

# Struts2 第三天

## 第1章 OGNL 表达式入门

### 1.1 OGNL 表达式概述

#### 1.1.1 什么是 OGNL 表达式

OGNL 的全称是对象图导航语言（Object-Graph Navigation Language），它是一种功能强大的开源表达式语言，使用这种表达式语言，可以通过某种表达式语法，存取 Java 对象的任意属性，调用 Java 对象的方法，同时能够自动实现必要的类型转换。如果把表达式看作是一个带有语义的字符串，那么 OGNL 无疑成为了这个语义字符串与 Java 对象之间沟通的桥梁。

#### OGNL

 编辑

OGNL是Object-Graph Navigation Language的缩写，它是一种功能强大的表达式语言，通过它简单一致的表达式语法，可以存取对象的任意属性，调用对象的方法，遍历整个对象的结构图，实现字段类型转化等功能。它使用相同的表达式去存取对象的属性。

问题：我们学了 OGNL 表达式之后，EL 表达式还用不用？

这个问题，我们将在第 3 章找到答案。

#### 1.1.2 OGNL 表达式的由来：

它原本是xwork2中的默认表达式语言，当年 OpenSymphony 和 apache 在合作开发 struts2 框架时，把这个表达式也引进来了，所以就变成了 struts2 的默认表达式语言。

结论：在 struts2 中，ognl 表达式就是默认的表达式语言。

#### 1.1.3 OGNL 表达式的使用要求：

要想使用 ognl 表达式，一般情况下都得需要使用 struts2 的标签库。

例如：

struts2: `<s:textfield name="username" label="用户名"/>`

html: 用户名: `<input type="text" name="username"/>`

细节：

struts2 框架本身有可能把 html 中的某些字符串看成是 ognl 表达式。用来给属性赋值。



### 1.1.4 它的特点：

它不仅可以用于取值，显示。还可以赋值。**取值**是我们程序员使用框架做的事情。**赋值**是框架为我们做的。

## 1.2 OGNL 表达式的基本用法

### 1.2.1 借助 s:property 标签输出内容到浏览器

#### 1.2.1.1 s:property 的作用

写法：

```
<%--导入标签库--%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<%--要想使用 OGNL 表达式，需要借助 struts2 的标签--%>
<s:property value="OGNLExpression"/>
```

作用：

把 value 属性取值所对应的内容输出到浏览器上。注意：它不是把 value 的值输出到浏览器上。

属性：

value：取值是一个 OGNL 表达式。

运行结果：

此时浏览器不会有任何内容显示。究其原因，我们先要理解【把 value 属性取值所对应的内容输出到浏览器上】这句话。这句话中有【所对应的内容】这几个字，这几个字就说明它要去某个地方找数据，那自然是找到了就显示，找不到就什么都不显示了。

那它是去哪找数据了呢？是我们第二章中要讲解的 OGNL 上下文。

既然我们现在无法得知 OGNL 上下文中有什么，那么我们就先来把 value="OGNLExpression" 里面的 OGNLExpression 看成是普通字符串吧。如何把一个 OGNL 表达式看成是字符串呢？请看下一小节。

#### 1.2.1.2 把一个 OGNL 表达式看成字符串的方式

把一个 OGNL 表达式看成是字符串的方式是：

**%{'OGNL 表达式'} 或者是 %{"OGNL 表达式"}**

到底是使用单引号还是双引号，是由外层引号决定的。以上两种都可以实现。我们也可以简写，就是把外面的%{}去掉，直接用引号括住表达式。

**'OGNL 表达式' 或者是 "OGNL 表达式"**

示例代码：



```
<s:property value="%{'OGNLEExpression'}"/>
<s:property value='%{"OGNLEExpression1"}'/>
```

### 1.2.1.3 把一个字符串看成 OGNL 表达式的方式

把一个字符串看成是 OGNL 表达式的方式是：**`%{字符串}`**

## 1.2.2 OGNL 表达式调用普通方法

OGNL 表达式的强大之处在于它可以让我们在表达式中访问对象的方法，例如下面的代码：

```
长度: <s:property value="'OGNLEExpression'.length()"/><br/>
转大写: <s:property value="'OGNLEExpression'.toUpperCase()"/><br/>
分隔: <s:property value="'OGNLEExpression'.split('E')"/><br/>
```

