**Struts1**

## 一、概述

Struts Framework是一个开源的**应用于MVC模式的WEB开发框架**，扮演的角色是控制层。

最早是作为Apache Jakarta项目的组成部分，框架的主要架构设计和开发者是Craig R.McClanahan。

## 二、基本运作流程

1. 服务器启动的时候读取web.xml文件，加载ActionServlet类并初始化struts-config-xxx.xml文件，把配置信息如action路径、action名称、formBean信息、全局跳转信息放到ActionMapping中，跳转路径封装为action的一个ActionFarword对象，ActionMapping包含ActionFarword。

2. 用户在前台客户端发送HTTP请求到后台，负责接收请求的控制器是ActionServlet类，该类继承HttpServlet类，如果ActionForm实例不存在，容器也会创建一个ActionForm对象,封装所有的请求参数，此时容器会根据配置信息决定是否需要表单验证。如果需要验证，就调用ActionForm的validate()方法。如果ActionForm的validate（）方法返回一个包含一个或多个ActionMessage的ActionErrors对象，就表示表单验证失败，此时ActionServlet将直接把请求转发给包含用户提交表单的JSP组件。在这种情况下，不会在创建Action对象并调用Action的execute()方法。

3. 如果返回null则表示验证成功，下一步ActionServlet把请求交付RequestProcessor类处理，该类负责进行action分发，从配置信息中找出请求路径对应的action类，如果不存在，就返回用户请求路径无效的信息。若找到有对应的action而且没有实例，则初始化一个实例，并调用里面的execute方法，execut调用业务逻辑和数据访问层进行业务流程处理，返回一个ActionForward对象，由视图显示处理的结果，視图部份可以是HTML、JSP、Struts自定义标签、PDF文件、Excel文件、一个请求等。

4. 四个主要类的请求顺序是：ActionMapping--ActionForm---Action---ActionForward.

5. RequestProcessor类包含一个HashMap，作为存放所有Action实例的缓存。每个Action实例在缓存中存放的key为Action类名。在RequestProcessor类的processActionCreate()方法中，首先检查在HashMap中是否存在Action实例，如果有直接使用，否则创建一个新的。创建Action实例的代码位于同步代码块中，以保证只有一个线程创建Action实例，然后放在HashMap中。供其他线程使用。所以说Servlet和Struts1都是单例模式。

## 主要的包、组件

**1. 主要的包及作用**

**org.apache.struts.Action：**控制整个struts1框架运行的核心类；如ActionServlet,RequestProcessor,Action,ActionMapping,ActionForm,ActionForward等

**org.apache.struts.Actions：**特定适配器转换功能；如DispatchAction

o**rg.apache.struts.Config：**提供对配置文件struts-config.xml元素的映射

**org.apache.struts.Taglib**：标签类的集合，包括html,logic,bean

**org.apache.struts.Util：**支持web Application的一些常用服务功能

**org.apache.struts.Validator：**用于动态的配置form表单验证(struts1.1以上)

**2 控制层的分类**

@ ActionServlet（公司的老总）

用来接受所有客户端的\*.do请求，但是他本身不会去处理请求，而是将请求委托给RequestProcessor全权处理；

@ RequestProcessor（项目总监）

用来接受ActionServlet转发过来的请求并处理请求，它会根据struts.xml中的配置调用不同的Action去全权处理客户端不同的请求；

@ Action：（程序员）

它的功能和Servlet一样作为控制器来控制和转发请求，由它去调用模型层中的业务接口方法，并根据不同的结果选择不同的视图显示；

**3 模型层中的ActionForm（需求分析师）**

a.封装提交表单中的字段信息

b.对提交表单中的字段信息进行验证(服务器端验证)

c.可以对表单中的字段信息重新赋值

**4 ActionMapping（美工）**

ActionServlet只是任务的分派者，它依请求分配任务给其它的对象来执行，而分配的依据是请求的URI以及struts-config.xml的<action-mappings>卷标所设定的内容。

<action-mappings>用来描述一组ActionMapping对象，当中的每一个<action>卷标都对应一个ActionMapping对象，当客户端发出请求至ActionServlet时，ActionServlet根据其URI及<action>卷标设定的path属性查看对应的ActionMapping对象，ActionMapping对象会告诉ActionServlet该使用哪一个Action对象（在<action>卷标中使用type属性设定），而ActionServlet再将工作交给该Action对象来执行。

**5 ActionForward（验收人员）**

ActionForword继承自ForwardConfig，它实际上是用来封装视图路径和转发请求方式的一个对象，目的是控制器将 Action 类的处理结果转发至目的地。

这个对象可以有两种创建方式：

return new ActionForward("/basic/error.jsp");

return mapping.findForward("error");

**6 ActionMessages和ActionErrors（测试人员）**

尽管Struts框架提供了有效的异常处理机制，但不能保证处理所有的错误，这时Struts框架会把错误抛给Web容器，在默认情况下Web容器会向用户浏览器直接返回原始信息。如果想避免直接让用户看到这些原始信息，可以在web.xml中配置<error-page>元素。

**7 引入的包介绍**

antlr.jar --- 接受词文法语言的描述

commons-beanutils.jar ---提供对JAVA的反射和自省API的包装

commons-digester.jar ---用于XML到Java对象的映射，处理struts-config.xml配置文件，已达到生成对应的对象。

commons-fileupload.jar ---支持文件上传功能

commons-validator.jar ---对表单进行验证，支持校验规则和错误消息的国际化

commons-logging.jar --- 支持日志的输出

jakarta-oro.jar --- 文本处理工具，支持正则表达式、blob表达式、替换、分割、文件名过滤的功能。

struts.jar---struts的核心jar包

## 四、代码开发主要步骤

**1 引入jar包（见3.7介绍）**

**2 web.xml配置**

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<servlet>

<servlet-name>action</servlet-name>

<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>

<init-param>

<param-name>config</param-name>

<param-value>/WEB-INF/struts-config.xml</param-value>

</init-param>

<init-param>

<param-name>debug</param-name>

<param-value>3</param-value>

</init-param>

<load-on-startup>0</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>action</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

