Struts2 值栈

一、咄咄怪事

1.奇怪的 EL 表达式

给一个目标 Action 类发送一个 Struts2 请求，假设目标 Action 类中包含如下方法：

```
public String getMessage() {
    return "I am very happy in atguigu!";
}
```

则在结果 JSP 页面上可以使用如下 EL 表达式获取上述方法的返回值：

```
${requestScope.message }
```

但毫无疑问我们根本没有将任何数据以 message 为属性名保存到请求域中，那么这个奇怪的 EL 表达式是如何读取到数据的呢？

2.偷梁换柱的 getAttribute()方法

我们逐步来探究一下上面提出的问题。

①第一步

根据 EL 表达式语法，${requestScope.message }会被翻译成如下代码：

```
request.getAttribute("message");
```

②第二步

在页面上直接输出 request 对象得到如下结果：

```
org.apache.struts2.dispatcher.StrutsRequestWrapper@756ea3b2
```

而由 Tomcat 创建的 request 对象本来应该是：

```
org.apache.catalina.connector.RequestFacade@39e2cdaa
```

说明 request 对象已经不是我们在纯 Servlet 容器环境下使用的 request 对象了，而是一个经过 Struts2 包装过的 request 对象。

③第三步

StrutsRequestWrapper 通过继承 javax.servlet.http.HttpServletRequestWrapper 类对原始的 request 对象进行了包装，仅修改了 getAttribute()一个方法的行为。代码如下：

```
/**
 * 首先从原始的请求域中获取属性值，如果找不到就从值栈中读取
 *
 * @param key  请求域数据的键
 */
```

```
public Object getAttribute(String key) {
    if (key == null) {
        throw new NullPointerException("You must specify a key value");
    }

    //如果禁用了从值栈读取数据的功能或key是以javax.servlet开始的则从原始的请求域中读取数据并直接返回
    if (disableRequestAttributeValueStackLookup ||
key.startsWith("javax.servlet")) {
        return super.getAttribute(key);
    }

    //获取ActionContext对象
    ActionContext ctx = ActionContext.getContext();

    //尝试从原始的请求域中读取数据
    Object attribute = super.getAttribute(key);

    //如果ActionContext对象不为空且从原始的请求域中没有获取到指定数据
    if (ctx != null && attribute == null) {

        boolean alreadyIn = isTrue((Boolean)
ctx.get(REQUEST_WRAPPER_GET_ATTRIBUTE));

        if (!alreadyIn && !key.contains("#")) {
            try {
                ctx.put(REQUEST_WRAPPER_GET_ATTRIBUTE, Boolean.TRUE);

                //通过ActionContext对象获取“值栈”对象
                ValueStack stack = ctx.getValueStack();

                if (stack != null) {
                    //如果值栈对象不为空，则尝试根据key“查找”数据
                    attribute = stack.findValue(key);
                }
            } finally {
                ctx.put(REQUEST_WRAPPER_GET_ATTRIBUTE, Boolean.FALSE);
            }
        }
    }
    return attribute;
}
```

结论就是在 Struts2 环境下，request 对象的 getAttribute()方法会首先从请求域中获取数据，如果获取不到，则通过 ValueStack 对象获取数据。

二、ValueStack

1.概述

ValueStack 是一个接口

```
com.opensymphony.xwork2.util.ValueStack
```

它的实现类是

```
com.opensymphony.xwork2.ognl.OgnlValueStack
```

①作用

Struts2 为每一个请求都分配了一个 ValueStack 对象，目的是为每一个请求都提供一个临时的数据存储空间。

②两个数据容器

分析 OgnlValueStack 源码，其中包含两个重要的数据容器

```
CompoundRoot root;//通常称为“对象栈”
transient Map<String, Object> context;//通常称为“Map 栈”
```

③对象栈

[1]CompoundRoot 类声明

```
//继承自 ArrayList
public class CompoundRoot extends ArrayList {
```

[2]CompoundRoot 是在 List 基础之上实现的“栈”——后进先出。

```
//返回栈顶对象
public Object peek() {
    return get(0);
```

```
    }

    //删除栈顶对象并返回
    public Object pop() {
        return remove(0);
    }

    //将对象压入栈顶
    public void push(Object o) {
        add(0, o);
    }
```

