**Struts2第4天课程**

**【学习目标】**

## 1，能够独立使用OGNL表达式

1. 说出OGNL表达式语言的概念
2. 说出值栈的概念
3. 说出值栈存储结构
4. 说出值栈内部对象
5. 分析值栈存取原理
6. 独立使用EL表达式获取值栈数据

## 2，能够独立使用OGNL表达式符号语法

1. 独立使用#号
2. 独立使用%号
3. 独立使用$号

## 3，能够独立使用Struts2标签库

1. 独立使用<s:property> 标签
2. 独立使用<s:iterator> 标签
3. 独立使用<s:if> <s:elseif> <s:else> 标签
4. 独立使用<s:a> 标签

## 4，能够独立使用struts2 UI 相关标签 From Tag

1. 独立使用<s:form> 表单元素标签
2. 独立使用textfield, password, hidden 标签
3. 独立使用submit、reset、label 标签
4. 独立使用textarea 标签
5. 独立使用radio、checkboxlist、select 标签
6. 设置form主题样式

# OGNL表达式

## OGNL表达式是什么

OGNL是Object Graphic Navigation Language（对象图导航语言）的缩写，是一个使用简单、功能强大的、开源的表达式语言，可以方便地操作任何的对象属性、方法等。

Struts2框架使用OGNL作为默认的表达式语言，主要用于页面的取值。它类似于EL表达式语言，但比EL语法强大很多。

EL Expression Language 表达式语言， 主要用来获取 JSP页面四个域范围数据 （page、 request、 session、 application ）

OGNL = EL +对象操作 ${name.toUpperCase()}

OGNL表达式的三个要素  
        \* 表达式            -- OGNL的表达式，编写表达式，OGNL负责解析表达式，从而存值和取值  
        \* Root对象        -- 根对象  
        \* Context对象    -- OGNL的上下文对象，运行的环境

测试的代码

|  |
| --- |
| //访问对象的方法  @Test  **public** **void** run1() **throws** OgnlException{  OgnlContext context = **new** OgnlContext();  // 获取对象的方法  Object obj = Ognl.getValue("'helloworld'.length()", context, context.getRoot());  System.***out***.println(obj);  }  // 获取OGNL上下文件的对象  @Test  **public** **void** run2() **throws** OgnlException{  OgnlContext context = **new** OgnlContext();  context.put("name", "美美");  // 获取对象的方法  Object obj = Ognl.getValue("#name", context, context.getRoot());  System.***out***.println(obj);  }  // 从root栈获取值  @Test  **public** **void** run3() **throws** OgnlException{  OgnlContext context = **new** OgnlContext();  Customer c = **new** Customer();  c.setCust\_name("haha");  context.setRoot(c);  String name = (String) Ognl.getValue("cust\_name", context, context.getRoot());  System.out.println(name);  } |

## OGNL的用法详解

【提示】先看后学。

1. Java对象的直接访问

例如：<s:property value=*"'igeek'"*/>

1. 实例方法调用:访问对象的方法

语法：object.doSomething()，object是对象实例

例如：<s:property value=*"'igeek'.toUpperCase()"*/>

1. 静态方法调用（类的静态方法或静态变量的访问）

语法：@[类全名(包括包路径)]@[方法名|值名]

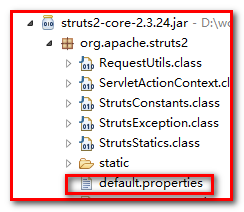
例如：<s:property value=*"@java.lang.Math@max(10,21)"*/>

类的静态方法或静态变量的调用访问。

格式：@[类全名(包括包路径)]@[方法名|值名]，如：@java.lang.Math@max(10,20)、@com.igeek.MyConstant@APP\_NAME

注意：必须通过配置struts2的常量来开启静态方法调用功能：

struts.ognl.allowStaticMethodAccess=true



1. 赋值操作或表达式串联。--可以运算

例如：<s:property value=*"1+2"*/>

OGNL可以访问上下文(OGNL Context)和ActionContext。(值栈相关访问-重要)

【示例】

目标：分别演示前几种基本用法。

参考步骤：创建web项目：struts2\_d04\_c01

【第一步】搭建struts2的环境：导入jar、struts.xml、web.xml。

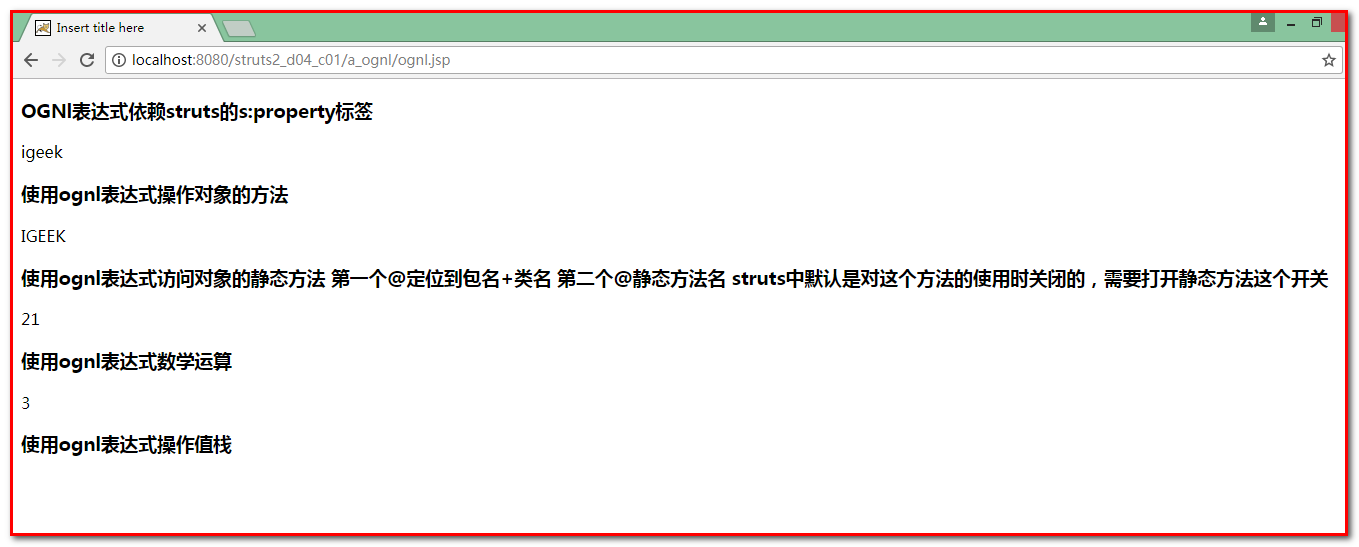
在WebContent下面创建a\_ognl文件夹，并且创建ognl.jsp页面，引入struts标签库，编写表达式代码。

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"*  prefix=*"s"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <h3>OGNl表达式依赖struts的s:property标签</h3>  <s:property value=*"'igeek'"*/><br/>  <h3>使用ognl表达式操作对象的方法</h3>  <s:property value=*"'igeek'.toUpperCase()"*/>  <h3>使用ognl表达式访问对象的静态方法  第一个@定位到包名+类名  第二个@静态方法名  struts中默认是对这个方法的使用时关闭的，需要打开静态方法这个开关  </h3>  <s:property value=*"@java.lang.Math@max(10,21)"*/>  <h3>使用ognl表达式数学运算</h3>  <s:property value=*"1+2"*/>  <h3>使用ognl表达式操作值栈</h3>  </body>  </html> |

【第二步】struts.xml文件中的配置如下：

|  |
| --- |
| <!-- 开启静态方法的 调用 -->  <constant name=*"struts.ognl.allowStaticMethodAccess"* value=*"true"*></constant> |

【第三步】测试：



【小结】

1、ognl 表达式，需要结合struts2的<s:property > 标签进行使用，它的value属性支持ognl表达式

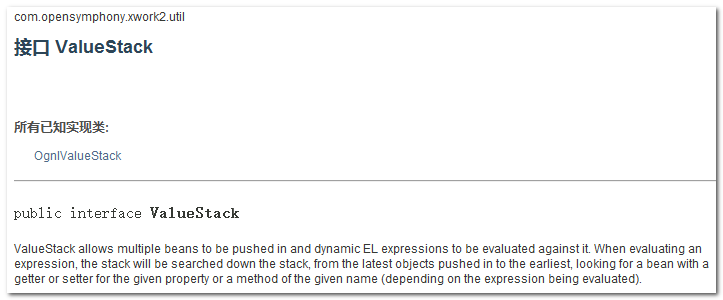
2、OGNL表达式最强大的功能是可以操作值栈(ValueStack)。

# 值栈概述（掌握）

## 值栈(ValueStack)是什么

值栈（ValueStack），是Struts2的数据中转站，栈中自动保存了当前Action对象和其他相关对象（包括常用的Web对象的引用，如request、session、application等），也可以手动保存自己的数据对象，同时也可以随时随地将对象从值栈取出或操作（通过OGNL表达式）

值栈（ValueStack），实际是一个接口的对象的称呼,接口是ValueStack类，实现类是OgnlValueStack类，该对象是Struts2利用OGNL的基础，或者说Struts2中Ognl使用都是基于值栈完成的。



Struts2框架将ValueStack对象保存在request域中，键为“struts.valueStack”，即值栈是request域中的一个对象，一个请求对应一个Action实例和一个值栈对象。

【示例1】值栈的获取方式：

1. request.getAttribute(“struts.valueStack”)：用的较少
2. ActionContext.getContext().getValueStack()：用的非常多:底层使用的还是第一种方式获取