<error-page>

<error-code>404</error-code>

<location>/exception/error404.jsp</location>

</error-page>

<error-page>

<error-code>500</error-code>

<location>/exception/error500.jsp</location>

</error-page>

<error-page>

<exception-type>java.lang.Exception</exception-type>

<location>/exception/default.jsp</location>

</error-page>

</web-app>

**3 struts-config.xml的配置**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-config\_1\_2.dtd">

<struts-config>

<data-sources />

<form-beans >

<form-bean name="loginForm" type="form.LoginForm">

</form-bean>

</form-beans>

<global-exceptions />

<global-forwards />

<global-results> ---全局页面导航

<result name="global">/jsp/login.jsp</result>

</global-results>

<action-mappings >

<action path="/login" parameter="method" name="loginForm" type="action.LoginAction">

<forward name="ok" path="/index.jsp"></forward>

<forward name="error" path="/login.jsp"></forward>

</action>

<action path="/manBookInfo" parameter="method" name="loginForm" type="action.ManBookAction">

<forward name="ok" path="/index.jsp"></forward>

<forward name="edit" path="/addBookInfo.jsp"></forward>

</action>

</action-mappings>

<message-resources parameter="action.ApplicationResources" />

</struts-config>

**4 login.jsp页面**

<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>

<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic"%>

entryForm.action="login.do?method=login";

entryForm.submit();

**5 LoginAction类**

public class LoginAction extends DispatchAction

{

public ActionForward login(ActionMapping map, ActionForm form,

HttpServletRequest req, HttpServletResponse res) throws Exception

{

LoginForm loginForm = (LoginForm) form;

String forward = null;

if(null!=loginForm.getUserName()&&"wade399".equals(loginForm.getUserName())

&&null!=loginForm.getPwd()&&"123".equals(loginForm.getPwd()))

{

forward = "ok";

List<Book> books = BookService.getBookService().getBookInfoAll();

req.setAttribute("books",books);

return map.findForward(forward);

}

else

{

forward = "error";

req.setAttribute("errorInfo", "The userName or pwd is error！");

return map.findForward(forward);

}

}

}

## 五、国际化的支持

浏览器设置语言，到Requestprosessor类中找到session中的Local,如果为空则取浏览器中的，否则取session的，如果Local没有对应的国际化资源文件，比如说浏览器设置的是意大利，或session中的是意大利，而没有意大利的MessageResource文件，则**从操作系统中读取相应的语言，如果您操作系统提供的语言也没有国际化文件的话则读取默认的MessageResource.properties。**

不同用户在不同的国家，可以自定义Local,再放到session里面，注意，一个用户对应一个session.当执行bean:meaasge标签命令的时候，标签类会获得session里面的Local,获取对应的国际资源文件内容显示出来。

1 struts-config.xml中配置message-resources标签，basename,要以此为基准命名

<message-resource parameter="MessageResource"/>

2 提供国际化资源文件 baseName+语言+国家.properties

中文：MessageResource\_zh\_cn.properties

英文：MessageResource\_en\_us.properties

缺省文件：MessageResource.properties，找不到对应的文件时读取此项文件

3 设置文字

MessageResource\_zh\_cn.properties文件中，采用native2ascii转换成unicode

user.title=\u7528\u6237\u767b\u5f55

user.username=\u7528\u6237

user.password=\u5bc6\u7801

user.button.login=\u767b\u5f55

MessageResource\_en\_us.properties文件中

user.title=User Login

user.username=User Name

user.password=Password

user.button.login=Login

4 jsp页面引用

<bean:message key="user.username"/>

5 类中设置

Locale currentLocale = new Locale("zh", "CN"); //中文

Locale currentLocale2 = new Locale("en", "US"); //英文

request.getSession().setAttribute(Globals.LOCALE\_KEY, currentLocale); //设置在SESSION中

this.setLocale(req, currentLocale2); //调用Action中的setLocale方法

6 动态文本国际化

user.title={0} {1}User Login

<bean:message key="user.name">

<bean:message key="user.title" arg1="caterpillar" arg2="good morning"/>

ActionMessages mes = new ActionMessages();

ActionMessage me = new ActionMessage("user.name",new Object[]{'wade','wade2'});

mes.add(me);

saveMeaasges(res,mes);

ActionMessage me2 = new ActionMessage("user.name",new Object[]{'wade','wade2'});

mes.add(me2);

saveErrors(res,mes);

## 三种常见的标签

**1 html标签**

<html:errors/>

<html:link href="http://www.caterpillar.onlyfun.net/phpBB2"/>

<html:messages id="messages" message="true">

<bean:write name="messages"/>

</html:messages>

<html:button property="button1" value="按我" onclick="alert('Hello!World!')" title="Hello!World!"/>

<html:form action="Login" focus="text1" method="POST">

......

<html:form/>

<html:img page="/images/caterpillar.jpg" alt="caterpillar" width="" height="" />

<html:image property="push" src="push.jpg" onclick="alter("send")"/>

<html:submit value="送出"/>

<html:reset value="重清"/>

<html:text property="name" size="20"/>

<html:password property="password" size="20"/>

<html:hidden property="hid" value="1234"/>

<html:textarea property="message" value="在此留言" cols="50" rows="5"/>

<html:radio property="sex" value="man"/>男

<html:radio property="sex" value="woman"/>女

<html:checkbox property="lang[0]" value="Java">Java</html:checkbox>

<html:checkbox property="lang[1]" value="C#">C#</html:checkbox>

<html:select property="lang" value="Java">

<html:option value="Java">Java</html:option>

<html:option value="C#">C#</html:option>

</html:select>

**2 bean标签**

<bean:write name="userForm" property="username"/>

<bean:message key="welcome.title"/>

<bean:message key="welcome.greeting" arg1="caterpillar" arg2="good morning"/>

<bean:include id="inc1" page="somepage.jsp"/>

<bean:define id="hello" value="Hello!World!"/>

3 logic标签

<logic:equal name="name" value="wade">

.....

</logic:equal>

<logic:notEqual name="name" value="wade">

.....

</logic:notEqual>

<logic:greatEqual name="name" value="5">

.....

</logic:greatEqual>

<logic:lessEqual name="name" value="5">

.....

</logic:lessEqual>

<logic:greatThan name="name" value="5">

.....

</logic:greatThan>

<logic:lessThan name="name" value="5">

.....

</logic:lessThan

<logic:present scope="request" name="valid\_user">

.....

</logic:present>

<logic:notPresent scope="request" name="valid\_user">

.....

