

2018

# Java EE框架 --- Struts2

Java EE framework --Struts2

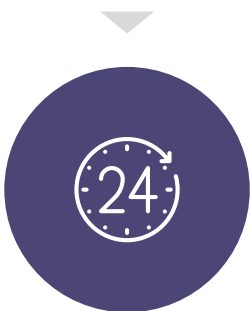
王磊

计算机工程学院

# CONTENTS



创建JavaWeb项目



搭建Stust2 开发环境



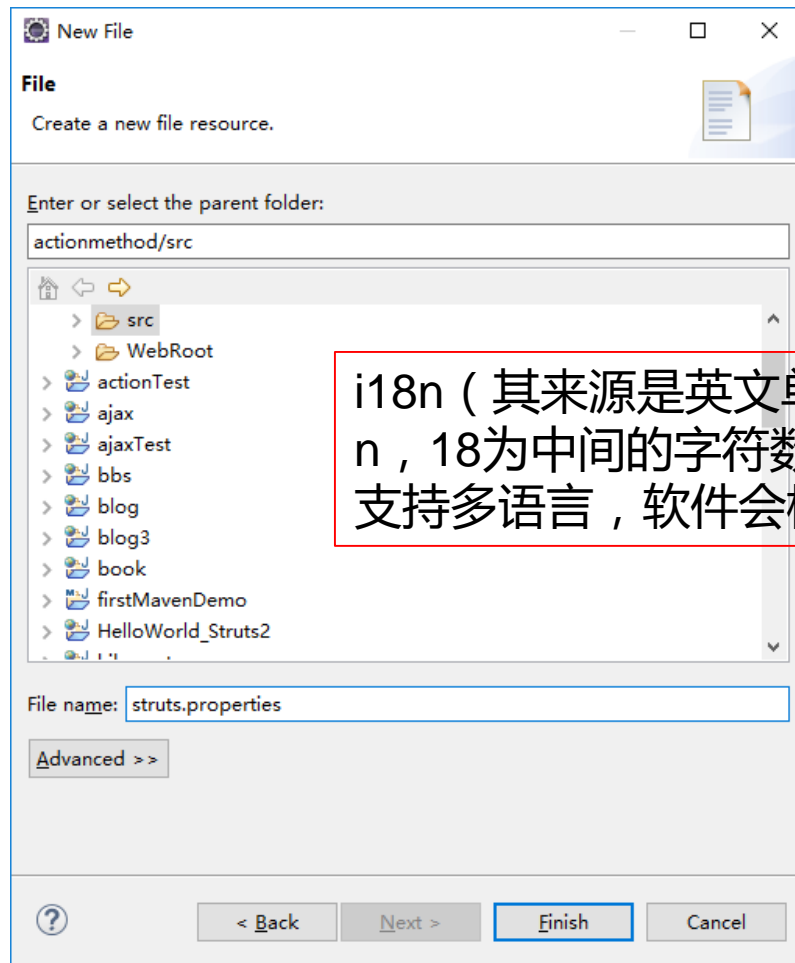
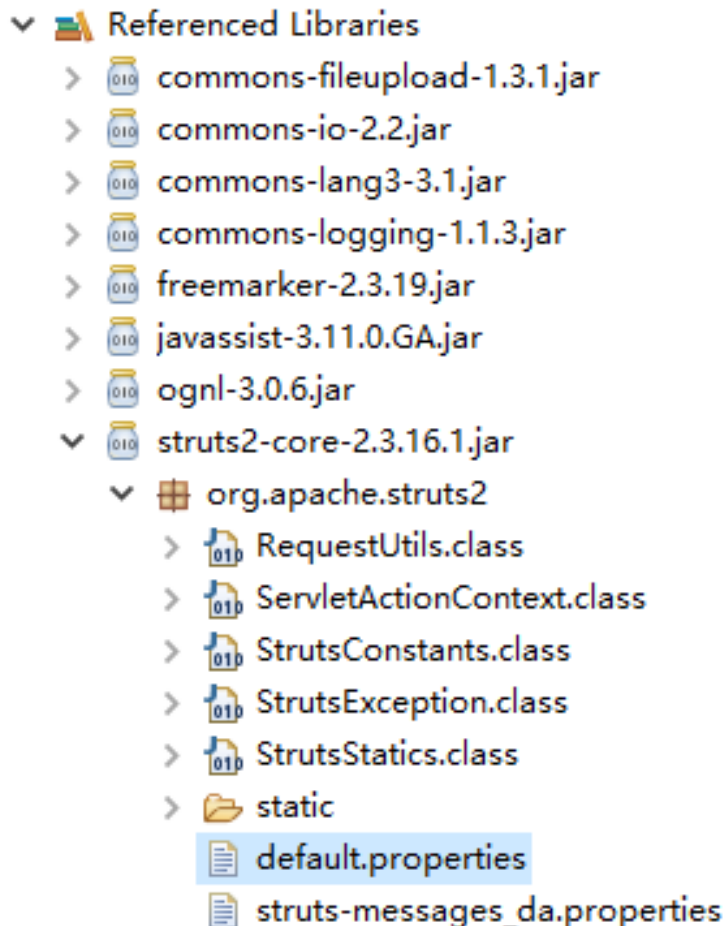
配置struts.xml



