**Struts2拦截器（重点）**

**1.拦截器概念：**

**什么是struts2拦截器：拦截器就是一个类，他能够拦截action的请求，并进行预处理。**

**Struts2拦截器在访问某个Action方法之前或之后实施拦截（在action之前调用称之为前置拦截，之后成为后置拦截）**

**拦截器是可插拔的，是一种AOP实现（AOP String 面向切面编程）**

**----------AOP理解为：代理思想，使用代理模式**

配置文件中的自动提示功能需要在窗口----》首选项----中进行设置。导入文件路径：

D:\eclipse\struts-2.3.34\src\xwork-core\src\main\resources

Action1

拦截器简单示意图： ac

记录日志

登录

拦截器

Action2

记录日志

下订单

加入现在有登录和下订单两项请求，都访问action。在action中都需要进行记录日志。那么此时就可以利用一个拦截器来完成记录日志。

**范例：自定义一个拦截器并使用**

自定义拦截器类：

**package** wp.demo\_03;

**import** com.opensymphony.xwork2.ActionInvocation;

**import** com.opensymphony.xwork2.interceptor.Interceptor;

**public** **class** MyInterceptor **implements** Interceptor{

//自定义拦截器

@Override

**public** **void** destroy() {}

@Override

**public** **void** init() {}

@Override

**public** String intercept(ActionInvocation invocation) **throws** Exception {

//调用action中的execute方法,返回的是执行execute方法的字符串

System.***out***.println("interceptor begin");

String result = invocation.invoke() ;

System.***out***.println(result);

System.***out***.println("interceptor end");

**return** result;

}

}

**package** wp.demo\_03;

**import** com.opensymphony.xwork2.ActionSupport;

**public** **class** MyAction **extends** ActionSupport{

@Override

**public** String execute() **throws** Exception {

System.***out***.println("执行了execute()方法");

**return** ***NONE***;

}

}

Struts.xml配置文件：

<!-- 配置拦截器 -->

<!-- 注册自定义的拦截器 -->

<interceptors>

<interceptor name = *"myCeptor"* class =*"wp.demo\_03.MyInterceptor"*></interceptor>

</interceptors>

<action name = *"myAction"* class = *"wp.demo\_03.MyAction"*>

<!-- 使用拦截器 -->

<interceptor-ref name=*"myCeptor"*></interceptor-ref>

</action>

注意该配置文件必须放在所有的action节点前。<interceptor-ref name=*"****myCeptor****"*></interceptor-ref>表示关联的拦截器。<interceptor name = *"****myCeptor****"* class =*"wp.demo\_03.MyInterceptor"*></interceptor>表示在xml文件中注册一个拦截器。

**2.拦截器运行原理：**

1. **自定义登录拦截器（重要）：**

**自己定义一个登录的拦截器，当用户进行正常登录后再进行书籍信息添加时，可正常操作。当用户没有登录就进行书籍信息添加时则会拦截书籍信息添加，并跳转到登录页面，提示用户登录。**

**范例：自定义拦截器：**

**登录界面：**

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

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

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>登录页面</title>

</head>

<body>

<s:fielderror/>

<form action=*"*${pageContext.request.contextPath }*/interceptor/user\_login.action"* method = *"post"*>

username:<input type = *"text"* name = *"username"*/><br></br>

password:<input type = *"password"* name = *"pwd"*/><br></br>

<input type = *"submit"* value = *"登录"*></input>

</form>

</body>

</html>

**书本信息添加界面：**

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>书籍信息添加页面</title>

</head>

<body>

<h1>书籍信息添加页面！</h1>

<form action=*"*${pageContext.request.contextPath }*/interceptor/book\_add.action"*>

bookname:<input type = *"text"* name = *"bookname"*></input><br></br>

p&nbsp;r&nbsp;i&nbsp;c&nbsp;e:<input type = *"text"* name = *"price"*></input><br></br>

<input type = *"submit"* value = *"提交"*/>

</form>

</body>

</html>

**书籍添加成功界面：**

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>书籍信息添加成功页面</title>

</head>

<body>

<h1>this book is add success</h1>

<h1>书籍信息添加成功了哦！</h1>

</body>

</html>

**登录action：**

package wp.demo\_01;

import org.apache.struts2.ServletActionContext;

import com.opensymphony.xwork2.ActionSupport;

import com.opensymphony.xwork2.ModelDriven;

import wp.demo\_01\_model.User;

