一、结果视图

1、局部和全局结果视图

a、局部视图：

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

        <action name="hello" class="com.qianfeng.struts.action.ActionDemo1"
method="helloMoto">
        <!--
            结果逻辑视图,用于action 方法运行结束后应该怎么操作
                    局部视图
                        只给当前的 action用,给谁配置了就是谁用
                    全局视图
                        不止当前 action 可以用
                    相同的 name情况下,局部的优先级会高于全局的优先级

          -->
            <result name="success">/demo2.jsp</result>
        </action>

        <action name="hello1" class="com.qianfeng.struts.action.ActionDemo1"
method="helloMoto">
        </action>
    </package>
```

b、全局视图：

多个动作对应同一个视图

```
<package name="p1" extends="struts-default">
    <!--
    全局视图
        给当前 package 内部的所有的 action 设置指定的结果集
     -->
        <global-results>
            <result name="success">/demo1.jsp</result>
        </global-results>

        <action name="hello" class="com.qianfeng.struts.action.ActionDemo1"
method="helloMoto">
        <!--
```

```
            结果逻辑视图,用于action 方法运行结束后应该怎么操作
                    局部视图
                        只给当前的 action用,给谁配置了就是谁用
                    全局视图
                        不止当前 action 可以用
                    相同的 name情况下,局部的优先级会高于全局的优先级

          -->
            <result name="success">/demo2.jsp</result>
        </action>


        <action name="hello1" class="com.qianfeng.struts.action.ActionDemo1"
method="helloMoto">

        </action>
    </package>
```

2、result元素的配置：

属性：

l name：逻辑视图名称。它对应的是动作方法的返回值。默认值：success。

l type：到达目标的形式。默认值：dispatcher。转发。

3、Struts2提供的结果类型（result type属性）

3.1在struts-default.xml中有定义

```
 <result-types>
            <result-type name="chain"
class="com.opensymphony.xwork2.ActionChainResult"/>
            <result-type name="dispatcher"
class="org.apache.struts2.dispatcher.ServletDispatcherResult" default="true"/>
            <result-type name="freemarker"
class="org.apache.struts2.views.freemarker.FreemarkerResult"/>
            <result-type name="httpheader"
class="org.apache.struts2.dispatcher.HttpHeaderResult"/>
            <result-type name="redirect"
class="org.apache.struts2.dispatcher.ServletRedirectResult"/>
            <result-type name="redirectAction"
class="org.apache.struts2.dispatcher.ServletActionRedirectResult"/>
            <result-type name="stream"
class="org.apache.struts2.dispatcher.StreamResult"/>
            <result-type name="velocity"
class="org.apache.struts2.dispatcher.VelocityResult"/>
            <result-type name="xslt"
class="org.apache.struts2.views.xslt.XSLTResult"/>
            <result-type name="plainText"
class="org.apache.struts2.dispatcher.PlainTextResult" />
            <result-type name="postback"
class="org.apache.struts2.dispatcher.PostbackResult" />
        </result-types>
```

3.2 常见 type

```
<package name="p2" extends="struts-default">
        <action name="mianhuai" class="com.qianfeng.struts.action.ActionDemo2"
            method="mianhuai">
            <!-- name 方法的返回值 默认是success 如果返回的是 success 那么不写也行
name 可以随便写值,只要和方法的返回值一致即可
                系统默认有几个返回值,是框架本身自带进行处理的
                 success 框架的默认值,建议我们在成功返回数据的情况下使用这个值,但是规则
都是用来打破的
                  error 指的是action 在运行期间出现错误了
                  input 输入校验出错,用于回显数据用的,要求是返回到输入页面 类似于注册登
录的时候 输入错了,网页上面还有你原先输入的信息
                  none 不需要配置 result
                  login 如果用户没有登录,建议返回此值,然后跳转的目标页是登陆页面

                    <result name="error">/demo2.jsp</result>
                    <result name="input">/demo1.jsp</result>
                    <result name="login">/index.jsp</result>

                type 结果集的处理类型
                    dispatcher 用于转发到另外一个jsp页面  默认值
```

```
                    redirect 重定向 到另外一个 jsp 页面
                    chain 用于转发到另外一个 action    转发(不是重定向), action
                    redirectAction 重定向到另外一个 action
                    stream 以二进制流的方式输出数据,用于文件下载
                    plainText 以纯文本的方式输出内容

                    freemarker 以freemarker模板的方式输出内容  用于页面静态化
                    velocity   velocity模板
                    xslt  以 xml 方式输出内容(肯定不用)
                    httpheader 输出 http 头
                    postback

                    type 的原理
                        将想要执行的操作 放到了对应的类里面,自己处理结果

                        如果要自定义 type   照着系统的抄,只要将我们自己想要做的事情放到
里面.然后设置返回值 type
                    -->
            <result name="success" type="chain">
            <!--
            跳转或者重定向到另外一个命名空间下的 action 的时候 需要指定参数
            namespace 命名空间
            actionName 目标 action name
             -->
                <param name="namespace">/testchain</param>
                <param name="actionName">testchaintoothernamesapce</param>
            </result>