部署运行

# • 一、Struts2常量配置 •

1、修改struts2常量方式1：在src下创建struts.properties，写上键值对



i18n (其来源是英文单词 internationalization 的首末字符 i 和 n, 18 为中间的字符数) 是“国际化”的简称  
支持多语言，软件会根据当前的语言使用环境，自动切换

```
*struts.properties
1 struts.i18n.encoding=UTF-8
2
```

## • 一、Struts2常量配置 •

2、修改struts2常量方式2：在struts根元素下面，使用constant定义常量(推荐使用)

struts.xml ☒

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuratio
3 <struts>
4 <constant name="struts.enable.DynamicMethodInvocation" value="true" />
5 <package name="struts2" namespace="/" extends="struts-default">
6
7     <action name="userlogin" class="com.bbc.action.UserAction" method="login">
8
```

## • 一、Struts2常量配置 •

3、修改struts2常量方式3：修改web.xml配置文件，使用

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
  <display-name>actiontest1</display-name>
  <context-param>
    <param-name>struts.i18n.encoding</param-name>
    <param-value>UTF-8</param-value>
  </context-param>
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.
  </filter>
```

## • 一、Struts2常量配置 •

4、在struts.xml中配置常量如下：常用的有4个

```
<struts>
<!-- java国际化，解决post提交中文乱码问题 -->
<constant name="struts.i18n.encoding" value="UTF-8"></constant>

<!-- 指定action的访问后缀名 -->
<constant name="struts.action.extension" value="action,,"></constant>

<!-- 指定struts是否以开发模式运行 -->
<constant name="struts.devMode" value="true"></constant>

<!-- 开启struts动态调用模式 -->
<constant name="struts.enable.DynamicMethodInvocation" value="true" />
```

热加载struts.xml配置文件，无需重启服务器  
提供更多的错误信息输出，方便开发时调试

后缀名或者为action或者为空  
要是配置为do，则必须是：  
<http://localhost:8080/xx/xx.do>形式

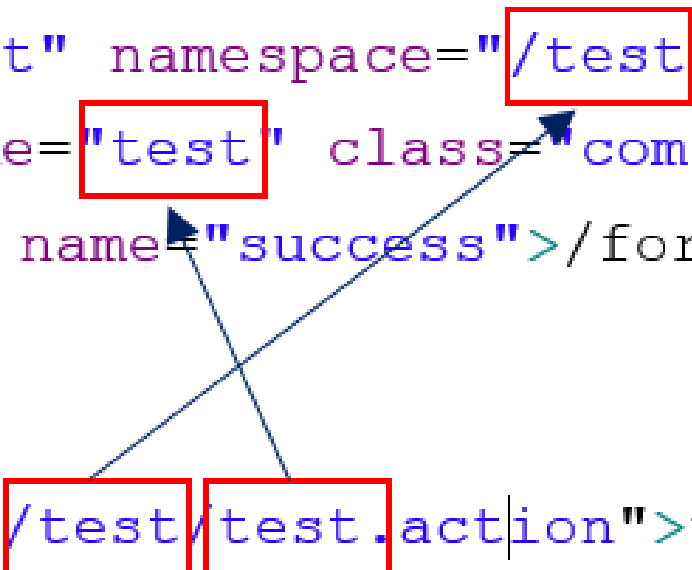
## • 二、Struts2命名空间、绝对路径 •

- 1、namespace，主要是避免多人共同开发项目出现名字冲突
- 2、namespace决定了action的访问路径，默认为“”，可以接收所有路径的action。Namespace可以写为/，或者/xxx，或者/xxx/yyy，对应的action访问路径为/index.action，/xxx/index.action，或者/xxx/yyy/index.action，namespace最好用模块进行命名。
- 3、有了namespace可以在项目开发时由项目经理给每一个人分不同的namespace，这样每个开发人员只需要保证自己所写的action不同名即可。