</logic:notPresent>

<logic:notEmpty name="authors">

<logic:iterate name="authors" id="author">

<tr>

<td height=20 align=left>

<bean:write property='authorId' name='author'/

</td>

<td height=20 align=left>

<bean:write property="authorName" name="author" />

</td>

</tr>

</logic:iterate>

</logic:notEmpty>

## 七、动态验证框架及错误处理

**1. 动态验证**

@ struts-config.xml中配置插件

<plug-in className="org.apache.struts.validator.ValidatorPlugIn">

<set-property property="pathnames" value="/org/apache/struts/validator/validator-rules.xml,/WEB-INF/validation.xml"/>

</plug-in>

@ 在web-inf目录下建立 validation.xml文件

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE form-validation PUBLIC

"-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.3.0//EN"

"http://jakarta.apache.org/commons/dtds/validator\_1\_3\_0.dtd">

<form-validation>

<formset>

<form name="logonForm">

<!-- depends 设置校验规则 若多个规则， 用逗号隔开 这里 required username必填-->

<field property="username" depends="required,maxlength,minlength">

<!-- 用于返回到页面上的校验结果， 如“\*\*必填” 这里在application.properties中找想要显示的对应的文字 -->

<arg key="logonForm.username" />

<!-- key用表达式取某变量值,name为校验方式,resourse为是否从资源文件中取值 -->

<arg key="${var:maxlength}" name="maxlength" resource="false" />

<var>

<var-name>maxlength</var-name>

<var-value>10</var-value>

</var>

<arg key="${var:minlength}" name="minlength" resource="false" />

<var>

<var-name>minlength</var-name>

<var-value>6</var-value>

</var>

</field>

<field property="password" depends="required,mask">

<arg key="logonForm.password" />

<var>

<var-name>mask</var-name>

<var-value>^[0-9a-zA-Z]\*$</var-value>

</var>

</field>

</form>

</formset>

</form-validation>

public class LogonForm extends ValidatorForm

{

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;

}

}

**2. 错误处理**

@ web.xml配置：

<error-page>

<error-code>404</error-code>

<location>/exception/error404.jsp</location>

</error-page>

<error-page>

<error-code>500</error-code>

<location>/exception/error500.jsp</location>

</error-page>

<error-page>

<exception-type>java.lang.Exception</exception-type>

<location>/exception/default.jsp</location>

</error-page>

当WEB容器捕获到exception-type或error-code指定的错误时将跳到由location指定的页面。

@ 提示信息和错误信息的发出

----------ActionError/ActionErrors/ActionMessage/ActionMessages:

ActionMessage：用于保存一个与资源束对应的提示信息。主要构造函数如：

ActionMessages:用于保存多个ActionMessage。并在html:errors 和html:messages中起作用。

html:messages/html:errors使用property属性访问某个资源

ActionErrors：用于保存一个与资源束对应的错误信息。用法跟ActionMessages差不多。

ActionError不赞成使用。

@ AtionErrors和ActionMessages的区别

& ActionErrors是ActionMessages的一个子类，功能几乎相同，不同点在于标签<html:errors/>和<html:messages>的使用上的区别。

html:errors指定了footer和header属性。默认值为errors.header和errors.footer,需要时可以自己指定。如果资源属性文件配置了 errors.header和errors.footer，则任何时候使用html:errors时开头和结尾都是这两个属性对应的资源信息。

而html:message默认情况下没有errors.header和errors.footer值，当然可以自己指定。

& html:errors可以根据property属性指定显示一个错误信息。html:messages有一个必添项id。html:messages不能直接显示信息，它将选出的信息放入一个用id标识的Iterator对象里，然后在用ben:write或JSTL c:out标签显示每个信息.例如：

<html:messages message="true" id="msg">

<c:out value="${msg}"/><br />

</html:messages>

ActionErrors errors = new ActionErrors();

ActionMessages ms = new ActionMessages();

ms.add("11111", new ActionMessage("error.password"));

ms.add("22222", new ActionMessage("error.eerrdf"));

errors.add(ms);

## 八、Tiles整合

**1 Tiles增加了layout的概念**，其实就是把一个页面划分为几块。通常的来说一个页面大概可以划分为如下几块

head页面头部：存放一个运用的公共信息：logo等，如果是网站可能是最上面的一块。

menu页面菜单：放置一个运用中需要使用的菜单，或者在每一个页面都使用的连接。

footer页面尾部：如版权信息等。

body页面主题内容：每个页面相对独立的内容。

如果按上面的划分那对每一个页面我们只要写body里面的内容，其他的就可以共享重用。

如果大多数页面的布局基本相同我们甚至可以使用一个jsp文件根据不同的参数调用不同的body。

**2 在struts-config.xml中配置信息**

<plug-in className="org.apache.struts.tiles.TilesPlugin" >

<set-property property="definitions-config" value="/WEB-INF/conf/tiles-defs.xml" />

<set-property property="moduleAware" value="true" />

<set-property property="definitions-parser-validate" value="true" />

</plug-in>

**3 tiles-defs.xml设置**

<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE tiles-definitions PUBLIC

"-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"

"http://jakarta.apache.org/struts/dtds/tiles-config\_1\_1.dtd">

<tiles-definitions>

<definition name=".myLayout" path="/tiles/myLayout.jsp">

<put name="title" value="Sample Page Title" />

<put name="header" value="/tiles/header.jsp" />

<put name="menu" value="/tiles/menu.jsp" />

<put name="footer" value="/tiles/footer.jsp" />

<put name="body" value="/tiles/body.jsp" />

</definition>

</tiles-definitions>

**4 页面引用**

<tiles:insert attribute="header"/>

<tiles:insert attribute="menu"/>

<tiles:insert attribute="body"/>

<tiles:insert attribute="footer"/>

**5 直接在页面引用，不用配置**

<%@page contentType="text/html; charset=Big5"%>

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

<tiles:insert page="/tiles/myLayout.jsp" flush="true">

<tiles:put name="title" value="Tiles範例" />

<tiles:put name="header" value="/tiles/header.jsp" />

<tiles:put name="menu" value="/tiles/menu.jsp" />

<tiles:put name="footer" value="/tiles/footer.jsp" />

<tiles:put name="body" value="/tiles/body.jsp" />

</tiles:insert>

## 九、文件的上传

1 页面

<html:form action="/Upload" method="post" enctype="multipart/form-data">

選擇檔案:<html:file property="file" />

<html:submit>上傳</html:submit>

</html:form>

2 后台

public class UploadForm extends ActionForm

{

private FormFile file;

public void setFile(FormFile file)

