一、输入验证

> 对用户输入的内容进行合法性校验

用户输入验证：开发中工作量极大，产出比较低。

客户端验证：用js,容易被篡改

服务器端验证：无论何时都必须写。

实际开发：客户端验证+服务器端验证

以下内容：属于服务器端验证

1.1、编程式验证

编写验证代码:

缺点：验证规则写到了代码中，硬编码。

优点：验证时可以控制的更加精细。（用的少）

1.1.1、针对动作类中的所有动作方法进行验证

1.1.1.1、动作类需要实现ActionSupport,覆盖掉public void validate(){}方法

1.1.1.2、方法内部：编写你的验证规则，不正确的情况调用addFieldError添加错误信息

```
/**
 * 方式一  对所有的方法进行验证
 */
    @Override
    public void validate() {
        String re ="[a-zA-Z]{5,20}";
        boolean matches = student.getName().matches(re);
        if (matches) {

        }else{
            //代表不符合规范
            //跳回验证页面,name要和表单中的 name 一致
            addFieldError("name", "名字必须是5-20的大小写字母");//添加一个错误
        }
        super.validate();
    }
```

1.1.1.3、验证失败：

视图：会自动转向一个name=input的逻辑视图

错误消息提示：建议使用struts2标签库。如果没有显示请使用s:fieldError标签

1.1.2、针对动作类中的指定动作方法进行验证

方式一：简单。使用一个注解。

```
    @SkipValidation//跳过验证  设置过这个注解后 方法就不在经过validate验证
    public String  find(){
        //不需要登陆
        System.err.println("find执行了");
        return SUCCESS;
    }
```

方式二：

如果一个动作方法名叫做save

只针对该方法进行验证，请编写public void validateSave(){}

```
public String save(){
        System.err.println("dEMO3save执行了");
        return SUCCESS;
    }

    public String  find(){
        System.err.println("dEMO3find执行了");
        return SUCCESS;
    }
/**
 * 方法的格式 validate+要验证的方法名
 */
    public void validateSave(){
        System.err.println("验证 save 了");
        Object object =
ServletActionContext.getRequest().getSession().getAttribute("user");
        if (object==null) {//等于空代表没有登陆 ,需要跳转到登陆页面
            addFieldError("name", "需要登陆");//添加错误信息，它其实是向一个 map 中存
放了数据
        }
    }
```

2、声明式验证

优点：把验证规则写到了配置文件中。（用得多）

缺点：不是很精细。

错误视图和消息提示和编程式一致。

2.1 针对动作类中的所有动作方法进行验证

在动作类所在的包中建立：动作类名-validation.xml配置文件。

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE validators PUBLIC
        "-//Apache Struts//XWork Validator 1.0.3//EN"
        "http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">
    <validators>
        <!--
            name 代表要给哪个属性做验证,要和表单的name 保持一致
          -->
         <field name="name">
            <!--
                验证规则  struts 提供了一些内置的验证规则
                required 必须填写,有时候不生效
                requiredstring 生效的 必须填写内容,不管具体内容是什么 -->

                <field-validator type="requiredstring">

            <!--    message 如果验证失败,提示什么信息 -->

                    <message>必须填写name</message>
                </field-validator>
        </field>
        <field name="name">
        <!--
            注意 什么都不写,这个验证器是不管的,必须得写一点内容才有效
            如果要验证必须填写,那么得增加一个验证规则 叫requiredstring
         -->
            <field-validator type="stringlength">
                <param name="minLength">5</param>
                <param name="maxLength">20</param>
                <message>长度必须在5-20之间</message>
            </field-validator>
        </field>

        <field name="name">

        <field-validator type="regex">
        <!--
        正则表达式 需要些在 CDATA 区里面
         -->
            <param name="regex"><![CDATA[[a-zA-Z]]]></param>
```

