一、文件下载

通过 type为stream的result结果类型处理文件下载。

1.1 Action 类

```
/**
 *
 * @author jackiechan
 *
 *文件下载的编码步骤
 *  1定义一个InputStream 变量 不能叫 in,提供 get 方法
 *  2 在action 方法中 只要给InputStream 赋值即可
 *
 *  3在 action 的 result 配置  type 为 stream
 *      下载文件需要知道下载哪个文件 问价名是什么名字,文件类型是什么类型,需要注入参数
 *  找stream这个处理下载文件的类 接收什么参数
 *      inputName 指定输入流的名字, 会反射调用对应 get 方法
 *      contentType告诉浏览器文件的类型,可以设置为二进制流
 *      contentDisposition  告诉浏览器文件名字attachment;filename=文件名
 *
 *  中文乱码问题
 *      在设置文件名的时候提前进行 URLENCODE 编码
 *  文明自己写死的问题
 *  在 action 里面定义一个变量名 提供 get set 方法 用于接收文件名字(可能是用户传递过来
的,也可以在代码中动态赋值)
 *  在 result 的contentDisposition参数中 filename=${变量名}
 */
public class ActionDemo3 extends ActionSupport {
    private InputStream inputStream;//作为文件的输入流,用于将要被下载的文件读取出来
    private String name;//文件名,告诉下载者文件的名字

    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public InputStream getInputStream() {
        return inputStream;
    }



    public String downLoadFile() throws FileNotFoundException,
UnsupportedEncodingException{
```

```
        //给文件名赋值(可能框架在封装参数的时候已经帮我们封装好了)
        name="中文.png";
        name = URLEncoder.encode(name, "UTF-8");//对 name 进行 urlEncode 编码
        //给inputStream 赋值,告诉框架你要下载哪个文件,文件实际上是通过用户的请求传递参数过来,自己找到 File 在哪
        String realPath = ServletActionContext.getServletContext().getRealPath("WEB-INF/classes/a.png");
        inputStream=new FileInputStream(new File(realPath));
        return SUCCESS;
    }

}
```

1.2 struts.xml配置文件

```
<action name="downLoadFile" class="com.qianfeng.struts.action.ActionDemo3" method="downLoadFile">
            <result type="stream">
            <!--
            既然是下载文件, 文件叫什么名? 文件是什么类型
                下载哪个文件 告诉它是哪个输入流
                inputName 告诉框架 下载哪个文件
                contentType 告诉浏览器 文件的类型
                contentDisposition告诉框架(浏览器)文件名
                ${name} OGNL表达式 不是 EL 表达式, EL 表达式只能在 jsp 用
                调用了当前 action 类的 getName 方法
              -->
                <param name="inputName">inputStream</param>
                <param name="contentType">application/octet-stream</param>

                <!-- <param name="contentDisposition">attachment;filename="a.png"</param> -->

              <!--此处使用 ognl 表达式动态获取编码下载文件的文件名,这样就不用像上面的方式写死了文件名-->
                <param name="contentDisposition">attachment;filename=@java.net.URLEncoder@encode(${name}, "UTF-8")</param>
            </result>
            <result name="input">/index.jsp</result>
        </action>
```

二、OGNL表达式简介

> 在struts2中要使用OGNL表达式，必须放到Struts2的标签中。

2.1 OGNL

> OGNL是ObjectGraphic Navigation Language（对象图导航语言）的缩写，它是一个开源项目。webwork用它作为表达式语言。

2.2其他重要的功能

```
<body>
 测试内容
<!--
相当于
c:out> 标签 用于输出内容为字符串
 value 用的是 ognl 表达式,从所有的作用域中找 key 或者是属性为 zhangsan 的内容
 如果想要输出纯字符串,需要用单引号括起来

 调用对象的方法
 -->
<s:property value="'zhangsan'.length()" /><hr>
<!--
调用静态方法和属性
-->
   <s:property value="@java.lang.Integer@MAX_VALUE"/><hr>
<s:property value="@java.lang.Math@random()"/><hr>
 </body>
```