### 1.2.3 OGNL 表达式访问静态属性和静态方法

OGNL 表达式还支持访问静态成员，这其中包括静态属性和静态方法，但是必须按照提供的格式编写，格式是：**`@包名.包名...类名@静态属性名称`**。这其中...的含义表示有几级包，就写几个包名。

示例代码如下：

```
int 的最大值: <s:property value="@java.lang.Integer@MAX_VALUE"/>

<!--但是会议用 OGNL 表达式访问静态方法，需要开启对静态方法访问的支持。
struts2 框架默认不支持 OGNL 表达式的静态方法调用，开启的方式如下：
<constant name="struts.ognl.allowStaticMethodAccess"
value="true"></constant>
-->

随机数: <s:property value="@java.lang.Math@random()"/>
```

## 1.2.4 OGNL 表达式创建集合

### 1.2.4.1 list 集合

用 HTML 在浏览器上输出一个单选性别：

```
<input type="radio" name="gender" value="男"/>男
<input type="radio" name="gender" value="女"/>女
<br/>
```

用 Struts2 的单选按钮标签输出一个单选



```
<%--s:radio 用于在浏览器上显示一个单选按钮
list 属性取值是一个 OGNL 表达式

{} 就表示创建了一个 List 集合 List list = new ArrayList();
{'男','女'}
list.add("男");list.add("女");

--%>

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

### 1.2.4.2 map 集合

```
用 HTML 在浏览器上输出一个单选性别：Map 结构

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

<%--#{ } 就表示创建了一个 Map
      里面的写法
      #{ 'key': 'value', 'key': 'value' ..... }

--%>

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

## 第2章 OGNL 上下文

### 2.1 ContextMap

#### 2.1.1 ContextMap 概述

它是 OGNL 上下文对象，是 struts2 中封装数据最大的对象。我们一次请求中所有用到的信息都可以在它里面找到。它是一个 Map 结构的对象，其中 key 是字符串，value 是一个 Object。

它里面到底封装了什么内容呢？请看下一章节

#### 2.1.2 ContextMap 中封装的数据

下图是我们在 struts2 官方文档中找到的，请以此为准：





```

context map---
    --application
    --session
    --value stack(root)
    --action (the current action)
    --request
    --parameters
    --attr (searches page, request, session, then application scopes)
    
```

我们把这些内容拿出来逐个分析一下，得到下面的表格：

Map 的 key (类型是 String)	Map 的 Value (类型是 Object)	说明信息
application	Java.util.Map<String,Object>	封装的应用域中的所有数据
session	Java.util.Map<String,Object>	封装的会话域中的所有数据
request	Java.util.Map<String,Object>	封装的请求域中的所有数据
valueStack(特殊)	com.opensymphony.xwork2.ognl.OgnlValueStack	它是 List 结构
parameters	Java.util.Map<String,String[]>	封装的是请求参数
attr	Java.util.Map<String,Object>	封装的是四大域的组合数据，从最小的域开始搜索
action	com.opensymphony.xwork2.ActionSupport	当前执行的动作类对象

用颜色把它区分开，目的是让我们明确一件事：

蓝色的：是我们已经会了的。

四大域对象在 **jsp** 那天就已经学过了。

绿色的：是我们不用管的。

**parameters**：现在我们已经会用模型驱动了，所以再也不用自己取了。

**attrs**：由于 **ContextMap** 中已经包含了三大域，**page** 域的范围又太小了。

**action**：在后面我们讲的值栈中，会提供这个当前执行的动作类。

红色的：它是我们第一次接触的，以前不会。而且是 **struts2** 中经常用的，所以它是重点！

我们关心的部分，结构如下图所示：

ContextMap 大Map	key    java.lang.String	value    java.lang.Object	
	application	key    String	value    Object 应用域中的    所有数据
	session	key    String	value    Object 会话域中的    所有数据
	request	key    String	value    Object 请求域中的    所有数据
	com.opensynphony.xwork2.util.ValueStack.ValueStack	ValueStack对象	

## 2.2 ActionContext

### 2.2.1 ActionContext 对象概述

它是一个工具类，是 **struts2** 框架提供给我们的，可以让我们调用其中的方法，快速的操作 **ContextMap**。用它操作 **OGNL** 上下文对象，比直接操作 **ContextMap** 要方便很多。

