 |
| 通配符映射 (推介使用) | 用action标签中使用※通配符 |
| Zero Configuration(注解配置/零配置) | 注解Action类 |

### DMI-动态方法调用:使用!(不推介)

|  |
| --- |
| 如果Action中存在多个方法时，我们可以使用!+方法名调用指定方法。  struts.xml里面的action标签不配置method属性.  如下：  public class HelloWorldAction{  private String message;  ....  public String execute() throws Exception{  this.message = "我的第一个struts2应用";  return "success";  }    public String other() throws Exception{  this.message = "第二个方法";  return "success";  }  }  假设访问上面action的URL路径为： /struts/test/helloworld.action  要访问action的other() 方法，我们可以这样调用：  /struts/test/helloworld!other.action  如果不想使用动态方法调用，我们可以通过常量struts.enable.DynamicMethodInvocation关闭动态方法调用。  <constant name="struts.enable.DynamicMethodInvocation" value="false"/> |

### 使用通配符映射action(推介使用)

|  |
| --- |
| <a href=”<%=path%>/test/add\_user” >通配符映射</a>  <a href=”<%=path%>/test/del\_user” >通配符映射</a>  <package name="itcast" namespace="/test" extends="struts-default">  <action name="\*\_\*" class="cn.itcast.action.{2}Action" method="{1}">  <result name="success">/WEB-INF/page/hello.jsp</result>  </action>  </package> |
| public class UserAction {  private String message;  ....  public String del() throws Exception{  this.message = "删除用户";  return "success";  }    public String add() throws Exception{  this.message = "添加用户";  return "success";  }  } |
| action的优先级  没有通配符的action  如果路径中有通配符(\*)不管有几个通配符,一律按顺序去找, 所以范围小的统配路径应该放在前面 |

各种通配符代表的意思

|  |  |
| --- | --- |
| \* | 0-n不包括/ |
| \*\* | 0-n包括/ |
| / | 转义符 |

### 零配置/注解配置

|  |
| --- |
| 概述:  ParentPackage 父包  Namespace 包名  Result (只有一个结果集)  Results (多个结果集) |
| 写在Action前面的注解  @ParentPackage(value="struts-default")  @Namespace(value=" package1")  @Result(name="success", value="/WEB-INF/page/main.jsp")  @Results({  Result(name="success", value="/WEB-INF/page/main.jsp", type=org.apache.struts2.dispatcher.ServletRedirectResult.class),//配置type(全类名,不要用双引号括起来):客户端跳转  Result(name="input", value="/WEB-INF/page/login.jsp")  })  public Class LognAction(){ // 请求路径: <%=path%>/package1/logn.action  这里必须有execute方法(默认就执行它)  } |
| web.xml的filter标签中需要声明包别名和包路径  <filter>  ……  <init-param>  <param-name>actionPackages</param-name>  <param-value>cn.itcast.org.action</param-value>  </init-param>  </filter> |

## Action中接收表单数据的3种方法:

|  |
| --- |
| 方法1.利用action类的属性接受用户输入(适合表单数据项比较少的情况)  需要getUsername(), setUsername() |
| 方法2.利用领域对象(VO)接受用户输入(适合表单数据项比较多的情况)  必需要一个参数为空的构造方法(默认)  jsp页面表单名称写成xx.xxx, 而不是xxx  对象属性不需要new Xxx(), 但需要写getXxx(), 和setXxx() |
| 方法3(不好用).使用ModelDriven(模型驱动)模式接受用户输入. (适合表单数据项比较多的情况)  必需要一个参数为空的构造方法(默认)  须实现接口ModelDriven, 和重写getModel()(返回Xxx对象)  需要new Xxx(), 访问数据:xxx.getXxx();  提交和结果jsp页面: 表单名称写成xxx, 而不是xx.xxx  <form action="<%=path%>*/first/login*">  username:<input type=*"text"* name=*"username"* />  <br>  password:<input type=*"password"* name=*"password"* />  <br>  <input type=*"submit"* value=*"登陆"* />  </form>  **import** vo.User;  **import** com.opensymphony.xwork2.ActionSupport;  **import** com.opensymphony.xwork2.ModelDriven;  /\*\*  \* 使用ModelDriven<T>(模型驱动)模式接受用户输入, 需要使用泛型  \* 使用模型驱动的好处是省略写getXxx(), setXxx().  \* **@author** Administrator  \*  \*/  **public** **class** LoginAction **extends** ActionSupport **implements** ModelDriven<User>{  **private** **static** **final** **long** *serialVersionUID* = 1L;    **private** User user = **new** User();//需要手动new一个对象    **public** String execute() **throws** Exception{  System.*out*.println(user.getUsername());  System.*out*.println(user.getPassword());    **return** **null**;  }    /\*\*  \* 模型驱动需要返回一个User对象  \*/  **public** User getModel(){    **return** user;  }  } |

## Action里使用(request/session/application)的4种方法

|  |
| --- |
| 第一种: 原始类型,依赖struts 2 (非IOC) **public** **class** Request\_Session\_Application **extends** ActionSupport {  **private** **static** **final** **long** *serialVersionUID* = 1L;  **private** HttpServletRequest request;  **private** HttpSession session;  **private** ServletContext application;  **private** Request\_Session\_Application() {  request = ServletActionContext.*getRequest*();  session = request.getSession();  application = ServletActionContext.*getServletContext*();  }  **public** String execute() {  request.setAttribute("k1", "v1");  session.setAttribute("k2", "v2");  application.setAttribute("k3", "v3");  **return** "success";  }  } |
| 第二种: Map类型, 依赖struts2 (非IOC) 16 **public** **class** Request\_Session\_Application **extends** ActionSupport {  **private** **static** **final** **long** *serialVersionUID* = 1L;  **private** Map request;  **private** Map session;  **private** Map application;  **private** Request\_Session\_Application() {  request = (Map) ActionContext.*getContext*().get("request");  session = ActionContext.*getContext*().getSession();  application = ActionContext.*getContext*().getApplication();  }  **public** String execute() {  request.put("k1", "v1");  session.put("k2", "v2");  application.put("k3", "v3");  **return** "success";  }  } |
| 第三种: Map类型, 不依赖struts 2 (IOC方法) (用得最多) **public** **class** Request\_Session\_Application **extends** ActionSupport **implements** RequestAware, SessionAware, ApplicationAware{  **private** **static** **final** **long** *serialVersionUID* = 1L;  **private** Map request;  **private** Map session;  **private** Map application;  **public** **void** setRequest(Map request) {  **this**.request = request;  }  **public** **void** setSession(Map session) {  **this**.session = session;  }  **public** **void** setApplication(Map application) {  **this**.application = application;  }  **public** String execute() **throws** Exception {  request.put("k1", "v1");  session.put("k2", "v2");  application.put("k3", "v3");  **return** *SUCCESS*;  }  } |
| 第四种: 原始类型, 不依赖struts 2 (IOC) **public** **class** Request\_Session\_Application **extends** ActionSupport **implements** ServletRequestAware, ServletContextAware {  **private** **static** **final** **long** *serialVersionUID* = 1L;    **private** HttpServletRequest request ;  **private** HttpSession session;  **private** ServletContext application;  **public** **void** setServletContext(ServletContext context) {  **this**.application = context;  }  **public** **void** setServletRequest(HttpServletRequest request) {  **this**.request = request;  }  **public** String execute() **throws** Exception {  session = request.getSession();    request.setAttribute("k1", "v1");  session.setAttribute("k2", "v2");  application.setAttribute("k3", "v3");    **return** *SUCCESS*;  }  } |