## 二、Struts2命名空间、绝对路径

当我们要想访问一个action所关联到的jsp文件时，应该用namespace+action的name

```
<package name="test" namespace="/test" extends="struts-default">
    <action name="test" class="com.xk.TestAction">
        <result name="success">/forward/test.jsp</result>
    </action>
</package>
<a href="<%=path%>/test/test.action">test3</a>
```



链接地址和action的对应关系



## 二、Struts2命名空间、绝对路径

为action配置了namespace时，访问此action的形式如下：

.../webappname/xxx/yyy/ActionName.action

而当此action成功执行跳转到某个jsp页面时，如想在此jsp页面写链接，一定要写绝对路径，因为相对路径是相对.../webappname/xxx/yyy/，而如果以后我们修改了action的namespace时，相对路径又要变，所以链接不能写成相对路径。

通常用myeclipse开发时建立一个jsp文件，默认总会有如下内容：

```
<%
```

```
    String path = request.getContextPath();
```

```
    String basePath = request.getScheme()+"://"+request.getServerName()+":“
```

```
    +request.getServerPort()+path+”/”;
```

```
%>
```

写绝对路径可以参此内容。也可以参考<head>下的<base href="<%=basePath%>"> 来完成绝对路径的书写。

## • 三、struts开发示例 •

- 1、创建动态项目
- 2、创建com.bbc.model，编写javabean

```
package com.bbc.model;
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;
    }
}
```

## • 三、struts开发示例 •

### 3、创建com.bbc.action包，创建UserAction类

```
package com.bbc.action;
import java.util.ArrayList;
import java.util.List;
import com.bbc.model.User;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport {
    private User user;
    private List<String> list=new ArrayList<String>();
    public List<String> getList() {
        return list;
    }
    public void setList(List<String> list) {
        this.list = list;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
}
```

## • 三、struts开发示例 •

### 3、创建com.bbc.action包，创建UserAction类

```
@Override
public String execute() throws Exception {
    return SUCCESS;
}
public String login(){
    if(user.getUsername().equals("admin") && user.getPassword().equals("123")){
        list.add("user1");
        list.add("user2");
        ActionContext.getContext().put("list",list);
        return SUCCESS;
    }else{
        return INPUT;
    }
}
```

## • 三、struts开发示例 •

### 3、创建com.bbc.action包，创建UserAction类

```
public String add(){
    return SUCCESS;
}
public String delete(){
    return SUCCESS;
}
public String update(){
    return "update";
}
public String query(){
    return "query";
}
public void validateLogin() {
    if(user.getUsername()==null||user.getUsername().equals("")){
        this.addFieldError("username", "用户名不能为空");
    }
    if(user.getPassword()==null||user.getPassword().equals("")){
        this.addFieldError("password", "密码不能为空");
    }
}
}
```

## • 三、struts开发示例 •

### 4、创建struts.xml配置文件

```
<!-- 指定struts是否以开发模式运行 -->
<constant name="struts.devMode" value="true"></constant>
<package name="struts2" namespace="/" extends="struts-default">
    <action name="userlogin" class="com.bbc.action.UserAction" method="login">
        <result name="success">/main.jsp</result>
        <result name="input">/login.jsp</result>
    </action>
    <action name="useradd" class="com.bbc.action.UserAction" method="add">
        <result name="success">/useradd.jsp</result>
    </action>
    <action name="userdelete" class="com.bbc.action.UserAction" method="delete">
        <result name="success">/userdelete.jsp</result>
    </action>
    <action name="user" class="com.bbc.action.UserAction">
        <result name="update">/userupdate.jsp</result>
        <result name="query">/userquery.jsp</result>
    </action>
</package>
</struts>
```