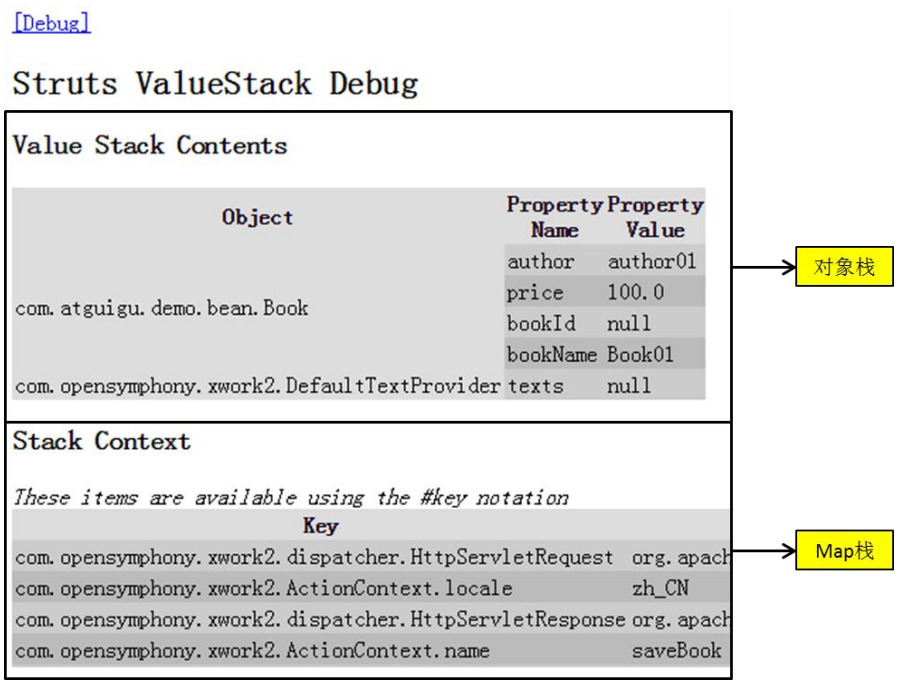
[3]如何方便的查看值栈中的数据？

(1)在 JSP 页面上导入 Struts2 标签库

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

(2)使用<s:debug></s:debug>标签

(3)点击[Debug]超链接即可展开值栈数据列表



[4]对象栈中的数据

(1)默认情况下对象栈中的数据

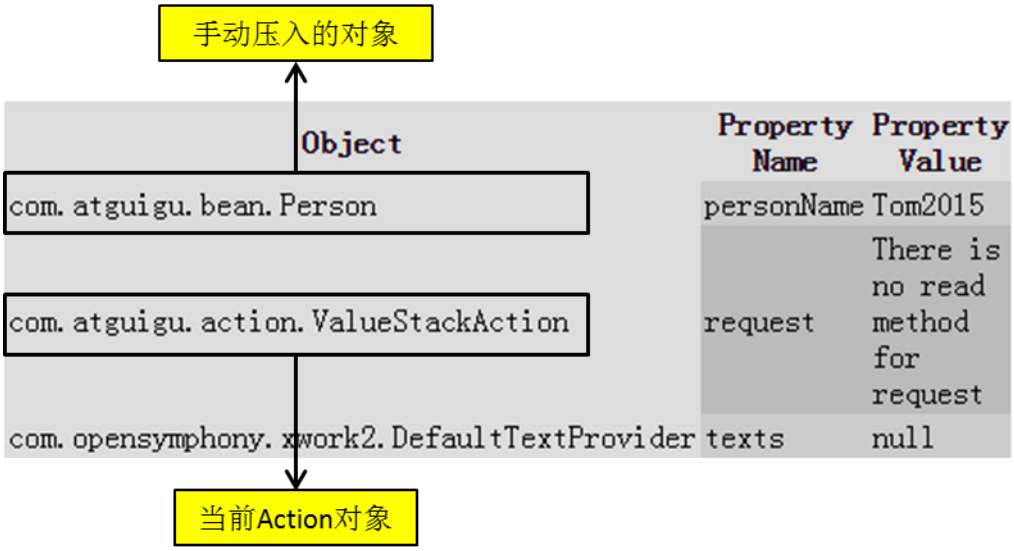
可以看到，默认情况下 Struts2 会将当前 Action 对象压入值栈的栈顶，前面提及的 message 属性值就是从栈顶 Action 对象中获取的。