```
            <message>必须是大小写字母</message>
        </field-validator>

        </field>


    </validators>
```

2.2 针对动作类中的指定动作方法进行验证

在动作类所在的包中建立：动作类名-动作名（是请求的struts.xml配置文件中的）-validation.xml配置文件。

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE validators PUBLIC
        "-//Apache Struts//XWork Validator 1.0.3//EN"
        "http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">
    <validators>
    <!--
    此文件的写法
        动作类名-action名-validation
        注意中间不是方法名是 action名
        注意,这里验证的内容不止是要在表单中存在,还需要在对象中存在这些属性,提供 get 和
set 方法
        这里的 name 必须和表单中 name 一致
    -->
    <field name="email">
        <!--
        如果不写任何内容,不会验证,必须得写内容才会生效
          -->
            <field-validator type="requiredstring">
                <message>必须填写邮箱</message>
            </field-validator>
        </field>


        <field name="email">
        <!--
        如果不写任何内容,不会验证,必须得写内容才会生效
          -->
            <field-validator type="email">
                <message>邮箱格式不对,起来重睡</message>
            </field-validator>
        </field>


    <field name="age">
        <field-validator type="int">
```

```
            <param name="min">10</param>
            <param name="max">10000</param>
            <message>年龄问题</message>
        </field-validator>

    </field>

    </validators>
```

说明：验证功能是由validation拦截器来负责处理的。回显错误信息是由workflow拦截器来负责处理的。

3、Struts2中提供的内置声明式验证器的使用

Struts2提供的声明式验证器在xwork-core-**.jar包的

com\opensymphony\xwork2\validator\validators\default.xml配置文件中。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC
        "-//Apache Struts//XWork Validator Definition 1.0//EN"
        "http://struts.apache.org/dtds/xwork-validator-definition-1.0.dtd">

<!-- START SNIPPET: validators-default -->
<validators>
    <validator name="required"
class="com.opensymphony.xwork2.validator.validators.RequiredFieldValidator"/>
    <validator name="requiredstring"
class="com.opensymphony.xwork2.validator.validators.RequiredStringValidator"/>
    <validator name="int"
class="com.opensymphony.xwork2.validator.validators.IntRangeFieldValidator"/>
    <validator name="long"
class="com.opensymphony.xwork2.validator.validators.LongRangeFieldValidator"/>
    <validator name="short"
class="com.opensymphony.xwork2.validator.validators.ShortRangeFieldValidator"/>
    <validator name="double"
class="com.opensymphony.xwork2.validator.validators.DoubleRangeFieldValidator"/>
    <validator name="date"
class="com.opensymphony.xwork2.validator.validators.DateRangeFieldValidator"/>
    <validator name="expression"
class="com.opensymphony.xwork2.validator.validators.ExpressionValidator"/>
    <validator name="fieldexpression"
class="com.opensymphony.xwork2.validator.validators.FieldExpressionValidator"/>
    <validator name="email"
class="com.opensymphony.xwork2.validator.validators.EmailValidator"/>
    <validator name="url"
class="com.opensymphony.xwork2.validator.validators.URLValidator"/>
```

```
    <validator name="visitor"
class="com.opensymphony.xwork2.validator.validators.VisitorFieldValidator"/>
    <validator name="conversion"
class="com.opensymphony.xwork2.validator.validators.ConversionErrorFieldValidator"
/>
    <validator name="stringlength"
class="com.opensymphony.xwork2.validator.validators.StringLengthFieldValidator"/>
    <validator name="regex"
class="com.opensymphony.xwork2.validator.validators.RegexFieldValidator"/>
    <validator name="conditionalvisitor"
class="com.opensymphony.xwork2.validator.validators.ConditionalVisitorFieldValidat
or"/>
</validators>
<!--  END SNIPPET: validators-default -->
```

4、自定义声明式验证(了解怎么运作的)

4.1编写一个类，继承FieldValidatorSupport

```
public class MyFieldValidatorSupport extends FieldValidatorSupport {
    private  String[] UpCase =
{"A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","
U","V","W","X","Y","Z"};
    private String[] LowCase =
{"a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","
u","v","w","x","y","z"};
    private String[] Number ={"1","2","3","4","5","6","7","8","9","0"};


    /**
 * 验证的方法
 * 在这里对传递过来的数据进行验证
 *  object 当前的 action 动作类
 */
    @Override
    public void validate(Object object) throws ValidationException {
            boolean isHaveUp=false;
            boolean isHaveLow=false;
            boolean isHaveNumber=false;


            //获取到当前验证的字段
        //获取到当前被验证的内容(传递过来的内容)
          String fieldName = getFieldName();//获取到验证的属性名
            Object value = this.getFieldValue(fieldName, object);//获取到当前属性传
递过来的值
```

```
        //判断内容是否符合规范
            if (!(value instanceof String)) {
                addFieldError(fieldName, object);
            } else {
               //是字符串,判断是佛符合我们的规则
                //密码必须是大小写加数字
                for (String string : UpCase) {
                    if (((String)value).contains(string)) {
                        isHaveUp=true;
                        break;
                    }

