OGNL是Object Graphic Navigation Language（对象图导航语言）的缩写，

它是一个开源项目。 Struts2框架使用OGNL作为默认的表达式语言。

OGNL相对其它表达式语言具有下面几大优势：

1、支持对象方法调用，如xxx.doSomeSpecial()；

2、支持类静态的方法调用和值访问，表达式的格式:

@[类全名（包括包路径）]@[方法名 |  值名]，例如：

@java.lang.String@format('foo %s', 'bar')

或@tutorial.MyConstant@APP\_NAME；

设置 struts.ognl.allowStaticMethodAccess=true

3、访问OGNL上下文（OGNL context）和ActionContext；

4、支持赋值操作和表达式串联，如price=100, discount=0.8,

calculatePrice()，这个表达式会返回80；

5、操作集合对象。

Ognl 有一个上下文（Context）概念，说白了上下文就是一个MAP结构，它实现

了java.utils.Map的接口.

ValueStack实际是一个接口,在Struts2中利用OGNL时，实际上使用的是实现了该接口的OgnlValueStack类,这个类是Struts2利用OGNL的基础

* **ValueStack(值栈): 贯穿整个 Action 的生命周期(每个 Action 类的对象实例都拥有一个**

**ValueStack 对象). 相当于一个数据的中转站. 在其中保存当前**

**Action 对象和其他相关对象.**

* **Struts 框架把 ValueStack 对象保存在名为 “struts.valueStack” 的请求属性中,request中**

ObjectStack: Struts 把动作和相关对象压入 ObjectStack 中--List

ContextMap: Struts 把各种各样的映射关系(一些 Map 类型的对象) 压入 ContextMap 中

**Struts 会把下面这些映射压入 ContextMap 中**

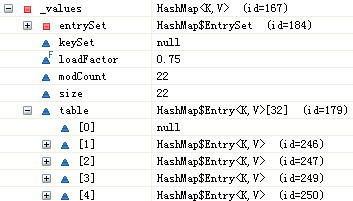
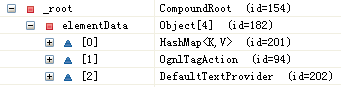
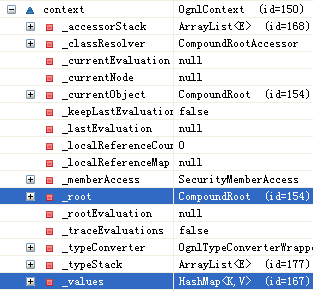
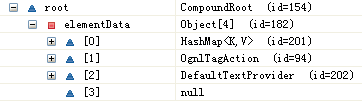
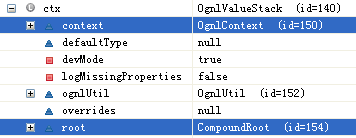
parameters: 该 Map 中包含当前请求的请求参数

request: 该 Map 中包含当前 request 对象中的所有属性

session: 该 Map 中包含当前 session 对象中的所有属性

application:该 Map 中包含当前 application 对象中的所有属性

attr: 该 Map 按如下顺序来检索某个属性: request, session, application



* **OgnlValueStack 类包含两个重要的属性 一个root和一个context。**

**\* 其中root本质上是一个ArrayList.**

**\* 而context 是一个Map（更确切的说是一个OgnlContext对象)**

* **在这个OgnlContext对象（context）中，有一个默认的顶层对象 \_root，OGNL访问context中这个默认顶层对象中的元素时，是不需要#号的，直接通过元素的名称来进行访问，**
* **而访问其他对象时，如 request、session、attr等，则需要#号引用。**

**注:Struts2将OgnlValueStack的root对象赋值给了OgnlContext 中的\_root对象，在OgnlValueStack的root对象中，保存着调用Action的实例，因此，在页面上通过Struts2标签访问Action 的属性时，就不需要通过#号来引用**

**总结:ognl Context包含 ObjectStack属性和ContextMap属性**

**当Struts2接受一个请求时，会迅速创建ActionContext，ValueStack，action 。然后把action存放进ValueStack，所以action的实例变量可以被OGNL访问。**

**注意： Struts2中，OGNL表达式需要配合Struts标签才可以使用。如：<s:property value="name"/>**

**向值栈保存数据**

**valueStack.set("username", "username\_valueStack");**

**valueStach.push("zrgk");**

在JSP中获取ValueStack数据

<s:property value="[0].top"/> //取值栈顶对象

* 直接在root中查找数据

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

* 在OgnlContext中获取数据

request:<s:property value="#request.username"/>

session:<s:property value="#session.username"/>

application:<s:property value="#application.username"/>

attr:<s:property value="#attr.username"/>

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

使用EL表达式能够访问valueStack中对象的属性

原因是Struts2对HttpServletRequest作了进一步的封装。简略代码如下：

public class StrutsRequestWrapper extends HttpServletRequestWrapper {

public StrutsRequestWrapper(HttpServletRequest req) {

super(req);

}

public Object getAttribute(String s) {

......