如何理解“当前 Action”？

●当前请求的目标 Action

●转发到当前页面的“来源”Action

(2)手动压入对象后



④Map 栈

[1]似曾相识的 Map 栈对象

大家是否还记得，在获取 Web 资源时用到的 ActionContext 类中，有一个成员变量指向了一个 Map，它的声明如下

```
private Map<String, Object> context;
```

而我们 OgnlValueStack 类中也有一个 context 成员变量指向了一个 Map 对象，声明如下

```
transient Map<String, Object> context;
```

他们指向的是同一个对象吗？我们可以比较一下

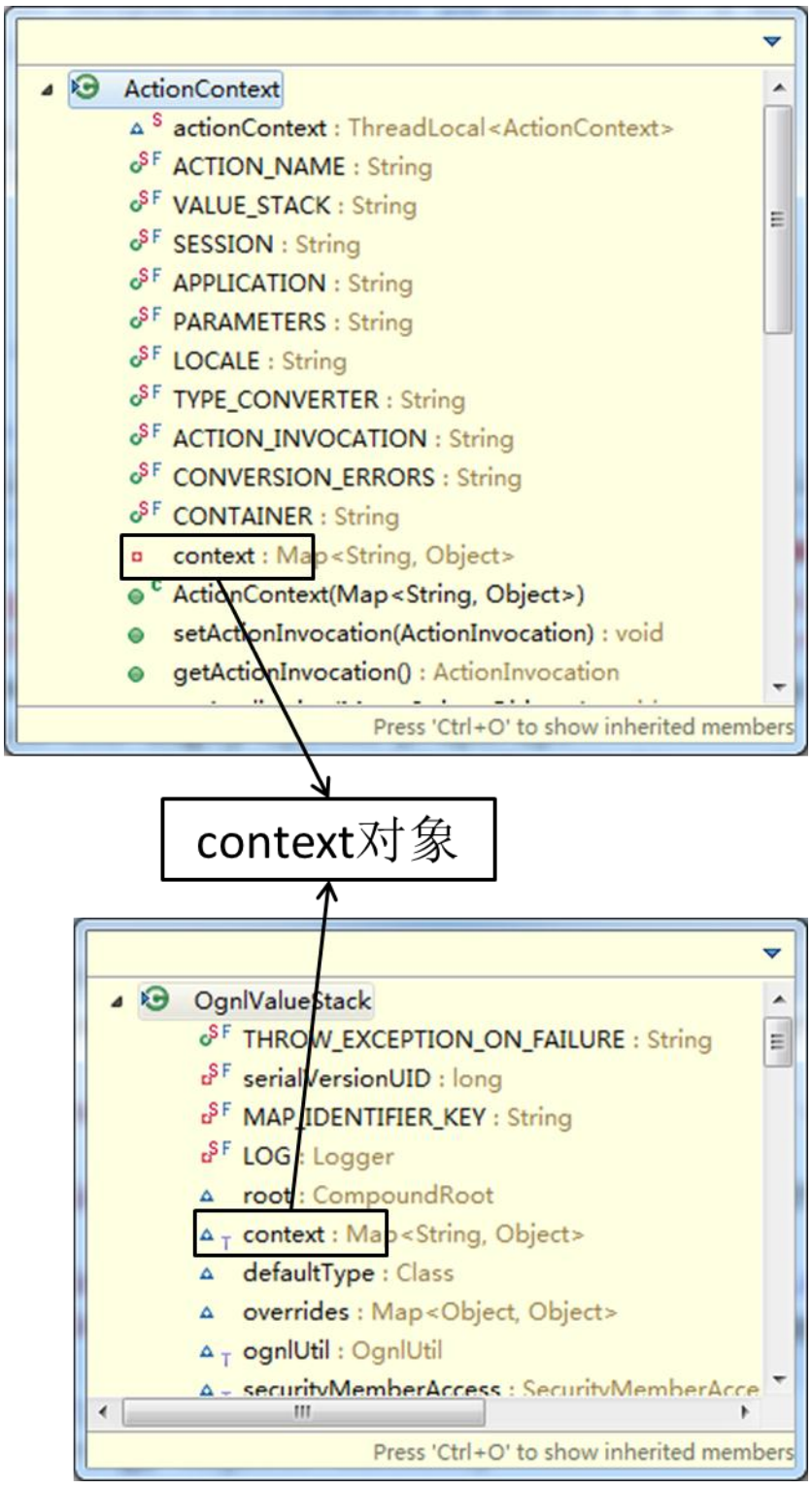
```
//1.从值栈对象中获取context对象
Map<String, Object> contextFromVS = valueStack.getContext();

//2.从ActionContext对象中获取context对象
Map<String, Object> contextFromAC =
```

```
ActionContext.getContext().getContextMap();

//比较这两种方式获取的context对象，返回true
System.out.println(contextFromVS == contextFromAC);
```