                }
                for (String string : LowCase) {
                    if (((String)value).contains(string)) {
                        isHaveLow=true;
                        break;
                    }
                }
                for (String string : Number) {
                    if (((String)value).contains(string)) {
                        isHaveNumber=true;
                        break;
                    }
                }
                //加到错误提示中
                if (!(isHaveNumber&&isHaveLow&&isHaveUp)) {//不包含大小写
                     addFieldError(fieldName, value);//告诉它出错了
                }

            }
    }

}
```

4.2、声明自定义的验证器

在构建路径src顶端，建立一个固定名称为validators.xml的配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC
        "-//Apache Struts//XWork Validator Definition 1.0//EN"
        "http://struts.apache.org/dtds/xwork-validator-definition-1.0.dtd">
        <!--
        声明了一个自定义的验证器
          -->
     <validators>
      <validator name="strongpassword"
class="com.qianfeng.struts.validator.MyFieldValidatorSupport"/>
     </validators>
```

4.3 使用验证器

> 可以像使用struts2内置的声明式验证器使用自己定义的验证器

二、Struts2的国际化

1、配置全局消息资源文件

1.1配置

1.1.1 在 struts.xml 中配置国际化资源文件的前缀

```
<struts>
  <!--指定国际化资源文件前缀为 msg-->
<constant name="struts.custom.i18n.resources" value="msg"></constant>
    <package name="p1" extends="struts-default" namespace="/student">

        <action name="*" class="com.qianfeng.struts.action.ActionDemo1">
            <result>/success.jsp</result>
            <result name="input">/index.jsp</result>
        </action>

    </package>


</struts>
```

1.1.2 编写国际化文件

> 在 src 目录下编写 前缀(基名)_语言_区域.properties文件 比如如果前缀是 msg 英文是 msg_en_us.properties

> 中文是 msg_zh_CN.properties在内部将需要国际化的内容以 key=value 的方式填写好

```
name=张三
```

```
name=zhangsan
```

1.1.3使用

1.1.3.1在动作类中使用

前提：必须经过动作方法。动作类要继承ActionSupport

```
public class ActionDemo1 extends ActionSupport {

        @Override
        public String execute() throws Exception {
        //获取值,根据不同的情况决定获取到什么语言的值
            String string = getText("name");//通过配置文件获取 name 的值
            System.err.println(string);
            return super.execute();
        }

}
```

1.1.3.2在jsp页面上使用

```
<s:text name="name" />
```

2、配置局部消息资源文件

1、消息资源文件：动作类名_zh_CN.properties

2、访问：

a、在JSP上使用s:text/，直接访问的JSP，显示全局消息资源文件的内容。因为没有经过动作类。

b、通过访问动作，转发到JSP，那么显示局部的。

c、通过动作访问，转发到JSP，如果动作类没有继承ActionSupport，那么会显示全局的。

资源查找顺序:

1. 动作类. properties
2. 父接口名. properties .如果没有则一直向上,直到找到最顶级的父接口
3. 如果实现了 ModelDriven,则查找模型. properties
4. 找当前类所在的包以及父包直到 src 目录下的名字为 package.properties 的文件
5. 查找对应基名的 properties
6. 全局

> Resource bundles are searched in the following order:
>
> 1. ActionClass.properties
> 2. Interface.properties (every interface and sub-interface)
> 3. BaseClass.properties (all the way to Object.properties)
> 4. ModelDriven's model (if implements ModelDriven), for the model object repeat from 1
> 5. package.properties (of the directory where class is located and every parent directory all the way to the root directory)
> 6. search up the i18n message key hierarchy itself
> 7. global resource properties

3、自由指定资源包

```
<s:i18n name="com.qianfeng.struts.package">
    <body>
  <!--默认找的是全局的  -->
      <s:i18n name="msg">
        <s:text name="name" />
      </s:i18n>s:i18n>
        <hr>
        <!--
        指定加载的国际化文件 com.qianfeng.struts下面的package.properties,如果指定包下没有,则向上找父包,直到 src
         -->
        <s:i18n name="com.qianfeng.struts.package">
            <s:text name="name"></s:text>
        </s:i18n>
        <hr>
        <s:i18n name="com/qianfeng/struts/package">
            <s:text name="name"></s:text>
        </s:i18n>
    </body>
```