ActionContext ctx = ActionContext.getContext();

Object attribute = super.getAttribute(s);//先从request范围获取属性值

if (ctx != null) {

if (attribute == null) {//如果从request范围没有找到属性值,即从ValueStack中查找对象的属性值

......

ValueStack stack = ctx.getValueStack();

attribute = stack.findValue(s);

......

}

}

return attribute;

}

}

**#号的用法: OGNL会设定一个根对象（root对象），**

**Action中代码:**

**ValueStack valueStack=ServletActionContext.*getContext*().getValueStack();**

**//set方法放置对象到map中,map再放入到栈(List集合)上**

**valueStack.set("student", new Student());**

**valueStack.set("employee", new Employee());**

**//直接放置对象到栈(List集合)上**

**valueStack.getRoot().add(0,new Student());**

**valueStack.getRoot().add(1,new Employee());**

**Jsp页面:**

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

**age::<s:property value="age"/><br>**

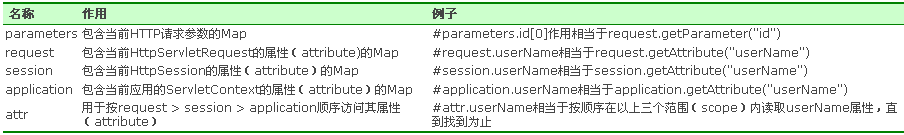
**number::<s:property value="number"/><br>**

**salary:<s:property value="salary"/><br>**

**用法1:访问OGNL上下文和Action上下文，#相当ActionContext.getContext()**

**1、 如果访问其他Context中的对象，由于他们不是根对象，所以在访问时，**

**需要添加#前缀。**



也可写为#request[‘userName’]或#session[‘userName’]或#appliction[‘userName’]

**集合的投影(只输出部分属性)** **（过滤）**

**1、集合的投影(只输出部分属性)**

collectionName.{ *expression* }

**<s:iterator value="allList.{name}" var="person">**

**<s:property/> <br>**

**</s:iterator>**

**1) 集合的过滤有以下三种方式：**     a.“?#”：过滤所有符合条件的集合，如：users.{?#this.age > 19}；  
     b.“^#”：过滤第一个符合条件的元素，如：users.{^#this.age > 19}；  
     c.“$#”：过滤最后一个符合条件的元素，如：users.{$#this.age > 19} 。  
 **.2) this 表示集合中的元素；**

**<s:iterator value="allList.{?#this.age>25}" var="person">**

**<s:property value="name"/> xxxxxx <s:property value="age"/> <br>**

**</s:iterator>**

**3、集合的投影和过滤**

**投影（过滤）操作返回的是一个集合，可以使用索引取得集合中指定的**

**元素，如：users.{?#this.age > 19}[0]**

**<s:iterator value="allList.{?#this.age>25}.{name}" var="person">**

**<s:property/><br>**

**</s:iterator>**

**<s:iterator value="allList.{?#this.age>25}[0]" var="person">**

**<s:property/><br>**

**</s:iterator>**

**用法3:构造Map，如#{‘foo1’:‘bar1’, ‘foo2’:‘bar2’}。**

(**%用法**)

**“%”符号的用途是在标签的属性值被理解为字符串类型时，告诉执行环境%{}里的是OGNL表达式。**

**形式一: { }中ognl表达式**

**Action中的代码:**

**ServletActionContext.*getRequest*().setAttribute("username", "username\_request");**

**jsp页面:**

**<s:textfield name="name" label="%{#request.username} "/>**

**形式二: { }中值用 ’ ’引起来,此时不再是ognl表达式,而是普通的字符串**

**jsp页面:**

**<s:textfield name="name" label="%{'foo'}"/>**

(**$用法**)