{

this.file = file;

}

public FormFile getFile()

{

return file;

}

public void reset(ActionMapping mapping, HttpServletRequest req)

{

file = null;

}

}

public class UploadAction extends Action

{

public ActionForward execute(ActionMapping mapping,

ActionForm form,

HttpServletRequest request,

HttpServletResponse response) throws Exception

{

UploadForm fileForm = (UploadForm) form;

FormFile file = fileForm.getFile();

FileOutputStream fileOutput = new FileOutputStream("/home/caterpillar/files/" + file.getFileName());

fileOutput.write(file.getFileData());

fileOutput.flush();

fileOutput.close();

return mapping.findForward("success");

}

}

## 十、Struts1的优缺点

**1 优点**

@ 利用Struts提供的taglib可以大大节约开发时间

@ 维护扩展比较方便。通过一个配置文件，即可把握整个系统各部分之间的联系，这对于后期的维护有着莫大的好处

@ 实现MVC模式，结构清晰,使开发者只关注业务逻辑的实现

@ 表单验证解决了请求数据的验证问题，增强了系统健壮性

@ 支持国际化I18N,internationalization

@ 页面导航,通过一个配置文件,即可把握整个系统各部分之间的联系,这对于后期的维护有着莫大的好处。

**2 缺点**

@ Struts1 Action是单例模式并且必须是线程安全的，因为仅有Action的一个实例来处理所有的请求。

@ Struts1 Action 依赖于Servlet API ,因为当一个Action被调用时HttpServletRequest 和 HttpServletResponse 被传递给execute方法

@ 类型的转换. Struts的FormBean把所有的数据都作为String类型,而且转化的类型是不可配置的。

@ 测试不方便. Struts的每个Action都同Web层耦合在一起，这样它的测试依赖于Web容器，单元测试也很难实现。

## 十一、Struts1和Struts2的对比

**@ 类:**

Struts1要求Action类继承一个抽象基类。Struts1的一个普遍问题是使用抽象类编程而不是接口。

Struts 2 Action类可以实现一个Action接口，也可实现其他接口，使可选和定制的服务成为可能。Struts2提供一个ActionSupport基类去实现常用的接口。Action接口不是必须 的，任何有execute标识的POJO对象都可以用作Struts2的Action对象。

**@ 线程模式:**

Struts1 Action是单例模式并且必须是线程安全的，因为仅有Action的一个实例来处理所有的请求。单例策略限制了Struts1 Action能作的事，并且要在开发时特别小心。 Action资源必须是线程安全的或同步的。

Struts2 Action对象为每一个请求产生一个实例，因此没有线程安全问题。（实际上，servlet容器给每个请求产生许多可丢弃的对象，并且不会导致性能和垃圾回收问题）

**Servlet 依赖:**

Struts1 Action 依赖于Servlet API ,因为当一个Action被调用时HttpServletRequest 和 HttpServletResponse 被传递给execute方法。

Struts 2 Action不依赖于容器，允许Action脱离容器单独被测试。如果需要，Struts2 Action仍然可以访问初始的request和response。但是，其他的元素减少或者消除了直接 访问HttpServetRequest 和 HttpServletResponse的必要性。

**@ 可测性:**

测试Struts1 Action的一个主要问题是execute方法暴露了servlet API（这使得测试要依赖于容器）。一个第三方扩展－－Struts TestCase－－提供了一套Struts1的模拟对象（来进行测试）。

Struts 2 Action可以通过初始化、设置属性、调用方法来测试，“依赖注入”支持也使测试更容易。

**@ 捕获输入:**

Struts1 使用ActionForm对象捕获输入。所有的ActionForm必须继承一个基类。因为其他JavaBean不能用作ActionForm，开发者经常创建多余的类捕获输入。动态Bean（DynaBeans）可以作为创建传统ActionForm的选择，但是，开发者可能是在重新描述(创建)已经存在的JavaBean（仍然会导致有冗余的javabean）。

Struts 2直接使用Action属性作为输入属性，消除了对第二个输入对象的需求。输入属性可能是有自己(子)属性的rich对象类型。Action属性能够通过 web页面上的taglibs访问。Struts2也支持ActionForm模式。rich对象类型，包括业务对象，能够用作输入/输出对象。这种 ModelDriven 特性简化了taglib对POJO输入对象的引用。

**@ 表达式语言：**

Struts1 整合了JSTL，因此使用JSTL EL。这种EL有基本对象图遍历，但是对集合和索引属性的支持很弱。

Struts2可以使用JSTL，但是也支持一个更强大和灵活的表达式语言－－"Object Graph Notation Language" (OGNL).

**@ 绑定值到页面（view）:**

Struts 1使用标准JSP机制把对象绑定到页面中来访问。

Struts 2 使用 "ValueStack"技术，使taglib能够访问值而不需要把你的页面（view）和对象绑定起来。ValueStack策略允许通过一系列名称相同但类型不同的属性重用页面（view）。

**@ 类型转换：**

Struts 1 ActionForm 属性通常都是String类型。Struts1使用Commons-Beanutils进行类型转换。每个类一个转换器，对每一个实例来说是不可配置的。

Struts2 使用OGNL进行类型转换。提供基本和常用对象的转换器。

**@ 校验：**

Struts 1支持在ActionForm的validate方法中手动校验，或者通过Commons Validator的扩展来校验。同一个类可以有不同的校验内容，但不能校验子对象。

Struts2支持通过validate方法和XWork校验框架来进行校验。XWork校验框架使用为属性类类型定义的校验和内容校验，来支持chain校验子属性

**@ Action执行的控制：**

Struts1支持每一个模块有单独的Request Processors（生命周期），但是模块中的所有Action必须共享相同的生命周期。

Struts2支持通过拦截器堆栈（Interceptor Stacks）为每一个Action创建不同的生命周期。堆栈能够根据需要和不同的Action一起使用。

**Struts2**

## 一、概述

Struts2是以Webwork的设计思想为核心，吸收了Struts1的优点，因此，可以认为Struts2是Struts1和Webwork结合的产物。Struts2是一个兼容Struts1和WebWork的MVC框架，