三、Struts2中的拦截器

常用的拦截器

```
 <interceptor name="alias"
class="com.opensymphony.xwork2.interceptor.AliasInterceptor"/>
            <interceptor name="autowiring"
class="com.opensymphony.xwork2.spring.interceptor.ActionAutowiringInterceptor"/>
            <interceptor name="chain"
class="com.opensymphony.xwork2.interceptor.ChainingInterceptor"/>
```

```
            <interceptor name="conversionError"
class="org.apache.struts2.interceptor.StrutsConversionErrorInterceptor"/>
            <interceptor name="cookie"
class="org.apache.struts2.interceptor.CookieInterceptor"/>
            <interceptor name="cookieProvider"
class="org.apache.struts2.interceptor.CookieProviderInterceptor"/>
            <interceptor name="clearSession"
class="org.apache.struts2.interceptor.ClearSessionInterceptor" />
            <interceptor name="createSession"
class="org.apache.struts2.interceptor.CreateSessionInterceptor" />
            <interceptor name="debugging"
class="org.apache.struts2.interceptor.debugging.DebuggingInterceptor" />
            <interceptor name="execAndWait"
class="org.apache.struts2.interceptor.ExecuteAndWaitInterceptor"/>
            <interceptor name="exception"
class="com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor"/>
            <interceptor name="fileUpload"
class="org.apache.struts2.interceptor.FileUploadInterceptor"/>
            <interceptor name="i18n"
class="com.opensymphony.xwork2.interceptor.I18nInterceptor"/>
            <interceptor name="logger"
class="com.opensymphony.xwork2.interceptor.LoggingInterceptor"/>
            <interceptor name="modelDriven"
class="com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor"/>
            <interceptor name="scopedModelDriven"
class="com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor"/>
            <interceptor name="params"
class="com.opensymphony.xwork2.interceptor.ParametersInterceptor"/>
            <interceptor name="actionMappingParams"
class="org.apache.struts2.interceptor.ActionMappingParametersInteceptor"/>
            <interceptor name="prepare"
class="com.opensymphony.xwork2.interceptor.PrepareInterceptor"/>
            <interceptor name="staticParams"
class="com.opensymphony.xwork2.interceptor.StaticParametersInterceptor"/>
            <interceptor name="scope"
class="org.apache.struts2.interceptor.ScopeInterceptor"/>
            <interceptor name="servletConfig"
class="org.apache.struts2.interceptor.ServletConfigInterceptor"/>
            <interceptor name="timer"
class="com.opensymphony.xwork2.interceptor.TimerInterceptor"/>
            <interceptor name="token"
class="org.apache.struts2.interceptor.TokenInterceptor"/>
            <interceptor name="tokenSession"
class="org.apache.struts2.interceptor.TokenSessionStoreInterceptor"/>
            <interceptor name="validation"
class="org.apache.struts2.interceptor.validation.AnnotationValidationInterceptor"/
>
            <interceptor name="workflow"
class="com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor"/>
```

```
            <interceptor name="store"
class="org.apache.struts2.interceptor.MessageStoreInterceptor" />
            <interceptor name="checkbox"
class="org.apache.struts2.interceptor.CheckboxInterceptor" />
            <interceptor name="datetime"
class="org.apache.struts2.interceptor.DateTextFieldInterceptor" />
            <interceptor name="profiling"
class="org.apache.struts2.interceptor.ProfilingActivationInterceptor" />
            <interceptor name="roles"
class="org.apache.struts2.interceptor.RolesInterceptor" />
            <interceptor name="annotationWorkflow"
class="com.opensymphony.xwork2.interceptor.annotations.AnnotationWorkflowIntercept
or" />
            <interceptor name="multiselect"
class="org.apache.struts2.interceptor.MultiselectInterceptor" />
            <interceptor name="deprecation"
class="org.apache.struts2.interceptor.DeprecationInterceptor" />
```