ActionContext（Action上下文，工具类）

【第一步】创建web项目struts2\_d04\_c02，新建com.igeek\_01包，并在包中创建ValueStackAction类，类中代码如下：

|  |
| --- |
| **package** com.igeek\_01;  **import** org.apache.struts2.ServletActionContext;  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **import** com.opensymphony.xwork2.util.ValueStack;  **public** **class** ValueStackAction **extends** ActionSupport{      @Override  **public** String execute() **throws** Exception {  //获取值栈 方式一  ValueStack valueStack1 = (ValueStack) ServletActionContext.*getRequest*().getAttribute("struts.valueStack");    //获取值栈的方式二:第二种方式的底层使用的还是第一种方式  ValueStack valueStack2 = ActionContext.*getContext*().getValueStack();    //我们在Action中获取了两次值栈，那这两个值栈是同一个对象吗？  //通过对结果的观察，发现，两种获取值栈的方式获取的值栈是同一个对象  System.*out*.println(valueStack1==valueStack2);  System.*out*.println(valueStack1.hashCode());  System.*out*.println(valueStack2.hashCode());  //当我们发出了两次请求,两次请求的值栈的hashcode不一致，表明每次请求都会重新创建值栈    **return** *NONE*;  }      } |

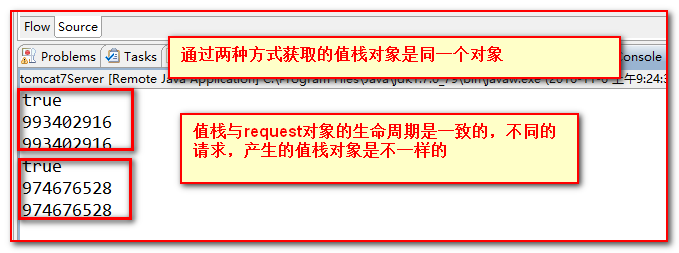
【第二步】配置struts.xml文件，具体配置如下：

|  |
| --- |
| <package name=*"default"* namespace=*"/"* extends=*"struts-default"*>  <action name=*"valueStack"* class=*"com.igeek\_01.ValueStackAction"*></action>  </package> |

【第三步】 测试

<http://localhost:8080/struts2_d04_c02/valueStack.action>

【运行结果】



【示例2】（重点）

目标：通过断点查看值栈的内部存储结构。

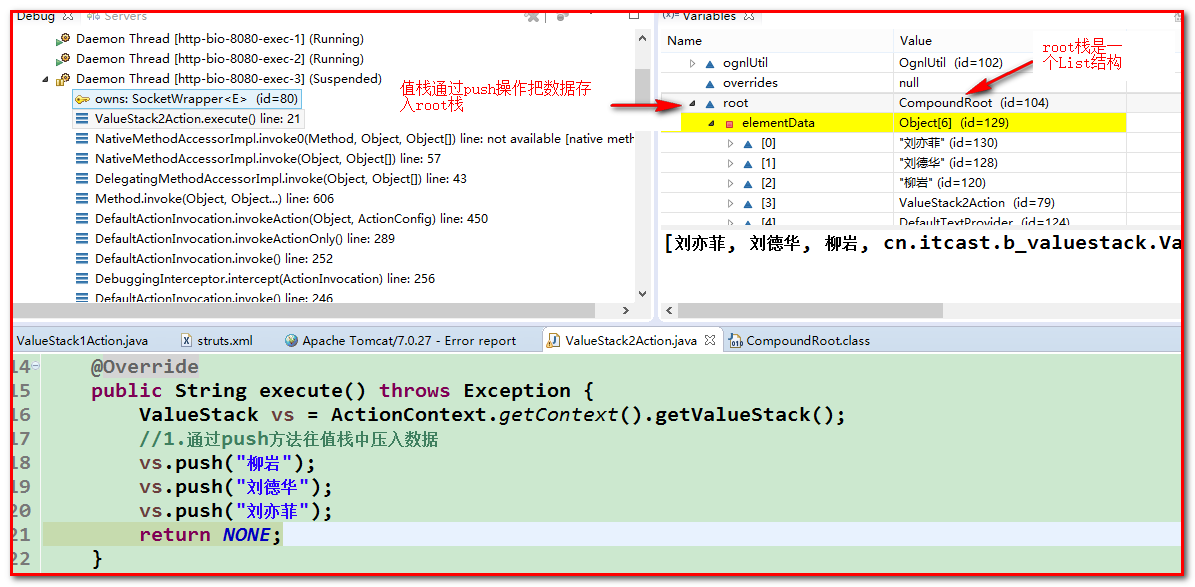
查看值是存在值栈的哪里？

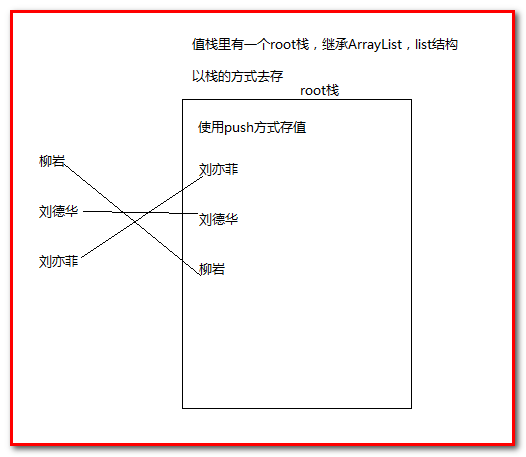
【第一步】

|  |
| --- |
| **package** com.igeek\_01;  **import** com.opensympionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **import** com.opensymphony.xwork2.util.ValueStack;  **public** **class** ValueStackAction1 **extends** ActionSupport{  @Override  **public** String execute() **throws** Exception {  ValueStack vs = ActionContext.*getContext*().getValueStack();  //1.通过push方法往值栈中压入数据  vs.push("柳岩");  vs.push("刘德华");  vs.push("刘亦菲");  **return** ***NONE***;  }  } |

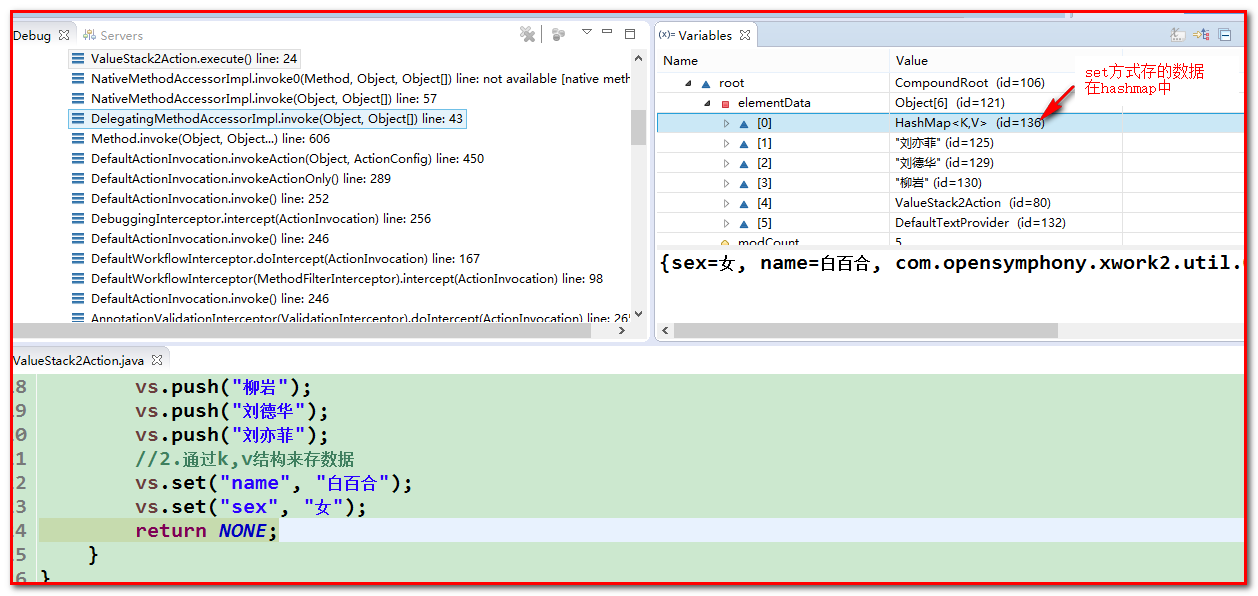
【第二步】

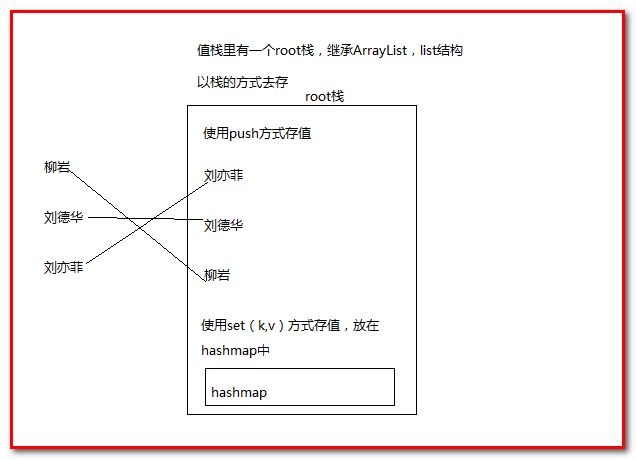
|  |
| --- |
| <!-- 查看valuesStack是什么？ -->  <action name=*"vs1"* class=*"com.igeek\_01.ValuesStackAction1"*></action> |