XWork----WebWork2---Struts2, XWork核心就是用拦截器处理HTTP请求。

**XWork采用拦截器的机制来处理用户请求的全新MVC框架**

## 二、开发流程

**1 引入的jar包**

struts2-core-2.x.x.jar--- Struts 2框架的核心库

xwork-2.x.x.jar--- XWork库，Struts 2在其上构建

ognl-2.6.x.jar--- 对象图导航语言（Object Graph Navigation Language），Struts

**2框架使用的一种表达式语言**

freemarker-2.3.x.jar--- Struts 2的UI标签的模板使用FreeMarker编写

commons-logging-1.1.x.jar--- ASF出品的日志包，Struts 2框架使用这个日志包来支持Log4J和JDK 1.4+的日志记录。

Commons-fileupload--- 文件上传

commons-io-1.3.2.jar--- 文件上传

第三方框架集成，比如说与Spring集成，struts2-spring-plugin-2.x.x.jar.

**3 commons项目是作为JDK的补充拓展及优化的一系列方案**

**由apache公司编写. 包名org.apache.commons,由反转的域名org.apache和包名commons组成.   
如:   
org.apache.commons.logging包,是用来记录日期.   
org.apache.commons.lang包,用来包装部份数据类型.**



整合第三方日志 工具包



对象的创建和销毁在一定程度上会消耗系统的资源，虽然jvm的性能在近几年已经得到了很大的提高，对于多数对象来说，没有必要利用对象池技术来进行对象的创建和管理。但是对于有些对象来说，其创建的代价还是比较昂贵的，比如线程、tcp连接、rpc连接、数据库连接等对象，因此对象池技术还是有其存在的意义。Apache-commons-pool-1.6提供的对象池主要有两种：一种是带Key的对象池，这种带Key的对象池是把相同的池对象放在同一个池中，也就是说有多少个key就有多少个池；另一种是不带Key的对象池，这种对象池是把生产完全一致的对象放在同一个池中，但是有时候，单用对池内所有对象一视同仁的对象池，并不能解决的问题。



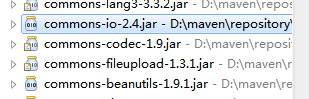
http请求工具包



包装部份数据类型. BooleanUtils 等



集合工具类，ComparatorUtils，



**4 web.xml的配置，在struts2框架是通过Filter启动的**

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

在FilterDispatcher的init()方法中将会读取类路径下默认的配置文件struts.xml完成初始化操作。

**5 struts.xml 的配置**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<package name="itcast" namespace="/test" extends="struts-default">

<action name="helloworld" class="cn.itcast.action.HelloWorldAction">

<result name="success">/WEB-INF/page/hello.jsp</result>

</action>

</package>

</struts>

http://localhost:端口/内容路径/test/helloworld.action

**6 Action类**

public class HelloWorldAction

{

private String message;

public String getMessage()

{

return message;

}

public void setMessage(String message)

{

this.message = message;

}

public String execute() throws Exception

{

this.message = "我的第一个struts2应用";

return "success";

}

}

**7 hello.jsp页面，用EL表达式**

<body>

${message}

</body>

**8 一组业务功能相关的Action放在同一个包下**，通常每个包都应该继承struts-default包， struts-default包是由struts内置的，它定义了struts2内部的众多拦截器和Result类型。Struts2很多核心的功能都是通过这些内置的拦截器实现的。

**9 http://server/struts2/path1/path2/path3/test.action的路径寻找**

path1/path2/path3/----path1/path2/-----path1/，如若找不到，则提示没有Action.

**10 result配置类似于struts1中的forward**，但struts2中提供了多种结果类型，如： dispatcher(默认值)、 redirect 、 redirectAction 、 plainText。

<result type="redirectAction">

<param name="actionName">helloworld</param>

<param name="namespace">/test</param>

</result>

<result type="redirect">view.jsp?id=${id}</result> ---${id}表示取相应action的id属性值

**11 FilterDispatcher默认处理的请求是\*.do，修改默认的配置方式：**

<struts>

<constant name="struts.action.extension" value="do,go"/>

</struts>

**12 配置多个struts配置文件**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<include file="struts-user.xml"/>

<include file="struts-order.xml"/>

</struts>

通过这种方式，我们就可以将Struts 2的Action按模块配置在多个配置文件中。

**13 动态方法调用**

http://server/struts/test/helloworld!other.action 表示调用helloworld类的other方法

**14 全局forward**

<package>

.....

<global-results>

<result name="message">/message.jsp</result>

</global-results>

</package>

**15 常量的配置和使用**

常量可以在下面多个文件中进行定义，struts2加载常量的搜索顺序如下，后面的设置可以覆盖前面的设置：

default.properties文件

struts-default.xml

struts-plugin.xml

struts.xml

struts.properties（为了与webwork向后兼容而提供）

web.xml

<constant name="struts.i18n.encoding" value="UTF-8"></constant>

指定默认编码集，作用于HttpServletRequest的setCharacterEncoding方法和freemarker、velocity的输出

<constant name="struts.action.extension" value="do,action"></constant>

指定需要struts2处理的请求后缀，默认值为action。如果需要制定多个属性则用“，”隔开。

<constant name="struts.serve.static.browserCache" value="false"></constant>

设置浏览器是否缓存静态内容，默认值为true。开发中设置为false。

<constant name="struts.configuration.xml.reload" value="true"></constant>

struts的配置文件在修改后是否自动加载。默认为false

<constant name="struts.devMode" value="true"></constant>

控制台输出更多的详细错误信息

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

本设置可以修改struts2中标签的视图主题。修改标签个性化代码样式。

<constant name="struts.objectFactory" value="spring"></constant>

与spring集成时指定spring负责action对象的创建。

<constant name="struts.DynamicMethodInvocation" value="false"></constant>

设置Struts2是否支持动态方法调用，默认值为true。如需关闭设置为false。

<constant name="struts.multipart.maxSize" value="1024"></constant>

上传文件总大小的定义

**16 获取HttpServletRequest / HttpSession / ServletContext / HttpServletResponse对象**

HttpServletRequest request = ServletActionContext.getRequest();

ServletContext servletContext = ServletActionContext.getServletContext();