/\*\*

**\* 要使用模型驱动接收属性，必须要实现ModelDriveren接口**

**\* 并且要手动创建接收参数的对象。**

\* \*/

public class UserAction extends ActionSupport implements **ModelDriven<User>**{

**private User user = new User() ;**

public String login() throws Exception {

//用户名为admin密码为123456

**if(user.getUsername().equals("admin") && user.getPwd().equals("123456")){**

//把用户信息放入session,即将提交过来的用户信息保存到user中以便于添加书信息页面的登录检验

**ServletActionContext.getRequest().getSession().setAttribute("user", "user");**

return SUCCESS ;

}else{

**addFieldError("user\_error", "用户名或密码不正确！");**

return INPUT ;

}

}

@Override

public User getModel() {

return user;

}

}

**书籍添加action：**

package wp.demo\_01;

import com.opensymphony.xwork2.ActionSupport;

import com.opensymphony.xwork2.ModelDriven;

import wp.demo\_01\_model.Book;

public class BookAction extends ActionSupport implements ModelDriven<Book>{

**private Book book = new Book(); //使用模型驱动接收参数必须要手动创建对象**

public String add() throws Exception { //添加书籍信息

System.out.println(book.toString()) ;

return SUCCESS ;

}

@Override

public Book getModel() {

return book ;

}

}

**书本信息添加拦截器：**

package wp.demo\_01;

import org.apache.struts2.ServletActionContext;

import com.opensymphony.xwork2.ActionInvocation;

import com.opensymphony.xwork2.ActionSupport;

import com.opensymphony.xwork2.interceptor.MethodFilterInterceptor;

import com.sun.corba.se.spi.orbutil.fsm.Action;

import wp.demo\_01\_model.User;

public class MyBookInterceptor extends **MethodFilterInterceptor**{

@Override

protected String doIntercept(ActionInvocation invocation) throws Exception {

//判断session中是否有用户存在。若有用户则可以进行书本信息添加。若不存在则不能添加

**User user = (User)ServletActionContext.getRequest().getSession().getAttribute("user") ;**

if(user == null){

//未登录 要添加提示信息

**ActionSupport action = (ActionSupport)invocation.getAction() ;**  //取得action

action.addFieldError("login\_error", "请登录后在进行添加信息操作！") ;

return "no\_login" ; //**需要在配置文件中的result节点中配置**

}else{

//登录了。执行BookAction

**return invocation.invoke() ;**

}

}

}

**配置文件struts.xml：**

<!-- 配置拦截器 -->

<package name=*"myInterceptor"* extends = *"struts-default"* namespace = *"/interceptor"*>

<!-- 注册自定义的拦截器 -->

<interceptors>

**<interceptor name=*"bookCeptor"* class=*"wp.demo\_01.MyBookInterceptor"*>**

<!-- 登录方法放行？

excludeMethods:放行的方法

includeMethods:拦截的方法

-->

<!-- 告诉自定义的拦截器login方法不需要拦截，需要放行 -->

**<param name=*"excludeMethods"*>login</param>**

</interceptor>

<!-- 如果要用框架和自定义的拦截器，我们可以定义一个拦截器栈 -->

**<interceptor-stack name=*"myStack"*>**

**<interceptor-ref name = *"bookCeptor"*></interceptor-ref>**

**<interceptor-ref name=*"defaultStack"*></interceptor-ref>**

**</interceptor-stack>**

</interceptors>

<!-- 告诉系统使用自定义的拦截器栈 -->

**<default-interceptor-ref name=*"myStack"*></default-interceptor-ref>**

<!-- 定义一个全局结果集来处理未登录情况 -->

**<global-results>**

**<result name = *"no\_login"*>/a\_interceptor/login.jsp</result>**

**</global-results>**

<!-- 配置登录的action -->

**<action name=*"user\_\*"* class = *"wp.demo\_01.UserAction"* method = *"{1}"*>**

<result>/a\_interceptor/addbook.jsp</result>

<result name = *"input"*>/a\_interceptor/login.jsp</result>

</action>

<!-- 配置book的action -->

**<action name=*"book\_\*"* class = *"wp.demo\_01.BookAction"* method = *"{1}"*>**

<result>/a\_interceptor/addbooksuccess.jsp</result>

</action>

</package>

在以上程序中，用户进行登录，然后跳转到“书籍信息添加”页面。输入要添加的书籍信息，点击提交按钮，此时会对调用自定义拦截器的doIntercept()方法，判断用户是否已经登录，若用户没有进行登录，则会对BookAction中的add()方法进行拦截。并且保存未登录的错误提示然后跳转到登录页面，并且输出未登录的错误提示信息。如果用户已经进行了登录，则通过ActionInvocation调用invoke方法进行执行BookAction类，在后台输出书籍信息。

