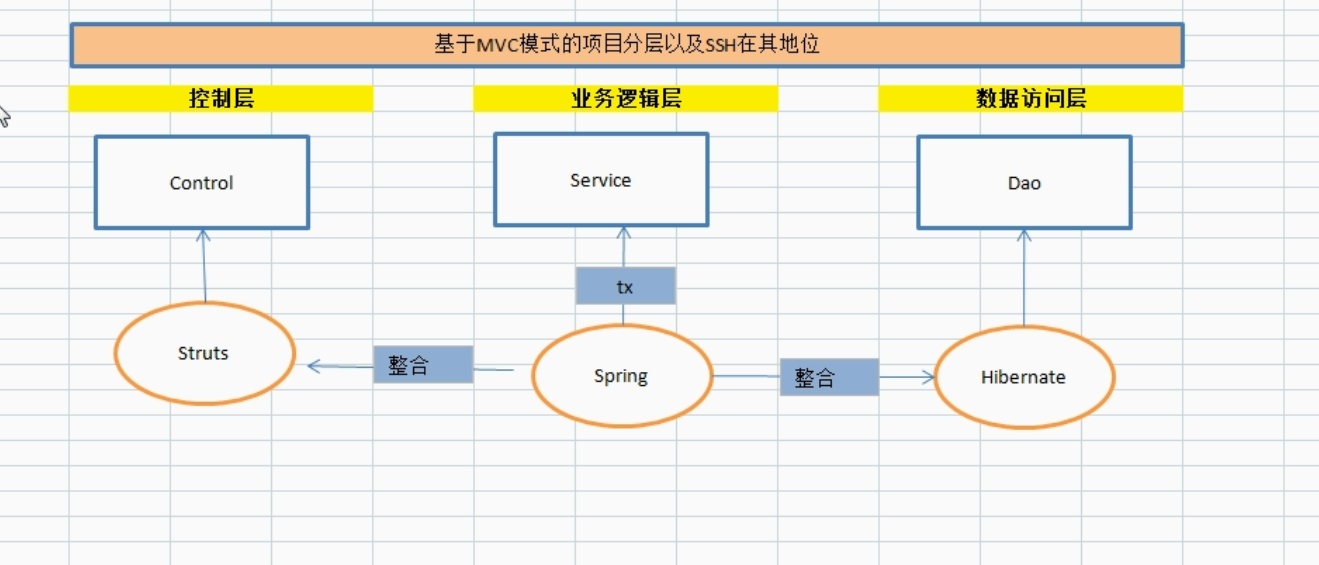
# 关于框架以及自定义struts



2017.4.26

自定义Structs框架的基本思想，更加简化servlet的功能，仅仅只是起到了一个控制的作用。

这就要求：获取请求信息的代码应该封装到一个Action类中，且根据不同的返回值灵活的对处理完请求后，是转发还是重定向，到哪个页面。

1. 创建一个ActionServlet类，让所有的客户端请求都转到这一个Servlet类中来。

这样就出现一个问题：怎么区分各种不同的Action呢？

根据请求获得url，进一步得到是哪个action，如/login.action ,最后获得的就是login。

String url=request.getServletPath(); // login.action;

String actionName=url.subString(string.lastIndexOf("/")+1,string.indexOf("."));

然后再通过配置文件找到相应的 类名 ,以及要使用的方法名。

String className=actionMapping.getClassName(); //actionMapping是一个bean。后面再讲。

String method=actionMapping.getMethod();

再根据反射，我们就可以获得对象，并执行相应的方法。

Class clazz=Class.forName(className);

Method method1=clazz.getDeclaredMethod(method,HttpServletRequest.class, HttpServletRequest.class);

第一个参数为 方法名，后面的参数为方法的 参数.class 参数类的字节码类

Object object=clazz.newInstance();

String returnFlag=(String)method1.invoke(object,req,reps);

第一个参数 为调用方法的对象 后面参数为方法参数。

这样根据配置文件以及反射技术，就可以完美的实现一个类，来处理不同的action。

1. 我们要获得的东西都应该要从配置文件中，弄出来。由上面可知，配置文件是关系型的，所以必须采用xml这种格式的。

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <myStructs>  <package>  <action name="login" class="action.LoginAction" method="login">  <result name="success" type="redirect">/index.jsp</result>  <result name="loginFail">/login.jsp</result>  </action>  </package>  </myStructs> |