2.3访问OGNL上下文（OGNL context）和ActionContext； 下面讲

2.4操作集合对象。

三、context上下文：数据中心

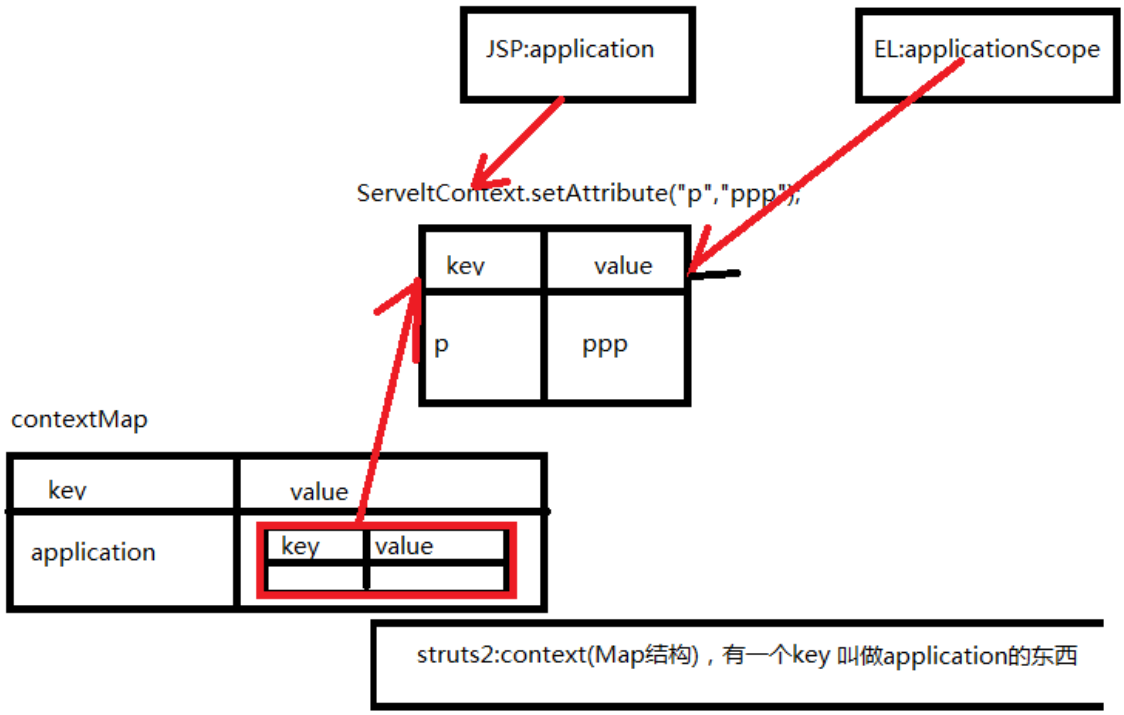
> Struts 中使用一个 context 来存放一些数据,其实是一个 map, 其中有一下 key, 对应的值
>
> application 对应一个 map, 存着 servletContext 中的 attribute,并不是servletContext对象
>
> session 对应一个 map 存着 session 中的 attribute ,不是 session 对象
>
> value stack 值栈

> action 当前访问的 action对象
>
> request 对应一个 map, 存放 request 中的 attribute,不是 request 对象
>
> parameters 对应一个 map, 存放所有的请求参数

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

contextmap中的数据

key	value	备注
application	ServeltContext中的所有属性attributes	相当于EL的内置对象applicationScope
session	HttpSession中的所有属性attributes	相当于EL的内置对象sessionScope
value stack(根)	他是一个List	
action	动作类	
request	ServletRequest中的所有属性	相当于EL的内置对象requestScope
parameters	Map map= request.getParameterMap()	相当于EL的内置对象paramValues
attr	从四大域范围中搜索	${}



> 动作类的生命周期：每次访问都会重新创建新的动作类的实例。还会创建ActionContext和ValueStack的实例。ActionContext和ValueStack一直保持在你的线程中。（ThreadLocal方式实现）。

> 开启 debug 模式,可以在jsp页面上面 通过< s:dbug> 来查看

Value Stack Contents

Object	Property Name	Property Value
com.itheima.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