HttpSession se = request.getSession();

HttpServletResponse response = ServletActionContext.getResponse();

## 三、基本运作流程

1 客户端初始化一个指向Servlet容器的http请求

2 这个请求经过一系列的Filter（有个叫做ActionContextCleanUp的过滤器）

3 接着FilterDispatcher被调用，FilterDispatcher询问ActionMapper来决定这个请是否需要调用某个Action

4 如果ActionMapper决定需要调用某个Action，FilterDispatcher把请求的处理交给ActionProxy

5 ActionProxy通过Configuration Manager询问框架的配置文件，找到需要调用的Action类

6 ActionProxy创建一个ActionInvocation的实例。

7 ActionInvocation实例使用命名模式来调用，在调用Action的过程前后，涉及到相关拦截器（Intercepter）的调用。

8 一旦Action执行完毕，ActionInvocation负责根据struts.xml中的配置找到对应的返回结果。返回结果通常是（但不总是，也可 能是另外的一个Action链）一个需要被表示的JSP或者FreeMarker的模版。在表示的过程中可以使用Struts2 框架中继承的标签。在这个过程中需要涉及到ActionMapper

在上述过程中所有的对象（Action，Results，Interceptors，等）都是通过ObjectFactory来创建的。

## 四、国际化的支持

1 转杯资源国际文件

baseName\_language\_country.properties

其中baseName是资源文件的基本名，我们可以自定义<constant name="struts.custom.i18n.resources" value="ApplicationResource" />

，但language和country必须是java支持的语言和国家

2 对于中文的属性文件，我们编写好后，应该使用jdk提供的native2ascii命令把文件转换为unicode编码的文件。

命令的使用方式如下：native2ascii 源文件.properties 目标文件.properties

3 JSP页面

<s:text name="user"/>，name为属性文件中的key

4 Action类中，可以继承ActionSupport，

String value = getText("user")

5 在表单标签中，通过key属性指定属性文件中的key，如：

<s:textfield name="realname" key="user"/>

6 动态国际化文本

welcome={0},你好,欢迎来到{1}

<s:text name="welcome">

<s:param><s:property value="realname"/></s:param>

<s:param>传智播客</s:param>

</s:text>

getText(String key, String[] args)

getText(String aTextName, List args)

## 五、自定义拦截器

<package name="itcast" namespace="/test" extends="struts-default">

<interceptors>

<interceptor name=“permission" class="cn.itcast.aop.PermissionInterceptor" />

<interceptor-stack name="permissionStack">

<interceptor-ref name="defaultStack" />

<interceptor-ref name=" permission " />

</interceptor-stack>

</interceptors>

<action name="helloworld\_\*" class="cn.itcast.action.HelloWorldAction" method="{1}">

<result name="success">/WEB-INF/page/hello.jsp</result>

<interceptor-ref name="permissionStack"/>

</action>

</package>

因为struts2中如文件上传，数据验证，封装请求参数到action等功能都是由系统默认的defaultStack中的拦截器实现的，所以我们定义的拦截器需要引用系统默认的defaultStack，这样应用才可以使用struts2框架提供的众多功能。如果希望包下的所有action都使用自定义的拦截器，可以通过<default-interceptor-ref name=“permissionStack”/>把拦截器定义为默认拦截器。注意：每个包只能指定一个默认拦截器。另外，一旦我们为该包中的某个action显式指定了某个拦截器，则默认拦截器不会起作用。

public class PermissionInterceptor implements Interceptor

{

private static final long serialVersionUID = -5178310397732210602L;

public void destroy() {}

public void init() {}

public String intercept(ActionInvocation invocation) throws Exception

{

System.out.println("进入拦截器");

if(session里存在用户)

{

String result = invocation.invoke();

}

else

{

return “logon”;

}

//System.out.println("返回值:"+ result);

//return result;

}

}

## 六、验证框架

**1 手工验证框架**

if(username==null && "".equals(username.trim()))

{

this.addFieldError("username", "用户名不能为空");

}

if(this.hasFieldError)

{

return input;

}

验证失败后，请求转发至input视图：

<result name="input">/WEB-INF/page/addUser.jsp</result>

addUser.jsp页面

<s:fielderror/>

**2 XML输入验证框架**

(1) Action类需要继承ActionSupport，然后提供校验文件

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd">

<validators>

<field name="username">

<field-validator type="requiredstring">

<param name="trim">true</param>

<message>用户名不能为空!</message>

</field-validator>

</field>

</validators>

(2) 在这个校验文件中，对action中字符串类型的username属性进行验证，首先要求调用trim()方法去掉空格，然后判断用户名是否为空。

该文件需要和action类放在同一个包下，文件的取名应遵守ActionClassName-validation.xml规则，

其中ActionClassName为action的简单类名，-validation为固定写法。如果Action类为cn.itcast.action.UserAction，

那么该文件的取名应为：UserAction-validation.xml。

(3) <field>指定action中要校验的属性，<field-validator>指定校验器，

<message>为校验失败后的提示信息,如果需要国际化，可以为message指定key属性，key的值为属性文件中的key。

上面指定的校验器requiredstring是由系统提供的，系统提供了能满足大部分验证需求的校验器，这些校验器的定义可以在xwork-2.x.jar中的com.opensymphony.xwork2.validator.validators下的default.xml中找到。

(4) 系统提供的校验器如下：

@ required (必填校验器,要求field的值不能为null)

@ requiredstring (必填字符串校验器,要求field的值不能为null,并且长度大于0,默认情况下会对字符串去前后空格)

@ stringlength(字符串长度校验器,要求field的值必须在指定的范围内,否则校验失败,minLength参数指定最小长度,maxLength参数指定最大长度)

@ regex(正则表达式校验器,检查被校验的field是否匹配一个正则表达式.expression参数指定正则表达式,

caseSensitive参数指定进行正则表达式匹配时,是否区分大小写,默认值为true)

@ int(整数校验器,要求field的整数值必须在指定范围内,min指定最小值,max指定最大值)

@ double(双精度浮点数校验器,要求field的双精度浮点数必须在指定范围内,min指定最小值,max指定最大值)

@ fieldexpression(字段OGNL表达式校验器,要求field满足一个ognl表达式,expression参数指定ognl表达式,该逻辑表达式基于ValueStack进行求值,