### 2.2.2 ActionContext 对象以及和 ContextMap 的关系

**ActionContext** 就相当于对 **ContextMap** 进行了一次再封装。

### 2.2.3 ActionContext 何时创建

由于 **ActionContext** 是操作的 **ContextMap**，而 **ContextMap** 中封了我们一次请求的所有数据，所以它的创建应该是每次请求访问 **Action** 时，即核心控制器(**StrutsPrepareAndExecuteFilter**)的 **doFilter** 方法执行时，下图是代码截取：



```
StrutsPrepareAndExecuteFilter.class  index.jsp  struts.xml  ActionContext.class  CustomerAction.java  PrepareOperations.class  ActionContext.class
78
79 public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException, S
80
81 HttpServletRequest request = (HttpServletRequest) req;
82 HttpServletResponse response = (HttpServletResponse) res;
83
84 try {
85     if (excludedPatterns != null && prepare.isUrlExcluded(request, excludedPatterns)) {
86         chain.doFilter(request, response);
87     } else {
88         prepare.setEncodingAndLocale(request, response);
89         prepare.createActionContext(request, response);
90         prepare.assignDispatcherToThread();
91         request = prepare.wrapRequest(request);
92         ActionMapping mapping = prepare.findActionMapping(request, response, true);
93         if (mapping == null) {
94             boolean handled = execute.executeStaticResourceRequest(request, response);
95             if (!handled) {
96                 chain.doFilter(request, response);
97             }
98         } else {
99             execute.executeAction(request, response, mapping);
100         }
101     }
102 } finally {
```

## 2.2.4 ActionContext 的线程安全

我们都知道，java 的 web 工程是多线程的，那么每个线程在访问 Action 时，都会创建自己的 ActionContext，那么是如何保证在获取 ActionContext 时，每个线程都能获取到自己的那个呢？

答案就是，每次创建 ActionContext 时，把对象绑定到当前线程上。下图是代码截图：

```
StrutsPrepareAndExecuteFilter.class  index.jsp  struts.xml  ActionContext.class  CustomerAction.java  PrepareOperations.class  ActionContext.class
78
79 public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException, S
80
81 HttpServletRequest request = (HttpServletRequest) req;
82 HttpServletResponse response = (HttpServletResponse) res;
83
84 try {
85     if (excludedPatterns != null && prepare.isUrlExcluded(request, excludedPatterns)) {
86         chain.doFilter(request, response);
87     } else {
88         prepare.setEncodingAndLocale(request, response);
89         prepare.createActionContext(request, response);
90         prepare.assignDispatcherToThread();
91         request = prepare.wrapRequest(request);
92         ActionMapping mapping = prepare.findActionMapping(request, response, true);
93         if (mapping == null) {
94             boolean handled = execute.executeStaticResourceRequest(request, response);
95             if (!handled) {
96                 chain.doFilter(request, response);
97             }
98         } else {
99             execute.executeAction(request, response, mapping);
100         }
101     }
102 } finally {
```

这是 ActionContext 类中的方法：





```
/**
 * Sets the action context for the current thread.
 *
 * @param context the action context.
 */
public static void setContext(ActionContext context) {
    actionContext.set(context);
}
// ThreadLocal<ActionContext> com.opensymphony.xwork2.ActionContext.actionContext
/**
 * Returns the ActionContext specific to the current thread.
```

那么接下来，新的问题产生了，既然把 `ActionContext` 绑定到线程上了，我们该如何获取这个对象呢？

请看下一章节。

## 2.2.5 ActionContext 的获取

使用 `ActionContext` 类中的静态方法，如下图所示：

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

图中的 `actionContext` 到底是什么数据类型呢？下图将告诉大家：

```
index.jsp ActionContext.class
39 * @author Patrick Lightbody
40 * @author Bill Lynch (docs)
41 */
42 public class ActionContext implements Serializable {
43
44     static ThreadLocal<ActionContext> actionContext = new ThreadLocal<ActionContext>();
45 }
```