是contextMap中的根

3.1、ActionContext的API

> 操作contextMap中的数据的

3.1.1 action 类,用于存放数据和打印数据

```
/**
 * Action 的声明周期, 每次请求都要创建一个对象 没有线程安全问题 尽情的使用成员变量 随着
Action 的创建,还有一个对象也一块创建
 * ActionContext 创建 value stack 值栈 存值,取值,存值,取值,存值,取值
 *
 * @author jackiechan
 *
 */
public class ActionDemo1 extends ActionSupport {

    public ActionDemo1() {
        System.err.println("吓清醒了.....");
    }

    @Override
    public String execute() throws Exception {
        ActionContext context = ActionContext.getContext();
        context.put("testkey1", "testvalue1");// 以testkey1为 key 以testvalue1值 放
到了
                                              // context map 中
        Object object = context.get("testkey1");// 从 context map 中获取 key
                                              // 为testkey1的值
        // System.err.println(object);

        Map<String, String> stringmapMap = new HashMap();
        stringmapMap.put("key1", "value1");
        context.put("testkey2", stringmapMap);

        Customer customer = new Customer();
        customer.setName("张三");
        context.put("cus", customer);

        ServletActionContext.getRequest().setAttribute("reqKey1", "reqvalue1");
        Map<String, Object> requestMap = (Map<String, Object>)
context.get("request");
        System.err.println(requestMap.get("reqKey1"));
        requestMap.put("aaaa1", "bbbbb2");// 等价于下面那行代码
        // ServletActionContext.getRequest().setAttribute("aaaa1", "bbbbb2");

        ServletActionContext.getRequest().getSession().setAttribute("sessionkey1",
"sessionvalue1");
        Map<String, Object> sessionMap = (Map<String, Object>)
context.get("session");
        Object sessionvalue = sessionMap.get("sessionkey1");
        System.err.println("sessionvalue------" + sessionvalue);
```

```
        Map<String, Object> session = context.getSession();// 内部执行的是
context.get("session")
                                                            // 然后强制转换;
        Object sessionvalue2 = session.get("sessionkey1");
        System.err.println("sessionvalue2==" + sessionvalue2);

        // Map<String, Object> parametersMap = (Map<String, Object>)
        // context.get("parameters");
        // String[] name = (String[]) parametersMap.get("name");//因为不知道一个
key
        // 有多少个参数传递过来,所以是一个数组封装的
        // Map<String, Object> parameters = context.getParameters();//等价于上面代
码
        // System.err.println(name[0]);//不要直接写0 ,这里是为了演示,正常要判断的
        //

        ServletActionContext.getServletContext().setAttribute("scontextKey1",
"scontextValue1");
        Map<String, Object> applicationMap = (Map<String, Object>)
context.get("application");// 内部调用context.getContextMap()的返回值

// map

// 的

// get

// 方法
        Object contextValue = applicationMap.get("scontextKey1");
        System.err.println("contextValue===" + contextValue);

        Map<String, Object> contextMap = context.getContextMap();// contextmap
        Map<String, Object> sessionMap2 = (Map<String, Object>)
contextMap.get("session");
        Object object2 = sessionMap2.get("sessionkey1");
        System.err.println("sessionMap2 sessionkey1===" + object2);

        HttpServletRequest request = ServletActionContext.getRequest();
        HttpServletResponse response = ServletActionContext.getResponse();
        HttpSession httpSession = ServletActionContext.getRequest().getSession();

        System.err.println(request);
        System.err.println(response);
        System.err.println(httpSession);

        return super.execute();

    }
}
```

3.1.2 jsp 用于显示在 action 中存放的数据

> 要使用 struts 的标签,需要引入标签库

```
<%@ 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>

    <title>My JSP 'login.jsp' starting page</title>

    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->

  </head>

  <body>
  11111111111111111111
  <!--
  #一定是从 context map 中找对应的 key

  <!--
  获取 数据的思路.
    你要获取数据在哪放着
        在一个小的 map 里面放着
                找到这个小的 map 然后.key 取出来
                    怎么找到小的 map
                        在  request 里面放着
                            找到 request, 然后调用.key 找到这个 map
                                怎么找到 request, 在 Context 大 map 中


   -->

  -->
  <s:property value="#testkey1" /><hr>
  <!--
  获取对象的属性或者是 map的value 的时候,直接 .属性或者是.key    .key 取的不是大 map 里面的对象
  如果#testkey2取到的是一个普通对象,那么.key1 调用的就是 getKey1这个方法
  如果#testkey2取到的是个 map 那么.key1 调用的是 map.get(key1);
```

```
  -->
  <s:property value="#testkey2.key1"/><hr>
  <!--
  获取对象和对象的属性
   -->
   <s:property value="#cus"/><hr>
   <s:property value="#cus.name"/><hr>

  <s:property value="#request.reqKey1"/><hr>

  <s:property value="#request.aaaa1"/><hr>

  <a>${sessionkey1}</a><hr>


  <s:property value="#session.sessionkey1"/><hr>

   <a>${scontextKey1}</a><hr>

  <s:property value="#application.scontextKey1"/><hr>

  <s:debug></s:debug>

  </body>

    </html>
```