        </action>

            <action name="testchain">
                <!-- 重定向到另外一个 action-->
                <result name="success"
type="redirectAction">testredirectAction</result>
        </action>

        <action name="testredirectAction">
                <result name="success" >/demo1.jsp</result>
        </action>
    </package>


    <!--
    命名空间不一致的包
    -->
    <package name="p3" extends="struts-default" namespace="/testchain">
        <action name="testchaintoothernamesapce">
        <result name="success" >/demo3.jsp</result>
        </action>
```

```
    </package>
```

l dispatcher：用于转发到另外一个JSP页面。

l freemarker：用于转发到另外一个freemarker模板。（页面静态化）

l velocity：用于转发到另外一个velocity模板。

l httpheader：用于输出http协议的消息头。

l xslt：XML有关的样式

l redirect：用于重定向到另外一个JSP页面。

l redirectAction：用于重定向到另外一个动作。

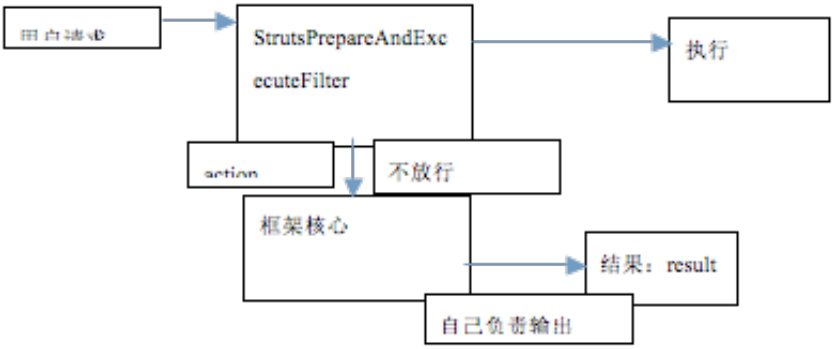
l stream：用于文件下载（日后再讲。文件上传和下载）

l plainText：以纯文本的形式展现页面。输出源码。

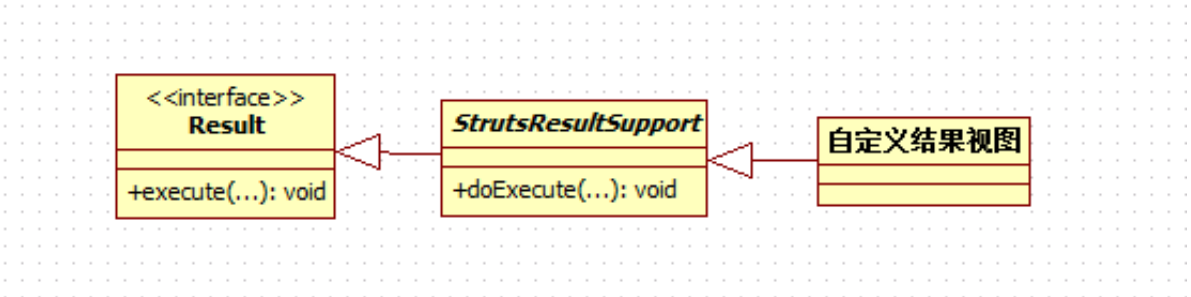
4、自定义结果视图

随机验证码图片

4.1、struts2的执行图



4.2、自定义结果视图步骤



a、编写一个类，直接或间接实现com.opensymphony.xwork2.Result接口。一般继承org.apache.struts2.dispatcher.StrutsResultSupport类

```
/**
 * 这个类将要变成一个 type 类
 *
 *  用于返回验证码给页面
 * @author jackiechan
 *
 */
public class CaptchaResult  extends StrutsResultSupport{
    /**
     *  需要了解自定义的流程,但是很少用
     *
     *  这个方法就是用于处理自己逻辑的方法
     *  创建验证码
     *  生成验证码图片
     *  返回给页面(页面中有一个 img 标签)  不会单独跳转页面
     *  如果不返回页面而是直接返回数据，那么应该在这里直接调用 response 处理数据
     */
    @Override
    protected void doExecute(String finalLocation, ActionInvocation invocation)
throws Exception {
            ValidateCode validateCode=new ValidateCode(200, 130, 4, 100);//创建验
证码
            //获取图片
            BufferedImage image = validateCode.getBuffImg();
            //返回给页面
            ServletOutputStream stream =
ServletActionContext.getResponse().getOutputStream();
            ImageIO.write(image, "jpg", stream);
    }