既然说 `ActionContext` 是对 `ContextMap` 的再封装，那么它是怎么封装的呢？下图将展示给大家：

```
index.jsp ActionContext.class
42 public class ActionContext implements Serializable {
43
44     static ThreadLocal<ActionContext> actionContext = new ThreadLocal<ActionContext>();
45
46     private Map<String, Object> context;
47
48     /**
49      * Creates a new ActionContext initialized with another context.
50      *
51      * @param context a context map.
52      */
53     public ActionContext(Map<String, Object> context) {
54         this.context = context;
55     }
56 }
```





## 2.2.6 获取 ContextMap 中的数据

### 2.2.6.1 s:debug 标签的使用

```
<%-- 引入标签库 --%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<%--1、struts2 的 debug 标签
    它是一个用于开发阶段的标签，查看我们 OGNL 上下文中内容的标签 --%>
<s:debug/>
```

### 2.2.6.2 使用 OGNL 表达式获取 Map 中的数据

此章节我们将的是，获取 Map 中的数据，包括 ContextMap，以及三大域中的数据。我们首先往 Map 和三大域中存入一些数据：

```
/**
 * 对 ContextMap
 * 和
 * 三大域 (ServletContext, HttpSession, ServletRequest)
 * 的存值操作
 */
public class DemolAction extends ActionSupport {

    /**
     * Map 部分的存值操作
     * @return
     */
    public String demol() {
        //1. 获取 ActionContext 对象          他就是一个 Map 结构
        ActionContext context = ActionContext.getContext(); //从当前线程上获取 ActionContext 对象

        //2. 往 Map 存入数据
        context.put("contextMap", "hello context map");

        //3. 获取 key 为 application 的 Map 对象
        //第一种方式：使用 map 操作，叫解耦的方式，不依赖原始 ServletAPI 对象
        Map<String, Object> applicationMap = context.getApplication(); // (Map<String, Object>) context.get("application");
        applicationMap.put("applicationMap", "hello application map");
        //第二种方式：使用原始 ServletAPI 对象操作，方便的方式。
        ServletContext application = ServletActionContext.getServletContext();
        application.setAttribute("applicationAttr", "hello application")
    }
}
```



```
attr");  
  
        return SUCCESS;  
    }  
}
```

在页面中使用 OGNL 表达式获取：

<%--2、使用 OGNL 表达式获取 Map 中的数据 需要借助 s:property

获取 map 的数据：都是根据 Key 获取 value，所以我们要提供的是 key。

如何表示 key：使用#，后面的内容就表示 key

写法：

```
#key
```

```
--%>
```

获取 ContextMap 中的数据：<s:property value="#contextMap"/>

```
<hr/>
```

<%--3、使用 OGNL 表达式获取三大域中的数据

也需要使用 s:property 标签

写法就是

```
#key.key
```

```
--%>
```

获取 key 为 application 的 map 中的数据

```
<s:property value="#application.applicationMap"/><br/>
```

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

## 2.3 ValueStack 对象

### 2.3.1 ValueStack 对象概述

ValueStack 是 Struts 的一个接口，字面意义为值栈，OgnlValueStack 是 ValueStack 的实现类，客户端发起一个请求 struts2 架构会创建一个 action 实例同时创建一个 OgnlValueStack 值栈实例，OgnlValueStack 贯穿整个 Action 的生命周期。

它是 ContextMap 中的一部分，里面的结构是一个 List，是我们可以快速访问数据一个容器。它的封装是由 struts2 框架完成的。

通常情况下我们是从页面上获取数据。它实现了栈的特性（先进后出）。

### 2.3.2 ValueStack 的内部结构

在 OgnlValueStack 中包含了一个 CompoundRoot 的对象，该对象继承了 ArrayList，并且提供了只能操作集合第一个元素的方法，所以我们说它实现了栈的特性。同时，它里面定义了一个 ContextMap 的引用，也就是说，我们有值栈对象，也可以通过值栈来获取 ContextMap。



```
index.jsp  ActionContext.class  OgnlValueStack.class
50  */
51  public class OgnlValueStack implements Serializable, ValueStack, ClearableValueStack, MemberAccessValueStack {
52
53      public static final String THROW_EXCEPTION_ON_FAILURE = OgnlValueStack.class.getName() + ".throwExceptionOnFailure";
54
55      private static final long serialVersionUID = 370737852934925530L;
56
57      private static final String MAP_IDENTIFIER_KEY = "com.opensymphony.xwork2.util.OgnlValueStack.MAP_IDENTIFIER_KEY";
58      private static final Logger LOG = LoggerFactory.getLogger(OgnlValueStack.class);
59
60      CompoundRoot root;
61      transient Map<String, Object> context;
62      Class defaultType;
63      Map<Object, Object> overrides;
64      transient OgnlUtil ognlUtil;
65      transient SecurityMemberAccess securityMemberAccess;
66      private transient XWorkConverter converter;
67
68      private boolean devMode;
69      private boolean logMissingProperties;
70
```