通过程序验证我们发现，contextFromVS 和 contextFromAC 指向同一块内存区域，证明他们指向的是同一个 Map 对象。它们之间的关系可以用下图表示：



[2]context 对象中的内容

context 对象中的内容可以分为如下几个部分[由于 context 对象是 Map 类型下面使用 key/vlaue 形式展示]：

(1)原生的 Web 资源对象

Key	Value

com.opensymphony.xwork2.dispatcher.HttpServletRequest	Struts2 包装过的 Request 对象
com.opensymphony.xwork2.dispatcher.HttpServletResponse	原生的 response 对象
com.opensymphony.xwork2.dispatcher.ServletContext	原生的 ServletContext 对象

补充说明：看到这里大家可能会想，之前获取 Web 资源时 ActionContext 给我们提供的是封装相关数据的 Map 对象呀？怎么还有原生的 Web 对象呢？原生的 Web 对象不是从 ServletActionContext 中获取的吗？其实大家看看 ServletActionContext 的源码就能够发现，它获取原生 Web 资源对象本质上也是从 ActionContext 中获取的。

```
public static HttpServletRequest getRequest() {
   return (HttpServletRequest) ActionContext.getContext().get(HTTP_REQUEST);
}
```

包括 Struts2 为实现了 XxxAware、ServletXxxAware 接口的 Action 对象注入 Web 资源对象本质上也都是从 ActionContext 中获取的。

当然 ActionContext 获取 Web 资源对象时也都是从 context 对象这个 Map 中获取的。

(2)Web 资源对应的 Map 对象

Key	Value
com.opensymphony.xwork2.ActionContext.application	封装 Application 域数据的 Map 对象
application	封装 Application 域数据的 Map 对象
request	封装请求域数据的 Map 对象
com.opensymphony.xwork2.ActionContext.parameters	封装请求参数数据的 Map 对象
parameters	封装请求参数数据的 Map 对象
com.opensymphony.xwork2.ActionContext.session	封装 Session 域数据的 Map 对象
session	封装 Session 域数据的 Map 对象
attr	可以按范围从小到大顺序在所有域对象中查询数据的特殊 Map 对象

补充说明：

[1]除了 request 的 Map 对象，其他几个域对象都有两个键指向，一个键较长，一个键较短，嗯，放心吧，他们指向的是同一个对象。

[2]关于 attr 这个 Map，它能够按照范围由小到大的顺序在各个域对象中查询数据，看源码就知道——它的实现类是 org.apache.struts2.util.AttributeMap

```
    public Object get(Object key) {
        PageContext pc = getPageContext();

        if (pc == null) {
            Map request = (Map) context.get("request");
            Map session = (Map) context.get("session");
```

```
            Map application = (Map) context.get("application");

            if ((request != null) && (request.get(key) != null)) {
                return request.get(key);
            } else if ((session != null) && (session.get(key) != null)) {
                return session.get(key);
            } else if ((application != null) && (application.get(key) != null)) {
                return application.get(key);
            }
        } else {
            try{
                return pc.findAttribute(key.toString());
            }catch (NullPointerException npe){
                return null;
            }
        }

        return null;
    }
```

(3)ValueStack 对象

Key	Value
com.opensymphony.xwork2.util.ValueStack.ValueStack	值栈对象

补充说明：看到这个你会不会困惑呢？是的，你没有看错，Struts2 确实是把 ValueStack 值栈对象放入了 context 这个 Map 当中，上面显示的 key 指向的是 ValueStack 对象的引用。ActionContext 对象就是通过这种方式获取 ValueStack 对象的。其实 Struts2 还将 ValueStack 对象的引用又复制了一份放到了请求域中——是不是有点狡兔三窟的感觉？通过很多方式都能够得到 ValueStack 对象的引用，之所以这样无非是为了在不同情况下方便的获取值栈。稍后我们会盘点获取值栈对象的所有方法。