**Struts2的文件的上传：**

1. **回顾文件上传：**

客户端：

<form>设置enctype编码类型(MIME类型)：multipart/form-data

<form>设置method提交方式post（只能是post不能是get）

<input type = “file” name = “”>必须提供type和name属性。

服务器端：

Apache-commons-fileupload组件。

Jsp-smartupload组件

Servlet3.0以后API内置文件上传API

cos文件上传组件。

最好是使用前两种。struts2默认导入文件上传jar包。

**2.struts2的文件上传机制（用的fileupload）**

Sturts2内部文件上传，默认采用Apache-commons-fileupload，struts2默认导入文件上传jar包。

在struts2中要实现文件上传是非常简单的。只需要三个参数（文件。文件名。文件类型）。

范例：struts2实现文件上传

**文件上传页面：**

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

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

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>struts2文件上传</title>

</head>

<body>

<s:fielderror></s:fielderror>

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

<input type = *"file"* name = *"upload"*></input><br></br>

<input type = *"submit"* value =*"提交"*/>

</body>

</html>

**文件上传的action：**

**package** wp.demo\_02;

**import** java.io.File;

**import** org.apache.commons.io.FileUtils;

**import** com.opensymphony.xwork2.ActionSupport;

**public** **class** UploadAction **extends** ActionSupport{

/\*\*

\* 需要三个参数（要上传的文件，文件名，文件类型）

\* \*/

**private File upload ; //要上传的文件**

**private String uploadContentType ; //文件类型**

**private String uploadFileName ; //文件名**

**public** File getUpload() {

**return** upload;

}

**public** **void** **setUpload**(File upload) {

**this**.upload = upload;

}

**public** String getUploadContentType() {

**return** uploadContentType;

}

**public** **void** **setUploadContentType**(String uploadContentType) {

**this**.uploadContentType = uploadContentType;

}

**public** String getUploadFileName() {

**return** uploadFileName;

}

**public** **void** **setUploadFileName**(String uploadFileName) {

**this**.uploadFileName = uploadFileName;

}

@Override

**public** String execute() **throws** Exception {

//一句话搞定文件上传 一个参数是要上传的文件，另一个参数是文件的保存路径 FileUtils.*copyFile*(upload, **new** File("d://" + uploadFileName)) ;

**return** ***SUCCESS***;

}

}

在以上程序中：通过表单的提交，将上传的文件自动设置到upload中。然后通过执行execute()方法进行文件的保存（上传）。在action中必须要写三个参数且三个setter方法的命名一定**是setUpload(),setUploadContentType(),setUploadFileName();**