→ 是个结构  
→ 引用的是OGNL上下文



```
/**
 * A Stack that is implemented using a List.
 *
 * @author plightbo
 * @version $Revision$
 */
public class CompoundRoot extends ArrayList {

    public CompoundRoot() {

    }

    public CompoundRoot(List list) {
        super(list);
    }

    public CompoundRoot cutStack(int index) {
        return new CompoundRoot(subList(index, size()));
    }

    public Object peek() {
        return get(0);
    }

    public Object pop() {
        return remove(0);
    }

    public void push(Object o) {
        add(0, o);
    }
}
```

继承了ArrayList接口

只能操作第一个对象

## 2.3.3 获取 ValueStack 中的数据

### 2.3.3.1 值栈中都有什么

首先我们要明确，值栈中存的都是对象。因为它本质就是一个 List，List 中只能存对象。

值栈中包含了我们通过调用 push 方法压栈的对象，当前执行的动作了和一个名称为 DefaultTextProvider 的类。值栈中的内容如下图：





Object	Property Name	Property Value
我们没有操作值栈时，默认的栈顶对象是当前执行的动作类对象  com.itheima.web.action.Demo2Action	texts	null
	actionErrors	[]
	errors	{}
	fieldErrors	{}
	errorMessages	[]
	container	There is no read method for container
	locale	zh_CN
com.opensymphony.xwork2.DefaultTextProvider	actionMessages	[]
	texts	null

在我们不操作值栈时，默认的栈顶对象是当前执行的动作类。

### 2.3.3.2 在动作类中往值栈中存入数据

```
/**
 * 对 ContextMap 中 ValueStack 的压栈操作
 */
public class Demo2Action extends ActionSupport {
    private String name = "呵呵";

    /**
     * 往 ValueStack 中压栈
     * @return
     */
    public String demo2() {
        //1. 获取 ActionContext 对象
        ActionContext context = ActionContext.getContext();
        //2. 获取 ValueStack 对象
        ValueStack vs = context.getValueStack();
        //3. 压栈操作
        Student s = new Student();
        s.setName("泰斯特");
        s.setAge(18);
        vs.push(s);
        return SUCCESS;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```



```

Student 类:
/**
 * 一个学生的模型
 */
public class Student implements Serializable {

    private String name;
    private Integer age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Integer getAge() {
        return age;
    }
    public void setAge(Integer age) {
        this.age = age;
    }
}

```

### 2.3.3.3 我们可以获取值栈中的什么

一般情况下，我们都是根据 **debug** 标签中显示的 **Property Name** 来获取 **Property Value**。  
当然我们也可以获取栈顶对象。

#### Struts ValueStack Debug

##### Value Stack Contents

值栈中存的都是对象

我们使用OGNL表达式获取值栈中的数据，都是获取右边的内容

使用OGNL表达式获取值栈中的数据，都是  
使用PropertyName来获取PropertyValue  
显示到浏览器的都是PropertyValue中的内容

Object	Property Name	Property Value
com.itheima.domain.Student	name	泰斯特
	age	24
	gender	male
com.itheima.web.action.Demo2Action	texts	null
	actionErrors	[]
	errors	{}
	fieldErrors	{}
	errorMessages	[]
	container	There is no read method for container
	locale	zh_CN
	actionMessages	[]
com.opensymphony.xwork2.DefaultTextProvider	texts	null



### 2.3.3.4 如何让 Action 中定义的成员出现在值栈中

在 Action 定义一个私有属性，并且提供公有 `get/set` 方法，那么该属性就会出现在值栈的 `Property Name` 中。显示的名称是根据 `get/set` 方法后面的内容决定的，与私有成员变量名称叫什么无关。

明确：当我们在动作类中提供了类的成员变量，并且提供它的公有 `get/set` 方法，我们从值栈中获取这个值的数据。在值栈中显示的 `Property Name` 和动作类中 `get/set` 方法后面的部分一致。

Object	Property Name	Property Value
com.itheima.domain.Student	name	泰斯特
	age	24
	gender	male
	texts	null
	actionErrors	[]
	errors	{}
	fieldErrors	{}
	errorMessages	[]
com.itheima.web.action.Demo2Action	container	There is no read method for container
	name	test
	locale	zh_CN
	actionMessages	[]
com.opensymphony.xwork2.DefaultTextProvider	texts	null

### 2.3.3.5 在页面上使用 OGNL 表达式获取数据

<%-- 1、获取值栈中的数据 需要借助 struts2 的标签 `s:property`

明确的事情：

- 1、获取值栈中的数据，我们只能根据值栈中对象的 `property name` 获取 `property value`。
- 2、获取值栈中的数据，都是直接写属性名称，获取的就是值，并不需要使用任何符号。

它是把 `value` 属性的取值作为值栈中对象的 `property name`，在值栈中从栈顶逐个查找，只要找到了就返回结果，并且不再继续查找。

--%>

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