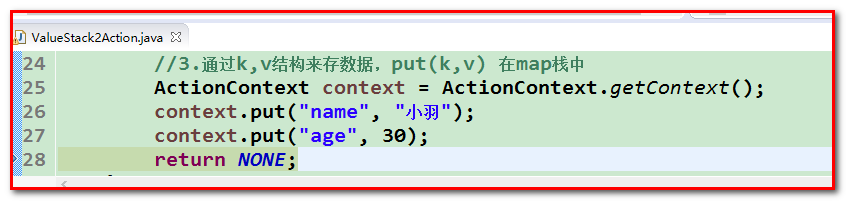


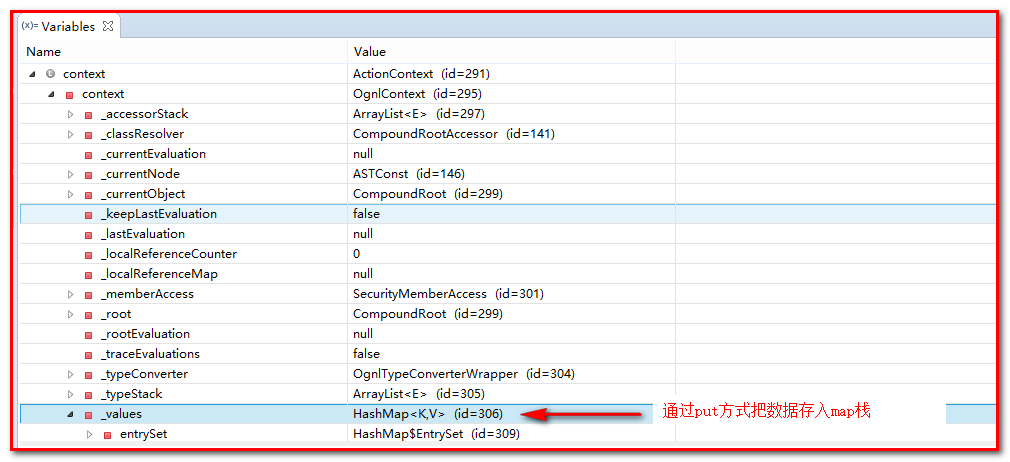
使用root栈来存数据时，取数据比较麻烦？ 使用Map来取数据比较方便

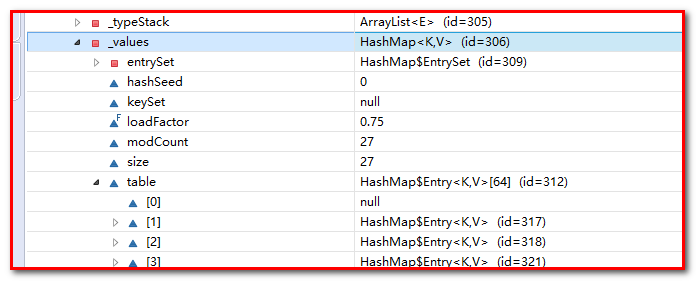


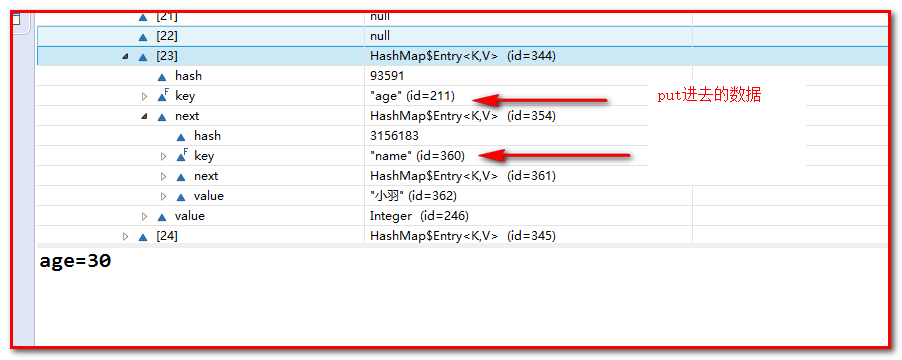


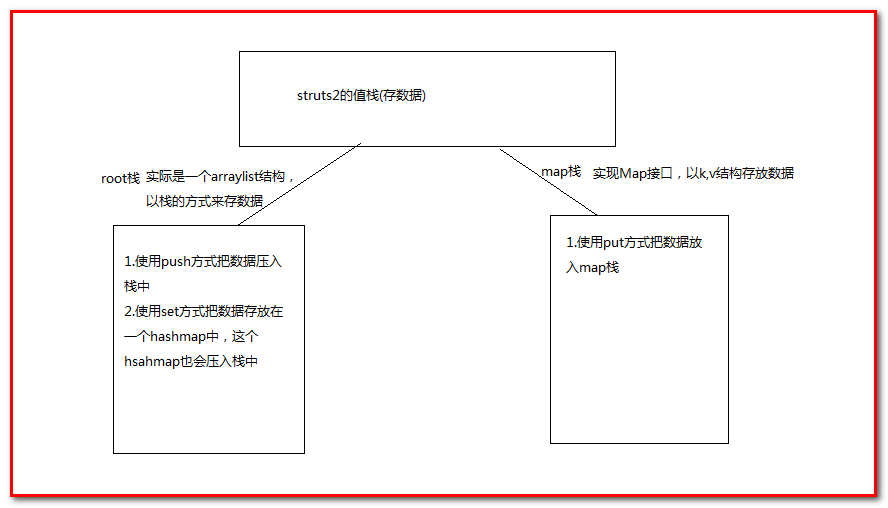
通过put方式存数据











【值栈小结】：

在值栈的内部有两个逻辑部分：

ObjectStack（对象栈）：又称为root栈，保存了Action的相关对象和动作，数据存储结构是List。

ContextMap（上下文栈）：又称为map栈，保存了各种映射关系，如常用的web对象的引用，数据存储结构是Map。

值栈存值有3种方式

针对root（compoundRoot）栈有push和set方法（stack）

针对map(ognlContext)栈有put方法（map）

就是java se中操作集合的方法

# 值栈的操作（掌握）

两部分：存值+取值

## 值栈的存值机制

【回顾知识了解】

栈的概念：

栈是一种数据结构，它按照先进后出的原则存储数据，即先进入的数据被压入栈底，最后进入的数据在栈顶，需要读取数据的时候，从栈顶开始弹出数据（即最后一个数据被第一个读出来）。

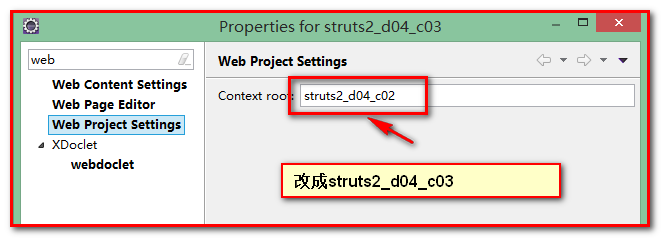
栈也被称为先进后出表，可进行插入和删除操作，插入称之为进栈（压栈）（push），删除称之为退栈（pop），允许操作的一端称为栈顶（top），另外一端就称为栈底（bottom）。栈底固定，而栈顶浮动。

对于栈就只能每次访问它的栈顶元素。

## 值栈的取值机制

|  |
| --- |
| OgnlValueStack类包含两个重要的属性，一个root（CompoundRoot）和一个context（OgnlContext）。  OGNL Context是struts2的数据中心，其结构示意图：    在这个OgnlContext对象中，有一个默认的顶层对象root（引用），俗称root栈，下面一个对象称之为Map栈。  对OGNL表达式的操作都是基于OgnlContext（map栈）对象，访问的规则如下：   * 如果访问 root栈内容（CompoundRoot 对象栈内容）， 不需要#，直接通过元素的名称来访问。 * 如果访问 Map栈内容 （如request、response、session、servletContext、attr、parameters）， 需要#key来引用访问，例如 #request.name 相当于 request.getAttribute("name" ) |

【示例1】在复制struts2\_d04\_c02项目，改成struts2\_d04\_c03,右键项目，选择Properties，在弹出框中修改web项目路径为struts2\_d04\_c03,如图：



valuestack.jsp页面中分别获取root栈和map栈中的内容,具体代码如下：

响应页面valuestack.jsp 代码如下：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'valuestack.jsp' starting page</title>  </head>    <body>  <!-- 访问值栈 -->  <!-- 访问root栈:如何访问这个匿名对象呢？ -->  <s:property value=*"[0].top"*/>|<s:property value=*"top"*/><br>  <!-- 如果我还想获取igeek1，如何获取呢？ -->  <s:property value=*"[1].top"*/><br>  <!-- 获取map栈中的内容 -->  <s:property value=*"#name"*/><br>    <!-- 如何获取root栈中的有名字对象 -->  <s:property value=*"name"*/>    <hr>  </body>  </html> |

## 值栈的默认搜索

在实际开发中，使用OGNL表达式时，会经常用到值栈的默认搜索，即自动搜索值栈，找到所需要的值。

【需求】在值栈中搜索需要的值

【示例】

【第一步】：修改ValueStackAction中的代码：

|  |
| --- |
| **package** com.igeek\_01;  **import** org.apache.struts2.ServletActionContext;  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **import** com.opensymphony.xwork2.util.ValueStack;  **public** **class** ValueStackAction2 **extends** ActionSupport {      **public** String execute() **throws** Exception {    //获取值栈的方式一:struts.valueStack区分大小写  ValueStack valueStack1 = (ValueStack) ServletActionContext.*getRequest*().getAttribute("struts.valueStack");  /\*//获取值栈的方式二：方式二底层使用的还是第一种方式获取的  ValueStack valueStack2 = ActionContext.getContext().getValueStack();    //证明两种方式获取的值栈是同一个对象  System.out.println(valueStack1.hashCode());  System.out.println(valueStack2.hashCode());  //==比较是内存地址  System.out.println(valueStack1==valueStack2);\*/    //向root栈存值  //方式一，以匿名的方式向root栈存值  //ActionContext.getContext().getValueStack().push("igeek1");  //ActionContext.getContext().getValueStack().push("igeek2");  //方式二：命名的方式压栈，有名字压栈  //ActionContext.getContext().getValueStack().set("name", "igeek3");      //向map栈存值  ActionContext.*getContext*().put("name", "map栈的name");        **return** *SUCCESS*;  }  } |