返回true时校验通过,否则不通过)

@ email(邮件地址校验器,要求如果field的值非空,则必须是合法的邮件地址)

@ url(网址校验器,要求如果field的值非空,则必须是合法的url地址)

@ date(日期校验器,要求field的日期值必须在指定范围内,min指定最小值,max指定最大值)

@ conversion(转换校验器,指定在类型转换失败时,提示的错误信息)

@ visitor(用于校验action中的复合属性,它指定一个校验文件用于校验复合属性中的属性)

@ expression(OGNL表达式校验器,expression参数指定ognl表达式,该逻辑表达式基于ValueStack进行求值,返回true时校验通过,否则不通过,

该校验器不可用在字段校验器风格的配置中)

required 必填校验器

<field-validator type="required">

<message>性别不能为空!</message>

</field-validator>

requiredstring 必填字符串校验器

<field-validator type="requiredstring">

<param name="trim">true</param>

<message>用户名不能为空!</message>

</field-validator>

stringlength：字符串长度校验器

<field-validator type="stringlength">

<param name="maxLength">10</param>

<param name="minLength">2</param>

<param name="trim">true</param>

<message><![CDATA[产品名称应在1-10个字符之间]]></message>

</field-validator>

int：整数校验器

<field-validator type="int">

<param name="min">1</param>

<param name="max">150</param>

<message>年龄必须在1-150之间</message>

</field-validator>

字段OGNL表达式校验器

<field name="imagefile">

<field-validator type="fieldexpression">

<param name="expression"><![CDATA[imagefile.length() > 0]]></param>

<message>文件不能为空</message>

</field-validator>

</field>

email：邮件地址校验器

<field-validator type="email">

<message>电子邮件地址无效</message>

</field-validator>

regex：正则表达式校验器

<field-validator type="regex">

<param name="expression"><![CDATA[^13\d{9}$]]></param>

<message>手机号格式不正确!</message>

</field-validator>

## 七、主要的标签

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

<s:set name="name" value="'kk'" />

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

<s:set name="list" value="{'zhangming','xiaoi','liming'}" />

<s:iterator value="#list" id="name" status="st">

<font color=<s:if test="#st.odd">red</s:if><s:else>blue</s:else>

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

</s:iterator>

<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>

如果集合为list

<s:checkboxlist name="list" list="{'Java','.Net','RoR','PHP'}" value="{'Java','.Net'}"/>

生成如下html代码：

<input type="checkbox" name="list" value="Java" checked="checked"/><label>Java</label>

<input type="checkbox" name="list" value=".Net" checked="checked"/><label>.Net</label>

<input type="checkbox" name="list" value="RoR"/><label>RoR</label>

<input type="checkbox" name="list" value="PHP"/><label>PHP</label>

如果集合为MAP

<s:checkboxlist name="map" list="#{1:'瑜珈用品',2:'户外用品',3:'球类',4:'自行车'}" listKey="key" listValue="value" value="{1,2,3}"/>

生成如下html代码：

<input type="checkbox" name="map" value="1" checked="checked"/><label>瑜珈用品</label>

<input type="checkbox" name="map" value="2" checked="checked"/><label>户外用品</label>

<input type="checkbox" name="map" value="3" checked="checked"/><label>球类</label>

<input type="checkbox" name="map" value="4"/><label>自行车</label>

<s:radio name="list" list="{'Java','.Net'}" value="'Java'"/>

生成如下html代码：

<input type="radio" name="list" checked="checked" value="Java"/><label>Java</label>

<input type="radio" name="list" value=".Net"/><label>.Net</label>

<s:select name="list" list="{'Java','.Net'}" value="'Java'"/>

<select name="list" id="list">

<option value="Java" selected="selected">Java</option>

<option value=".Net">.Net</option>

</select>

## 八、文件上传

<form enctype="multipart/form-data" action="${pageContext.request.contextPath}/xxx.action" method="post">

<input type="file" name="uploadFile">

</form>

public class HelloWorldAction

{

private File uploadFile;//得到上传的文件

private String uploadFileContentType;//得到文件的类型

private String uploadFileFileName;//得到文件的名称

。。。这里略省了属性的getter/setter方法。。。。。

public String upload() throws Exception

{

String realpath = ServletActionContext.getServletContext().getRealPath("/images");

File file = new File(realpath);

if(!file.exists()) file.mkdirs();

FileUtils.copyFile(uploadFile, new File(file, uploadFileFileName));

return "success";

}

}

## OGNL表达式和JSTL、EL的区别

在开头引入了java类<%@ page import="java.util.\*" %> 在<% %>中可加入了大段的java代码，还有<%=user.getUserName() %> 获取后台的变量值。

在HTML中加入Java代码可以让页面动起来了。但是这样会有一些问题存在，比如这样分工不够明确，页面是用来显示的，可现在需要插入大段JAVA代码，这也是最初我们使用model1模式的做法。耦合性大，不够灵活，项目变大了以后会有很多麻烦，所以逐渐发展到model2模式，也就是MVC模式---把业务逻辑写在Controller层，页面显示写在View层。采用MVC模式后，我们尽量做到VC的分离，但是即使这样，我们还是需要动态往HTML中绑定值，尤其遇到页面复杂的情况，这时el和jstl就应运而生了。el可以解决<%=user.getUserName() %>这类问题。而jstl可以解决<%@ page import="java.util.\*" %> ，<% %>这类问题。

**标签和表达式不一样，经常要配合使用**

**标签就是在web.xml中定义名称和类路径，页面处理一些逻辑关系。比如说jstl,struts1标签，struts2标签，自定义标签**

**表达式就是输出值的作用，做一些简单的运算和判断。如EL表达式，ognl表达式。**

**1 OGNL: 对象图导航语言**

OGNL: user.name在xml中

<result type="redirect">view.jsp?id=${id}</result>

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

<s:property value="#request.request\_data"/>

<s:property value="#session.session\_data"/>

<s:property value="#application.application\_data"/><br/>

访问对象

<s property value="user.username"/>

访问对象中的一般属性

<s property value="user.address.addr"/>

访问对象的静态属性

<s property value="@com.czj.vo.Address@YIPS"/>

访问Aaction静态方法

<s property value="@com.czj.action.LoginAction@get()"/>

访问值栈对象的方法

<s property value="user.get()"/>

访问JDK静态方法

<s property value="@java.lang.Math@float(22.4)"/>或<s property value="@@float(22.4)"/>

