OGNL表达式：

Object Graphic Navigation Language 对象图导航语言

它的写法就是在浏览器使用：对象.对象的方式

明确：使用OGNL表达式取值时，需要借助struts2的标签库

字符串 转 OGNL 表达式加： %{OGNL}

OGNL 表达式 转 字符串 加：

"'OGNL'"

常用标签：

s:property标签：

作用:数据输出到浏览器上。 和<%=jsp表达式%>作用类似

属性：value：取值是一个OGNL表达式，把value属性取值所对应的内容输出到浏览器上。

OGNL表达式的最基本使用：

1、借助struts2的s:property标签输出数据

把s:property标签value属性取值所对应的内容输出到浏览器上

OGNL的最基本使用：<s:property value="OGNLExpression"/> （输出这个值是OGNLExpression对就应的数据而不是OGNLExpression）

2、把一个OGNL表达式强制看成一个普通的字符串，用引号把OGNL表达式括起来就是一个字符串了。

写法：

输出字符串-简写：<s:property value="'OGNLExpression22'"/>

输出字符串-标准写法：<s:property value="%{'OGNLExpression111'}"/>

3、OGNL表达式功能强大：

表达式：是用于获取数据的。一般情况下都有一个明确的返回值。

OGNL表达式：它不仅可以用于取值。还可以用于赋值。

取值都是我们干的。赋值都是框架干的。

3.1、OGNL表达式它支持普通的方法调用。(和EL表达式的区别：EL表达式只能调用静态的自定义方法)

普通方法调用：<s:property value="'OGNLExpression".length()'/>

普通方法调用：<s:property value="'OGNLExpression".toUpperCase()'/>

普通方法调用：<s:property value="'OGNLExpression".split("E")'/>

3.2、OGNL表达式访问静态属性

OGNL表达式访问静态属性有固定格式：

@包名.包名...类名@属性名称

访问静态属性：<s:property value="@java.lang.Integer@MAX_VALUE"/>

3.3、OGNL表达式访问静态方法

固定格式：

@包名.包名...类名@方法名称

访问静态方法：<s:property value="@java.lang.Math@random()"/>

struts2默认情况下是不开启静态方法访问的。

需要在配置文件中开启

```
<struts>
 <!-- 开启静态方法访问 -->
 <constant name="struts.ognl.allowStaticMethodAccess" value="true"></constant>
</struts>
```

3.4、OGNL表达式可以操作集合 List/Map

首先，使用HTML标签创建一个性别的选择

性别：<input type="radio" name="gender" value="男">男

<input type="radio" name="gender" value="女">女

list属性取值是一个OGNL表达式。

当我们写一个{}，表示创建了一个List集合。List list = new ArrayList();

括号中的内容就是集合元素。list.add("男");list.add("女");

s:radio标签会遍历集合。生成input标签，type是radio

<s:radio list="{'男','女'}" label="性别" name="gender"></s:radio> 这就等同于上面的两行**HTML**

3.5、struts2的迭代标签：

s:iterator：它的作用就是用于遍历集合的

属性：

value：取值是一个**OGNL**表达式

var:用了该属性：**var**的取值就是一个普通的字符串。它会把**var**值作为**key**，把当前遍历的对象作 为**value**，存入**contextMap**中

没用该属性：**iterator**标签遍历时会把当前遍历的对象压入栈顶。当下次遍历之前，再弹栈。

begin：遍历的开始索引

end：遍历的结束索引

step：遍历的步长。

status：遍历的计数器对象

getCount:当前遍历的序号。从**1**开始

getIndext:当前遍历的索引。从**0**开始

isOdd:是否是奇数行

isEven:是否是偶数行

isFirst:是否第一行

isLast:是否最后一行

```
<%-- 不使用 var属性
<s:iterator value="customers">
        <TR style="FONT-WEIGHT: normal; FONT-STYLE: normal; BACKGROUND-COLOR: white; TEXT-DECORATION: none">
         <TD>${custName }</TD>
         <TD>${custLevel }</TD>
         <TD>${custSource }</TD>
         <TD>${custIndustry }</TD>
         <TD>${custAddress }</TD>
         <TD>${custPhone }</TD>
         <a href="${pageContext.request.contextPath }/customer/CustomerServlet?method=editCustomerUI&custId=${custId}">修改</a>
         <a href="${pageContext.request.contextPath }/customer/CustomerServlet?method=removeCustomer&custId=${custId}">删除</a>
        </s:iterator>
--%>
```

使用了 var属性

```
<s:iterator value="customers" var="cust">
          <TD><s:property value="#cust.custName"/></TD>
          <TD><s:property value="#cust.custLevel"/></TD>
          <TD><s:property value="#cust.custSource"/></TD>
          <TD><s:property value="#cust.custIndustry"/></TD>
          <TD><s:property value="#cust.custAddress"/></TD>
          <TD><s:property value="#cust.custPhone"/></TD>
</s:iterator>
```

使用了Status属性

<s:iterator value="customers" var="cust" status="vs">

以下可以设置不相行显示的颜色

<TR ${vs.odd ? "style='background-color:#FFFACD'":"style='background-color:#F0F8FF'"} style="FONT-WEIGHT: normal; FONT-STYLE: normal; BACKGROUND-COLOR: white; TEXT-DECORATION: none">

<TD><s:property value="#cust.custName"/></TD>

<TD><s:property value="#cust.custLevel"/></TD>

<TD><s:property value="#cust.custSource"/></TD>

<TD><s:property value="#cust.custIndustry"/></TD>

<TD><s:property value="#cust.custAddress"/></TD>

<TD><s:property value="#cust.custPhone"/></TD>

<TD>

<a href="${pageContext.request.contextPath }/customer/CustomerServlet?method=editCustomerUI&custId=<s:property value="#cust.custId"/>">修改****

** **

<a href="${pageContext.request.contextPath }/customer/CustomerServlet?method=removeCustomer&custId=<s:property value="#cust.custId"/>">删除****

</TD>

</TR>

</s:iterator>

3.5、操作map

用**HTML**标签创建的

性别：<input type="radio" name="gender" value="male">男

<input type="radio" name="gender" value="female">女

用**struts2**标签创建

当我们写了一个#{}，表示创建了一个Map。 Map map = new HashMap();

括号中的内容就是map中的数据。map.put("male","男");map.put("female","女");

s:radio标签会遍历map,生成input标签

<input type="radio" name="gender" value="male"/>男

<s:radio list="#{'male':'男','female':'女'}" label="性别" name="gender"></s:radio>