## 自定义类型转换器(converter)

|  |
| --- |
| **public** **class** DateConverter **extends** DefaultTypeConverter {  @Override  **public** Object convertValue(Map<String Object> context, Object value, Class toType) {// toType(要转换成的目标类型) value(存入或取出的数据)  SimpleDateFormat dateFormat = **new** SimpleDateFormat("yyyyMMdd");  **try** {  **if** (toType == Date.**class**) { // 存入数据(字符串-转换->Date)  String[] params = (String[]) value; // Request.getParameterValues()  **return** dateFormat.parse(params[0]);  } **else** **if** (toType == String.**class**) { // 取出数据(Date-转换->字符串)  Date date = (Date) value;  **return** dateFormat.format(date);  }  } **catch** (Exception e) {  }  **return** **null**;  }  } |
| 注册为局部类型转换器： 在Action类所在的包下放置ActionClassName-conversion.properties文件，ActionClassName是Action的类名，后面的-conversion.properties是固定写法，对于本例而言，文件的名称应为HelloWorldAction-conversion.properties 。在properties文件中的内容为：  **要转换的参数名=类型转换器的全类名**  对于本例而言， HelloWorldAction-conversion.properties文件中的内容为：  createtime= cn.itcast.conversion.DateConverter |
| 注册为全局类型转换器： 在WEB-INF/classes下放置xwork-conversion.properties文件 。在properties文件中的内容为：  待转换的类型=类型转换器的全类名  对于本例而言， xwork-conversion.properties文件中的内容为：  (对所有Date类型都作转换)java.util.Date= cn.itcast.conversion.DateConverter |

## struts.xml向Action里的属性注入值

|  |
| --- |
| 这跟struts1的action单例管理方式不同，struts2每个请求到来时都会创建一个action实例  当有些Action的配置不想硬编码的时候使用:  public class HelloWorldAction{  private String savePath;  public String getSavePath() {  return savePath;  }  public void setSavePath(String savePath) {  this.savePath = savePath;  }  ......  } |
| <package name="itcast" namespace="/test" extends="struts-default">  <action name="helloworld" class="cn.itcast.action.HelloWorldAction" >  <param name="savePath">/images</param>  <result name="success">/WEB-INF/page/hello.jsp</result>  </action>  </package>  上面通过<param>节点为action的savePath属性注入“/images” |

# 文件上传

|  |
| --- |
| 第一步：在WEB-INF/lib下加入jar包commons-fileupload-1.2.1.jar、commons-io-1.3.2.jar。这两个文件可以从http://commons.apache.org/下载。 |
| 第二步：把form表单的enctype设置为：“multipart/form-data“，如下：  <form enctype=*"multipart/form-data"* action="${pageContext.request.contextPath}*/first/hello*" method=*"post"*>  <input type=*"file"* name=*"uploadImage"*><br />  <input type=*"submit"* value=*"提交"* />  </form> |
| 第三步：在Action类中添加以下属性，属性红色部分对应于表单中文件字段的名称：  **import** java.io.File;  **import** java.io.IOException;  **import** org.apache.commons.io.FileUtils;  **import** org.apache.struts2.ServletActionContext;  **public** **class** HelloWorldAction {  **private** File uploadImage;// 得到上传的文件(临时文件, 本类运行结束后就清理)  **private** String uploadImageContentType;// 得到文件的类型  **private** String uploadImageFileName;// 得到文件的名称  //这里略省了属性的getter/setter方法    /\*\*  \* 主方法  \* @return  \*/  **public** String execute() {    ServletContext **application** = ServletActionContext.*getServletContext*();  String realpath = **application**.getRealPath("/images");//得到现在这个类在服务器上的绝对路径  File file = **new** File(realpath);//新建文件夹  **if** (!file.exists()) {//如果文件夹不存在,则创建.  file.mkdirs();  }  **try** {  FileUtils.*copyFile*(uploadImage, **new** File(file, uploadImageFileName));//一个非常好用的保存文件的方法:FileUtils.copyFile(源文件, 目标文件.)  } **catch** (IOException e) {  e.printStackTrace();  }  **return** "success";  }  } |

## 多文件上传

|  |
| --- |
| 第一步：在WEB-INF/lib下加入commons-fileupload-1.2.1.jar、commons-io-1.3.2.jar。这两个文件可以从http://commons.apache.org/下载。 |
| 第二步：把form表的enctype设置为：“multipart/form-data“，如下：  <form enctype=*"multipart/form-data"* action="${pageContext.request.contextPath}*/first/hello*" method=*"post"*>  <input type=*"file"* name=*"uploadImages"*><br />  <input type=*"file"* name=*"uploadImages"*><br />  <input type=*"file"* name=*"uploadImages"*><br />  <input type=*"submit"* value=*"提交"* />  </form> |
| 第三步：在Action类中添加以下属性，属性红色部分对应于表单中文件字段的名称：  **import** java.io.File;  **import** org.apache.commons.io.FileUtils;  **import** org.apache.struts2.ServletActionContext;  **public** **class** HelloWorldAction {  **private** File uploadImages[];// 得到上传的文件(临时文件, 本类运行结束后就清理)  **private** String uploadImagesContentType[];// 得到文件的类型  **private** String uploadImagesFileName[];// 得到文件的名称  //这里略省了属性的getter/setter方法    /\*\*  \* 主方法  \* @return  \*/  **public** String execute() **throws** Exception{  String realpath = ServletActionContext.*getServletContext*().getRealPath("/images");//得到现在这个类在服务器上的绝对路径  File file = **new** File(realpath);//新建文件夹  **if** (!file.exists()) {//如果文件夹不存在,则创建.  file.mkdirs();  }  **for**(**int** i=0; i<uploadImages.length; i++){  File img = uploadImages[i];  FileUtils.*copyFile*(img, **new** File(file, uploadImagesFileName[i]));//一个非常好用的保存文件的方法:FileUtils.copyFile(源文件, 目标文件.)  }  **return** "success";  }  } |