**“$”有两个主要的用途**

**\* 用于在国际化资源文件中，引用OGNL表达式**

**\* 在Struts 2配置文件中，引用OGNL表达式**

Properties 文件中：

**item.ognlname=${#request.msgxxx}**

**<s:text name="item.ognlname"></s:text>**

**<action >**

**<result name="ognl">/ognl/ongl.jsp?msg=${#request.msgxx}</result>**

**</action>**

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

**property标签用于输出指定值：**

<s:property value=“#name" **default**="a **default** value" />

**\* 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。**

**例子** **:**

**<s:property/>:输出栈顶的值**

**输出: username=username\_valueStack, msgxx=msg\_valueStack**

**<s:property value="%{'<hr> hr的使用'}" escape="false" />**

**输出**:**<hr> hr的使用**

**注:escape=“false”时,hr作为html标签使用**

**set标签用于将某个值放入指定范围。**

**var:变量的名字,name,id和var表达的含义是一样的,name,id被var替代**

**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，则默认放置在值栈的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。**

**例子:**

**<s:set value="#request.username" var="xxx“ scope=“request” /><br>**

**<s:property value=“#request.xxx" /><br>**

**<s:set value="#request.username" var="xxx" scope="page" /><br>**

**<s:property value="#attr.xxx" /><br>**

**<s:set value="#request.username" var="xxx" /><br>**

**<s:property value="xxx" /><br>**

**或者：<s:property value=“#xxx" /><br>**

iterator标签

**Iterator:标签用于对集合进行迭代，这里的集合包含List、Set和数组。**

**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。**

**var： 可选属性，引用变量的名称.**

**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。**

例子：

Action类，组织allList集合

for(int i=0;i<10;i++){

Person p=new Person();

p.setId(i);

p.setName("tom"+i);

p.setAge(20+i);

allList.add(p);

}

ServletActionContext.getRequest().setAttribute("allList", allList);