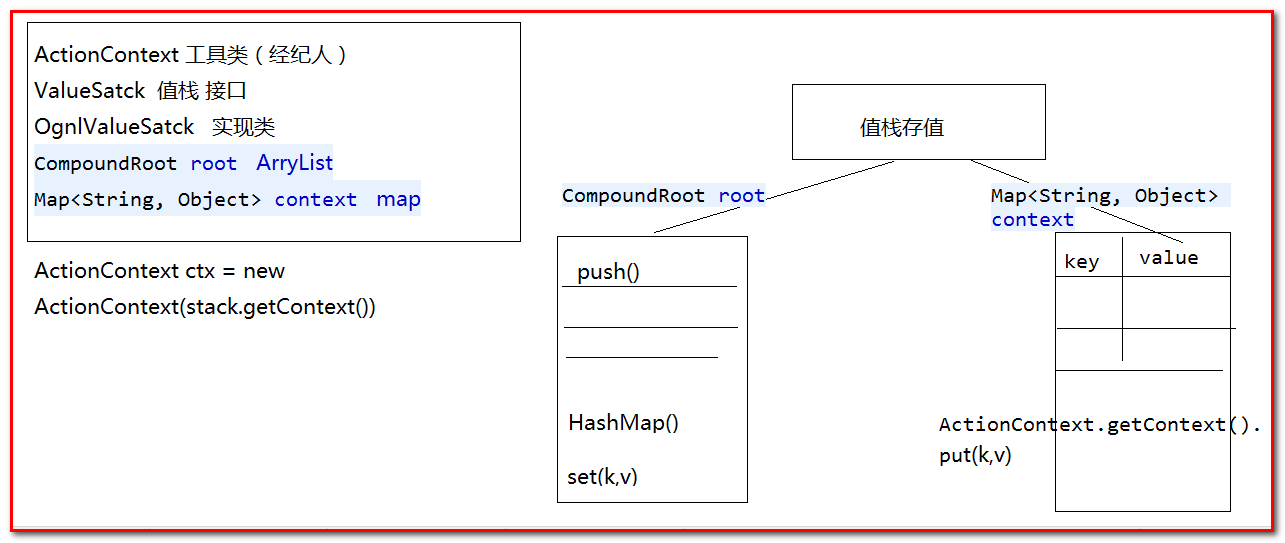
【第二步】编写valuestack.jsp页面

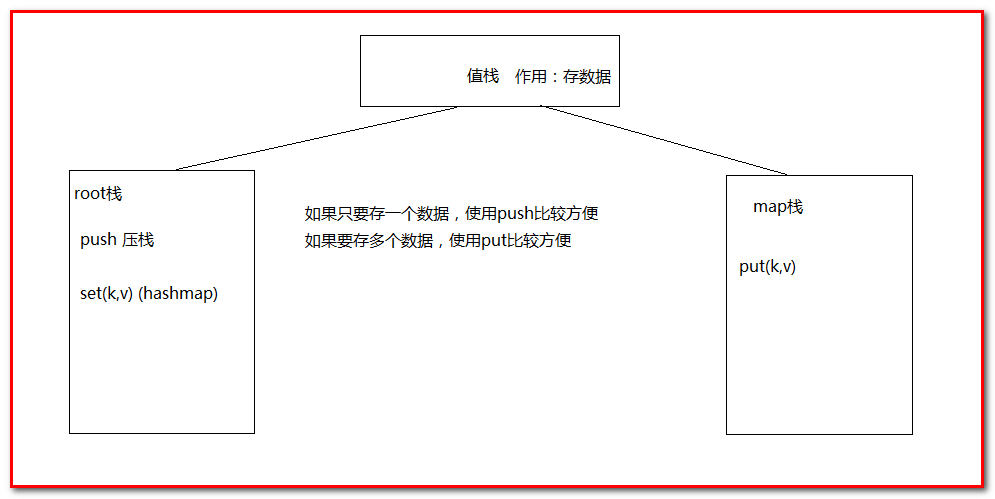
|  |
| --- |
| <!-- 值栈的默认搜索 -->  <s:property value=*"name"*/><br/>  <!-- 通过默认搜索，假如root栈找不到，找map --> |

那查找的顺序是怎么样的呢？

先搜索root栈，再搜索map栈，一找到就停止搜索，直接返回值

root栈的操作效率非常高





## 值栈的存取小结

值栈的主要作用就是数据的保存和获取（可以在任何地方获取）。

注意要点：怎么存，就怎么取

存进root栈，就不通过#获取<s:property value=”xxx”>；

存进map栈，就通过#获取<s:property value=”#xxx”>；

1. 如何向值栈保存数据
2. ValueStack.push(obj) ：保存数据到Root栈顶-压栈顶（对象本身）-匿名
3. ActionContext.getContext().put(key,value) ：保存数据到Map栈中
4. ValueStack.set(key,value)：将数据保存到Root栈顶（数据对象自动被封装为Map来保存，栈顶是个map，map里面有个属性是对象）--有名字

2．ognl表达式如何获取值栈的数据

* JSP页面获取

1. <s :property value= “name”/> 先搜索root栈对象属性(getter方法：getXxx-->xxx)，再搜索map的key
2. <s:property value=”#name” /> 搜索map的key
3. 通过 [index].top 指定访问root栈某层对象 ，例如 [0].top 栈顶对象

* Action代码获取

ValueStack.findValue(ognl表达式) ; 获取值.栈数据

|  |
| --- |
| //在代码中获取root值栈中的值  String str1=ActionContext.*getContext*().getValueStack().findString("username");  //在代码中获取map值栈中的值  String str2=ActionContext.*getContext*().getValueStack().findString("#username");  System.*out*.println("str1:"+str1);  System.*out*.println("str2:"+str2); |

## 模型驱动优先属性驱动接收参数

【问题】为什么在赋值的时候，模型驱动的属性会优先赋值？模型驱动是如何具备更高的优先级别？

【示例】新增user数据模型，使用模型驱动

【第一步】在com.igeek\_01包中创建User类，代码如下：

|  |
| --- |
| **package** com.igeek\_01;  **public** **class** User {  **private** String username;  **private** String password;  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPassword() {  **return** password;  }  **public** **void** setPassword(String password) {  **this**.password = password;  }  } |

【第二步】com.igeek\_01包中创建ValueStack2Action.java类，代码如下：

|  |
| --- |
| **package** com.igeek\_01;  **import** com.opensymphony.xwork2.ActionSupport;  **import** com.opensymphony.xwork2.ModelDriven;  **public** **class** ValueStack2Action **extends** ActionSupport **implements** ModelDriven<User>{  //创建user  **private** User user = **new** User();  //创建一个属性  **private** String username;  @Override  **public** String execute() **throws** Exception {  **return** *SUCCESS*;  }  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  @Override  **public** User getModel() {  // **TODO** Auto-generated method stub  **return** user;  }  } |

【第三步】在WebContent中的b\_valuestack文件夹下，创建valuestack2.jsp页面，加入<s:debug />标签,具体代码如下：

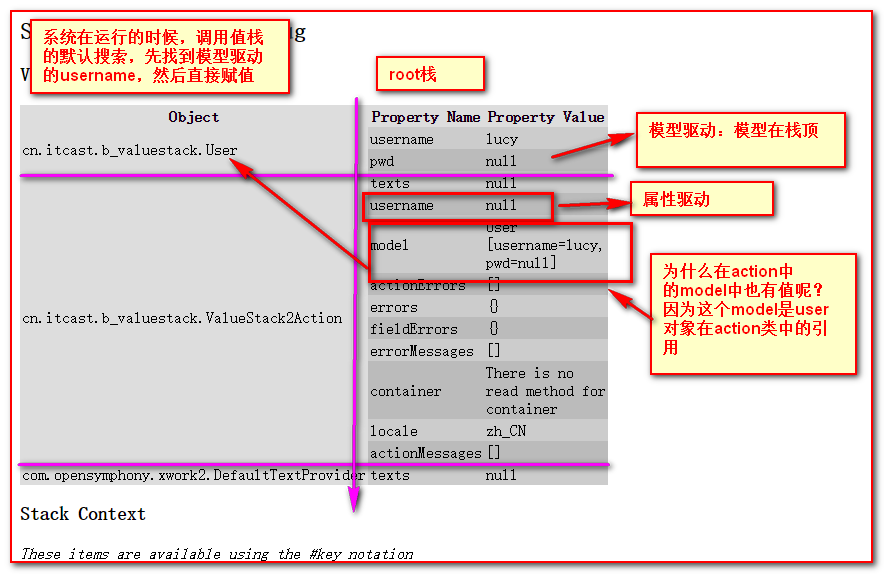
|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>值栈测试内容</title>  </head>    <body>  <s:debug></s:debug>  </body>  </html> |

【第四步】在struts.xml文件中配置：

|  |
| --- |
| <action name=*"valueStack2"* class=*"com.igeek\_01.ValueStack2Action"*>  <result>/b\_valuestack/valuestack2.jsp</result>  </action> |