Struts2中功能核心。是一种AOP编程思想的具体应用。

modelDriven：模型驱动

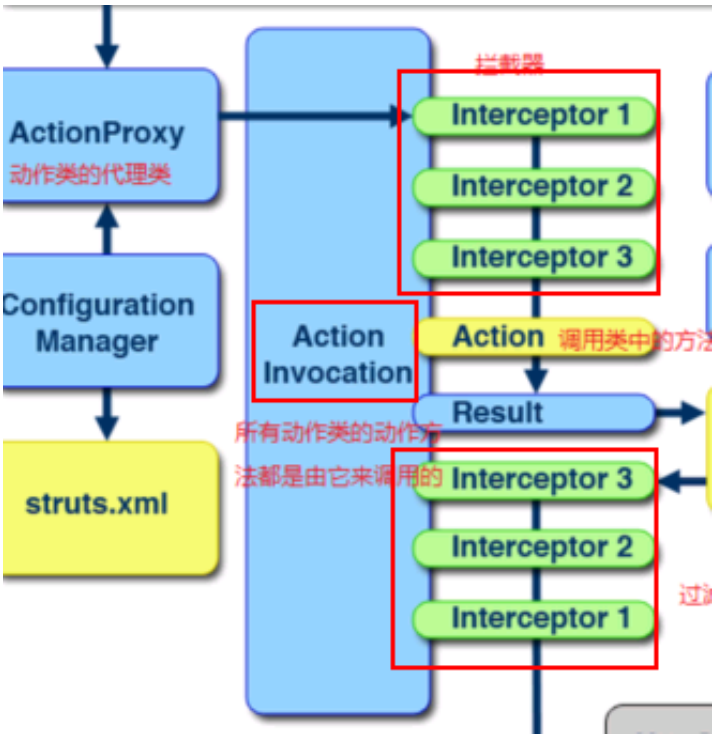
servletConfig：获取ServletAPI

staticParams：静态参数注入

params：动态参数注入

validation：输入验证，声明式验证。

等等



3.1、自定义拦截器

3.1.1编写一个类，继承AbstractInterceptor抽象类

```
public class MyInterceptor extends AbstractInterceptor {
/**
 * 定义 拦截器的功能
 *      用于判断用户是否登陆,没有登陆要求登陆,登陆了直接放行
 *
 * 统计成功数据
 *  统计错误收
 * 统计执行时间
 *
 * 返回值 Action 的返回值(可以中途拦截直接返回)
 * 参数  Action 的封装对象
 */
    @Override
    public String intercept(ActionInvocation invocation) throws Exception {
        // Object action = invocation.getAction();//获取到了 Action 对象
        //从 session 里面获取值,判断是否有值
            Object user =
ServletActionContext.getRequest().getSession().getAttribute("user");
            if (user==null) {//没有登陆
                //让它跳到登陆页面
                return "login"; //相当于咱们遇到转换异常,没有经过 action 直接到了结果
视图页面 ,会被当做返回结果直接进行返回
            }
            System.err.println(System.currentTimeMillis());
            System.err.println("执行之前");
            String invoke = invocation.invoke();//返回值,同步的方法,调用了第二个拦截
器的intercept方法
            System.err.println("继续写代码"+invoke);
            System.err.println(System.currentTimeMillis());
            //有很多个拦截器,拦截器有顺序
                // 1 2 3 4 5五个拦截器
            // 1拦截器调用 invocation.invoke() 其实是调用第2个拦截器的intercept方法
            //2 进行invocation.invoke()调用的是 3拦截器intercept
            // 直到 5调用invocation.invoke() 放心到 action  其实调用的是对应 action
方法

        return invoke;
    }
```

3.1.2配置拦截器

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>
<constant name="sturts.devMode" value="true"></constant>
<constant name="struts.custom.i18n.resources" value="msg"></constant>
    <package name="p1" extends="struts-default">
      <!--定义拦截器-->
        <interceptors>
        <interceptor name="myint"
class="com.qianfen.struts.interceptor.MyInterceptor"></interceptor>
        </interceptors>

    </package>


</struts>
```

3.1.3使用拦截器

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>
<constant name="sturts.devMode" value="true"></constant>
<constant name="struts.custom.i18n.resources" value="msg"></constant>
    <package name="p1" extends="struts-default">
        <!--定义拦截器-->
        <interceptors>
        <interceptor name="myint"
class="com.qianfen.struts.interceptor.MyInterceptor"></interceptor>
        </interceptors>

        <action name="test1" class="com.qianfen.struts.interceptor.ActionDemo1"
method="test1">
            <!--
            注意:一旦引用了自定义的拦截器,struts原始的默认拦截器就失效了
             -->
            <interceptor-ref name="myint"></interceptor-ref>
```

```
            <!-- 引入 struts 原始的拦截器-->
            <interceptor-ref name="defaultStack"></interceptor-ref>
            <result name="test1">/index.jsp</result>
            <result name="login">/login.jsp</result>
        </action>

    </package>


</struts>
```