# 自定义拦截器

|  |
| --- |
| 要自定义拦截器需要  可以实现Interceptor接口;  或者继承Abstractinterceptor抽象类;  或者继承MethodFilterInterceptor方法过滤类(这个拦截器类可以配置不需要拦截的方法和需要拦截的方法)  MethodFilterInterceptor  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionInvocation;  **import** com.opensymphony.xwork2.interceptor.Interceptor;  **public** **class** *myInterceptor* **implements** Interceptor {  **private** **static** **final** **long** *serialVersionUID* = 1L;  **public** **void** destroy() {  System.*out*.println("系统关闭时执行");  }  **public** **void** init() {  System.*out*.println("系统初始化执行的最后一个方法");  }  **public** String intercept(ActionInvocation invocation) **throws** Exception {  System.*out*.println("intercept开始拦截");  String result = null  // Object user = ActionContext.getContext().getSession().get("user");  // if (user != null) {  // result = invocation.invoke();//如果还有拦截器, 则调用下一个拦截器; 如果没有拦截器你, 则调用Action指定的方法.  // }else{  result = “login”;//用户没有登录, 返回登录页面.  }  System.*out*.println("intercept结束拦截");  **return** result;//跳转到登陆页面  }  } |
| <package name=*"name1"* namespace=*"/first"* extends=*"struts-default"*>  <interceptors>  <interceptor name=*"myInterceptor"* class=*"cn.itcast.action.myInterceptor"* /> //注册自定义的拦截器  <interceptor-stack name=*"myInterceptorStack"*> <!-- 注册自定义拦截器栈 -->  <interceptor-ref name=*"defaultStack"* /> <!-- 引用默认的拦截器 -->  <interceptor-ref name=*"myInterceptor"* /> //引用自定义的拦截器  </interceptor-stack>  </interceptors>    <global-results>  <result name=*"login"*>/WEB-INF/page/login.jsp</result>  </global-results>    <action name=*"hello"* class=*"cn.itcast.action.HelloWorldAction"* method=*"execute"*>  <interceptor-ref name=*"* *myInterceptorStack" /*> <!-- 只有这个action才调用拦截器栈 -->  <result name=*"success"*>/WEB-INF/page/main.jsp</result>  </action>  </package>  因为struts2中如文件上传，数据验证，封装请求参数到action等功能都是由系统默认的defaultStack中的拦截器实现的，所以我们定义的拦截器需要引用系统默认的defaultStack，这样应用才可以使用struts2框架提供的众多功能。  如果希望包下的所有action都使用自定义的拦截器，可以通过<default-interceptor-ref name=“permissionStack”/>把拦截器定义为默认拦截器。注意：每个包只能指定一个默认拦截器。另外，一旦我们为该包中的某个action显式指定了某个拦截器，则默认拦截器不会起作用。 |

## execAndWait拦截器: 服务器忙时显示一个等待页面

|  |
| --- |
| struts.xml  <action name=*"login"* class=*"action.Login"*>  <interceptor-ref name=*"timer"* />  <interceptor-ref name=*"execAndWait"*>  <param name=*"delay"*>2000</param> //等待2秒进入wait.jsp, 如果2秒内服务器忙完了, 则不会进入wait.jsp  </interceptor-ref>  <interceptor-ref name=*"defaultStack"* />  <result name=*"wait"*>/wait.jsp</result>  <result name=*"success"*>/success.jsp</result>  </action>  Action  **public** String execute() **throws** Exception {  System.*out*.println("开始等待...");  Thread.*sleep*(5000); //服务器总共忙五秒,  System.*out*.println("等待结束");  **return** *SUCCESS* ;  }  wait.jsp中需要加入: <meta http-equiv=*"refresh"* content=*"1"*><%-- 刷新 --%>  浏览器效果:  等待2秒进入wait.jsp, 再等待3秒进入success.jsp |

## token拦截器: 防止表单重复提交

|  |
| --- |
| 在JSP页面<form>中加入令牌: <s:token/> |
| struts.xml 第一种: 重复提交则跳转到一个resut name=”invalid.token”的页面  <interceptor-ref name=*"* token*"* />  <resut name=”invalid.token”>/error.jsp</result> |
| struts.xml 第二种:如果发现重复提交, 则返回success对应的视图. 后台忽略提交操作  <interceptor-ref name=*"tokenSession"* /> |

## 自定义权限控制拦截器

|  |
| --- |
|  |

## timer拦截器: 控制台打印一次request到response之间等待的时间

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

# 校验

## 校验----手工编写代码方式

|  |
| --- |
| 继承ActionSupport方法;  过重写validate()或validateXxx()方法实现，  validate()方法会校验action中所有与execute方法签名相同的方法。  validateXxx()只会校验action中方法名为xxx的方法。其中Xxx的第一个字母要大写。  (公共校验部分可以放在一个单独的方法内被调用)  当某个数据校验失败时，我们应该调用addFieldError()方法往系统的fieldErrors添加校验失败信息（为了使用addFieldError()方法，action可以继承ActionSupport ），  如果系统的fieldErrors包含失败信息，struts2会将请求转发到名为input的result。在input视图中可以通过<s:fielderror/>显示失败信息。 |
| JSP页面  <FONT color=*"red"*>**<s:fielderror />**</FONT>(错误显示区域)  <form action="${pageContext.request.contextPath}*/first/manage\_add*" method=*"post"*>  用户名:<input type=*"text"* name=*"username"* />(用户名不能为空)<br />  手机号:<input type=*"text"* name=*"mobile"* />(手机号不能为空,并且要求符合1开头,第二位要求是3/5/8, 长度为11位) <br />  <input type=*"submit"* value=*"保存"* />  </form> |
| action.xml的package中必须配一个input视图  <result name=*"input"*>/index.jsp</result> <!-- 有且仅有一个input视图 --> |
| Action  **import** java.util.regex.Pattern;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** PersonManageAction **extends** ActionSupport {  **private** **static** **final** **long** *serialVersionUID* = 1L;  **private** String username;  **private** String mobile;  //省略了getXxx(), setXss();  @Override  **public** **void** validate() {  **if** (**this**.username == **null** || "".equals(**this**.username.trim())) {  **this**.addFieldError("username", "用户名不能为空");  }  **if** (**this**.mobile == **null** || "".equals(**this**.mobile.trim())) {  **this**.addFieldError("mobile", "手机号不能为空");  } **else** **if** (!Pattern.*compile*("^1[358]\\d{9}").matcher(**this**.mobile.trim()).matches()) {// 正则表达式  **this**.addFieldError("mobile", "手机号格式不正确");  }  **super**.validate();  }  /\*\*  \* 主方法  \* @return  \*/  **public** String add() {  **return** "message";  }  } |
| 校验----流程  1。**类型转换器**对请求参数执行类型转换，并把转换后的值赋给action中的属性。  2。如果在执行类型转换的过程中出现异常，系统会将异常信息保存到ActionContext，**conversionError拦截器**将异常信息封装到fieldErrors里，然后执行第3步。  如果类型转换没有出现异常，则直接进入第3步。  3。系统通过反射技术调用action中的**validateXxx()**方法，Xxx为方法名。  4。调用action中的**validate()**方法。  5。经过上面4步，如果系统中的fieldErrors存在错误信息（即存放错误信息的集合的size大于0)，系统自动将请求转发至名称为input视图。  如果系统中的fieldErrors没有任何错误信息，系统将执行action中的处理方法。 |