不写类自动会从lang包中找到

访问JDK静态方法

<s property value="@java.util.Canlendar@getInstance().getYear()"/>

访问普通类构造方法

<s property value="new com.czj.vo.Address('wade,'stu',24).name"/>

访问lList <s property value="nameList.size"/>

访问List某一个值 <s property value="nameList[0]"/> 类似于数组的下标形式

访问Set <s property value="nameSet"/>

访问Set某一个值 <s property value="nameList[0]"/>

访问Map <s property value="nameMap"/>

访问Map某一个值 <s property value="nameMap['m1']"/>

访问数组 <s property value="nameArray"/>

访问数组某一个值 <s property value="nameArray[4]"/>

**2 JSTL （ JSP Standard Tag Library ），也就是JSP标准标签库。**

要循环显示后台list中的值绑定到前台table中。那我们就封装一个通用的标签,这个标签的命名可以随意，但是要参照<HTML>里面的格式，<> ,那我们可以命名为<s:iterator> <c:forEach> 等等

1.核心标签库

a.JSTL表达式标签

b.条件标签

c.循环标签

d.URL操作标签

2.格式标签库(处理国际化，数字和日期格式)

3.SQL标签库(可以访问数据库)

4.XML标签库(可操作XML标记)

5.函数标签库(主要是字符串操作函数)

**3 EL表达式:**

1.基本格式：${}

2.4种取值范围：pageScope，requestScope，sessionScope，applicationScope

${requestScope.user.name}

3.基本运算：.和[] .用于取属性[]用于取Array，List，Map，Set

${requestScope.map['apple']},${requestScope.list[1]}

4.{}里允许的运算：算术运算，关系运算，逻辑运算，empty/not empty空判断

${1+1}

${user.age<18}

${(user.age>18)&&(user.sex=='男')}

${empty user}判断user对象是否为空

5.EL的11个隐含对象

pageContext,param和paramValues,header和headerValues，cookie,initParam和4种取值范围对象

\*pageContext可获取jsp的request,response,out,session,config,servletContext等对象

${pageContext.session.uesr}

header

${header.host}

\*cookie

${cookie.key}

## 十、Struts1和Struts2的对比

**@ 类:**

Struts1要求Action类继承一个抽象基类。Struts1的一个普遍问题是使用抽象类编程而不是接口。

Struts 2 Action类可以实现一个Action接口，也可实现其他接口，使可选和定制的服务成为可能。Struts2提供一个ActionSupport基类去实现常用的接口。Action接口不是必须 的，任何有execute标识的POJO对象都可以用作Struts2的Action对象。

**@ 线程模式:**

Struts1 Action是单例模式并且必须是线程安全的，因为仅有Action的一个实例来处理所有的请求。单例策略限制了Struts1 Action能作的事，并且要在开发时特别小心。 Action资源必须是线程安全的或同步的。

Struts2 Action对象为每一个请求产生一个实例，因此没有线程安全问题。（实际上，servlet容器给每个请求产生许多可丢弃的对象，并且不会导致性能和垃圾回收问题）

**Servlet 依赖:**

Struts1 Action 依赖于Servlet API ,因为当一个Action被调用时HttpServletRequest 和 HttpServletResponse 被传递给execute方法。

Struts 2 Action不依赖于容器，允许Action脱离容器单独被测试。如果需要，Struts2 Action仍然可以访问初始的request和response。但是，其他的元素减少或者消除了直接 访问HttpServetRequest 和 HttpServletResponse的必要性。

**@ 可测性:**

测试Struts1 Action的一个主要问题是execute**方法暴露了servlet API（这使得测试要依赖于容器）**。一个第三方扩展－－Struts TestCase－－提供了一套Struts1的模拟对象（来进行测试）。

Struts 2 Action可以通过初始化、设置属性、调用方法来测试，“依赖注入”支持也使测试更容易。

**@ 捕获输入:**

Struts1 使用ActionForm对象捕获输入。所有的ActionForm必须继承一个基类。因为其他JavaBean不能用作ActionForm，开发者经常创建多余的类捕获输入。动态Bean（DynaBeans）可以作为创建传统ActionForm的选择，但是，开发者可能是在重新描述(创建)已经存在的JavaBean（仍然会导致有冗余的javabean）。

Struts 2直接使用Action属性作为输入属性，消除了对第二个输入对象的需求。输入属性可能是有自己(子)属性的rich对象类型。Action属性能够通过 web页面上的taglibs访问。Struts2也支持ActionForm模式。rich对象类型，包括业务对象，能够用作输入/输出对象。这种 ModelDriven 特性简化了taglib对POJO输入对象的引用。

**@ 表达式语言：**

Struts1 整合了JSTL，因此使用JSTL EL。这种EL有基本对象图遍历，但是对集合和索引属性的支持很弱。

Struts2可以使用JSTL，但是也支持一个更强大和灵活的表达式语言－－"Object Graph Notation Language" (OGNL).

**@ 绑定值到页面（view）:**

Struts 1使用标准JSP机制把对象绑定到页面中来访问。

Struts 2 使用 "ValueStack"技术，使taglib能够访问值而不需要把你的页面（view）和对象绑定起来。ValueStack策略允许通过一系列名称相同但类型不同的属性重用页面（view）。

**@ 类型转换：**

Struts 1 ActionForm 属性通常都是String类型。Struts1使用Commons-Beanutils进行类型转换。每个类一个转换器，对每一个实例来说是不可配置的。

Struts2 使用OGNL进行类型转换。提供基本和常用对象的转换器。

**@ 校验：**

Struts 1支持在ActionForm的validate方法中手动校验，或者通过Commons Validator的扩展来校验。同一个类可以有不同的校验内容，但不能校验子对象。

Struts2支持通过validate方法和XWork校验框架来进行校验。XWork校验框架使用为属性类类型定义的校验和内容校验，来支持chain校验子属性

**@ Action执行的控制：**

Struts1支持每一个模块有单独的Request Processors（生命周期），但是模块中的所有Action必须共享相同的生命周期。

Struts2支持通过拦截器堆栈（Interceptor Stacks）为每一个Action创建不同的生命周期。堆栈能够根据需要和不同的Action一起使用。