**<s:iterator value="#request.allList“ var=“person”>**

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

**<s:property value=“#person.name”/>**

**</s:iterator>**

**<s:iterator value="#request.allList" var="person" begin="2" end="7" step="2">**

**<s:property value="#person.name"/><s:property value="#person.age"/><br>**

**</s:iterator>**

**<s:iterator value="allList" status="st">**

**st.getCount():<s:property value="#st.count"/> &nbsp;&nbsp;**

**st.getIndex():<s:property value="#st.index"/> &nbsp;&nbsp;**

**st.isEven():<s:property value="#st.even"/>&nbsp;&nbsp;**

**st.isOdd():<s:property value="#st.odd"/>&nbsp;&nbsp;**

**st.isFirst:<s:property value="#st.first"/>&nbsp;&nbsp;**

**st.isLast():<s:property value="#st.last"/><br>**

**</s:iterator>**

**<table border="1">**

**<s:iterator value="allList" var="person" status="st">**

**<tr class=<s:property value="#st.even?'even':'odd'"/> >**

**<td><s:property value="#person.name"/></td>**

**</tr>**

**</s:iterator>**

**</table>**

**if/elseif/else** **基本的流程控制.‘If’标签可单独使用也可以和‘Else If’标签和(或)一个多个‘Else’一起使用**

**<s:if test="#age==23">**

**23**

**</s:if>**

**<s:elseif test="#age==21">**

**21**

**</s:elseif>**

**<s:else>**

**都不等**

**</s:else>**

**<table border="1">**

**<s:iterator value="allList" var="person">**

**<tr>**

**<td><s:property value="#person.name"/></td>**

**<td><s:property value="#person.age"/></td>**

**<td><s:if test="#person.age<24">少年</s:if>**

**<s:elseif test="#person.age<26">中年</s:elseif>**

**<s:else>老年</s:else>**

**</td>**

**</tr>**

**</s:iterator>**

**</table>**

**url:该标签用于创建url,可以通过"param"标签提供request参数.**

**value:如果不提供就用当前action,使用value后缀必须加.action**

**action:用来生成url的action,如果没有则使用value**

**namespace :命名空间**

**var:引用变量的名称.**

**使用action<br>**

**<s:url action="ognlTagAction\_test" namespace="/ognl" var="myurl">**

**<s:param name="name" value="%{'张老师'}"></s:param>**

**<s:param name="id" value="12"></s:param>**

**</s:url>**

**注意：当使用url时，应配合使用struts2的a标签**

**<s:a href="%{#url}" >xxxx</s:a><br>**

**使用value<br>**

**<s:url value="ognlTagAction\_test.action" namespace="/ognl" var="myurl">**

**<s:param name="id" value="12"></s:param>**

**<s:param name="cnname" value="%{'zhang'}"></s:param>**

**</s:url>**

**<s:a href="%{#myurl}" >xxxx</s:a><br>**

**比较：<a href=“<s:property value=‘%{#myurl}’/>”>链接</a><br>的不同**

**ognl操作集合**

**Action中:**

**private List allList=new ArrayList();**

**Jsp:**

**集合的长度:<s:property value="allList.size"/>**

**输出第一个元素:<s:property value="allList[0]"/>~~**

**<s:property value="allList[0].name"/>~~**

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

**ognl也可直接创建集合对象**

**Jsp:**

**<s:iterator value="{1,2,3,4}">**

**<s:property/> <br>**

**</s:iterator>**

**<s:iterator value="{'s1','s2','s3','s4'}" var="s">**

**<s:property value="#s"/> <br>**

**</s:iterator>**

**使用ognl操作map**

**ognl用多种方式使用#号，每种是不同的.动态map对于动态单选按扭组很有用.**

**创建map与创建list语法很相似,不同的是map前需要加"#"号.**

**<s:iterator value="#{'key01':'value01','key02':'value02'}">**

**<s:property/> <br>**

**</s:iterator>**

**<s:iterator value="#{'key01':'value01','key02':'value02'}">**

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

**</s:iterator>**

**<s:iterator value="#{'key01':'value01','key02':'value02'}" var="map">**

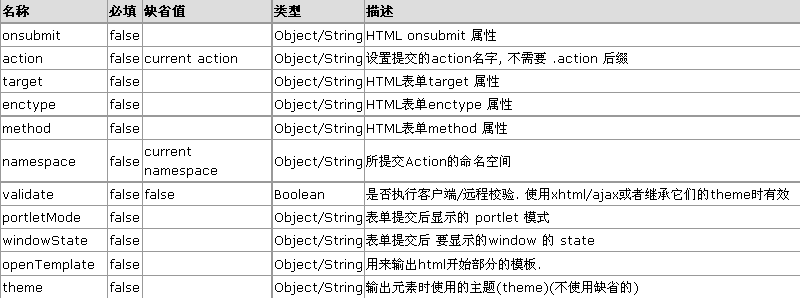
**key=<s:property value="#map.key"/>**

**value=<s:property value="#map.value"/> <br>**

**</s:iterator>**

**UI标签：**

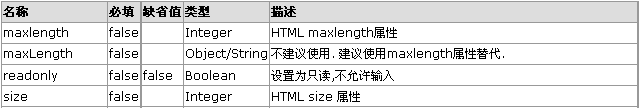
**form 标签**



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



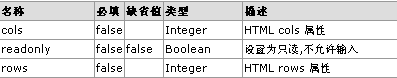
password 标签扩展自 textfield 标签, 多了一个 showPassword 属性.该属性是布尔型.

默认值为 false, 它决定着在表单回显时是否显示输入的密码. true显示密码

submit 标签

reset 标签

textarea 标签



checkboxlist 标签

**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="id" listValue="name"/>

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.**

**该标签的使用和checkboxlist复选框相同。**

**如果集合里存放的是javabean(id和name为Person的属性)**

**<s:radio 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>**

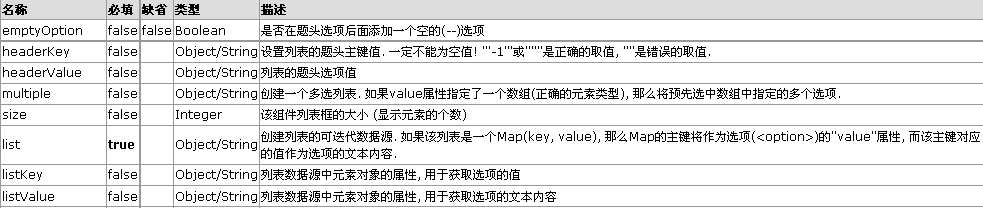
**<input type="radio" name="map" id="map3" value="3"/><label for="map3">球类</label>**

**<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'"/>**

select 标签将呈现一个select 元素



如果集合为list

<s:select name="list" list="{'Java','.Net'}" value="'Java'"/>

生成如下html代码：

<select name="list" id="list">

<option value="Java" selected="selected">Java</option>

<option value=".Net">.Net</option>

</select>

如果集合里存放的是javabean(id和name为Person的属性)

<s:select name="beans" list="#request.persons" listKey="personid" listValue="name"/>

如果集合为MAP

<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"/>

生成如下html代码：

<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>