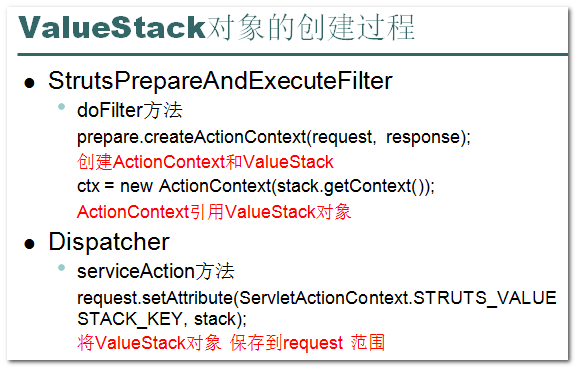
【第五步】测试

【运行结果分析】

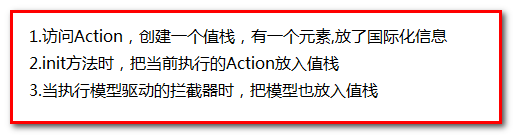


注意：值栈在使用过程中要注意同名覆盖的问题。

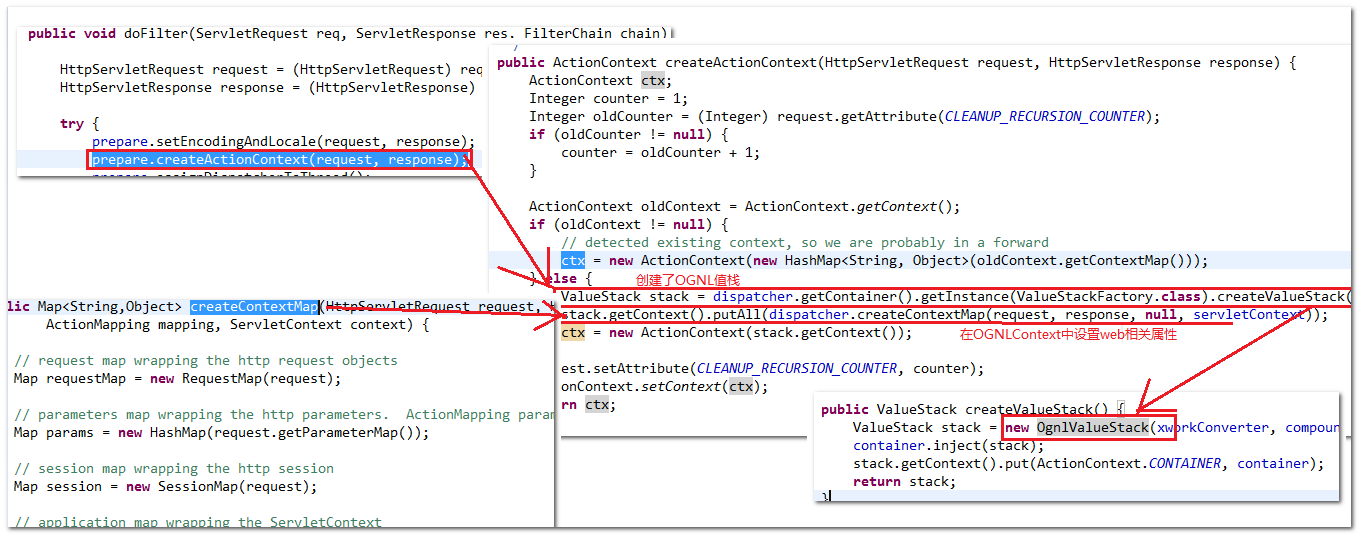
## 值栈的创建过程源代码分析（了解）



【值栈的运行流程源码详解】：



1．访问action的时候，走过滤器，StrutsPrepareAndExecuteFilter中创建了值栈（ValueStack<--OgnlValueStack），并将常用web对象保存到OgnlContext的Map中。



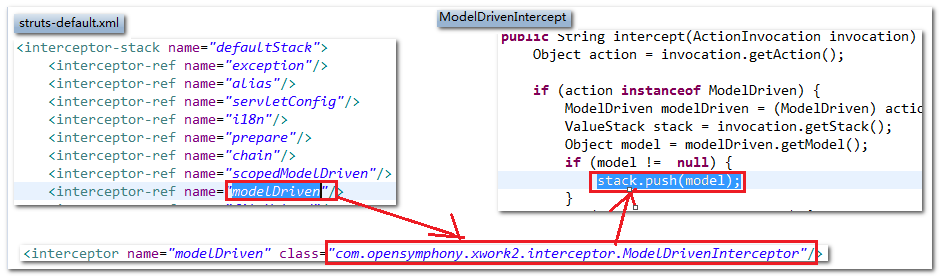
注意：此时root栈为空。

2．当访问Action的时候，过滤器会通过ActionProxy代理对象来执行。通过ActionProxyFactory的createActionProxy方法来得到代理对象。在获取代理对象的时候，需要创建一个ActionInvocation接口的默认实现类DefaultActionInvocation的对象，该默认对象在创建的时候会自动调用init方法来初始化，在初始化方法中会自动调用stack.push(action)方法，将当前Action对象，压入Root栈的顶部。

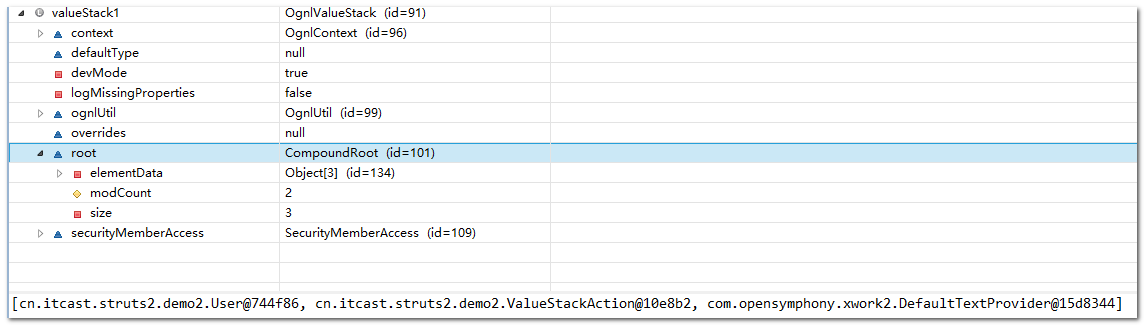


注意：再通过debug调试发现，默认情况下ValueStack的root栈存在2个对象 ，ValueStackAction（Action在栈顶）和DefaultTextProvider（用来初始化获取国际化资源信息）

3．在初始化Action代理对象后，会自动调用拦截器（增强），而执行默认拦截器栈中的ModelDriven拦截器，struts2会判断该Action是否实现了ModelDriven接口，如果实现了，则会调用ModelDrivenInterceptor拦截器中的intercept方法，在该方法中会调用stack.push(model)，将model压入Root栈顶。



注意：此时root栈就有3个对象了，栈顶是model。



原理目的：

* 知道值栈的初始化时机，访问action的时候，才创建（这个过程中会创建actionProxy对象，并且同时创建值栈，并且，将一些对象放入值栈。）
* 值栈初始化之后，里面主要默认有：root栈(model(user),action类对象、DefaultTextProvider)，map栈(servlet相关对象、action对象一个引用)
* 要了解：哪些对象在栈顶！因为后面我们从值栈取值，都从栈顶往下取。所以，一定要知道哪个对象在栈上面。

## 值栈的生命周期

【分析】：

|  |
| --- |
| * 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。 * Struts2框架将ValueStack对象保存在名为“struts.valueStack”的（request）请求属性中，即值栈是request中的一个对象，一个请求对应一个Action实例和一个值栈对象。 |

值栈以线程安全的方式为每个请求提供公共的数据存取服务。当有请求到达的时候，Struts2会为每个请求创建一个新的值栈，也就是说，值栈和请求是一一对应的，不同的请求，值栈也不一样，而值栈封装了一次请求所有需要操作的数据。

因此：值栈的生命周期，就是request生命周期，也就是Action的生命周期

【示例】通过重定向跳转页面时，目标页面无法获取到Action中放入值栈的值。

|  |
| --- |
| <!-- 值栈的生命周期证明 -->  <action name=*"valueStack3"* class=*"com.igeek\_01.ValueStack3Action"*>  <result type=*"redirect"*>/b\_valuestack/valuestack3.jsp</result>  </action> |

通过重定向传递数据可实现。

【示例】证明值栈的生命周期是与request一致的（下面是一个完整的例子测试值栈的生命周期）

【第一步】创建ValueStack3Action类，具体代码如下：

|  |
| --- |
| **package** com.igeek\_01;  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** ValueStack3Action **extends** ActionSupport{    @Override  **public** String execute() **throws** Exception {    ActionContext.*getContext*().getValueStack().set("username", "igeek");    **return** *SUCCESS*;  }  } |

【第二步】编写valuestack3.jsp页面，在页面上获取值栈中的值

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'valuestack2.jsp' starting page</title>  </head>    <body>  <!-- 获取值栈中的值 -->  <s:property value=*"username"*/>  </body>  </html> |

【第三步】去编写struts.xml,具体代码如下：

|  |
| --- |
| <!-- 配置ValueStack3Action -->  <action name=*"valueStack3"* class=*"com.igeek\_01.ValueStack3Action"*>  <result>/b\_valuestack/valuestack3.jsp</result>  </action> |

【第四步】测试

http://localhost:8080/struts2\_d04\_c03/valueStack3.action

【第五步】修改xml文件中的跳转方式，默认是请求跳转，现在改成重定向

|  |
| --- |
| <!-- 配置ValueStack3Action -->  <action name=*"valueStack3"* class=*"com.igeek\_01.ValueStack3Action"*>  <result type=*"redirect"*>/b\_valuestack/valuestack3.jsp</result>  </action> |

【第六步】测试，发现获取不到数据，表明值栈的生命周期是与request一致的

# Struts2使用表达式语言来操作值栈

## Struts2支持的表达式语言

|  |
| --- |
| Struts 2支持以下几种表达式语言：   * OGNL（Object-Graph Navigation Language），可以方便地操作对象属性的开源表达式语言； * EL(Expression Language)，可以方便的操作JSP页面四个域范围数据 （page、 request、 session、 application ）； * JSTL（JSP Standard Tag Library），JSP 2.0集成的标准的表达式语言； * Groovy，基于Java平台的动态语言，它具有时下比较流行的动态语言（如Python、Ruby和Smarttalk等）的一些起特性； * Velocity，严格来说不是表达式语言，它是一种基于Java的模板匹配引擎，据说性能要比JSP好。 |