## 校验----基于XML配置方式

|  |
| --- |
| Action类需要继承ActionSupport， |
| 然后提供校验文件，如下：  <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.3//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.3.dtd">  <validators>  <field name=*"username"*> <!-- 指定要校验的类属性 -->  <field-validator type=*"requiredstring"*> <!-- 系统提供的校验器(*requiredstring:*字符串校验器) -->  <param name=*"trim"*>true</param> <!-- 校验之前是否去前后空格, true代表是 -->  <message>用户名不能为空!</message> <!-- 校验失败时,设置进去这个值 -->  </field-validator>  </field>  <field name=*"mobile"*>  <field-validator type=*"requiredstring"*>  <param name=*"trim"*>true</param>  <message>手机号不能为空!</message>  </field-validator>    <field-validator type=*"regex"*>  <param name=*"expression"*> <![CDATA[^1[358]\d{9}$]]> </param>  <message>手机号格式不正确!</message>  </field-validator>  </field>  </validators> |
| 对**所有方法**进行校验的配置方法:( ActionClassName-validation.xml)  该文件需要和action类放在同一个包下，校验文件的取名应为:ActionClassName-validation.xml规则，其中ActionClassName为action的简单类名，-validation为固定写法。如果Action类为cn.itcast.action.UserAction，那么该文件的取名应为：UserAction-validation.xml。 |
| 对**指定方法**进行校验的配置方法:( ActionClassName-ActionName-validation.xml)  该文件需要和action类放在同一个包下，校验文件的取名应为:ActionClassName-ActionName-validation.xml，其中ActionName为struts.xml中为action配置的名称。例如：在实际应用中，常有以下配置：  <action name="user\_\*" class="cn.itcast.action.UserAction" method="{1}">  <result name="success">/WEB-INF/page/message.jsp</result>  <result name="input">/WEB-INF/page/addUser.jsp</result>  </action>  UserAction中有以下两个处理方法：  public String add() throws Exception{  ....  }  public String update() throws Exception{  ....  }  要对add()方法实施验证，校验文件的取名为： UserAction-user\_add-validation.xml  要对update()方法实施验证，校验文件的取名为： UserAction-user\_update-validation.xml |
| 系统提供的校验器如下：  required (必填校验器,要求field的值不能为null)  requiredstring (必填字符串校验器,要求field的值不能为null,并且长度大于0,默认情况下会对字符串去前后空格)  stringlength(字符串长度校验器,要求field的值必须在指定的范围内,否则校验失败,minLength参数指定最小长度,maxLength参数指定最大长度,trim参数指定校验field之前是否去除字符串前后的空格)  regex(正则表达式校验器,检查被校验的field是否匹配一个正则表达式.expression参数指定正则表达式,caseSensitive参数指定进行正则表达式匹配时,是否区分大小写,默认值为true)  int(整数校验器,要求field的整数值必须在指定范围内,min指定最小值,max指定最大值)  double(双精度浮点数校验器,要求field的双精度浮点数必须在指定范围内,min指定最小值,max指定最大值)  fieldexpression(字段OGNL表达式校验器,要求field满足一个ognl表达式,expression参数指定ognl表达式,该逻辑表达式基于ValueStack进行求值,返回true时校验通过,否则不通过)  email(邮件地址校验器,要求如果field的值非空,则必须是合法的邮件地址)  url(网址校验器,要求如果field的值非空,则必须是合法的url地址)  date(日期校验器,要求field的日期值必须在指定范围内,min指定最小值,max指定最大值)  conversion(转换校验器,指定在类型转换失败时,提示的错误信息)  visitor(用于校验action中的复合属性,它指定一个校验文件用于校验复合属性中的属性)  expression(OGNL表达式校验器,expression参数指定ognl表达式,该逻辑表达式基于ValueStack进行求值,返回true时校验通过,否则不通过,该校验器不可用在字段校验器风格的配置中) |
| required 必填校验器  <field-validator type="required">  <message>性别不能为空!</message>  </field-validator>  requiredstring 必填字符串校验器  <field-validator type="requiredstring">  <param name="trim">true</param>  <message>用户名不能为空!</message>  </field-validator>  stringlength：字符串长度校验器  <field-validator type="stringlength">  <param name="maxLength">10</param>  <param name="minLength">2</param>  <param name="trim">true</param>  <message><![CDATA[产品名称应在1-10个字符之间]]></message>  </field-validator>  int：整数校验器  <field-validator type="int">  <param name="min">1</param>  <param name="max">150</param>  <message>年龄必须在1-150之间</message>  </field-validator>  字段OGNL表达式校验器  <field name="imagefile">  <field-validator type="fieldexpression">  <param name="expression"><![CDATA[imagefile.length() > 0]]></param>  <message>文件不能为空</message>  </field-validator>  </field>  email：邮件地址校验器  <field-validator type="email">  <message>电子邮件地址无效</message>  </field-validator>  regex：正则表达式校验器  <field-validator type="regex">  <param name="expression"><![CDATA[^13\d{9}$]]></param>  <message>手机号格式不正确!</message>  </field-validator> |
| 多个校验文件的执行顺序  1.某个action提供了ActionClassName-validation.xml和ActionClassName-ActionName-validation.xml两种规则的校验文件：  2.Accion有继承关系(eg: BoyAction继承FatherAction)  访问上面名为user的action，系统先搜索到  FatherAction-validation.xml  FatherAction-login-validation.xml，  接着搜索到  BoyAction-validation.xml  BoyAction-login-validation.xml。  校验规则是这四个文件的总和。  总结:  先所有方法校验后单个方法校验  先有father亲后有boy |

# 国际化

|  |
| --- |
| **资源文件命名规范**  准备资源文件，资源文件的命名格式如下： (语言的范围大于国家, 所以语言放在国家前面, 国家是世界独一无二的,所以大写)  baseName\_language\_country.properties  baseName\_language.properties  baseName.properties  其中baseName是资源文件的基本名，我们可以自定义，但language和country必须是java支持的语言和国家。如：  中国大陆： baseName\_zh\_CN.properties  美国： baseName\_en\_US.properties |
| **注册国际化文件基本名**  使用上面的资源文件，在struts.xml中使用常量加载全局资源文件如下：  <constant name="struts.custom.i18n.resources" value="itcast" /> <!-- itcast为资源文件的基本名。 --> |
| * jsp中引入标签: <%@ taglib uri="/struts-tags" prefix="s" %> * 在JSP页面中使用<s:text name=“”/>标签输出国际化信息： <s:text name=“user”/>，name为属性文件中的key * 在表单标签中，通过key属性指定属性文件中的key，如： <s:textfield name="realname" key="user"/> * 在Action类中，可以继承ActionSupport，使用getText()方法，该方法的第一个参数用于指定属性文件中的key。 |
| 国际化—**输出带占位符**的国际化信息  属性文件中的内容如下：  welcome={0},你好,欢迎来到{1}.  在jsp页面中输出带占位符的国际化信息  <s:text name="welcome">  <s:param>  <s:property value="realname"/> <!-- realname是从值栈中取的 -->  </s:param>  <s:param>传智播客</s:param>  </s:text>  显示结果: 老金, 你好, 欢迎来到传智播客.  在Action类中获取带占位符的国际化信息，  可以使用getText(String key, String[] args)或getText(String aTextName, List args)方法。  国际化—**JSP中使用**  使用<s:i18n>标签指定某个特定的资源文件中取数据  <s:i18n name=*"包名.类名"*>  <s:text name=*"keyName"* />  </s:i18n>  如果要访问某个包下的资源文件，如下：  <s:i18n name=“*包名.package*">  <s:text name="*keyName*" />  </s:i18n>  表单中使用国际化资源文件:  <s:textfield name=”realname” key=”*keyName*” /> |
| 国际化—**搜索key的顺序**  可以为某个action单独指定资源文件，  在java的包下放置package\_language\_country.properties资源文件，package为固定写法，处于该包及子包下的action都可以访问该资源。  (类名) ActionClassName\_language\_country.properties  (包名, 父包) PackageName\_language\_country.properties  全局资源文件(默认) baseName\_language\_country.properties  Action范围:  当前Action类对应资源  当前Action类实现接口对应资源  当前Action类父类对应资源  包范围  当前包对应资源  当前包父包对应资源  全局范围资源文件  Struts2默认资源文件 |