PS:ContextMap是xwork中的东西,struts2的核心就是xwork中的ognl,但是为了方便使用,所以就在ContextMap的基础上添加了root栈,ContextMap和root栈组成了值栈,值栈如下图所示:

3.2、ValueStack的API

> 操作contextMap中根的数据

3.2.1 action 类

```
/**
操作值栈
 * @author jackiechan
 *
 */
public class ActionDemo2  extends ActionSupport{
    private String actionNNNNNNN="默认值";
    public void setActionNNNNNNN(String actionNNNNNNN) {
        this.actionNNNNNNN = actionNNNNNNN;
```

```
    }
    public String getActionNNNNNNN() {
        return actionNNNNNNN;
    }
    public ActionDemo2() {
    }
    @Override
    public String execute() throws Exception {
        ValueStack valueStack = ActionContext.getContext().getValueStack();
        //看栈顶是不是一个 map，是的话就把内容放到 map 里，不是,就创建一个 map，把内容放进去,然后放到栈顶
        valueStack.set("valueStackkey1", "valueStackvalue1");

        valueStack.push("pushvalue");//注意不要往值栈里面直接扔字符串
         Date date=new Date();
         valueStack.push(date);

//       Date date2=new Date();
//       valueStack.push(date2);
         Customer customer=new Customer();
         customer.setName("张三");

         Customer customer2=new Customer();
         customer2.setName("李四");

         valueStack.push(customer);
         valueStack.push(customer2);

//       valueStack.pop();
//       valueStack.pop();
         valueStack.getContext().put("name", "contextName");
         //尽量不要往栈顶 放 list 集合,因为不好迭代
        Object findValue = valueStack.findValue("name");
        System.err.println("findValue==="+findValue);//从栈顶开始往下找这个属性,找到了就取出来,找不到,作为 key 去 contextmap 中找
        //String findString = valueStack.findString("name111", true); //第二个参数,代表找不到的时候 要不要抛出异常
        valueStack.setValue("actionNNNNNNN", "setvalue1");
        //表单中有隐藏域 放一些不需要用户输入的内容,有时候隐藏域里面的值是固定的,所以不需要再传递,直接在代码中添加,也不是直接写死到封装对象中
        valueStack.setParameter("canshu", "zhi");//往请求参数中添加了一个参数
        return super.execute();
    }
}
```

3.2.1 jsp

```
<%@ 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>

    <title>My JSP 'login.jsp' starting page</title>

    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->

  </head>

  <body>
  222222222<hr>
  <!--
     只有在大的 context 里面的数据才需要用#key 的方式取出来
    从值栈中找数据的方式
        从栈顶的对象上一个一个找,有没有这个 valueStackkey1名字的 key 或者是属性,找到了
就显示,找到之后就不会往下找了
        所以直接写属性名或者是 key 即可,不需要写任何#或者是对象名之类的



        struts 封装了 httpservletrequest
                通过表达式取值,找到了就返回
                        找不到,改成用 ValueStack 去查找值
                                ValueStack 的查找之的方法内部,如果没找到,会被当做 key
去 contextmap 中找
    -->

  <s:property value="valueStackkey1"/><hr>
  <s:property value="bytes"/><hr>

  <s:property value="year"/><hr>
  <s:property value="name"/><hr>
  <!--
  执行了 cutstack 方法,返回了一个新的集合
  [1] 就是索引从下标1开始,
  栈顶尽量不要放集合,集合放 context 或者 request 等 map 里面
   -->
  <s:property value="[1].name"/><hr>
  111111<hr>
```

```
  <s:property value="setkey"/><hr>
  222222<hr>
  <s:property value="#setkey"/><hr>
  3333333<hr>
  <s:property value="actionNNNNNNN"/><hr>

  <s:property value="canshu"/><hr>

  <a>${year}</a><hr>

  <a>${actionNNNNNNN}</a><hr>


  <s:debug></s:debug>

  </body>

    </html>
```