采用DOM4J.jar包对xml解析。

SAXReader reader=new SAXReader();

Document document=Reader.read(this.getClass().getSourceAsStream(‘mystructs.xml’));

通过解析器获得了Document 对象后，再把所有的元素标签封装到一个bean中，最后存入map集合中。

|  |
| --- |
| InputStream fis = this.getClass().getResourceAsStream("/myStructes.xml");  Document document = saxReader.read(fis);  //3.获取根  Element root = document.getRootElement();  Element Package = root.element("package");  List<Element> listAction = Package.elements("action");  for (Element ele\_action : listAction) {  ActionMapping actionMapping = new ActionMapping();  actionMapping.setName(ele\_action.attributeValue("name"));  actionMapping.setClassName(ele\_action.attributeValue("class"));  actionMapping.setMethod(ele\_action.attributeValue("method"));  Map<String, Result> resultMap = new HashMap<String, Result>();  List<Element> listResult = ele\_action.elements();  for (Element ele\_result : listResult) {  Result result = new Result();  result.setName(ele\_result.attributeValue("name"));  result.setType(ele\_result.attributeValue("type"));  result.setPage(ele\_result.getTextTrim());  resultMap.put(result.getName(), result);  }  actionMapping.setResultMap(resultMap);  allActions.put(actionMapping.getName(), actionMapping);  } |

# Struts全部内容：

1. **struts的开发环境搭建:**
   1. 引入jar包，
   2. 配置核心过滤器
   3. Action开发
   4. Struts.xml的配置。
2. **Struts的执行流程：**
   1. 服务器启动的时候：加载web项目，加载核心过滤器并初始化它的方法：

Struts-default.xml 核心功能初始化

Struts-plugin.xml struts相关插件

Struts.xml 用户编写的配置文件

同时，创建默认拦截器的对象，并对其初始化。

* 1. 第一次访问的时候：创建action实例，初始化，执行拦截器对象的intercept方法，执行就代表通过走向下一个拦截器。然后，再执行action的业务方法。

创建

**请求 ---- StrutsPrepareAndExecuteFilter 核心控制器 ----- Interceptors 拦截器（实现代码功能 ） ----- Action 的execute --- 结果页面 Result**

**\* 拦截器 在 struts-default.xml定义**

**\* 执行拦截器 是 defaultStack 中引用拦截器**

---- 通过源代码级别断点调试，证明拦截器是执行

1. **Struts的核心功能：**
   1. 数据处理（用servlet的方式处理 或者 用action的方法处理）
   2. 数据自动封装（对象需要 set、get方法都要写）
   3. 文件上传与下载

上传：struts可以自动获取file文件，FileName文件名，FileContentType，

然后通过FileUtiles直接把上传的文件存储起来就好了。

|  |
| --- |
| private File file1;  private String file1FileName;  private String file1ContentType;  public void setFile1(File file1) {  this.file1 = file1;  }  public void setFile1FileName(String file1FileName) {  this.file1FileName = file1FileName;  }  public void setFile1ContentType(String file1ContentType) {  this.file1ContentType = file1ContentType;  }  @Override  public String execute() throws Exception {  String path = ServletActionContext.getServletContext().getRealPath("/upload");  File destFile = new File(path,file1FileName);  FileUtils.copyFile(file1, destFile);  return SUCCESS;  } |

同时，可以通过注入参数，来设置上传文件的大小，以及类型。

|  |
| --- |
| <interceptor-ref name="defaultStack">  <param name="fileUpload.maximumSize" >300000000</param>  <!-- 限制运行的文件的扩展名 -->  <param name="fileUpload.allowedExtensions">txt,jpg,jar</param>    </interceptor-ref> |

下载：

文件下载，就是通过在action中配置result的type为stream来实现的。 需要配置：文件类型（统一用二进制的下载类型），文件下载流、文件下载头、以及缓冲区类型。

|  |
| --- |
| <result name="download" type="stream">  <!-- 运行下载的文件的类型:指定为所有的二进制文件类型 -->  <param name="contentType">application/octet-stream</param>  <!-- 对应的是Action中属性： 返回流的属性【其实就是getAttrInputStream()】 -->  <param name="inputName">attrInputStream</param>    <!-- 下载头，包括：浏览器显示的文件名 -->  <param name="contentDisposition">attachment;filename=${downFileName}</param>    <!-- 缓冲区大小设置 -->  <param name="bufferSize">1024</param>  </result> |