# 异常处理

|  |  |
| --- | --- |
| 局部异常集 | <action name=*"login"* class=*"cn.cstp.org.action.LoginAction"*>  <result name=*"login\_success"*>/WEB-INF/page/login\_success.jsp</result>  <result name=*"login\_fail"*>/WEB-INF/page/login\_fail.jsp</result>  <result name=*"input"*>/index.jsp</result>    <exception-mapping result=*"exception"* exception=*"java.lang.notPointException" /*>  <result name=*"exception"* type=*"dispatcher"*>/WEB-INF/page/exception.jsp</result>  </action> |
| 全局异常集 | <global-results>  <result name=*"exception"* type=*"dispatcher"*>/WEB-INF/page/exception.jsp</result>  </global-results>  <global-exception-mappings>  <exception-mapping result=*"exception"* exception=*"java.lang.Exception" /*>  </global-exception-mappings> |

# OGNL

## OGNL表达式语言

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

|  |
| --- |
| OGNL是(Object Graphic Navigation Language对象图导航语言)    Struts 2中的OGNL Context就是ActionContext(上下文)，结构示意图如下:  当Struts2接受一个请求时，它迅速创建ActionContext,ValueStack,action对象。然后action被存到ValueStack，所以action的实例变量可以被OGNL访问。    标准的OGNL会设定一个根对象（root对象）。假设使用标准OGNL表达式来求值（不使用Struts 2的OGNL表达式），如果OGNL上下文（OgnlContext Map类型）有两个对象：  foo对象，在OgnlContext中名称为foo；  bar对象，在OgnlContext中名称为bar。  同时foo对象被设置为根对象（root）。则利用下面的OGNL表达式求值：   1. #foo.blah // 返回foo.getBlah() 2. blah // 返回foo.getBlah() ，因为foo为根对象 3. #bar.blah //返回bar.getBlah()   访问Ognl Context中的对象需要使用#符号标注命名空间，如#bar，如果要访问的属性属于根对象，则可以省略命名空间，直接访问该属性。  由于ValueStack(值栈)是Struts 2中OGNL的根对象，如果用户需要访问值栈中的对象，则可以直接通过下面的EL表达式访问ValueStack(值栈)中对象的属性：  ${foo} //获得值栈中某个对象的foo属性  如果访问其他Context中的对象，由于不是根对象，在访问时，需要加#前缀。   * **request对象**：用来访问HttpServletRequest属性（attribute）的Map，例如#request.userName或者#request['userName']，相当于调用request.getAttribute("userName")。 * **session对象**：用来访问HttpSession，例如#session.userName或者#session['userName']，相当于调用session.getAttribute("userName")。 * **application对象**：用于访问ServletContext，例如#application.userName或者#application['userName']，相当于调用Servlet的getAttribute("username")。 * **attr对象**：用于按page->request->session->application顺序访问其属性。 * **parameters对象**：用于访问HTTP的请求参数，例如#parameters.userName或者#parameters['userName']，相当于调用request.getParameter("username")。   #request.username查找顺序: request.getAttribute("username") --> request.getParameter(“username”) |

## OGNL能访问哪些东西:

|  |
| --- |
| 可以调用 值栈 的 普通方法:<s:property value=*"user.getUsername()"*/><br>  可以调用 普通类 的 构造方法:<s:property value=*"new cn.cstp.org.vo.Student('username', 'password').username"*/><br>  可以调用 普通类 的 静态属性:<s:property value=*"@cn.cstp.org.action.LoginAction@NAME"*/><br>  可以调用 普通类 的 静态方法:<s:property value=*"@java.lang.Math@floor(3.14)"*/><br>  可以调用 默认类 的 静态方法:<s:property value=*"@@.floor(3.14)"*/><br> //(java.lang.Math类)  数组和list一样,但长度不可变,所以使用数组的地方一般用链表代替.  可以获取list: <s:property value=*"testList"* /><br>//按加入顺序输出list中的数据,  可以获取list中的某个元素:<s:property value=*"testList[0]"*/><br>//类似数组的下标导航  可以获取set: <s:property value=*"testSet"* /><br>//无序输出set中的数据, 重复元素被忽略掉  可以获取set中的某个元素:<s:property value=*"testSet.iterator.next "*/><br>//由于set无序, 所以不可用脚标导航, 只能用iterator迭代  可以获取map: <s:property value=*"testMap"* /><br>//无序输出map中的键值对  可以获取map中的某个元素:<s:property value=*"testMap['key1']"*/><br>//由于set无序, 所以不可用脚标导航, 只能用['键']导航  输出list中对象, 需要覆写对象的toString()方法. |

## OGNL可以访问集合的伪属性(无get, set方法)

|  |  |
| --- | --- |
| Map, List, Set 共有 | size, isEmpty |
| Map | keys, values |
| List | iterator |
| Set | iterator |
| iterator | next, hasNext |
| Enumeration | next, hasNext, nextElement, hasMoreElements |

## ValueStack(值栈, 直接访问)

|  |
| --- |
| 生命周期=request, 每次请求产生一个值栈 |

## OGNL中#的使用:

|  |  |  |  |
| --- | --- | --- | --- |
| parameters | 通过get传来的参数:  (url地址栏里的参数&method=”get”提交的表单)  包含当前HTTP请求参数的Map | #parameters.id[0] | request.getParameter(“id”); |
| request | 包含当前HttpServletRequest的属性(attribute)的Map | #request.username | request.getAttribute(“username”); |
| session | 包含当前HttpSession的属性(attribute)的Map | #session.username | session.getAttribute(“username”); |
| application | 包含当前应用的ServletContext的属性(attribute)的Map | #application.username | application.getAttribute(“username”); |
| attr | request>session>application顺序访问其属性(attribute) | #application.username | request.getAttribute(“username”);-->  session.getAttribute(“username”);-->  application.getAttribute(“username”); |
| actionContext | Action类的成员 | #username |  |

## OGNL中%的使用:

|  |
| --- |
| 1. 用%{}可以去除存在值栈中的Action对象, 直接调用它的方法.   例如:如果你的Action如果继承你ActionSupport, 那么在页面标签中, 用%{getText(‘key’)}的方式可以拿出国际化信息.   1. %{}将本来是文本属性解析为ognl, 对于本来就是ognl的属性不起作用. |

## OGNL中$的使用:

|  |
| --- |
| 国际化资源文件 和 Struts.xml中, 在值栈中匹配相应的值 |
| JSP中先到request中getParameter(username), 若找不到再到值栈中寻找username |

将ognl表达式解析成字符串:

|  |
| --- |
| <s:property value=”’username’” /> |

ognl表达式默认过滤其中的html标签 escape=”false” //让他逃走吗? false不让他逃走(解析出来)

## 查看值栈中的信息:

|  |
| --- |
| <s:debug></s:debug> |

## 选择取值栈中的对象

|  |  |
| --- | --- |
| top语法 | <s:property value=”[1].top.studet” /> //值栈中的第二个对象: [1].top |
| N语法 | <s:property value=”[1].studet” /> //值栈中的第二个对象: [1] |
| @语法(调用值栈中的静态成员) | <s:property value=”@vs1@get()”/> //vs1和vs都代表栈顶对象 |

## 压栈

每次请求(request)创建一个值栈, Action对象自动压栈.

|  |  |
| --- | --- |
| 手动往值栈中压对象(压栈) | ActionContext.getContext().getValueStack().push(new Student(“jin”, 23)); |