## • 三、struts开发示例 •

### 5、创建login.jsp文件

- ▼ WebRoot
  - > META-INF
  - ▼ WEB-INF
    - > lib
      - web.xml
    - login.jsp
    - main.jsp
    - useradd.jsp
    - userdelete.jsp
    - userquery.jsp
    - userupdate.jsp

```
<form action="userlogin" method="post">  
用户名: <input type="text" name="user.username"/><br>  
密 码: <input type="password" name="user.password"/><br>  
      <input type="submit" value="提交"/>  
</form>  
<s:fielderror></s:fielderror>
```

## • 三、struts开发示例 •

### 6、创建main.jsp文件

```
<body>
<s:debug></s:debug>
<center>    <h1>用户登录成功</h1>
```

当前登录用户名: <s:property value='User.username' /> </br></br></br>

```
    <a href="useradd">添加用户</a>
    <a href="userdelete">删除用户</a>
    <a href="user!update">修改用户</a>
    <a href="user!query">查询用户</a>
</center></br>
```

显示在action中存入 Value Stack Contents 中的 list , 直接使用s标签和 OGNL 表达式即可 </br>

```
    <s:property value="list[0]" />
    <s:property value="list[1]" />
```

```
<hr>
```

显示在action中存入Stack Context 中的 list , 必须在OGNL表达式前加 # </br>

```
    <s:property value="#list[0]" />
    <s:property value="#list[1]" />
```

```
</body>
```



## • 三、struts开发示例 •

### 7、测试结果

http://localhost:8080/actionmethod/login.jsp

用户名:

密 码:

[\[Debug\]](#)

http://localhost:8080/actionmethod/useradd

添加用户成功

http://localhost:8080/actionmethod/userdelete

用户删除成功

http://localhost:8080/actionmethod/user!update

用户修改页面

http://localhost:8080/actionmethod/user!query

用户查询页面

## 用户登录成功

当前登录用户名: admin

[添加用户](#) [删除用户](#) [修改用户](#) [查询用户](#)

显示在action中存入 Value Stack Contents 中的 list ,直接使用s标签和 OGNL 表达式即可  
user1 user2

显示在action中存入Stack Context 中的 list , 必须在OGNL表达式前 加 #  
user1 user2

## • 四、Action的拦截器 •

拦截器能在action被调用之前和被调用之后执行一些“代码”。Struts2框架的大部分核心功能都是通过拦截器来实现的，如防止重复提交、类型转换、对象封装、校验、文件上传、页面预装载等，都是在拦截器的帮助下实现的。每一个拦截器都是独立装载的(pluggable)，我们可以根据实际的需要为每一个action配置它所需要的拦截器。

## • 四、Action的拦截器 •

如果用户没有登录不允许访问action中的方法，提示“没有执行的权限”

### 1、定义拦截器（继承Interceptor接口）

```
package com.xk.interceptor;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.Interceptor;
public class PermissionInterceptor implements Interceptor {
    public String intercept(ActionInvocation invocation) throws Exception {
        Object users = ActionContext.getContext().getSession().get("users");
        if (users != null) // 如果用户不为null，代表用户已经登录，执行action方法
        {
            invocation.invoke(); // 如果已登录，通过过滤器执行result视图
        } else {
            ActionContext.getContext().put("message", "没有权限");
        }
        return "message"; // 没有就返回到message视图
    }
    public void destroy() {}
    public void init() {}
}
```

## • 四、Action的拦截器 •

如果用户没有登录不允许访问action中的方法，提示“没有执行的权限”

2、注册过滤器（在<package>下输入以下代码）

```
<interceptors>
  <interceptor name="permission"
class="com.xk.interceptor.PermissionInterceptor"/>
  <interceptor-stack name="permissionStack">
    <interceptor-ref name="defaultStack"/>
    <interceptor-ref name="permission"/>
  </interceptor-stack>
</interceptors>
```

interceptor-stack是一个interceptor栈，可以放入多个interceptor  
interceptor-ref 是注册的过滤器，必须把系统默认的过滤器  
*defaultStack*放在第一个位置，然后再放自己的过滤器

## • 四、Action的拦截器 •

如果用户没有登录不允许访问action中的方法，提示“没有执行的权限”