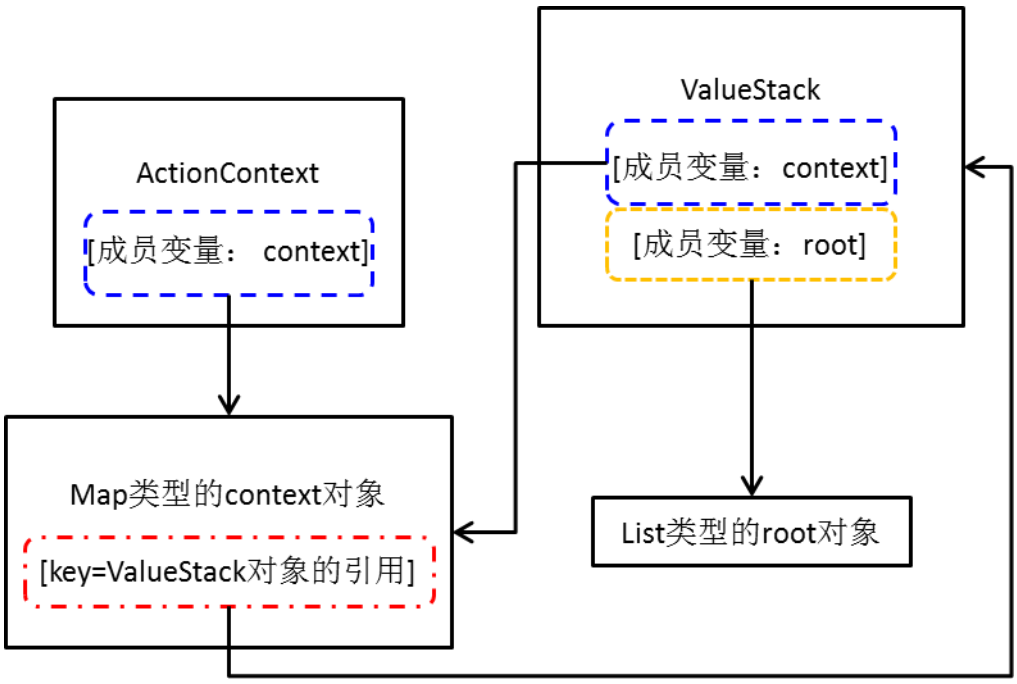
(4)与 Action 类运行相关的环境信息

Key	Value
com.opensymphony.xwork2.ActionContext.name	当前 action 的 name 属性值
action	当前 Action 类的实例对象
com.opensymphony.xwork2.ActionContext.conversionErrors	{}空 Map
com.opensymphony.xwork2.ActionContext.locale	en_US 或 zh_CN 等
last.property.accessed	null
last.bean.accessed	null
xwork.NullHandler.createNullObjects	false
current.property.path	null
com.opensymphony.xwork2.ActionContext.container	com.opensymphony.xwork2.inject.ContainerImpl 对象
com.opensymphony.xwork2.ActionContext.actionInvocation	com.opensymphony.xwork2.DefaultActionInvocation 对象

xwork.MethodAccessor.denyMethodExecution	false
report.conversion.errors	false
struts.actionMapping	ActionMapping Map 对象

⑤总结

ValueStack 对象与其他对象的关系可以概括为下面这个图



获取 ValueStack 对象的途经

通过上面对 ValueStack 对象数据结构的分析我们很容易得出获取 ValueStack 对象的两个较为直接的途经：

①通过 ActionContext 对象获取

```
ValueStack valueStack = ActionContext.getContext().getValueStack();
```

②通过读取请求域获取

```
ValueStack valueStack = (ValueStack) requestMap.get("struts.valueStack");
```

ValueStack 对象和请求的关系

在 Action 类中打印 ValueStack 对象发现每一个 ValueStack 对象的 HashCode 值都不同，证明 Struts2 为每一个请求创建一个新的 ValueStack 对象。这就保证了读写值栈中数据的时候不会有线程安全的问题。

三、OGNL

1.OGNL 概述

①OGNL 是 Apache Commons 下的一个子项目，同在 Commons 下的子项目

还有我们用过的 FileUpload、IO、BeanUtils、DbUtils、Logging 等。

②OGNL 并不依赖 Struts2 存在，创建一个 Java 工程就能够对 OGNL 进行测试，此时需要导入两个 JAR 包

javassist-3.11.0.GA.jar

ognl-3.0.6.jar

[这两个 JAR 包可以在 Struts2 提供的 blank 例子工程的 lib 目录下找到]