Struts 2默认的表达式语言是OGNL，原因是它相对其它表达式语言更简单、强大， 最重要的是可以直接操作值栈。

## OGNL对Struts2的值栈的操作方式

明确一点：OGNL表达式对值栈的操作，必须依赖Struts2的标签,所以每个jsp页面都需要引入struts2的标签库

Struts2 通过值栈存放数据，那如何取数据呢？ 通过Ognl表达式访问值栈中的数据

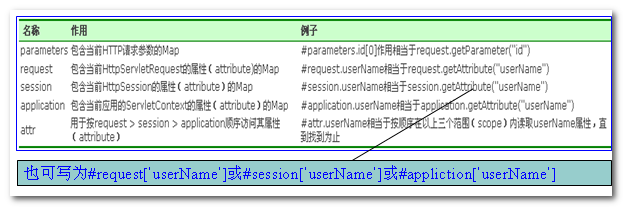
在Struts2中，OGNL表达式三种特殊符号的使用方式 ：

* # 号用法
* % 号用法
* $ 号用法

### #号的使用

【#用法一】：

访问 Map栈中常用对象,包括web对象（request/response/session/application..）， 添加#进行访问



提示：#attr 按照 page --- request --- session --- application的顺序依次进行搜索

【示例】在action中添加代码，向request、session、application域中放入对象，然后在页面通过ognl表达式获取，具体代码如下：

【第一步】创建struts2\_d04\_c04项目，新建com.igeek\_01包，在包中，创建OgnlAction类，在类中编写如下代码：

|  |
| --- |
| **package** com.igeek\_01;  **import** org.apache.struts2.ServletActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** OgnlAction **extends** ActionSupport{    @Override  **public** String execute() **throws** Exception {  //向Servlet的API中放入信息  //request  ServletActionContext.*getRequest*().setAttribute("name", "request域中的name");  //session  ServletActionContext.*getRequest*().getSession().setAttribute("name", "session域中的name");  //application  ServletActionContext.*getServletContext*().setAttribute("name", "application域中的name");    **return** *SUCCESS*;  }  } |

【第二步】在WebContent下面，创建c\_ognl文件夹，在文件夹中创建ognl.jsp页面，在页面中，编写如下代码：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'ognl.jsp' starting page</title>  </head>    <body>  request域中的name:<s:property value=*"#request.name"*/>|${requestScope.name }|<s:property value=*"#request['name']"*/><br>  session域中的name:<s:property value=*"#session.name"*/>|${sessionScope.name }|<s:property value=*"#session[\"name\"]"*/><br>  application域中的name:<s:property value=*"#application.name"*/>|${applicationScope.name }|<br>    <%  //在此处将值放入poge域  pageContext.setAttribute("name", "page域中的name");    %>  <%--#attr调用默认搜索 --%>  page域中的name:<s:property value=*"#attr.name"*/>|${name }<br>    <!-- 如何获取地址栏传过来的参数呢？ -->  地址栏的信息:<s:property value=*"#parameters.name"*/>    </body>  </html> |

【第三步】在struts.xml中配置OgnlAction类，代码如下：

|  |
| --- |
| <!-- 配置OGNLAction -->  <action name=*"ognl"* class=*"com.igeek\_01.OgnlAction"*>  <result>/c\_ognl/ognl.jsp</result>  </action> |

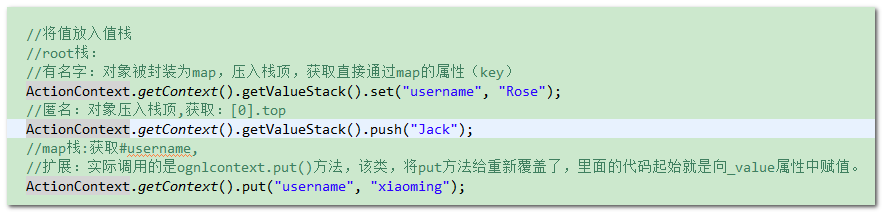
【第四步】测试：

【#用法二】：

如果不加 # 会直接调用ValueStack搜索功能（findValue），先搜索root栈对象的属性，后搜索Map栈,加#直接搜索map栈

【第一步】：

Action 代码



JSP代码



【#用法三】：用来构造map集合，必须要配合标签进行使用，

list：{‘value1’,’value2’}----new arraylist()

map：#{‘key1’ :’value1’,’key2’ :’value2’} ---new hashmap() 相当于map对象



【实际应用场景】:注册的时候的性别、学历，就可以构造list集合，也可以通过 # 构造map集合

（本章第五节讲解标签的使用的时候，会详解）

### %号的使用

主要作用：

1 强制解析

【示例】

【第一步】在com.igeek\_01包中，创建Ognl2Action类，在类中编写如下代码：

|  |
| --- |
| **package** cn.igeek.c\_ognl;  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** Ognl2Action **extends** ActionSupport{  @Override  **public** String execute() **throws** Exception {  ActionContext.*getContext*().getValueStack().set("username", "tom");  **return** *SUCCESS*;  }  } |

【第二步】在c\_ognl包中， 创建ognl2.jsp页面，页面中代码如下：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'ognl2.jsp' starting page</title>  </head>    <body>  <hr>  <!-- 默认不解析 -->  <s:textfield value=*"username"* /><br>  <!-- 强制解析 -->  <s:textfield value=*"%{username}"* /><br>  </body>  </html> |

【第三步】在struts.xml文件中，配置Action，代码如下：

|  |
| --- |
| <!-- 配置OGNL2Action -->  <action name=*"ognl2"* class=*"com.igeek\_01.Ognl2Action"*>  <result>/c\_ognl/ognl2.jsp</result>  </action> |

【第四步】访问：

<http://localhost:8080/struts2_d04_c01/ognl2.action>

### $号的使用

此处的$不能理解为el表达式，因为此处的$必须用在struts特有的文件中，只能用在struts2的文件中

用法：是用在struts2特有的文件中的

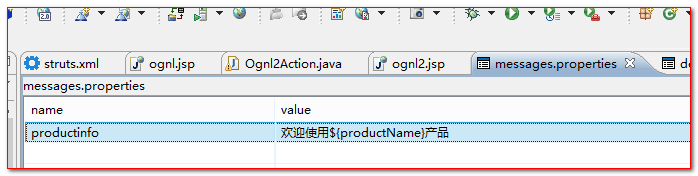
$主要作用：允许我们在配置文件中使用OGNL表达式，(换句话说：$可以在xml文件中获取值栈的值。)

配置文件主要指：struts.xml、国际化的文件(xxx.properties)、校验配置文件(xxx-validation.xml)。

【示例1】在资源文件中使用：在页面读取携带参数 --国际化文件中使用

【第一步】在src新建 messages.properties

点击ok之后，出现如下信息，点击保存



在上图中出现的${productName}-----获取值栈的值

【第二步】在struts.xml 配置国际化文件



【第三步】在com.igeek\_01包中创建Ognl3Action类，在类中编写如下代码：

|  |
| --- |
| **package** com.igeek\_01;  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** Ognl3Action **extends** ActionSupport{  @Override  **public** String execute() **throws** Exception {    //向root栈中添加内容  ActionContext.*getContext*().getValueStack().set("productName", "iphone");    **return** *SUCCESS*;  }  } |

【第四步】：在c\_ognl文件夹中，创建ognl3.jsp文件，代码如下：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'ognl.jsp' starting page</title>  </head>  <body>  <!--通过键获取资源文件中的内容-->  <s:text name=*"productinfo"*></s:text>  </body>  </html> |

【第五步】：在struts.xml文件中进行如下配置：

|  |
| --- |
| <!-- 配置OGNL3Action -->  <action name=*"ognl3"* class=*"com.igeek\_01.Ognl3Action"*>  <result>/c\_ognl/ognl3.jsp</result>  </action> |

【第六步】测试

<http://localhost:8080/struts2_d04_c04/ognl3.action>

【说明】通过上述证明，表明在资源文件中可以通过${productName}的方式获取值栈中的内容

【测试二】struts.xml中使用$的方式获取值栈中的值：URL请求重定向时，携带参数

【第一步】在com.igeek\_01包中创建Ognl4Action类，在类中编写如下代码：

|  |
| --- |
| **package** com.igeek\_01;  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** Ognl4Action **extends** ActionSupport{  @Override  **public** String execute() **throws** Exception {    //向root栈中添加内容  ActionContext.*getContext*().getValueStack().set("productName", "iphone");    **return** *SUCCESS*;  }  } |

【第二步】编写ognl4.jsp页面，代码如下：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'ognl4.jsp' starting page</title>  </head>    <body>  This is my JSP page. <br>  <!-- 当重定向只会，我们发现，值传不过来了，这个时候，这个页面还必须要使用那个productName，那怎么办呢？ -->  <%--  <s:property value="productName"/>  --%>  <!-- 如何获取参数的？ -->  <s:property value=*"#parameters.productName"*/>        </body>  </html> |

【第三步】当将结果集的跳转类型修改为redirect的时候，结果会如何呢？

|  |
| --- |
| <!-- 配置OGNL3Action -->  <action name="ognl3" class="com.igeek\_01.Ognl3Action">  <result type="redirect">/c\_ognl/ognl3.jsp?productName=${productName}</result>  </action> |

【第四步】测试运行

<http://localhost:8080/struts2_d04_c04/ognl3.action>

【注意】：

1 大括号{}不能是中文状态下的。

如果是重定向，则无法在下一个页面上获取到值栈的值，原因是：重定向是重新发起请求，值栈随着第一次的request请求的消亡而消亡。

2 关于ServletActionContext和ActionContext的选择？

他们之间继承：



ServletActionContext拥有ActionContext的所有功能。

但是，我们习惯这么操作：

操作值栈用：ActionContext

操作Servlet相关的API用：ServletActionContext

### EL表达式获取值栈的数据

除了OGNL表达式外，EL表达式也可以获取值栈数据。

【代码如下】

【第一步】在com.igeek\_01包中创建Ognl5Action类，在类中编写如下代码：

|  |
| --- |
| **package** com.igeek\_01;  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** Ognl5Action **extends** ActionSupport{  @Override  **public** String execute() **throws** Exception {    //向root栈中添加内容  ActionContext.*getContext*().getValueStack().set("username", "tom");    **return** *SUCCESS*;  }  } |

【第二步】：在c\_ognl文件夹中，创建ognl5.jsp文件，代码如下：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'ognl.jsp' starting page</title>  </head>  <body>  <!--通过键值栈中的内容-->  ${username}  </body>  </html> |