* 1. 自定义拦截器实现校验

自定义拦截器：定义一个类，实现interceptor接口 ，并重写方法。

|  |
| --- |
| public String intercept(ActionInvocation actionInvocation) throws Exception {  actionInvocation.invoke();  ActionProxy actionProxy=actionInvocation.getProxy();  actionProxy.getMethod();  return null;  } |

这个方法intercept是在访问后，在action实力创建并初始化后，执行的。执行ActionInvocation对象的invoke(),就代表不拦截了，执行下一个拦截器。

将自定义的拦截器，配置：



注意：通过ActionInvocation对象可以获取当前action的代理对象，通过代理对象，可以获取当前执行的action类名，以及当前要执行的方法名。 这样就可以进行控制。

* 1. Ognl表达式，以及jsp页面上取值，struts标签的使用。

根元素：

在action中的全局变量；

非根元素：

一些存在在域对象中的元素；

根元素取值，不用加# ，非根元素需要。

**为什么可以在jsp中取到值呢？**

因为struts把所有的数据都放到了valuStack值栈对象中了。ValueStack中分根元素，以及非根元素，取法不同。

* 1. 数据校验

通过xml来配置数据校验信息。

Struts中的拦截器已经配置了相应的拦截器。

1.Jsp中的要写：

<s:fielderror fieldName="user.userName"></s:fielderror>

2.xml配置

Xml名字：ActionClassName-actionName(user\_login)-validation.xml

Xml里面的细节：

引入约束：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE validators PUBLIC  "-//Apache Struts//XWork Validator 1.0.3//EN"  "http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd"> |

|  |
| --- |
| <!--需要验证的参数--> <**field** name="user.password">  <!--选择验证的校验器-->  <**field-validator** type="requiredstring">  <!--错误信息-->  <**message**>密码不能为空</**message**>  </**field-validator**>  <**field-validator** type="stringlength">  <!--参数-->  <**param** name="minLength">3</**param**>  <**param** name="maxLength">8</**param**>  <**message**>密码长度必须为3-8位</**message**>  </**field-validator**> </**field**> |

1. **Struts的其他细节**

数据回显，防止表单重复提交，模型驱动

**5. structs的配置**

structs.xml中可以配置action，也可以在相应的action文件夹下，新建一个xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* **<!DOCTYPE** struts **PUBLIC** "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"  "http://struts.apache.org/dtds/struts-2.3.dtd"***>*** <**struts**>  <!--配置action-->  <**package** name="zzk" extends="struts-default" > <!--继承了struts-default包-->  <**action** name="login" class="dynamicProxy.action.LoginAction" method="login" >  <**result** name="failed">/login.jsp</**result**>  <**result** name="success">/book.jsp</**result**>  </**action**>  <**action** name="book\_{1}" class="dynamicProxy.action.BookAction" method="{1}">  <**result** name="failed">/book.jsp</**result**>  <**result** name="success">/index.jsp</**result**>  </**action**>  </**package**> </**struts**> |

只要把它包括在struts.xml中就好了

<**include** file="dynamicProxy/user.xml"/>

# 难点：ognl与valueStack

## 问题1：ognl是什么，它有什么用？

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。

\* xwork 提供 OGNL表达式

\* ognl-3.0.5.jar

OGNL 是一种比EL 强大很多倍的语言

OGNL 提供五大类功能

1、支持对象方法调用，如xxx.doSomeSpecial()；

2、支持类静态的方法调用和静态属性访问

3、访问OGNL上下文（OGNL context）和ActionContext； （重点 操作ValueStack值栈 ）

4、支持赋值操作和表达式串联

5、操作集合对象。

## 问题2：valueStack是什么，它有什么用？

valueStack是值栈。

他是什么，我们从两个方面来说：

1.从技术角度 ValueStack就是一个接口。

2.从使用角度 ValueStack是一个容器。

我们使用valueStack最大的作用就是将action相关的数据以及web相关的对象，携带到页面上。

简单说：我们在struts2中通过valueStack将action中的数据携带到jsp页面上进行展示。

在页面上通过ognl表达式将valueStack中数据获取出来。

## ognl本身的使用 ：

ognl本身是一个独立的用来方便获取值的语言。并不是struts2中的。它本身的用法如下：

@Test  
**public void** main() **throws OgnlException** {  
 //ognl可以通过对象调用方法。  
 //1.创建一个ognl上下文  
 **OgnlContext** context=**new** OgnlContext();  
  