## 获得集合对象的子集(ONGL的选择, 投影功能, 不推介, 数据在查询出来就应该过滤好)

|  |
| --- |
| 除了in和not in之外，OGNL还允许使用某个规则获得集合对象的子集，常用的有以下3个相关操作符。  #this: 当前对象  [1]:第2个元素. 结果放在List中,输出时无[]  .{表达式} 结果放在List中, 输出时数据包含在[]中  ?：所有符合逻辑的元素。  ^：第一个元素。  $：最后一个元素。 |

## (兰布达表达式, 用于递归算法)Lambda Expressions

|  |
| --- |
| <s:property value="#fib = :[#this==0 ? 0 : #this==1 ? 1 : #fib(#this-2)+#fib(#this-1)], #fib(11)" />  注释:  #fib 函数名  :[] 递归函数体(表达式)写在里面  #this 函数的参数  4的阶乘:  <s:property value=*"#jie = :[#this==1 ? 1 : #this\*#jie(#this-1) ], #jie(4)"*/> |

# 覆盖struts 2的主题样式

|  |
| --- |
| 这个fielderror.ftl就是自定义的错误显示样式 |

# SSH整合

## Struts2+Spring2.5+Hibernate3.3整合开发

|  |
| --- |
| 下面给出整合开发时Struts 2、 Hibernate、Spring需要的JAR。  struts2-core-2.x.x.jar :Struts 2框架的核心类库  xwork-2.x.x.jar :XWork类库，Struts 2在其上构建  ognl-2.6.x.jar :对象图导航语言（Object Graph Navigation Language），struts2框架通过其读写对象的属性  freemarker-2.3.x.jar :Struts 2的UI标签的模板使用FreeMarker编写  commons-fileupload-1.2.x.jar 文件上传组件，2.1.6版本后需要加入此文件  struts2-spring-plugin-2.x.x.jar ：用于struts2集成Spring的插件  **hibernate核心安装包下的**(下载路径：http://www.hibernate.org/，点击“Hibernate Core”右边的“Downloads”):  hibernate3.jar  lib\bytecode\cglib\hibernate-cglib-repack-2.1\_3.jar  lib\required\\*.jar  **hibernate 注解安装包下的**(下载路径：www.hibernate.org，点击“Hibernate Annotations”右边的“Downloads”):  hibernate-annotations.jar  lib\ejb3-persistence.ja  hibernate-commons-annotations.jar  **Hibernate针对JPA的实现包**(下载路径：www.hibernate.org，点击“Hibernate Entitymanager”右边的“Downloads”):  hibernate-entitymanager.jar  lib\test\log4j.jar  slf4j-log4j12.jar  **Spring安装包下的**  dist\spring.jar  lib\c3p0\c3p0-0.9.1.2.jar  lib\aspectj\aspectjweaver.jar  aspectjrt.jar  lib\cglib\cglib-nodep-2.1\_3.jar  lib\j2ee\common-annotations.jar  lib\log4j\log4j-1.2.15.jar  lib\jakarta-commons\commons-logging.jar  MYSQL数据库驱动jar |

**Tag(标签库)**

**Generic Tags**

**Control Tags**

**if** **/elseIf /****else**

|  |
| --- |
| if/elseif/else  <s:if test=*"'book1' in {'book1', 'book2'}"*>  ...  </s:if>  <s:elseif test=*""*>  ...  </s:elseif>  <s:else>  ...  </s:else> |

**append**

**generator**

**iterator**

|  |
| --- |
| <s:iterator value=*"#request.books"* status=*"zhuangTai"* var=*"LingShiBianLiang"* >  <s:property value=*"#LingShiBianLiang"* /> <%--等同于<s:property/>--%>  <s:property value=*"#zhuangTai.count"*/> 遍历过的总数  <s:property value=*"#zhuangTai.index"*/> 当前的元素索引  <s:property value=*"#zhuangTai.even"*/> 当前是偶数吗?  <s:property value=*"#zhuangTai.odd"*/> 当前是奇数吗?  <s:property value=*"#zhuangTai.first"*/> 当前是第一个吗?  <s:property value=*"#zhuangTai.last"*/> 当前是最后一个吗?  </s:iterator> |
| List:<br>  使用OGNL表达式生成一个list对象  <s:set name=”bookList” value=”{‘book1, ‘book2’, ‘book3’}” />  <s:iterator value=*"#list"* >  <s:property /><br>  </s:iterator>  <p /> |
| <%--使用OGNL表达式生成一个Map对象--%>  <s:set var=*"myMap"* value="#{'k1':'v1', 'k2':'v2'}" />  Map1:<br>  <s:iterator value=*"myMap"* >  <s:property /><br>  </s:iterator>  Map2:<br>  <s:iterator value=*"myMap"* >  <s:property value=*"key"* />=  <s:property value=*"value"* /><br>  </s:iterator>  Map3:<br>  <s:iterator value=*"myMap"* >  <s:property value=*"#map.key"* />=  <s:property value=*"#map.value"* /><br>  </s:iterator> |

**merge**

**sort**

**subset**

**Data Tags**

**a**

**action**

**bean**

|  |
| --- |
| <s:bean name=*"action.Dog"* >//没有var, 创建的是一个Dog类型的匿名对象, bean结束即销毁  <s:param name=*"name"*>oudy</s:param>//给对象的属性name赋值oudy.  <s:property value=*"#myDog.name"*/> //访问匿名对象, 取值只能写在bean里面  </s:bean>  <s:bean name=*"action.Dog"* var=*"myDog"*>//创建一个Dog类型的myDog对象  <s:param name=*"name"*>oudy</s:param>//给对象的属性name赋值oudy.  </s:bean>  <s:property value=*"#myDog.name"*/> |

**date**

**debug**

|  |
| --- |
| <s:debug></s:debug>  取异常信息:  <s:property value=”errors.name[0]” />  ectionErrors: Action出异常  fieldErrors: 校验异常  errors是以上两个的和, 里面装的是map<key, value[]>  name是key, 对应的value是一个数组 |

**i18n**

|  |
| --- |
| 指定Action的资源文件(找得到指定的-->本Action-->本包-->父包-->默认  <s:i18n name=*"包名.ActionName"*>  <s:text name=*"keyName"* />  </s:i18n>  指定包下的资源文件(找不到就追溯到父包下找)  <s:i18n name=*"包名.package"*>  <s:text name=*"keyName"* />  </s:i18n>  依次找: 本Action-->本包-->父包-->默认  <s:i18n name=*"错误路径或空字符串"*>  <s:text name=*"keyName"* />  </s:i18n> |

**include**

|  |
| --- |
| 动态嵌入一个jsp或servlet的输出(不提倡, 中文易乱码)  <s:include value=*"inner.jsp"*>  <s:param name=*"参数名"*>参数值</s:param>//向inner.jsp传参数  </s:include>  inner.Jsp里#parameters拿到的是outer.jsp的数据,  因此在拿inner.jsp中的数据用:<%=request.getParameter(“name”)%> |

**param**

|  |
| --- |
| 保存数据  <s:param name=*"变量名"* value=*"变量值"* /> |

**property**

|  |
| --- |
| 输出变量的值  <s:property value=*"变量名"*/> |

**push**

**set**

|  |
| --- |
| <s:set var=*"给数据起的别名, name属性已经被var取代"* value=*"设置进去的数据值"* id=*""* scope=*"page/request/session/application或action"* />  scope默认等于request和actionContext |