【第三步】：在struts.xml文件中进行如下配置：

|  |
| --- |
| <!-- 配置OGNL5Action -->  <action name=*"ognl5"* class=*"com.igeek\_01.Ognl5Action"*>  <result>/c\_ognl/ognl5.jsp</result>  </action> |

【第四步】测试

<http://localhost:8080/struts2_d04_c04/ognl5.action>

【说明】通过上述证明，表明通过el表达式可以获取值栈中的内容

【原理分析】

EL 表达式原理， 在page、request、session、application 四个范围，调用getAttribute 获取数据。为什么也可以获取值栈的值呢？

【原理分析】：

阅读源码

Struts2 对request进行包装了,对request 的 getAttribute 方法增强

Struts2 框架 提供 StrutsRequestWrapper 包装类，

上述代码发现：优先使用 request.getAttribute取值，如果取不到，执行 valueStack的findValue方法

【因此】

request的 getAttribute方法被覆写，因此我这里称其为“神奇的request”。

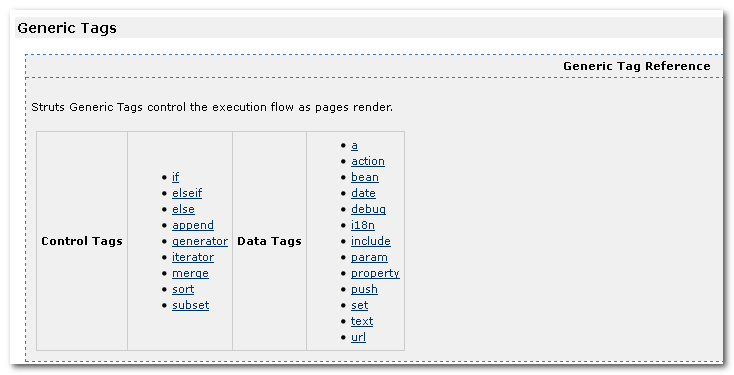
# Struts2的标签

## Struts2的页面标签概述

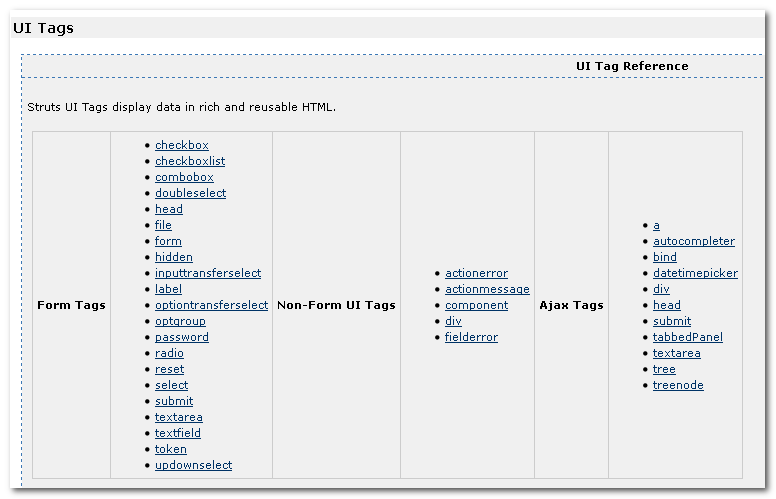
|  |
| --- |
| * 表单标签将在 HTML 文档里被呈现为一个表单元素 * 使用表单标签的优点:   + 表单回显   + ~~对页面进行布局和排版~~ * 标签的属性可以被赋值为一个静态的值或一个 OGNL 表达式. 如果在赋值时使用了一个 OGNL 表达式并把它用 %{} 括起来, 这个表达式将会被求值. |

参考： docs/WW/docs/tag-reference.html

通用标签库（控制标签、数据标签）：



表单标签库：



## 通用(Generic)标签

通用标签主要指两类：数据类标签和控制类标签。

### <s:property> --数据类标签

作用：将OGNL表达式的内容输出到页面

属性：

* + value属性，接收OGNL表达式,从值栈取值
  + default 属性：显示默认值，如果当通过OGNL表达式没有获取到值，default设置显示默认值
  + escapeHtml 属性， 是否对HTML标签转义输出 （默认是不转义 true，可以关闭）

【示例】标签属性演示。

【第一步】创建struts2\_d04\_c05项目，新建com.igeek\_01包，在包中创建TagAction类，

|  |
| --- |
| **package** com.igeek\_01;  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** TagAction **extends** ActionSupport{      **public** String execute() **throws** Exception {    //ActionContext.getContext().put("name", "lucy");    ActionContext.*getContext*().put("html", "<table border='1' width='100' height='100'><tr><td>A</td></tr></table>");    **return** *SUCCESS*;  }  } |

【第二步】在WebContent下面创建d\_tag文件夹，创建tag.jsp页面，编写如下代码：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'tag.jsp' starting page</title>  </head>    <body>    <s:property value=*"#name"* default=*"igeek"*/><br>  <!-- escapeHtml:是否解析html文件  true:默认值，默认不解析html文本  false:解析html文本  -->  <s:property value=*"#html"* escapeHtml=*"false"*/><br>      </body>  </html> |

【第三步】配置struts.xml文件

|  |
| --- |
| <!-- property的用法测试的Action -->  <action name=*"tag"* class=*"com.igeek\_01.TagAction"*>  <result>/d\_tag/tag.jsp</result>  </action> |

【第四步】访问

[http://localhost:8080/struts2\_d04\_c05/tag.action](http://localhost:8080/struts2_day05/tag.action)

### <s:iterator>（至关重要！！！！$$%%^&&\*\*\*\*）

作用：遍历集合对象（可以是List、set和数组等），显示集合对象的数据。（跟jstl的<c:foreach>功能一样）

属性：

value：迭代的集合，支持OGNL表达式，如果没有设置该属性，则默认使用值栈栈顶的集合来迭代。

var：引用变量的名称，该变量是集合迭代时的子元素。

status：引用迭代时的状态对象IteraterStatus实例，其有如下几个方法：

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 odd：返回当前迭代元素的索引是否是奇数

boolean even：返回当前迭代元素的索引是否是偶数

boolean isFirst()：返回当前迭代元素的索引是否是第一个

boolean isLast()：返回当前迭代元素的索引是否是最后一个

【示例一】： 在jsp页面中使用循环的方式输出1- 10 （打印 1- 10）

在WebContent下面创建d\_tag文件夹，在文件夹中创建tag1.jsp页面,具体代码如下:

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'tag1.jsp' starting page</title>  </head>    <body>  <!-- 在页面打印输出1--10  begin:起始值  end:结束值  step:步长  var:别名  status：状态    iterator的工作原理：  1 、将值以匿名的方式压入root栈顶  2、将该值以命名的方式放入map栈，名字就是var后面定义的名字  3、mystatus也是存放在map栈中  -->  <s:iterator begin=*"1"* end=*"10"* step=*"1"* var=*"num"* status=*"myStatus"*>  <s:property value=*"[0].top"*/>|<s:property value=*"top"*/>|  <s:property value=*"#num"*/>|<s:property value=*"#myStatus.odd"*/>|  <s:property value=*"#myStatus.index"*/>|<s:property value=*"#myStatus.isLast()"*/>|  <s:property value=*"#myStatus.isFirst()"*/>  <br>    </s:iterator>    </body>  </html> |

【示例二】： 遍历集合对象

【第一步】在com.igeek\_01包中，创建User对象，具体代码如下：

|  |
| --- |
| **package** com.igeek\_01;  **public** **class** User {  **private** String username;  **private** String pwd;    **public** User() {  **super**();  }  **public** User(String username, String pwd) {  **super**();  **this**.username = username;  **this**.pwd = pwd;  }  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPwd() {  **return** pwd;  }  **public** **void** setPwd(String pwd) {  **this**.pwd = pwd;  }  } |

【第二步】创建Tag2Action类，在类中创建一个List集合，放入数据，具体代码：

|  |
| --- |
| **package** com.igeek\_01;  **import** java.util.ArrayList;  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** Tag2Action **extends** ActionSupport{    @Override  **public** String execute() **throws** Exception {    ArrayList<User> list = **new** ArrayList<User>();  list.add(**new** User("lucy", "123"));  list.add(**new** User("tom","123"));  list.add(**new** User("rose", "123"));    ActionContext.*getContext*().getValueStack().set("list", list);    **return** *SUCCESS*;  }        } |

【第三步】在d\_tag文件夹中，创建tag2.jsp页面，在页面中通过<iterator>标签访问值栈中的集合,代码如下：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'tag2.jsp' starting page</title>  </head>    <body>  <!-- value：从值栈获取值 -->  <s:iterator value=*"list"* var=*"user"*>  <!-- 遍历的过程：  1 将值以匿名的方式压入root栈顶  2 以有名字的方式放入map栈中(var定义的键)  -->  <%--  此处获取方式较多，建议先掌握一种  <s:property value="username"/>：<s:property value="pwd"/>  --%>  <s:property value=*"[0].top.username"*/>:<s:property value=*"[0].top.pwd"*/>|  <s:property value=*"username"*/>：<s:property value=*"pwd"*/>|  <s:property value=*"#user.username"*/>:<s:property value=*"#user.pwd"*/>|  ${user.username }:${user.pwd }<br>  ${username }:${pwd }<br>    </s:iterator>      </body>  </html> |

【第四步】在struts.xml文件中配置如下:

|  |
| --- |
| <!-- s:iterator的用法测试的Actioin -->  <action name=*"tag2"* class=*"com.igeek\_01.Tag2Action"*>  <result>/d\_tag/tag2.jsp</result>  </action> |

【第五步】访问测试：

<http://localhost:8080/struts2_d04_c05/tag2.action>

遍历对象的属性

* <s:property value=”name” /> 从root栈顶取值
* <s:property value=”#product.name” /> 从map取值
* ${name} 搜索,先从request对象中取值，如果取不到，就去值栈进行默认搜索