3.1.4 多个拦截器

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>
<constant name="sturts.devMode" value="true"></constant>
<constant name="struts.custom.i18n.resources" value="msg"></constant>
    <package name="p1" extends="struts-default">
        <interceptors>
        <!--
        声明一个拦截器
         -->
        <interceptor name="myint1" class="指定 class"></interceptor>
        <interceptor name="myint2" class=""></interceptor>
        <interceptor name="myint3" class=""></interceptor>
        <interceptor name="myint4" class=""></interceptor>
        <interceptor name="myint5" class=""></interceptor>

        <!--
        定义一个拦截器的栈
         -->
        <interceptor-stack name="myintstack">
        <!--
        引入定义好的拦截器
        -->
        <interceptor-ref name="myint1"></interceptor-ref>
        <interceptor-ref name="myint2"></interceptor-ref>
        <interceptor-ref name="myint3"></interceptor-ref>
        <interceptor-ref name="myint4"></interceptor-ref>
        <interceptor-ref name="myint5"></interceptor-ref>
        <!--
        将struts 自带的defaultStack栈里面的拦截器引用过来
```

```
        -->
        <interceptor-ref name="defaultStack"></interceptor-ref>
        </interceptor-stack>
        </interceptors>

        <!--
            修改默认的拦截器, 修改成功后,当前包以及子包内的所有的 action 默认引用这个拦截器
          -->
    <default-interceptor-ref name="myintstack"></default-interceptor-ref>

    <action name="test">
    <!--
    引入拦截器或者是拦截器栈
     -->
        <interceptor-ref name="myintstack"></interceptor-ref>

    </action>
    <action name="test2">
    <!--
    引入拦截器或者是拦截器栈
     -->
        <interceptor-ref name="myintstack"></interceptor-ref>

    </action>


    <action name="test3">
    <!--
    引入拦截器或者是拦截器栈
    因为上面配置了 默认的拦截器 ,所以这里就不再引用了
     -->

    </action>

    </package>


</struts>
```

3.1.5 执行顺序

```
@Override
    public String intercept(ActionInvocation invocation) throws Exception {
        // Object action = invocation.getAction();//获取到了 Action 对象
```

```
        //从 session 里面获取值,判断是否有值
            Object user =
ServletActionContext.getRequest().getSession().getAttribute("user");
            if (user==null) {//没有登陆
                //让它跳到登陆页面
                return "login"; //相当于咱们遇到转换异常,没有经过 action 直接到了结果
视图页面 ,会被当做返回结果直接进行返回
            }
            System.err.println(System.currentTimeMillis());
            System.err.println("执行之前");
            String invoke = invocation.invoke();//返回值,同步的方法,调用了第二个拦截
器的intercept方法
            System.err.println("继续写代码"+invoke);
            System.err.println(System.currentTimeMillis());
            //有很多个拦截器,拦截器有顺序
                // 1 2 3 4 5五个拦截器
            // 1拦截器调用 invocation.invoke() 其实是调用第2个拦截器的intercept方法
            //2 进行invocation.invoke()调用的是 3拦截器intercept
            // 直到 5调用invocation.invoke() 放心到 action  其实调用的是对应 action
方法

        return invoke;
    }
```

1. 首先执行拦截器String invoke = invocation.invoke();之前的代码
2. 执行String invoke = invocation.invoke();放行,进入下一个拦截器,直到进入 action 方法
3. action 执行,返回结果到结果视图
4. 结果视图继续返回,按照拦截器的执行顺序逆向返回
5. 最终到 System.err.println("继续写代码"+invoke); 开始的代码
6. 多个拦截器的时候,按照执行顺序,从前到后执行调用,执行完成后按照从后到前的方式返回

3.2、自定义验证登录的拦截器

3.2.1针对所有动作方法进行拦截

```
/**
 * 定义 拦截器的功能
 *      用于判断用户是否登陆,没有登陆要求登陆,登陆了直接放行
 *
 *
 * 返回值 Action 的返回值(可以中途拦截直接返回)
 * 参数  Action 的封装对象
 */
    @Override
    public String intercept(ActionInvocation invocation) throws Exception {
        // Object action = invocation.getAction();//获取到了 Action 对象
        //从 session 里面获取值,判断是否有值
```