2、注册过滤器（在<package>下输入以下代码）

```
<interceptors>
  <interceptor name="permission"
class="com.xk.interceptor.PermissionInterceptor"/>
  <interceptor-stack name="permissionStack">
    <interceptor-ref name="defaultStack"/>
    <interceptor-ref name="permission"/>
  </interceptor-stack>
</interceptors>
```

interceptor-stack是一个interceptor栈，可以放入多个interceptor  
interceptor-ref 是注册的过滤器，必须把系统默认的过滤器  
*defaultStack*放在第一个位置，然后再放自己的过滤器

## • 四、Action的拦截器 •

### 3、局部过滤器的使用

```
<action name="user_*" class="com.xk.action.UserFilter" method="{1}">  
    <result name="success">hello.jsp </result>  
    <interceptor-ref name="permissionStack"/>  
</action>
```

## • 五、Action执行的方法 •

Action在执行的时候，不一定要执行execute()方法，可以在配置文件中配置action的时候，用method=来指定执行哪个方法，也可以在url地址中动态指定(动态方法调用DMI)，推荐使用DMI方法。

```
<struts>
  <constant name="struts.enable.DynamicMethodInvocation" value="true"/>
  <constant name="struts.devMode" value="true" />
  <package name="user" extends="struts-default" namespace="/user">
    <action name="userAdd" class="com.xk.action.UserAction" method="add">
      <result>/userAddSuccess.jsp</result>
    </action>
    <action name="userDelete" class="com.xk.action.UserAction" method="delete">
      <result>/userDeleteSuccess.jsp</result>
    </action>
    <action name="user" class="com.xk.action.UserAction">
      <result>/success.jsp</result>
    </action>
  </package>
```

## • 五、Action执行的方法 •

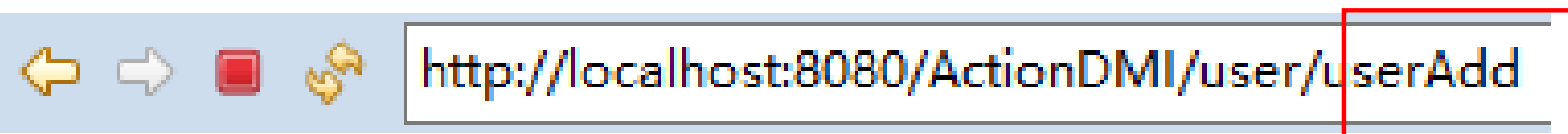
Action在执行的时候，不一定要执行execute()方法，可以在配置文件中配置action的时候，用method=来指定执行哪个方法，也可以在url地址中动态指定(动态方法调用DMI)，推荐使用DMI方法。

```
package com.xk.action;
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String add() {
        return SUCCESS;
    }
    public String delete() {
        return SUCCESS;
    }
}
```

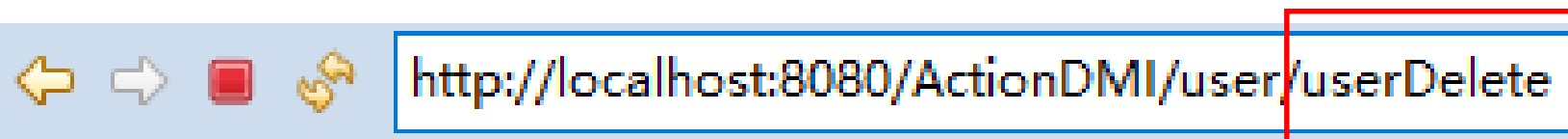


## • 五、Action执行的方法 •

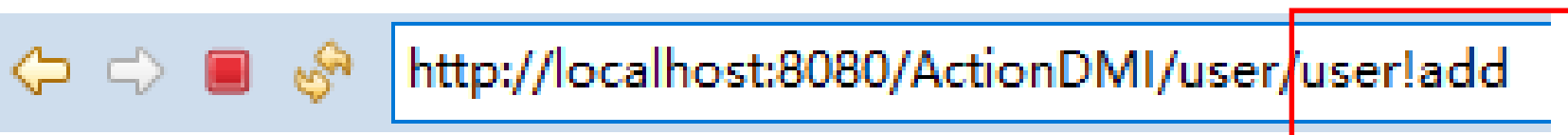
Action在执行的时候，不一定要执行execute()方法，可以在配置文件中配置action的时候，用method=来指定执行哪个方法，也可以在url地址中动态指定(动态方法调用DMI)，推荐使用DMI方法。



添加用户成功!



删除用户成功!



执行成功!