### <s:if> <s:elseif> <s:else>

作用：页面判断，其中的test属性可以接收OGNL表达式。

【示例】

根据后台传入的用户状态值（0，1，2），在页面判断并显示中文（管理员、普通用户、游客）

|  |
| --- |
| action：    页面： |

### <s:a>超链接标签

作用：生成a标签链接

注意：传值的两种方式

|  |
| --- |
| <s:a action=*"tag"* namespace=*"/"*>  <!--第一种方式，value中填写ognl表达式-->  <s:param name=*"name"* value=*"’tom’"*></s:param>  <!--第二种方式：直接在标签中间定义-->  <s:param name=*"name"*>tom</s:param>  new链接  </s:a> |

【示例】使用<s:a>标签生成一个链接

【第一步】创建tag3.jsp页面，在页面中编写如下代码：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'tag3.jsp' starting page</title>  </head>    <body>    <a href=*"*${pageContext.request.contextPath }*/tag3.action?name=lucy"*>tag3--old请求</a><br>  <!-- action:请求的URL  namespace:请求的是哪个namespace下面的action  超链接传值方式一  -->  <s:a action=*"tag3"* namespace=*"/"*>  <s:param name=*"name"*>tom</s:param>  new请求1  </s:a><br>  <!-- 超链接传值方式二  错误的传值写法：<s:param name="name" value="igeek"></s:param>  -->  <s:a action=*"tag3"* namespace=*"/"*>  <!-- value:ognl表达式，访问是对象 -->  <s:param name=*"name"* value=*"'igeek'"*></s:param>  new请求2  </s:a>      </body>  </html> |

【第二步】创建Tag3Action，具体代码如下：

|  |
| --- |
| **package** com.igeek\_01;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** Tag3Action **extends** ActionSupport{  **private** String name;    @Override  **public** String execute() **throws** Exception {    System.*out*.println(name);    **return** *NONE*;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  } |

【第三步】配置struts.xml

|  |
| --- |
| <!-- a的用法 -->  <action name=*"tag3"* class=*"com.igeek\_01.Tag3Action"*> </action> |

【第四步】访问测试

[http://localhost:8080/struts2\_d04\_c05/d\_dag/tag3.jsp](http://localhost:8080/struts2_day05/d_dag/tag3.jsp)

## 用户界面(UI)标签：用户界面标签主要包括两类：表单类标签和其他类标签

form表单的Struts2标签和传统标签的对比：（参考）





### <s:form>

作用：生成form标签。

属性：

* action属性，对应 struts.xml <action>元素name属性；
* namespace属性，对象 struts.xml <package>元素 namespace属性

【示例】创建tag4.jsp页面：代码如下：

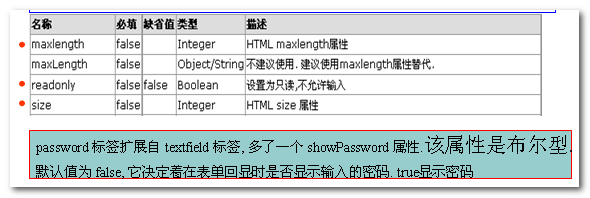
|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'tag4.jsp' starting page</title>  </head>    <body>  <!-- 传统的表单： -->  <form action=*"*${pageContext.request.contextPath }*/tag.action"* method=*"post"*>    </form>  <!-- struts2的表单 -->  <!--  action:请求的URL  namespace:请求的名称空间  -->  <s:form action=*"tag"* namespace=*"/"*>        </s:form>      </body>  </html> |

### <s:textfield>, <s:password>, <s:hidden>, <s:textarea>

作用：

* <s:textfield> 文本框 ，生成 <input type=”text” >
* <s:password> 密码域 ，生成<input type=”password” >
* <s:hidden> 隐藏域 ， 生成 <input type=”hidden” >
* <s:textarea>文本域，生成<textarea></textarea>

属性：



【示例】

文本框、密码框、隐藏域、文本域编写

|  |
| --- |
| <s:form action=*"tag"* namespace=*"/"*>  <!-- 文本域  name:文本框的名字  label:生成文本框前面的提示的  -->  <s:textfield name=*"username"* label=*"username"* /><br>  <!-- 密码框 -->  <s:password name=*"password"* label=*"password"* /><br>  <!-- 隐藏域 :页面是看不到的-->  <s:hidden name=*"age"* />  <!-- 文本域 -->  <s:textarea />  </s:form> |

### <s:radio>、<s:checkboxlist>、<s:select>

作用：（#构造map集合）

* <s:radio> 接收list或者map 生成一组单选按钮
* <s:select> 接收list或者map ，生成一组下拉列表
* <s:checkboxlist> 接收list或者map ，生成一组复选框



【示例】

|  |
| --- |
| <!-- radio：单选按钮 -->  <!-- 构造是list集合 -->  <s:radio list=*"{'男','女'}"* name=*"sex"* />  <!-- 构造map集合 -->  <s:radio list="#{'male':'男','female':'女' }" name=*"sexmap"* />    <!-- checkboxlist:复选框 -->  <s:checkboxlist list=*"{'唱歌','跳舞','溜冰'}"* name=*"favor1"* /><br>  <!-- 构造map集合 -->  <s:checkboxlist list="#{'sing':'唱歌','dance':'跳舞','skiing':'溜冰' }" name=*"favor2"* />  <br>  <!-- select:下拉列表框 -->  <s:select list=*"{'上海','北京'}"* name=*"city1"* /><br>  <s:select list="#{'shanghai':'上海','beijing':'北京' }" name=*"city2"* /> |

### <s:submit>、<s:reset>

作用：

* <s:submit>、<s:reset>分别对应html中的提交和重置

【示例】

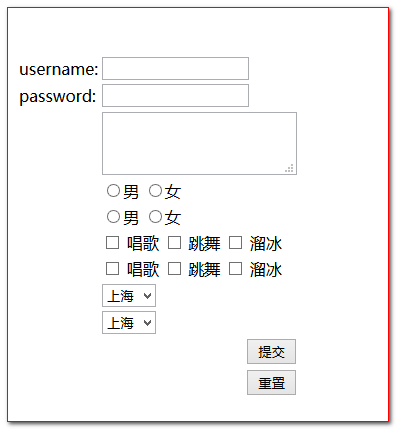
|  |
| --- |
| <s:submit value=*"提交"*/>  <s:reset value=*"重置"* /> |

【示例完整代码】

【第一步】在d\_tag文件夹中创建tag4.jsp页面,代码如下：

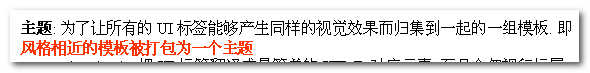
|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'tag4.jsp' starting page</title>  </head>    <body>  <!-- 传统的表单： -->  <form action=*"*${pageContext.request.contextPath }*/tag.action"* method=*"post"*>    </form>  <!-- struts2的表单 -->  <!--  action:请求的URL  namespace:请求的名称空间  -->  <s:form action=*"tag"* namespace=*"/"*>  <!-- 文本域  name:文本框的名字  label:生成文本框前面的提示的  -->  <s:textfield name=*"username"* label=*"username"* /><br>  <!-- 密码框 -->  <s:password name=*"password"* label=*"password"* /><br>  <!-- 隐藏域 :页面是看不到的-->  <s:hidden name=*"age"* /><br>  <!-- 文本域 -->  <s:textarea /><br>    <!-- radio：单选按钮 -->  <!-- 构造是list集合 -->  <s:radio list=*"{'男','女'}"* name=*"sex"* />  <!-- 构造map集合 -->  <s:radio list="#{'male':'男','female':'女' }" name=*"sexmap"* />    <!-- checkboxlist:复选框 -->  <s:checkboxlist list=*"{'唱歌','跳舞','溜冰'}"* name=*"favor1"* /><br>  <!-- 构造map集合 -->  <s:checkboxlist list="#{'sing':'唱歌','dance':'跳舞','skiing':'溜冰' }" name=*"favor2"* />  <br>  <!-- select:下拉列表框 -->  <s:select list=*"{'上海','北京'}"* name=*"city1"* /><br>  <s:select list="#{'shanghai':'上海','beijing':'北京' }" name=*"city2"* /><br>    <s:submit value=*"提交"*/>  <s:reset value=*"重置"* />    </s:form>      </body>  </html> |

【第二步】运行：



## 主题样式

主题的作用：

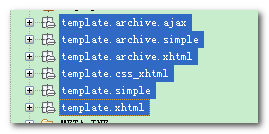


Struts2 模板文件，支持两种Freemarker生成 （\*.ftl模板文件） ， Velocity生成 （\*.vm 模板文件）

提供四种主题 ：

* Simple 最简单主题
* Xhtml 通过 布局表格 自动排版 （默认主题 ）
* css\_xhtml 通过CSS进行排版布局
* ajax 以Xhtml模板为基础，增加ajax功能

在struts核心包，提供模板主题

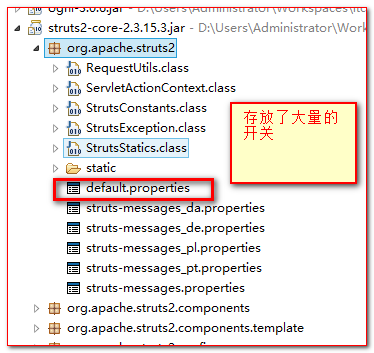


问题： 如何修改主题



开发中，在struts.xml 配置常量，修改默认主题样式，对所有form生效

|  |
| --- |
| <!-- 修改struts标签的默认模板 -->  <constant name=*"struts.ui.theme"* value=*"simple"*></constant> |



记忆方法：

表单的s标签中的name中接收OGNL表达式，除此之外，其他的s标签的value接收OGNL表达式

重点和小结

1 理解值栈内部结构

2 掌握存值+取值：怎么存就怎么取

3 标签+ognl表达式

4 跟踪代码