 **Object** object=Ognl.getValue("'aaa'.length()",context.getRoot());  
 **System**.out.println(object);  
  
 /\*\*  
 \* ognl 本质就是 特定格式的字符串 然后通过解析器来解析，  
 \*/  
}  
**public void** ognlDemo2() **throws OgnlException** {  
 //ognl调用静态方法和静态属性。都要用@  
 **OgnlContext** context=**new** OgnlContext();  
 **Object** object=Ognl.getValue("@java.lang.Math@max(10,20)",context.getRoot());  
 **System**.out.println(object);  
}  
**public void** ognlDemo3 () **throws OgnlException** {  
 /\*\*  
 \* ognl中有根元素和非根元素的区别。  
 \* struts2中通常把action的数据放到根中，把web数据放到非根中  
 \* 取值的时候，根元素直接用名字就好了，非根元素前面要加#  
 \* ognl是对像导航语言 a.b.c.d; 通过 . 来一直往下导航。  
 \*  
 \* 根和非根的本质就是：  
 \* 根：map集合的key为 root， value为存入根元素的值。 通过context.setRoot(value);  
 \* 非根 ：map集合的key为其他值 通过 context.put(key,value);  
 \*  
 \* 但是在struts中使用ognl的时候，谁是ognlContext，谁是root ，谁是 非根 这都是指定的。  
 \*  
 \* \*/  
 **OgnlContext** context=**new** OgnlContext(); //本质上就是一个Map集合。  
  
 **Category** category=**new** Category();  
 category.setCname("root");  
 context.setRoot(category);  
 **Object** object=Ognl.getValue("cname",context,context.getRoot());  
 **System**.out.println(object);  
}

## ognl在struts2中的使用：

前提：必须要结合struts2的标签来使用。

<**s:property** value="'aaa'.length()"/>

value中就是ognl的表达式

调用 实例方法 ： 对象.方法() ---- <s:property value="'hello,world'.length()"/>

调用 静态方法 ： @[类全名（包括包路径）]@[方法名] --- <s:property value="@java.lang.String@format('您好,%s','小明')"/>

\* 使用 静态方法调用 必须 设置 struts.ognl.allowStaticMethodAccess=true

### 问题1：什么是valueStack？

1.ValueStack有一个实现类叫OgnlValueStack

2.一个请求就对应一个action（一个action就是一个方法，而不是一个继承了ActionSupport的类，而是其中的方法），一个action就有一个ValueStack。

（一个模块一个继承了ActionSupport的类，其中的action都是围绕这个模块的请求）

|  |
| --- |
| ValueStack stack = (ValueStack)request.getAttribute("struts.valueStack"); **boolean** nullStack = stack == **null**; **if**(nullStack) {  ActionContext ctx = ActionContext.getContext();  **if**(ctx != **null**) {  stack = ctx.getValueStack();  } }  根据源码，可以看出来，第一次请求的时候，stack不存在，是通过ActionContext.getValueStack() 创建的  **if**(!nullStack) {  request.setAttribute("struts.valueStack", stack); }  然后把valueStack 保存request域中属性为struts.valueStack 中去。 |