四、OGNL的其他用法

4.1 jsp 中可以使用 el 表达式获取 OGNL 的内容

> EL表达式在Struts2中，被做了小小的改动。Struts2对原始的HttpServletRequest（服务器提供）进行包装，org.apache.struts2.dispatcher.StrutsRequestWrapper

```
/**
     * Gets the object, looking in the value stack if not found
     *
     * @param key The attribute key
     */
    public Object getAttribute(String key) {
        if (key == null) {
            throw new NullPointerException("You must specify a key value");
        }

        if (disableRequestAttributeValueStackLookup ||
key.startsWith("javax.servlet")) {
            // don't bother with the standard javax.servlet attributes, we can
short-circuit this
            // see WW-953 and the forums post linked in that issue for more info
            return super.getAttribute(key);
```

```
        }

        ActionContext ctx = ActionContext.getContext();//获取上下文
        Object attribute = super.getAttribute(key);//调用原始的属性方法,也就是 EL 表达式原始的方法

        if (ctx != null && attribute == null) {//如果 EL 表达式中没有找到并且有 context
            boolean alreadyIn = isTrue((Boolean) ctx.get(REQUEST_WRAPPER_GET_ATTRIBUTE));

            // note: we don't let # come through or else a request for
            // #attr.foo or #request.foo could cause an endless loop
            if (!alreadyIn && !key.contains("#")) {//如果没有#号,则进来查找
                try {
                    // If not found, then try the ValueStack
                    ctx.put(REQUEST_WRAPPER_GET_ATTRIBUTE, Boolean.TRUE);
                    ValueStack stack = ctx.getValueStack();
                    if (stack != null) {
                        attribute = stack.findValue(key);//从值栈中查找,内部跟着方法进去,可以发现,最终如果找不到,会去 contextmap 中找,也就是和 加#一样
                    }
                } finally {
                    ctx.put(REQUEST_WRAPPER_GET_ATTRIBUTE, Boolean.FALSE);
                }
            }
        }
        return attribute;
    }
```