## • 六、使用通配符定义action •

```
<action name="actiondemo_*"
class="com.xk.action.ActionDemo2" method="{1}">
    <result name="success">hello.jsp</result>
</action>
```

//actiondemo\_\*表示匹配actiondemo2后的所有内容，{1}标示第一个通配符\*，因为可能出现多个\*；{1}可以作为类名的一部分

输入：actiondemo\_add.action就调用add方法

actiondemo\_execute.action就调用execute方法

## • 七、struts接受请求参数 •

基本类型接收请求参数（get/post），在Action类中定义与请求参数同名的属性，struts2便能自动的接受请求参数并赋给同名属性

请求路径：

<http://localhost:8080/actiondemo!add.action?id=12&name=xike>

```
package com.xk.model;
public class ActionDemo { //id=xXx&name=xxx
    private String msg;
    private int id;
    private String name;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    ...
}
```

在jsp中：

id=\${id} name=\${name}  
可以获得属性信息了

## • 七、struts接受请求参数 •

复合类型接收请求函数

请求路径:

<http://localhost:8080/MyStruts2/index.jsp>

web.xml

```
<action name="actiondemo2_*" class="com.xk.action.ActionDemo2" method="{1}">
    <result name="success">/hello.jsp</result>
</action>
```

Person类

```
package cn.com.hewei.action;
```

```
public class Person {
```

```
    private Integer id;
```

```
    private String name;
```

```
    public Integer getId() { return id; }
```

```
    public void setId(Integer id) { this.id = id; }
```

```
    public String getName() { return name; }
```

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

```
}
```

ActionDemo2

```
package cn.com.hewei.action;
```

```
public class ActionDemo2 {
```

```
    private Person person;
```

```
    public Person getPerson() {
        return person;
    }
```

```
    public void setPerson(Person person) {
        this.person = person;
    }
```

```
    public String addUI() {
        return "success";
    }
```

```
    public String execute() {
        return "success";
    }
```

```
}
```

## • 七、struts接受请求参数 •

```
index.jsp
<form action="<%=request.getContextPath() %>/actiondemo2!add.action"
method="post">
    id:<input type="text" name="person.id" >
    name:<input type="text" name="person.name" >
    <input type="submit" value="发送">
</form>
```

得到参数:

```
id=${person.id}
name=${person.name}
```

## • 八、OGNL表达式 •

OGNL是Object Graphic Navigation Language(对象图导航语言)的缩写，是一个开源项目。Struts框架使用OGNL作为默认的表达式语言。