valueStack生命周期就是request生命周期。

（请求来-🡪 相应去）

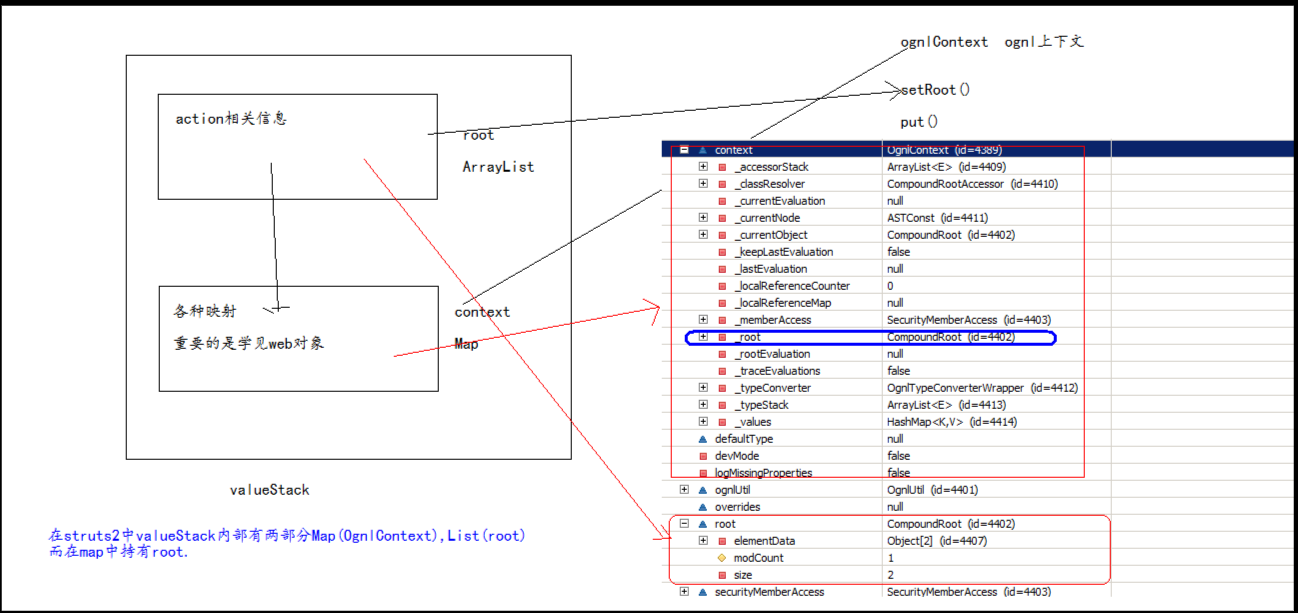
3.valueStack中存储了当前action对象 以及其他常用web对象（request，session,application.parameters）

### 问题2：值栈的内部结构？

valueStack中主要分为两部分：

一个root属性：本质是一个ArrayList。

一个context属性：本质是一个Map集合。（这等价于 OgnlContext）



**valueStack的结构图**

ArrayList 中存放action相关信息。

Map中存放各种映射，比如 常用web对象（request，session,application.parameters），

但是，这些都是放在Map的非根元素中的！！

在Map的根元素中影射了 ArrayList 所以，就相当于：

等价于OgnlContext的Map中存放了所有的数据。

因此访问ValueStack中数据，root集合中的Action相关数据 不用 # 号

访问web对象信息 需要 # 号。

### 问题3： 值栈对象的创建和 ActionContext与 ValueStack 的关系？

值栈对象 是请求时 创建的

doFilter中 prepare.createActionContext(request, response);

\* 创建ActionContext 对象过程中，创建 值栈对象ValueStack

\* ActionContext对象 对 ValueStack对象 有引用的 （在程序中 通过 ActionContext 获得 值栈对象 ）

值栈对象的获得：

ValueStack valueStack2 = ActionContext.getContext().getValueStack();

### 问题4：向值栈中保存数据（根）

为什么是学习向根中保存呢？因为要向非根中保存，就可以直接通过actionContext获取到session，存到session中去。

**像根中存数据有两种方式：**

push(Object obj)------->底层就是 root.add(0,obj) 将数据存储到栈顶。

set(String name,Object obj);----->底层是将数据封装到HashMap中，在将这 个HashMap通过push存储。

在jsp中 通过 <s:debug /> 查看值栈的内容

### 问题5：怎么通过jsp取出数据呢？

一般在后台取得数据，是通过：

ValueStack valueStack=**ActionContext**.getContext().getValueStack();  
//把得到的数据存入到值栈中，最好的方法就是采用valueStack.set("hotList",hotList);  
valueStack.set("hotList",hotList);

把数据存到值栈中。

在页面取数据有两种方式：

<**s:iterator** value="categoryList" var="c">  
 <**li**><**s:property** value="#c.cname" /></**li**>  
</**s:iterator**>

如果加了一个var=“c” 之后，取值就必须用# ，这是因为遍历得到的放入到了context中。

<**s:iterator** value="categoryList">  
 <**li**><**s:property** value="cname" /></**li**>  
</**s:iterator**>