```
        Object user =
ServletActionContext.getRequest().getSession().getAttribute("user");
            if (user==null) {//没有登陆
                //让它跳到登陆页面
                return "login"; //相当于咱们遇到转换异常,没有经过 action 直接到了结果
视图页面 ,会被当做返回结果直接进行返回
            }
            System.err.println("执行之前22222");
            String invoke = invocation.invoke();//返回值,同步的方法,调用了 action 的
动作方法
            System.err.println("继续写代码2222"+invoke);
            //有很多个拦截器,拦截器有顺序
                // 1 2 3 4 5五个拦截器
            // 1拦截器调用 invocation.invoke() 其实是调用第2个拦截器的intercept方法
            //2 进行invocation.invoke()调用的是 3拦截器intercept
            // 直到 5调用invocation.invoke() 放心到 action  其实调用的是对应 action
方法

        return invoke;
    }


}
```

3.2.2 针对特定方法拦截

```
public class MyInterceptor3 extends MethodFilterInterceptor {
    /**
     * 拦截的方法的另外一种方式
     */
    @Override
    protected String doIntercept(ActionInvocation invocation) throws Exception {
        Object user =
ServletActionContext.getRequest().getSession().getAttribute("user");
        if (user==null) {//没有登陆
            //让它跳到登陆页面
            return "login"; //相当于咱们遇到转换异常,没有经过 action 直接到了结果视图
页面 ,会被当做返回结果直接进行返回
        }
        return invocation.invoke();
    }

}
```

3.2.2.1 配置使用

```
<package name="p1" extends="struts-default">
        <interceptors>

        <interceptor name="myint3"
class="com.qianfen.struts.interceptor.MyInterceptor3">
            <!--配置需要排除拦截或者需要拦截的方法名,此配置可以不在这里写,可以写在引入拦
截器的时候
                排除和引入方法的作用是一致的,排除的代表除了之外的都拦截， 引入的代表引入
的拦截,具体用谁取决于自己,理论上哪个少写哪个
            -->
          <!--
            <param name="excludeMethods">test2</param>
             -->
            <param name="includeMethods">test2</param>
        </interceptor>
        </interceptors>


        <action name="test2" class="com.qianfen.struts.interceptor.ActionDemo2"
method="test2">
            <!--
            一旦引用了自定义的拦截器,原始的默认拦截器就失效了
             -->
            <interceptor-ref name="myint3">
                <!--可以在此处指定需要拦截的方法-->
            </interceptor-ref>
            <result name="test1">/index.jsp</result>
            <result name="login">/login.jsp</result>
        </action>

    </package>
```

四、文件的上传和下载

4.1、文件上传

必要前提：

表单的method必须是post

表单的enctype必须是multipart/form-data

提供input type=”file”类型上传输入域

在Struts2中，文件上传是由一个叫做fileUpload完成的。

4.1.1单文件上传

4.1.1.1 form

```
    <form action="fileUpload" enctype="multipart/form-data" method="post">
      名字:<input type="text" name="name" /><br>
      靓照:<input type="file" name="photo" /><br>
      <input type="submit" value="提交" /><br>
    </form>
```

4.1.1.2 Action

```
public class ActionDemo1 extends ActionSupport {
/**
 *  表单中必须是 multipart/form-data
 *  method 必须是post
 *
 *  比如表单中的 file  name 为 photo
 *  那么在 action 中  创建一个私有的变量 File 类型  名字就叫 photo  和表单名保持一致
 *      文件名的获取  起一个 表单名+FileName 格式 的字符串
 *      文件的类型   创建一个 表单名+ContentType 固定格式的字符串
 *
 *  在对应的 action 方法中将文件写到磁盘即可
 */
    private File   photo;// 名字不能随便写,表单中叫什么名这里就得写什么
    private String photoFileName;// 固定写法 表单名+ FileName
    private String photoContentType;//MIME TYPE
    private String name;//form 表单的中普通字段


    public String fileUpload() throws Exception {
            //保存文件
    String realPath =
ServletActionContext.getServletContext().getRealPath("files");