**text**

|  |
| --- |
| <s:text name=*"keyName"* />  国际化资源文件必须和Action同包,同名.  如果没有找到相应的message, tag body将被当做默认的message;  如果没有tag body, message的name会作为默认的message |

**url**

|  |
| --- |
| 不传参数url:  <a href="<s:url action=*"login"* namespace=*"p1"* />">i18n</a>  传参数url)  <s:url id=*"urlName"* value=*"/condition.jsp"* > //这个value默认不当做OGNL表达式, 可以%{OGNL表达式}  <s:param name=*"name"*>Max</s:param>  </s:url>  <s:a href=*"%{urlName}"*>if/elseif/else</s:a> |

**UI Tags**

**Form Tags**

**checkbox**

**checkboxlist**

|  |
| --- |
| <s:checkboxlist name=*"list"* list=*"{'book1','book2’,‘book3’}"* value=*"{'book2’, ‘book3’}"* />  <s:checkboxlist name=*"map"* list="#{1:'v1', 2:'v2',3:'v3'}" listKey=*"key"* listValue=*"value"* value=*"{1, 2}"*/>  <s:checkboxlist name=*"javaBean"* list=*"#request.persons"* listKey=*"id"* listValue=*"name"*/> |

**combobox**

**doubleselect**

**head**

**file**

**form**

**hidden**

**label**

**optiontransferselect**

**optgroup**

**password**

**radio**

|  |
| --- |
| <s:radio name=*"list"* list=*"{'book1','book2’,‘book3’}"* value=*"{'book2’, ‘book3’}"* />  <s:radio name=*"map"* list="#{1:'v1', 2:'v2',3:'v3'}" listKey=*"key"* listValue=*"value"* value=*"{1, 2}"*/>  <s:radio name=*"javaBean"* list=*"#request.persons"* listKey=*"id"* listValue=*"name"*/> |

**reset**

**select**

|  |
| --- |
| <s:select name=*"list"* list=*"{'book1','book2’,‘book3’}"* value=*"{'book2’, ‘book3’}"* />  <s:select name=*"map"* list="#{1:'v1', 2:'v2',3:'v3'}" listKey=*"key"* listValue=*"value"* value=*"{1, 2}"*/>  <s:select name=*"javaBean"* list=*"#request.persons"* listKey=*"id"* listValue=*"name"*/> |

**submit**

**textarea**

**textfield**

**token**

**updownselect**

**Non-Form UI Tags**

**actionerror**

**actionmessage**

**component**

**div**

**fielderror**

|  |
| --- |
| <s:fielderror fieldName=”name” theme=”simple” /> |

**Ajax Tags**

**a**

|  |
| --- |
| <s:a |

**autocompleter**

**bind**

**datetimepicker**

**div**

**head**

**submit**

**tabbedpanel**

**textarea**

**tree**

**treenode**



**模板相关**

|  |  |
| --- | --- |
| theme=”xhtml/ simple/ ajax” | 设置当前标签的主题 |
| template | 设置标签引用的组件 |
| templateDir | 组件所在的目录 |

鼠标

|  |  |
| --- | --- |
| onclick | 单击 |
| ondbclick | 双击 |
| onmousedown | 按下鼠标 |
| onmousemove | 松开鼠标 |
| onmouseover | 悬停在上 |
| onmouseout | 移开 |
| onkeypress | 按了某个键盘键 |
| onkeydown | 按下某个键盘键 |
| onkeyup | 松开某个键盘键 |
| onfocus | 得到焦点 |
| onblur | 失去焦点 |
| onchange | select/checkbox被改变 |
| onselect | 被选中 |

tooltip相关

|  |  |
| --- | --- |
| tooltip | 鼠标提示信息 |
| tooltipConfig | 提示信息的样式, 字体 |

其它属性

|  |  |
| --- | --- |
| cssClass | class属性 |
| cssStyle | css属性 |
| disabled | 不可操作标签, 灰色显示 |
| label | 表单标签前面的说明文字 |
| labelPosition | 表单标签说明文字位置 |
| name |  |
| required | 是否是必填表单(在前加※) |
| requiredposition | 必填表单※所在位置 |
| size | 表单长度 |
| tabIndex | 按下tab键是第几个 |
| title | 标签标题 |
| value | 初始值 |

下拉列表框—属性

|  |  |
| --- | --- |
| list | 父列表框的选项内容 |
| doubleList | 子列表框的选项内容 |
| name | 父下拉列表框提交数据的名称 |
| doubleName | 子下拉列表框提交数据的名称 |
| listKey | 父下拉列表框中被选定的内容 |
| listValue | 父下拉列表框在页面上显示的内容 |
| doubleListKey | 子下拉列表框中被选定的内容 |
| doubleListValue | 子下拉列表框在页面上显示的内容 |
| doubleSize | 子下拉列表框显示内容最大个数 |

doubleList参数

|  |
| --- |
| top属性指定父下拉列表与子下拉列表的级联关系  静态级联列表框:  动态级联列表框: |
|  |

**非表单UI**

|  |
| --- |
| 单行文本框 |
| 文本域 |
| 下拉列表 |
| 复选框(对应Action中的集合属性) |
| 单一复选框 |
| 文件选择组件 |
| 按钮 |