Ognl必须配置Struts标签，不能离开Struts标签直接使用。

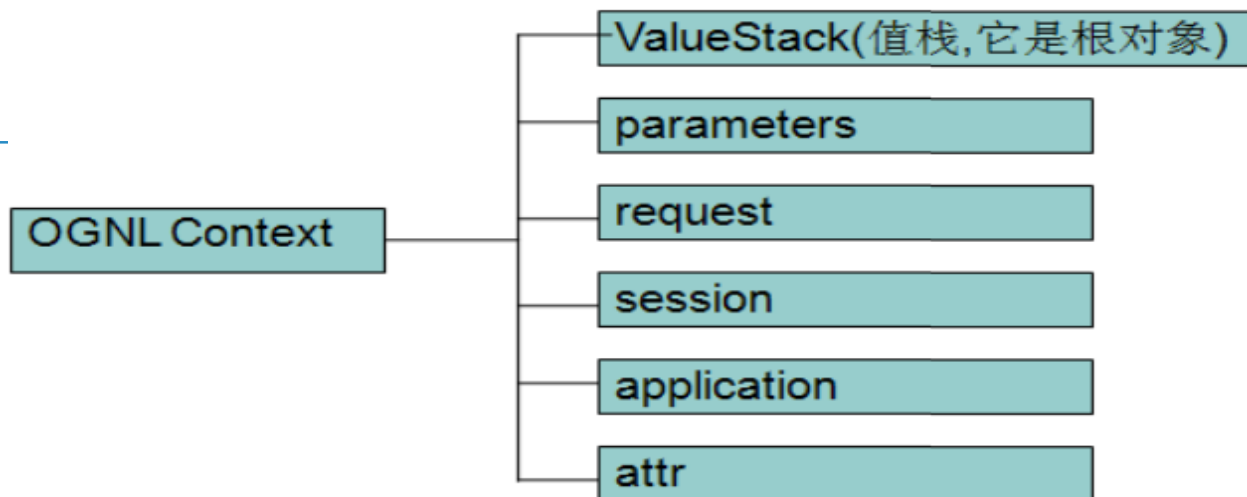
OGNL中的上下文即struts2中的actionContext

OGNL中的root即struts2中的valueStack

访问上下文（Context）中的对象需要使用#符号标注命名空间，如

#application、#session

访问根对象（即ValueStack）中对象的属性，则可以省略#命名空间，直接访问该对象的属性即可。



## 八、OGNL表达式

@Test

```
public void testOgnl()
```

```
{
```

```
    User user = new User("wl", "12345");
```

```
    Address address = new Address("233000", "安徽蚌埠");
```

```
    user.setAddress(address);
```

```
    try
```

```
    {
```

```
        System.out.println(Ognl.getValue("name", user));
```

```
        System.out.println(Ognl.getValue("address", user));
```

```
        System.out.println(Ognl.getValue("address.port", user));
```

```
    }
```

```
    catch (OgnlException e)
```

```
    {
```

```
        e.printStackTrace();
```

```
    }
```

```
}
```

//输出结果:

//wl

//com.wl.ognl.Address@dda25b

//233000

## 八、OGNL表达式

```
public class OperatorAction extends ActionSupport {
    private User user;
    ArrayList<User> al = new ArrayList<User>();
    private String id;
    public String Delete() {
        UserService us = new UserService();
        String id = ServletActionContext.getRequest().getParameter("id");
        User u = us.selectUserById(id);
        us.deleteUser(u);
        ActionContext ac = ActionContext.getContext();
        al = us.getAllUser();
        ac.put("al", al);
        return SUCCESS;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
}
```



## 八、OGNL表达式

```
<s:iterator value="al">
<tr>
  <td width="40" ><s:property value="#request.id"/></td>
  <td width="100"><s:property value="#request.username"/></td>
  <td width="100"><s:property value="password"/></td>
  <td width="70"><s:property value="grade"/></td>
  <td width="200"><s:property value="email"/></td>
  <td><a href="userModify?id=<s:property value="#request.id"/>"修改</a></td>
  <td><a href="userDelete?id=<s:property value="#request.id"/>"删除</a></td>
</tr>
</s:iterator>
```

## 九、struts2的常用标签

### 1. property标签

property标签用于输出指定值：

```
<s:set name="name" value="kk" />
<s:property value="#name"/>
```

### 2. iterator标签

iterator标签用于对集合进行迭代，这里的集合包含List、Set和数组。

```
<s:set name="list" value="{ 'zhangming','xiaoli','liming' }" />
<s:iterator value="#list" status="st">
    <font color=<s:if test="#st.odd">red</s:if><s:else>blue</s:else>>
        <s:property /></font><br>
</s:iterator>
```

### 3.if/elseif/else标签

```
<s:set name="age" value="21" />
<s:if test="#age==23">
    23
</s:if>
<s:elseif test="#age==21">
    21
</s:elseif>
<s:else>
    都不等
</s:else>
```

## • 十、action中操作request等对象 •

### 1、session域

```
Map<String,Object> session = ActionContext.getContext().getSession();  
session.put("name","sessionTom"); //放入session中
```

### 2、application域

```
Map<String,Object> application = ActionContext.getContext().getApplication();  
application.put("name","applicationTom"); //放入application中
```

### 3、request域

```
Map<String,Object> request = (Map<String,Object>) ActionContext.getContext().get("request");
```

注意：request域的声明周期和actionContext的生命周期是一样的。那么，struts2不推荐使用原生的request域。如有需要，可以把数据直接放到actionContext中（actionContext本身就是map）

例如：ActionContext.getContext().put("name","tom"); //推荐方式

## • 十一、action中获得原生Servlet API •

在jsp页面中，分别从request、session、application中取值

request: EL表达式取值：\${request.name}

S标签取值：<s:property value="#name"/> 或者 <s:property value="#request.name"/>

session: EL表达式取值：\${session.name}

S标签取值：<s:property value="#session.name"/>

application: EL表达式取值：\${application.name}

S 标签取值：<s:property value="#application.name"/>

## •使用Struts2•



可以在此基础上，进一步拓展业务逻辑层，链接后台数据库，写出更为复杂的框架程序