2.OGNL：对象图导航语言

①对象图：Java 对象可连续访问的级联属性形成的图形结构

例如：department.getEmpList().get(3).getAddress().getCity();

当对象嵌套层次较深时，按照对象自身的属性进行访问往往过于冗长。

②导航语言：以表达式的形式快速定位到对象图中的某个属性

类似：jQuery 选择器、xPath 表达式等

③OGNL 表达式语法

[1]Ognl.getValue()方法未指定上下文对象：从根对象中读取数据

(1)读取普通对象的属性："对象.属性名"或"对象[属性名]"

(2)读取 List/数组对象的元素："List/数组对象[下标]"

(3)读取 Map 对象中键对应的值："Map 对象['键']"或"Map 对象.键"

[2]Ognl.getValue()方法指定了上下文对象

(1)OGNL 表达式中没有使用#号，表示在根对象中查找数据

(2)OGNL 表达式中以#key 的形式表示在上下文对象中以 key 为键查找对应的值

④其他语法

[1]调用对象非静态方法

"对象.方法名()"或"对象.方法名('字符串或基本数据类型参数')"

[2]调用对象静态方法：

"@全类名@方法名()"或"@全类名@方法名('字符或基本数据串类型参数')"

[3]访问对象静态属性："@全类名@属性名"

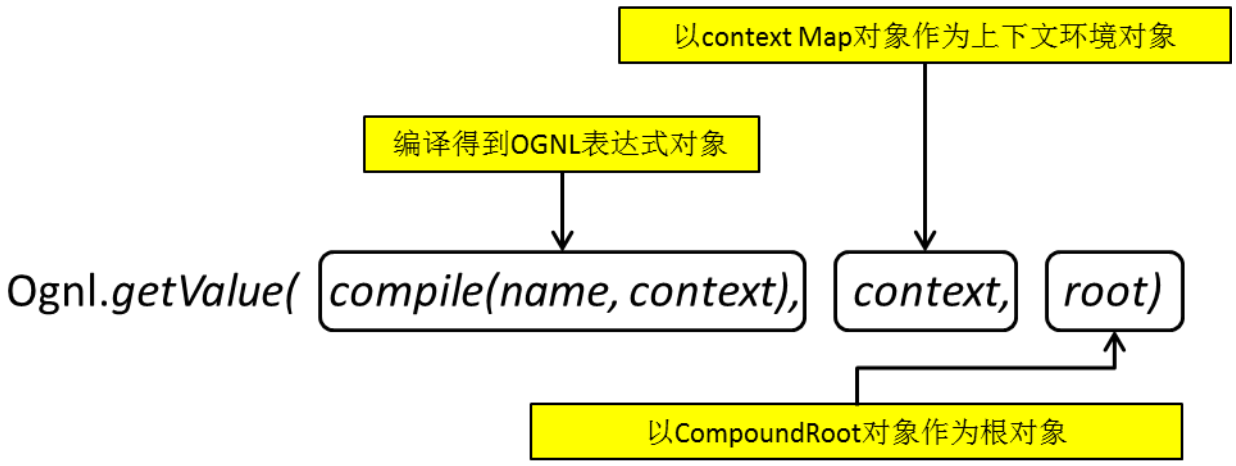
3.ValueStack 中的 OGNL

①对应关系

[1]CompoundRoot 对象对应 OGNL 中的根对象

[2]context 对象对应 OGNL 中的上下文环境对象

源码：com.opensymphony.xwork2.ognl.OgnlUtil.getValue(String, Map<String, Object>, Object)



②重要区别

在 OGNL 语法中，访问 List 或数组类型的根对象时必须使用“[]”指定下标，但如果根对象是 CompoundRoot 对象，则可以省略下标。

如果省略了下标，仅指定了对象的属性名时，将从栈顶元素向下开始查找包含这个属性名的对象，直到为止。

即使指定了下标，也不是仅从下标指向的对象开始查找，而是从指向的对象开始向下查找直到找到为止。

③访问静态资源

默认情况下 Struts2 不允许通过 OGNL 表达式访问静态资源，必须加入如下设置

```
<constant name="struts.ognl.allowStaticMethodAccess" value="true"></constant>
```

④通过 ValueStack 对象调用 public abstract Object findValue(String expr);方法即可执行 OGNL 表达式