4.2 jsp 中使用 ognl

```
<%@ 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>

<title>My JSP 'index.jsp' starting page</title>
<meta http-equiv="pragma" content="no-cache">
<meta http-equiv="cache-control" content="no-cache">
<meta http-equiv="expires" content="0">
<meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
<meta http-equiv="description" content="This is my page">
```

```
<!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->
</head>

<body>
    <%--
      构架了一个集合 集合的写法 {值1,值2,值3...}
       --%>
    <s:property value="{'a','b','c'}.size()" />
    <hr>
    <%--
        构建了一个 map  map的写法 #{'key1':value1,'key2':value2....}
         --%>
    <s:property value="#{'a':'a1','b':'b1','c':'c1'}.getClass()" />
    <hr>


    <!--
  单选框
  复选框
  s:radio 是一个单选的分组 name 一致

  radio name 不一致,可以同时选中
  list 所有要显示的数据
  内部如果使用了集合 map  必须要加 name 属性  不加 name 出错,
   -->

    <s:radio list="{'吃饭','睡觉','打豆豆'}" name="hobby1"></s:radio>
    <hr>
    <s:radio list="{'吃饭','睡觉','打豆豆'}" name="hobby2"></s:radio>
    <hr>
    <s:radio list="{'男','女','女博士'}" name="gender"></s:radio>
    <hr>
    <s:radio list="#{'1':'男','0':'女','2':'女博士'}" name="gender1"></s:radio>
    <hr>
    <!--
  根据数据的多少显示一组多选框
  如果用的是 list 集合构建多选框,那么显示的和传递的值是一样的
   -->
    <s:checkboxlist list="{'吃饭','睡觉','打豆豆'}" name="aaa"></s:checkboxlist>
    <hr>
    <!--
  如果时候map 构建多选框,那么map 的 key 会被传递到服务端, value 是显示的内容
   -->
    <s:checkboxlist list="#{'0':'吃饭','1':'睡觉','2':'打豆豆'}" name="aaa">
</s:checkboxlist>

    <hr>
```

```
    <!--
  listKey 哪个值是用来传递到服务端的， key 或者 value
  listValue 哪个值是用来显示的 key 或者 value
    -->
    <s:checkboxlist list="#{'0':'吃饭','1':'睡觉','2':'打豆豆'}" listKey="value"
        listValue="key" name="aaa"></s:checkboxlist>

    <hr>

    <s:checkboxlist list="#customers" name="dsasdas" listKey="age"
listValue="name"></s:checkboxlist>

    <!--
 value 里面默认是 ognl 表达式
    如果想将 value 的值作为字符串原样输出,需要用''包裹起来

  -->
    <s:property value="'name'" />
    <hr>
    <!--
 %{aaaaa} 用 %{}包裹起来,强制使用 ognl 表达式
   -->
    <s:property value="%{aaaaa}" />
    <hr>

    <s:property value="name" />
    <hr>
    <!--
    如果不写 value 取出来的是栈顶的对象
   -->
    <s:property />
    <hr>
    <!--
  scope 取值范围application|session|request|page|action
    默认范围 context和 request
    注意有很多 bug
  -->
    <s:set var="setvar" value="11111"></s:set>
    <hr>
    <hr>
    <s:property value="setvar" /><br>
    <hr>
    <hr>
    <s:property value="#request.setvar" /><br>
    <hr>
    <hr>
    <s:property value="#session.setvar" /><br>
    <!--
    往栈顶放数据
```

```
        push 标签一旦结束,就从栈顶弹掉
     -->
    <s:push value="1234567890">
    </s:push>
    这是华丽的分割线
    <!--
        escapeHtml碰到 html 内容是否转义 转义之后会原样输出内容,如果想以 html 的方式输出 就设置为 false
     -->
    <s:property value="'<hr>'" escapeHtml="false"/>
        <!--
        name 类的全限定名称
        var 变量名
        以 var 作为 key  name 对应的类创建一个对象作为 value 存放contextmap 中
          -->
        <s:bean name="java.util.Date" var="date"></s:bean>
        <s:property value="#date.time" /><hr>


        <!--
         value 你要遍历的集合或者是 map  默认是 ognl 表达式
         var 是每次遍历的对象对应的 key, 会以他作为 key 存放到 context 中
         status 当前的状态  会将状态封装的对象 以 status 的值作为 key 放到 contextmap 中
         isEven 是否是偶数
         isOdd 是否是基数
         isFirst
         isLast
         getIndex 当前的索引 当前遍历到了第几条数据 从0开始
         getCount 当前的 count  从1开始
          -->
        <%-- <s:iterator value="#customers" var="customer" status="vs">
            <s:property value="#customer.name"/><hr>
             <s:property value="#vs.count"/><hr>
        </s:iterator> --%>


        <table>
            <tr>
            <td>序号</td>
            <td>名字</td>
            </tr>
            <s:iterator value="#customers" var="customer" status="vs">

            <tr>
                <td><s:property value="#vs.count"/></td>
                <td><s:property value="#customer.name"/></td>
            </tr>
```

```
    </s:iterator>

    </table>

        /////////////////
        <table>
        <tr>
        <td>序号</td>
        <td>名字</td>
        </tr>
        <s:iterator value="#customers"  status="vs">

        <tr>
            <td><s:property value="#vs.count"/></td>
            <td><s:property value="name"/></td>
        </tr>

    </s:iterator>

    </table>









        //////////////
遍历 map request 的 map<hr>

<table>
    <tr>
        <td>序号</td>
        <td>key</td>
        <td>value</td>
    </tr>

    <!--
    key 从栈顶开始往下找,有没有 属性名是 key 的对象

            或者是有没有一个 map 里面有 key 这个 key
        没有 var 将每次遍历的对象放到栈顶,这个对象是一个 Map.EntrySet的对象
            这个对象身上有 key 和 value 两个属性
            所以我们写 key 和 value 从栈顶每个对象里面找的时候能找到 然后显示出来

    -->
    <s:iterator value="#request" status="vs">
    <tr>
    <!--
        内部每次遍历到 压倒栈顶,用完就弹栈
```

```
        -->
            <td><s:property value="#vs.count"/></td>
            <td><s:property value="key"/></td>
            <td><s:property value="value"/></td>
        </tr>

        </s:iterator>
    </table>

    /////////////////

        <table>
        <tr>
            <td>序号</td>
            <td>key</td>
            <td>value</td>
        </tr>

        <!--
        key 从栈顶开始往下找,有没有 属性名是 key 的对象

                或者是有没有一个 map 里面有 key 这个 key
            没有 var 将每次遍历的对象放到栈顶,这个对象是一个 Map.EntrySet的对象
                这个对象身上有 key 和 value 两个属性
                所以我们写 key 和 value 从栈顶每个对象里面找的时候能找到 然后显示出来

        -->
        <s:iterator value="#request" var ="req" status="vs">
        <tr>
        <!--
            内部每次遍历到 压倒栈顶,用完就弹栈
          -->
            <td><s:property value="#vs.count"/></td>
            <td><s:property value="#req.key"/></td>
            <td><s:property value="#req.value"/></td>
        </tr>

        </s:iterator>
    </table>

        <!--
            test 是一个表达式 内部可以使用 ongl 表达式
            如果多个判断条件  就写多个 if
         -->
        <s:if test="name=='李四1'||name=='李四'">
            <s:property value="'真的是赵四'"/>
        </s:if>

        <s:elseif test="name=='李四1'">
```

```
                <s:property value="'千锋有好多慧慧'"/>
        </s:elseif>
        <s:else>
            <s:property value="'复合肥哈地方控件很大喝咖啡的境况'"/>
        </s:else>
        <!--
        以下三个只有标签名不一样
          -->
        <s:select list="{'吃饭','睡觉','打豆豆'}" name ="select"></s:select>
        <s:checkboxlist list="{'吃饭','睡觉','打豆豆'}" name ="select">
</s:checkboxlist>
        <s:radio list="{'吃饭','睡觉','打豆豆'}" name ="select"></s:radio>

        <s:select list="#{'0':'吃饭','1':'睡觉','2':'打豆豆'}" listKey="value"
        listValue="key" name="aaa"></s:select>


    <s:debug></s:debug>
</body>
</html>
```