```
<s:property value="age"/>
```

```
<hr/>
```

<%-- 2、默认的栈顶对象是：当前执行的动作类

获取栈顶对象的方式：

使用 `s:property` 标签，不要写 `value` 属性

--%>

```
<s:property/>
```

```
<hr/>
```

<%-- 3、获取指定位置的对象中的属性值 --%>

```
<s:property value="[0].name"/><br/>
```



```
<s:property value="[1].name"/>
```

### 2.3.3.6 OGNL 表达式执行时调用的方法

```
<%--4、ValueStack 中的 findValue 方法 --%>
<%
    ActionContext context = ActionContext.getContext();
    ValueStack vs = context.getValueStack();
    //其实之前的所有 OGNL 的操作，最终调用的方法都是 findValue。
    //Object value = valueStack.findValue(String expression);
    Object o1 = vs.findValue("name");
    out.print("findValue 方法输出的: "+o1+"<br/>");

    Object o2 = vs.findValue("[1].name");
    out.print("findValue 方法输出的: "+o2+"<br/>");

    Object o3 = vs.findValue("#application.applicationMap");
    out.print("findValue 方法输出的: "+o3+"<br/>");
%>
```

## 第3章 Struts2 中使用 EL 表达式

### 3.1 EL 表达式回顾

EL 表达式的写法：\${表达式}。

它是从四大域中，由小到大逐个域搜索，根据名称获取值。只要找到了，就不再继续搜索。

它的原理：使用的是 PageContext 类中的 findValue 方法。

### 3.2 Struts2 对 EL 表达式的改变

Struts2 框架中对 EL 表达式做了如下改变：

EL 表达式原来的搜索顺序：

```
page Scope——>request Scope——>sessionScope——>application Scope
```

EL 表达式改变后的搜索顺序：

```
page Scope —>request Scope —>valueStack —>contextMap —>sessionScope
—>application Scope
```

它是如何做到的呢？答案就是，struts2 框架对 request 对象进行了包装，并且对 getAttribute 方法进行了增强，代码如下：