|  |
| --- |
| <s:set>标签  如果需要一个集合元素的时候（例如List对象或者Map对象），可以使用OGNL中同集合相关的表达式。  使用如下代码直接生成一个List对象：  <s:set name=*"list"* value=*"{'list1', 'list2', 'list3'}"* />  <s:iterator value=*"#list"* id=*"n"*>  <s:property value=*"n"* /><br />  </s:iterator>  生成一个Map对象：  <s:set name=*"map"* value=*"*#{'k1':'v1', 'k2':'v2'}" />  <s:iterator value=*"#map"* >  <s:property value=*"key"*/>=<s:property value=*"value"*/><br>  </s:iterator>  Set标签用于将某个值放入指定范围。  scope：指定变量被放置的范围，该属性可以接受application、session、request、 page或action。如果没有设置该属性，则默认放置在OGNL Context中。  value：赋给变量的值.如果没有设置该属性,则将ValueStack栈顶的值赋给变量。  对于集合类型，OGNL表达式可以使用in和not in两个元素符号。  in表达式用来判断某个元素是否在指定的集合对象中；  not in判断某个元素是否不在指定的集合对象中，如下所示。  **in表达式：**  <s:if test=" 'foo' in {'foo','bar'}">  在  </s:if>  <s:else> |
| 不在  </s:else>  **not in表达式：**  <s:if test=" 'foo' not in {'foo','bar'}">  不在  </s:if>  <s:else>  在  </s:else>  除了in和not in之外，OGNL还允许使用某个规则获得集合对象的子集，常用的有以下3个相关操作符。  ?：获得所有符合逻辑的元素。  ^：获得符合逻辑的第一个元素。  $：获得符合逻辑的最后一个元素。  例如代码：  <s:iterator value="books.{?#this.price > 35}">  <s:property value="title" /> - $<s:property value="price" /><br>  </s:iterator>  在上面代码中，直接在集合后紧跟.{}运算符表明用于取出该集合的子集，{}内的表达式用于获取符合条件的元素，本例的表达式用于获取集合中价格大于35的书集合。  public class BookAction extends ActionSupport {  private List<Book> books;  ....  @Override  public String execute() {  books = new LinkedList<Book>();  books.add(new Book("A735619678", "spring", 67));  books.add(new Book("B435555322", "ejb3.0",15));  }  }  property标签用于输出指定值：  <s:set name="name" value="'kk'" />  <s:property value="#name"/>  default：可选属性，如果需要输出的属性值为null，则显示该属性指定的值  escape：可选属性，指定是否格式化HTML代码。  value：可选属性，指定需要输出的属性值，如果没有指定该属性，则默认输出ValueStack栈顶的值。  id：可选属性，指定该元素的标识  iterator标签用于对集合进行迭代，这里的集合包含List、Set和数组。  <s:set name="list" value="{'zhangming','xiaoi','liming'}" />  <s:iterator value="#list" id="name" status="st">  <font color=  <s:if test="#st.odd">red</s:if>  <s:else>blue</s:else>>  <s:property value="name"/></font><br>  </s:iterator>  value：可选属性，指定被迭代的集合，如果没有设置该属性，则使用ValueStack栈顶的集合。  id：可选属性，指定集合里元素的id。  status：可选属性，该属性指定迭代时的IteratorStatus实例。该实例包含如下几个方法：  int getCount()，返回当前迭代了几个元素。  int getIndex()，返回当前迭代元素的索引。  boolean isEven()，返回当前被迭代元素的索引是否是偶数  boolean isOdd()，返回当前被迭代元素的索引是否是奇数  boolean isFirst()，返回当前被迭代元素是否是第一个元素。  boolean isLast()，返回当前被迭代元素是否是最后一个元素。  <s:set name="age" value="21" />  <s:if test="#age==23">  23  </s:if>  <s:elseif test="#age==21">  21  </s:elseif>  <s:else>  都不等  </s:else> |
| url标签  <s:url action="helloworld\_add" namespace="/test">  <s:param name="personid" value="23"/>  </s:url>  生成类似如下路径：  /struts/test/helloworld\_add.action?personid=23  红色部分为内容路径。  “%”符号的用途是在标志的属性为字符串类型时，计算OGNL表达式的值。  <s:set name="myurl" value="'http://www.foshanshop.net'"/>  <s:url value="#myurl" /><br>  <s:url value="%{#myurl}" />  输出结果：  #myurl  http://www.foshanshop.net |
| 表单标签\_checkboxlist复选框  **如果集合为list**  **<s:checkboxlist**  **name="list"**  **list="{'Java','.Net','RoR','PHP'}"**  **value="{'Java','.Net'}"/>**  生成如下html代码：  <input type="checkbox" name="list" value="Java" checked="checked"/><label>Java</label>  <input type="checkbox" name="list" value=".Net" checked="checked"/><label>.Net</label>  <input type="checkbox" name="list" value="RoR"/><label>RoR</label>  <input type="checkbox" name="list" value="PHP"/><label>PHP</label>  **如果集合为MAP**  **<s:checkboxlist**  **name="map"**  **list="#{1:'瑜珈用品',2:'户外用品',3:'球类',4:'自行车'}"**  **listKey="key"**  **listValue="value"**  **value="{1,2,3}"/>**  生成如下html代码：  <input type="checkbox" name="map" value="1" checked="checked"/><label>瑜珈用品</label>  <input type="checkbox" name="map" value="2" checked="checked"/><label>户外用品</label>  <input type="checkbox" name="map" value="3" checked="checked"/><label>球类</label>  <input type="checkbox" name="map" value="4"/><label>自行车</label>  **如果集合里存放的是javabean**  <%  Person person1 = new Person(1,"第一个");  Person person2 = new Person(2,"第二个");  List<Person> list = new ArrayList<Person>();  list.add(person1);  list.add(person2);  request.setAttribute("persons",list);  %>  <s:checkboxlist  name="beans"  list="#request.persons"  listKey="personid"  listValue="name"/>  **Personid和name为Person的属性**  生成如下html代码：  <input type="checkbox" name=“beans" value="1"/><label>第一个</label>  <input type="checkbox" name=“beans" value="2"/><label>第二个</label> |
| 表单标签\_radio单选框  该标签的使用和checkboxlist复选框相同。  如果集合里存放的是javabean(personid和name为Person的属性)  <s:checkboxlist name="beans" list="#request.persons" listKey="personid" listValue="name"/>  生成如下html代码：  <input type="radio" name="beans" id="beans1" value="1"/><label>第一个</label>  <input type="radio" name="beans" id="beans2" value="2"/><label>第二个</label>  如果集合为MAP  <s:radio name="map" list="#{1:'瑜珈用品',2:'户外用品',3:'球类',4:'自行车'}" listKey="key" listValue="value“ value="1"/>  生成如下html代码：  <input type="radio" name="map" id="map1" value="1"/><label for="map1">瑜珈用品</label>  <input type="radio" name="map" id="map2" value="2"/><label for="map2">户外用品</label>  <input type="radio" name="map" id="map3" value="3"/><label for="map3">球类</label>  <input type="radio" name="map" id="map4" value="4"/><label for="map4">自行车</label>  如果集合为list  <s:radio name="list" list="{'Java','.Net'}" value="'Java'"/>  生成如下html代码：  <input type="radio" name="list" checked="checked" value="Java"/><label>Java</label>  <input type="radio" name="list" value=".Net"/><label>.Net</label> |
| 表单标签\_select下拉选择框  <s:select name="list" list="{'Java','.Net'}" value="'Java'"/>  <select name="list" id="list">  <option value="Java" selected="selected">Java</option>  <option value=".Net">.Net</option>  </select>  <s:select name="beans" list="#request.persons" listKey="personid" listValue="name"/>  <select name="beans" id="beans">  <option value="1">第一个</option>  <option value="2">第二个</option>  </select>  <s:select name="map" list="#{1:'瑜珈用品',2:'户外用品',3:'球类',4:'自行车'}" listKey="key" listValue="value" value="1"/>  <select name="map" id="map">  <option value="1" selected="selected">瑜珈用品</option>  <option value="2">户外用品</option>  <option value="3">球类</option>  <option value="4">自行车</option>  </select> |
| <s:token />标签防止重复提交  用法如下：  第一步：在表单中加入<s:token />  <s:form action="helloworld\_other" method="post" namespace="/test">  <s:textfield name="person.name"/><s:token/><s:submit/>  </s:form>  第二步：  <action name="helloworld\_\*" class="cn.itcast.action.HelloWorldAction" method="{1}">  <interceptor-ref name="defaultStack" /> <!-- 应用Struts2提供的核心功能, 这个必须写在拦截器前面 -->  <interceptor-ref name="token" /> <!-- 拦截器 -->  <result name="invalid.token">/WEB-INF/page/message.jsp</result> <!-- 当发生重复提交,就发送到这个视图 -->  <result>/WEB-INF/page/result.jsp</result>  </action>  以上配置加入了“token”拦截器和“invalid.token”结果，因为“token”拦截器在会话的token与请求的token不一致时，将会直接返回“invalid.token”结果。  在debug状态,控制台出现下面信息,是因为Action中并没有struts.token和struts.token.name属性,我们不用关心这个错误：  严重: ParametersInterceptor - [setParameters]: Unexpected Exception caught setting 'struts.token' on 'class xxx: Error setting expression 'struts.token' with value '[Ljava.lang.String;@39f16f'  严重: ParametersInterceptor - [setParameters]: Unexpected Exception caught setting 'struts.token.name' |