五 Struts2中的UI主题

5.1、提供的主题：

> simple（实际用）、xhtml（默认）、css_xhtml、ajax

5.2、修改struts2使用的主题

> 配置全局参数,在 struts.xml

```
<constant name=*"struts.ui.theme"*value=*"simple"*></constant>
```

5.3 jsp 中使用主题

```
    <!-- theme 设置主题, 这个是局部配置,具有最高的优先级-->
    <s:form theme="xhtml" action="test3">
        <s:textfield value="张胜男" label="姓名"></s:textfield>
        <s:textfield value="fsdfsd" label="姓名"></s:textfield>
```

```
    </s:form>
    <hr>

    <s:form theme="xhtml" action="test3">
        <s:textfield name="name" value="aaaaa" label="姓名"></s:textfield>
        <s:token></s:token>
        <s:submit value="tijiao"></s:submit>
    </s:form>
    <hr>
    <s:form theme="xhtml" action="test4">
        <s:textfield name="name" value="aaaaa" label="姓名"></s:textfield>
        <!--
        防止重复提交的隐藏域
          -->
        <s:token></s:token>
        <s:submit value="tijiao"></s:submit>
    </s:form>
    <hr>

    <s:form >
        <s:textfield value="张胜男" label="姓名" name =""></s:textfield>
        <s:textfield value="fsdfsd" label="姓名"></s:textfield>
    </s:form>
</body>
```

六 防止表单重复提交

6.1 方式一

> 此方式需要在 jsp 中并且使用 struts 标签

6.1.1在表单中添加一个s:token

6.1.2 执行流程

1. 服务器在解析 jsp 的时候向HttpSession中存放了一个令牌
2. 服务器在解析 jsp 的时候会向表单中产生一个隐藏域，存了令牌,这个令牌和 session 中的一致
3. 执行动作方法前，有一个叫做token的拦截器负责处理,当拦截后,首先将两个令牌取到,如果一致,则删除 session 中的令牌,当重复提交的时候, session 中就没有令牌了,这时候两个比较就不一致,可以确定是重复提交

6.2 方式二

> 在需要拦截的 action 对应的配置文件中添加拦截器,并且编写好重复提交的页面接口

6.2.1

```
<action name="token" class="com.suo.actions.TokenAction">
            <result name="success">/WEB-INF/result/LoginResult.jsp</result>
            <!-- 若重复提交，则会跳转到这个页面，注意这里result的名字，一定要是
invalid.token -->
             <result name="invalid.token">/WEB-INF/result/TokenFailed.jsp</result>

            <interceptor-ref name="token"></interceptor-ref>
            <interceptor-ref name="defaultStack"></interceptor-ref>
            <!-- 这里一定要有这两个拦截器 -->
        </action>
```

6.2.2 流程

> 是由tokenSession 拦截器实现的,流程类似于上面的流程,不过就是发现是重复提交后会跳转到配置的页面