    FileUtils.copyFile(photo, new File(realPath, photoFileName));//写入文件
        System.err.println(name);
        return SUCCESS;
    }

    public File getPhoto() {
        return photo;
    }

    public void setPhoto(File photo) {
        this.photo = photo;
```

```
    }

    public String getName() {
        return name;
    }

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

    public String getPhotoFileName() {
        return photoFileName;
    }

    public void setPhotoFileName(String photoFileName) {
        this.photoFileName = photoFileName;
    }

    public String getPhotoContentType() {
        return photoContentType;
    }

    public void setPhotoContentType(String photoContentType) {
        this.photoContentType = photoContentType;
    }

}
```

4.1.2多文件上传

> 类似于单文件上传

```
public class ActionDemo2 extends ActionSupport {
/**
 * 一个 name 对应多个文件的情况下, 使用数组接收
 * 普通表单 也一样,就和兴趣爱好一样,传过来的也是一个数组
 */
    private File[]   photo;// 名字不能随便写,表单中叫什么名这里就得写什么
    private String[] photoFileName;// 固定写法 表单名+ FileName
    private String[] photoContentType;//MIME TYPE
//  private File   photo1;// 名字不能随便写,表单中叫什么名这里就得写什么
//  private String photo1FileName;// 固定写法 表单名+ FileName
//  private String photo1ContentType;//MIME TYPE
    private String name;
```

```
    public String fileUpload() throws Exception {
            //保存文件
    String realPath =
ServletActionContext.getServletContext().getRealPath("files");
    for (int i = 0; i < photo.length; i++) {
        FileUtils.copyFile(photo[i], new File(realPath, photoFileName[i]));//写入文
件
    }
        System.err.println(name);
        return SUCCESS;
    }


    public File[] getPhoto() {
        return photo;
    }


    public void setPhoto(File[] photo) {
        this.photo = photo;
    }


    public String[] getPhotoFileName() {
        return photoFileName;
    }


    public void setPhotoFileName(String[] photoFileName) {
        this.photoFileName = photoFileName;
    }


    public String[] getPhotoContentType() {
        return photoContentType;
    }


    public void setPhotoContentType(String[] photoContentType) {
        this.photoContentType = photoContentType;
    }


    public String getName() {
        return name;
    }


    public void setName(String name) {
```

```
        this.name = name;
    }

}
```

4.1.3上传中出现问题

l 上传失败时，会转向一个那么为input的视图。

4.1.3.1 错误显示：

```
在  input 对应的jsp 文件中通过 <s:actionerror/> 可以显示错误信息
```

4.1.3.2 修改上传文件的大小,在 struts.xml 文件中

> 注意在 tomcat6.0.43之后的版本,如果超出大小,会直接出现断开连接的页面,而不是显示错误信息,如果要测试这个功能,需要使用 tomcat6.0.43版本

```
<constant name="struts.multipart.maxSize" value="10737400"></constant>
```

4.1.3.3限制上传文件的类型

allowedTypesSet：允许上传的文件的MIME类型，多个用逗号分隔

allowedExtensionsSet：允许上传的文件的扩展名，多个用逗号分隔

```
<action name="fileUpload2" class="com.qianfeng.struts.action.ActionDemo2"
method="fileUpload">
        <interceptor-ref name="defaultStack">
          <!--配置允许的后缀名和 mime 类型-->
            <param name="fileUpload.allowedExtensions">.png,.jpg,.jpeg</param>
            <param name="fileUpload.allowedTypes">image/jpeg</param>
        </interceptor-ref>
            <result>/success.jsp</result>
            <result name="input">/index.jsp</result>
        </action>
```

4.1.3.4修改错误消息提示：与国际化有关

文件上传所有的默认消息提示在：

struts2-core.jar org\apache\struts2\struts-messsages.properties

覆盖掉默认的消息提示：局部消息资源文件无效,全局资源文件有效：

配置 struts.xml

```
<!--配置基名-->
<constant name="struts.custom.i18n.resources" value="msg"></constant>
```

在 src 下创建msg_zh_CN.properties 其中{0}和{1}等是占位符,用于显示错误的具体值,我们只需要将默认的错误消息中的 非占位符部分按照我们自己的需要改成对应内容即可

```
struts.messages.upload.error.SizeLimitExceededException=提示内容{0}提示内容{1}
```

常见错误

```
struts.messages.error.uploading=上传错误: {0}
struts.messages.error.file.too.large=文件太大: {0} "{1}" "{2}" {3}
struts.messages.error.content.type.not.allowed= MIME 类型不符: {0} "{1}" "{2}" {3}
struts.messages.error.file.extension.not.allowed=后缀名不符: {0} "{1}" "{2}" {3}
```