```

62-
63-
64- /**
65-  * Gets the object, looking in the value stack if not found
66-  *
67-  * @param key The attribute key
68-  */
69- public Object getAttribute(String key) {
70-     if (key == null) {
71-         throw new NullPointerException("You must specify a key value");
72-     }
73-
74-     if (disableRequestAttributeValueStackLookup || key.startsWith("javax.servlet")) {
75-         // don't bother with the standard javax.servlet attributes, we can short-circuit this
76-         // see WW-953 and the forums post linked in that issue for more info
77-         return super.getAttribute(key);
78-     }
79-
80-     ActionContext ctx = ActionContext.getContext();
81-     Object attribute = super.getAttribute(key);
82-
83-     if (ctx != null && attribute == null) {
84-         boolean alreadyIn = isTrue((Boolean) ctx.get(REQUEST_WRAPPER_GET_ATTRIBUTE));
85-
86-         // note: we don't let # come through or else a request for
87-         // #attr.foo or #request.foo could cause an endless loop
88-         if (!alreadyIn && !key.contains("#")) {
89-             try {
90-                 // If not found, then try the ValueStack
91-                 ctx.put(REQUEST_WRAPPER_GET_ATTRIBUTE, Boolean.TRUE);
92-                 ValueStack stack = ctx.getValueStack();
93-                 if (stack != null) {
94-                     attribute = stack.findValue(key);
95-                 }
96-             } finally {
97-                 ctx.put(REQUEST_WRAPPER_GET_ATTRIBUTE, Boolean.FALSE);
98-             }
99-         }
100-     }
101-     return attribute;

```

## 第4章 OGNL 表达式中的各种符号总结

我们在今天的学习中接触了很多的符号，在加上之前我们在学习 el 表达式的 \$ 符号，大家可能会容易混乱，接下来我们就来总结一下常用的符号，在这里要着重强调一下，OGNL 表达式在取值时，只能作用在 struts2 的标签中，不能写在 HTML 标签里

例如：

正确的用法：<s:property value="username"/>

错误的用法：<input type="text" name="username" value="%{ username }"/>

EL 表达式不能出现在 struts2 的标签中，如果写在 struts2 的标签中会报错：

例如：

正确的用法：<input type="text" name="username" value="\${username}"/>

错误的用法：<s:property value="\${ username }"/>

### 4.1#:

1、获取大 Map 中数据，把后面的内容看成是 key。

2、在使用 struts2 标签，创建 Map 对象时使用。<s:radio list="#{}"/>



## 4.2 %:

- 1、使用 `{' '}` 把 OGNL 表达式强制看成是普通字符串
- 2、使用 `{}` 把普通字符串强制看成是 OGNL 表达式

## 4.3 \$:

- 1、使用 `EL` 表达式的标识
- 2、在 `struts` 的配置文件中使用 OGNL 表达式

# 第5章 案例-优化客户列表的展示

## 5.1 环境搭建

我们直接把上次课的环境拿过来用就行了。

## 5.2 改造 Action

我们把之前查询所有客户的动作方法改造一下，之前我们是把查询结果存入请求域中了，而此时我们只需要在 `Action` 中定义一个集合，并且提供 `get/set` 方法，它就会出现在值栈中。就可以在页面中使用 `OGNL` 表达式获取。

```
/**
 * 查询所有客户
 * @return
 */
private List<Customer> customers;

public String findAllCustomer() {
    customers = customerService.findAllCustomer();
    return "findAllCustomer";
}

public List<Customer> getCustomers() {
    return customers;
}

public void setCustomers(List<Customer> customers) {
    this.customers = customers;
}
```

### 5.3 改造 jsp

在显示客户列表时，我们之前采用的是 `jstl` 标签库的 `c:forEach` 标签，今天我们将使用 `struts2` 提供的迭代标签 `s:iterator`。

```
<%--s:iterator 标签：
        作用：用于遍历集合，在 jsp 中显示
        属性：
            value: 取值是一个 OGNL 表达式
            var: 写了该属性: var 的值是一个字符串。他会把 var 的值作为 key，把当前遍历
的对象作为 value，存入 contextMap 中
                没写该属性：把当前遍历的对象压栈，每次遍历结束后弹栈。
            begin: 遍历的开始索引
            end: 遍历的结束索引
            step: 遍历的步长
            status: 计数器对象
            isOdd: 是否是奇数行
            isEven: 是否是偶数行
            isFirst: 是否是第一行
            isLast: 是否是最后一行
            getCount: 获取当前遍历的个数 从 1 开始
            getIndex: 获取当前遍历的索引 从 0 开始
--%>
<s:iterator value="customers" var="cust">
    <TR 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="#"${pageContext.request.contextPath }/customer/CustomerServlet?method=editCustomerUI&custId=<s:property value='#cust.custId'/>">修改</a>
                &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
            <a
href="#"${pageContext.request.contextPath }/customer/CustomerServlet?method=removeCustomer&custId=<s:property value='#cust.custId'/>">删除</a>
        </TD>
    </TR>
</s:iterator>
<%--<s:iterator value="customers">
    <TR style="FONT-WEIGHT: normal; FONT-STYLE: normal; BACKGROUND-COLOR:
```



北京市昌平区北七家宏福创业园修正大厦 3 层 电话：400-618-9090