如果不加，就直接写属性就好了。

这样就够了。其他不要深入了。

ActionContext是存在每一个线程的threadLocals 中的，

threadLocal是一个中介，为每一个线程开启一个threadLocalMap ，并把变量放入到map中，与此同时，map的key为threadLocal，value为actionContext类型。因此每一次请求，都是一个新的线程，每一个线程都有一个自己的ActionContext，存在threadLocal中。

ActionContext.getContext()源码：

**static** ThreadLocal<ActionContext> actionContext = **new** ThreadLocal();

**public static** ActionContext getContext() {  
 **return** (ActionContext)actionContext.get();  
}

本质是用了threadLocal中的get()方法：

**public** T get() {  
 **Thread** t = **Thread**.currentThread();  
 **ThreadLocalMap** map = getMap(t);  
 **if** (map != **null**) {  
 **ThreadLocalMap**.**Entry** e = map.getEntry(**this**);  
 **if** (e != **null**)  
 **return** (T)e.value;  
 }  
 **return** setInitialValue();  
}

### 问题七：为什么 EL也能访问值栈中的数据

StrutsPreparedAndExecuteFilter的doFilter代码中 request = prepare.wrapRequest(request);

\* 对Request对象进行了包装 ，StrutsRequestWrapper

\* 重写request的 getAttribute

Object attribute = super.getAttribute(s);

if (attribute == null) {

attribute = stack.findValue(s);

}

访问request范围的数据时，如果数据找不到，去值栈中找 request对象 （具备访问值栈数据的能力 （查找root的数据））

因为Struts2中使用的request、session等web对象都是被struts2中包装过的，重写了getAttribute方法，用el表达式获取数据的时候，访问request范围的数据时，如果数据找不到，去值栈中找 request对象

## OGNL表达式 常见使用

#、 % 、$ 符号使用

**1） # 的 使用**

用法一 # 代表 ActionContext.getContext() 上下文

<s:property value="#request.name" /> ------------> ActionContext().getContext().getRequest().get("name");

#request

#session

#application

#attr

#parameters

用法二 ： 不写# 默认在 值栈中root中进行查找

<s:property value="name" /> 在root中查找name属性

\* 查询元素时，从root的栈顶元素 开始查找， 如果访问指定栈中元素 <s:property value="[1].name" /> 访问栈中第二个元素name属性

\* 访问第二个元素对象 <s:property value="[1].top" />

用法三 ：进行投影映射 （结合复杂对象遍历 ）

1）集合的投影(只输出部分属性

<h1>遍历集合只要name属性</h1>

<s:iterator value="products.{name}" var="pname">

<s:property value="#pname"/>

</s:iterator>

2）遍历时，对数据设置条件

<h1>遍历集合只要price大于1500商品</h1>

<s:iterator value="products.{?#this.price>1500}" var="product">

<s:property value="#product.name"/> --- <s:property value="#product.price"/>

</s:iterator>

3）综合

<h1>只显示价格大于1500 商品名称</h1>

<s:iterator value="products.{?#this.price>1500}.{name}" var="pname">

<s:property value="#pname"/>

</s:iterator>

用法四： 使用#构造map集合

经常结合 struts2 标签用来生成 select、checkbox、radio

<h1>使用#构造map集合 遍历</h1>

<s:iterator value="#{'name':'aaa','age':'20', 'hobby':'sport' }" var="entry">

key : <s:property value="#entry.key"/> , value: <s:property value="#entry.value"/> <br/>

</s:iterator>

**2) %的使用**

用法一： 结合struts2 表单表单使用， 通过%通知struts， %{}中内容是一个OGNL表达式，进行解析

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

用法二： 设置ognl表达式不解析 %{'ognl表达式'}

<s:property value="%{'#request.username'}"/>

**3）$的使用**

用法一 ：用于在国际化资源文件中，引用OGNL表达式

在properties文件 msg=欢迎您, ${#request.username}

在页面

<s:i18n name="messages">

<s:text name="msg"></s:text>

</s:i18n>

\* 自动将值栈的username 结合国际化配置信息显示

用法二 ：在Struts 2配置文件中，引用OGNL表达式

<!-- 在Action 提供 getContentType方法 -->

<param name="contentType">${contentType}</param>

\* ${contentType} 读取值栈中contentType数据，在Action提供 getContentType 因为Action对象会被压入值栈，

contentType是Action属性，从值栈获得

结论： #使用ognl表达式获取数据，% 控制ognl表达式是否解析 ，$ 用于配置文件获取值栈的数据