**文件上传参数的设置：**

1. 可以设置允许上传文件的类型。
2. 可以设置允许上传文件的最大值。

**方法1**：<!--通过配置常量来限制上传文件的大小(第一种限制上传文件大小的方法) -->

<constant name = *"struts.multipart.maxSize"* value = *"1000000"*></constant>

**方法2：**<action name=*"upload\_demo"* class =*"wp.demo\_02.UploadAction"*>

<result>/b\_upload/upload\_01.jsp</result>

<result name = *"input"*>/b\_upload/upload\_01.jsp</result>

<!-- 配置文件上传参数 -->

<interceptor-ref name=*"defaultStack"*>

<!-- 限制上传文件后缀 -->

<param name=*"fileUpload.allowedExtensions"*>.txt</param>

<!-- 第二种限制上传文件大小的方式 2m -->

<param name=*"fileUpload.maximumSize"*>2097152</param>

</interceptor-ref>

</action>

1. 可以设置允许上传文件的后缀名。

<action name=*"upload\_demo"* class =*"wp.demo\_02.UploadAction"*>

<result>/b\_upload/upload\_01.jsp</result>

<result name = *"input"*>/b\_upload/upload\_01.jsp</result>

<!-- 配置文件上传参数 -->

**<interceptor-ref name=*"defaultStack"*>**

**<param name=*"fileUpload.allowedExtensions"*>.txt</param>**

**</interceptor-ref>**

</action>

**OGNL表达式：**

1. **OGNL表达式是什么？**

**OGNL是Object Graphic Navigation Language(对象图导航语言)的编写，是一个使用简单功能强大的开源的表达式语言，可以方便的操作任何对象属性，方法等。**

**Struts2框架使用OGNL作为默认的表达式语言，主要用于页面的取值。他类似于EL表达式，但比EL表达式强大许多。**

**EL表达式语言，主要用来获取jsp页面四个域范围数据（page ,request,session.application）.**

**OGNL = EL + 对象操作。**

**OGNL表达式的三个要素：**

1. **表达式 --------OGNL的表达式，编写表达式**
2. **Root对象 --------根对象**
3. **Context对象 --------ognl的上下文对象，运行的环境。**

**OGNL主要是用来设置值课取值的**

1. 利用OGNL来操作对象中的方法。

@Test

**public** **void** testOGNL\_01() **throws** OgnlException{

//创建OgnlContext对象

//可以通过OGNL来操作对象的方法 第一个参数为要操作的表达式

OgnlContext context = **new** OgnlContext() ;

**Object obj = Ognl.*getValue*("'helloworld'.length()", context, context.getRoot()) ;**

System.***out***.println(obj);

}

在以上程序中，首先通过OgnlContext 来创建一个对象context。然后利用Ognl.getValue(expression,context,root).方法来操作表达式中的内容。直接 对象.方法().

1. 利用OGNL来操作通过OgnlContext.put(key,value)方法保存的值。

@Test

**public** **void** testOGNL\_02() **throws** OgnlException{

//创建OgnlContext对象

//可以通过OGNL来操作对象的方法

OgnlContext context = **new** OgnlContext() ;

**context.put("name", "文攀") ;**

//通过context.put(key,value)方式保存的值要利用#key来获取value值

**Object obj = Ognl.*getValue*("#name",context,context.getRoot()) ;**

System.***out***.println(obj) ;

}

在以上程序中通过键值对的形式将一个属性保存到name中，然后取出的时候要在属性前面加一个#。eg:#name.

1. 利用OGNL来操作通过OgnlContext.setRoot(object obj)方法保存的对象中的属性值。

@Test

**public** **void** testOGNL\_03() **throws** OgnlException{

//创建OgnlContext对象

//可以通过OGNL来操作对象的方法

OgnlContext context = **new** OgnlContext() ;

Custorm custorm = **new** Custorm() ;

custorm.setCust\_name("文攀") ;

custorm.setAge(20) ;

//将客户放入一个root栈

context.setRoot(custorm) ;

//取得root栈中的对象的属性

**Object obj = Ognl.*getValue*("age",context,context.getRoot()) ;**

System.***out***.println(obj) ;

}

在以上程序中，是通过OnglContext的setRoot()方法将custom对象保存到root栈中。要取出该对象的属性内容时，只需通过属性名即可取出：eg:**Object obj = Ognl.*getValue*("age",context,context.getRoot()) ;**

**OGNL表达式详解：**

1. **java对象直接访问：**

例如：<s:property value = “emp”/>

1. **实例方法调用访问对象的方法**

语法：object.dosomething()

例如：<s:property value = “’hello’.toUpperCase()”/>

1. **静态方法调用（类的静态方法或静态变量的访问）**

语法：@包名+类名+@方法名()

**注意：必须要配置struts2的常量来开启静态方法的调用功能**

<!-- 开启静态方法常量 -->

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

范例：三种使用方式

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

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

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"* />

<title>ognl表达式测试</title>

</head>

<body>

<h3>访问java对象</h3>

<s:property value = *"'wenpan nihao '"*/>

<h3>操作对象属性</h3>

<s:property value = *"'wenpan'.toUpperCase()"*/>

<h3>静态方法的访问</h3>

<h3>语法：@包名+类名+@方法名()</h3>

<s:property value = *["@java.lang.Math@max(20,50)"](mailto:\"@java.lang.Math@max(20,50)\"/>)*[/>](mailto:\"@java.lang.Math@max(20,50)\"/>)

<h3>类似EL表达式的使用过</h3>

<s:property value = *"1+2"*/>

</body>

</html>

实际上OGNL表达式只是比EL表达式功能更加强大，可以操作对象。