}
```

b、声明结果类型，然后才能使用

```
    <package name="p100" extends="struts-default">
        <!--
        定义了一个结果集  名字是captcha,在别的地方使用的时候用的就是名字  处理类
com.qianfeng.struts.result.CaptchaResult
        -->
            <result-types>
                <result-type name="captcha"
class="com.qianfeng.struts.result.CaptchaResult"></result-type>
            </result-types>

        </package>
```

c、使用

```
    <package name="p100" extends="struts-default">
        <!--
        定义了一个结果集  名字是captcha,在别的地方使用的时候用的就是名字  处理类
com.qianfeng.struts.result.CaptchaResult
        -->
            <result-types>
                <result-type name="captcha"
class="com.qianfeng.struts.result.CaptchaResult"></result-type>
            </result-types>


            <action name="captcha">
        <!--
        name 用默认值
            返回验证码不需要页面 所以不跳页面
            出错误 There is no result type defined for type 'captcha' mapped with
name 'success'.
            没有captcha这个 type
            需要做一件事情
                第一件
                得让框架知道有captcha type
                第二件 captcha type 的处理类得配置一下
         -->
                <result type="captcha"></result>
            </action>


        </package>
```

二、封装请求参数

2.1动态参数注入

2.1.1、方式一: request 获取

页面,后面步骤除非特殊,会略过当前内容

```
 <form action="${pageContext.request.contextPath }/action1">
        <input type="text" name ="name" /><br>
        <input type="password" name ="password" /><br>
        <input type="submit" value="提交"/>
    </form>
```

映射文件,后面步骤除非特殊,会略过当前内容

```
<action name="action1" class="com.qianfeng.struts.action.ParamAction1"
method="m1"></action>
```

```
/**
 * 获取用户传递的参数 方式一
 * @author jackiechan
 *
 */
public class ParamAction1 extends ActionSupport {

    public String m1(){
        //获取到用户传递参数
        //
        HttpServletRequest request = ServletActionContext.getRequest();
        String name = request.getParameter("name");
        String password = request.getParameter("password");
        System.err.println("name="+name+"  password="+password);
        return NONE;
    }
}
```

2.1.2、方式二：Action 作为模型接收参数

用Action动作类作为模型对象。要求 form 表单中的 name 属性必须和 Action 中用于接收参数的属性名一致(属性 get 或者 set 方法去掉 get set 后首字母小写)

```
 <form action="${pageContext.request.contextPath }/action2">
        <input type="text" name ="name" /><br>
        <input type="password" name ="password" /><br>
        <input type="submit" value="提交"/>
    </form>
```

```
/**
 * 获取用户传递的参数 方式二
 * 框架自动封装参数
 *  Action 类就是 Model 类 所有的参数都声明 Action 类里面
 *  要求 表单中的 name 必须和属性名(get set 方法去掉 首字母小写)一致
 * @author jackiechan
 *
 */
public class ParamAction2 extends ActionSupport {
    private String name;
    private String password;
    public String getName1() {
        return name;
    }

    public void setName1(String name) {
        this.name = name;
    }

    public String getPassword1() {
        return password;
    }

    public void setPassword1(String password) {
        this.password = password;
    }

    public String m1(){
        System.err.println("name="+name+"  password="+password);
        return NONE;
    }
}
```

2.1.3、方式三：动作类和模型分开（推荐）

> 通过额外的对象接收参数

表单页面:

> 表单页面中的 name 属性必须是 在 action 中接收参数的对象属性名.属性名 比如用 user 对象接收参数,参数的名字是name 则为 user.name

```
<form action="${pageContext.request.contextPath }/action3">
        <input type="text" name ="user.name" /><br>
        <input type="password" name ="user.password" /><br>
        <input type="submit" value="提交"/>
    </form>
```

模型:

```
public class User implements Serializable {
    private String name;
    private String password;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    @Override
    public String toString() {
        return "User [name=" + name + ", password=" + password + "]";
    }

}
```

动作:

```
/**
 * 获取用户传递的参数 方式三
 * 框架自动封装参数
```

```
 *  Action 和 Model 分离
 *      要求,表单中的 name 不 在是直接写属性名 要写 user.name
 *  实现的方式 ognl 表达式
 *  因为表单中是 user.name 先执行 getUser 获取下 User 对象,判断是不是空,是空就利用反射创建一个对象然后调用 Set 方法设置进去
 *  然后再调用一次 get 获取值,然后再调User 上面的 setName 方法将 name 设置进去 ,如果getUser 不为空,那么就再调用一次 getUser
 *      第一次 getUser 获取到的结果只用于判断是否为空
 * @author jackiechan
 *
 */
public class ParamAction3 extends ActionSupport {
    private  User user;// =new User();

    public void setUser(User user) {
        System.err.println("setUser 执行了");
        this.user = user;
    }

    public User getUser() {
        System.err.println("getUser 执行了");
        return user;
    }

    public String m1(){
        System.err.println("name="+user.getName()+"password="+user.getPassword());
        return NONE;
    }
}
```

2.1.4、Acion实现ModelDriven接口

> 此方式与方式三相类似,不过是 action 实现的ModelDriven接口,然后 form 表单中的 name 为接收参数的对象的内部的属性名,没有了前置属性名

模型:

```
public class User implements Serializable {
    private String name;
    private String password;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
```

```
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    @Override
    public String toString() {
        return "User [name=" + name + ", password=" + password + "]";
    }

}
```

动作:

```
/**
 * 获取用户传递的参数 方式四
 * 框架自动封装参数
 *  Action 和 Model 分离
 *  实现ModelDriven 接口,泛型的类型就是Model 的类型
 *
 * @author jackiechan
 *
 */
public class ParamAction4 extends ActionSupport implements ModelDriven<User>{
    private  User user =new User();//实现了ModelDriven 对象要手动创建


    public String m1(){
        System.err.println("name="+user.getName()+"
password="+user.getPassword());
        return NONE;
    }
/**
 * 返回的就是用于封装参数的对象
 */
    @Override
    public User getModel() {
        return user;
    }
}
```

form 表单

```
<form action="${pageContext.request.contextPath }/action4">
      <input type="text" name ="name" /><br>
      <input type="password" name ="password" /><br>
      <input type="submit" value="提交"/>
  </form>
```

原因：与Struts2的值栈有关

该功能是由一个叫做modelDriven的拦截器完成的。

```
@Override
    public String intercept(ActionInvocation invocation) throws Exception {
        Object action = invocation.getAction();//action 代表执行我们的 action 类

        if (action instanceof ModelDriven) {//如果属于ModelDriven
            ModelDriven modelDriven = (ModelDriven) action;
            ValueStack stack = invocation.getStack();//获取值栈
            Object model = modelDriven.getModel();//调用 action 中的方法,因为实现了
ModelDriven所以一定有这个方法
            if (model !=  null) {
                stack.push(model);//放入栈顶,按照 struts 的原理,封装参数会从栈顶向下找
这个属性,此处可以找到
            }
            if (refreshModelBeforeResult) {
                invocation.addPreResultListener(new
RefreshModelBeforeResult(modelDriven, model));
            }
        }
        return invocation.invoke();
    }
```

2.2静态参数注入

```
<action name="action5" class="com.qianfeng.struts.action.ParamAction5"
method="m1">
        <!--
        name 就是 action 里面的属性名
        静态参数注入, 第一个 name 代表要给哪个参数赋值,第二个 name 代表属性的具体的名字
,中间的值代表具体的值
          -->
            <param name="name">待宰的羔羊</param>

        </action>
```

```
/**
 静态参数注入, 是由staticParams拦截器执行的,它会在动态参数注入 params拦截器之前就执行
 * @author jackiechan
 *
 */
public class ParamAction5 extends ActionSupport {
    private  String name;
    public void setName(String name) {
        this.name = name;
    }

    public String m1(){
        System.err.println("name==="+name);
        return NONE;
    }

}
```

2.3动态参数和静态参数注入功能实现（知道）

是由两个拦截器来完成。

静态参数注入：staticParams

动态参数注入：params

三、类型转换

> 因为我们 form 表单中传递过来的参数都是字符串格式,可能和我们最终想要的类型不一致,因此我们需要进行类型转换

1、实现字符串转日期

a、编写一个类，继承StrutsTypeConverter

```
/**
 * 将表单中的字符串转成日期格式
 * @author jackiechan
 *
 */
public class MyDateConvert extends StrutsTypeConverter{
    private SimpleDateFormat simpleDateFormat=new
SimpleDateFormat("yyyy/mm/dd");//指定格式
/**
 * 从字符串转到先要的类型
 *  values 表单中传递过来的数据
 * toClass 你想要的类型
 */
    @Override
    public Object convertFromString(Map context, String[] values, Class toClass) {
        if (values!=null&&values.length>0) {//传递了参数
            String  dataString= values[0];//传递过来的数据
            //将dataString 转成 Date
            try {
                Date date = simpleDateFormat.parse(dataString);//进行转换
                return date;//转成对应的对象后返回
            } catch (ParseException e) {
                e.printStackTrace();
                throw new RuntimeException(e);
            }
        }
        return null;
    }
/**
 * 根据类型转到字符串
 *  o 你目标类型的对象,你怎么将它转成字符串 当前是 date 类型
 */
    @Override
    public String convertToString(Map context, Object o) {
        String format = simpleDateFormat.format(o);//将日志转成字符串
        return format;
    }

}
```

b、注册类型转换器

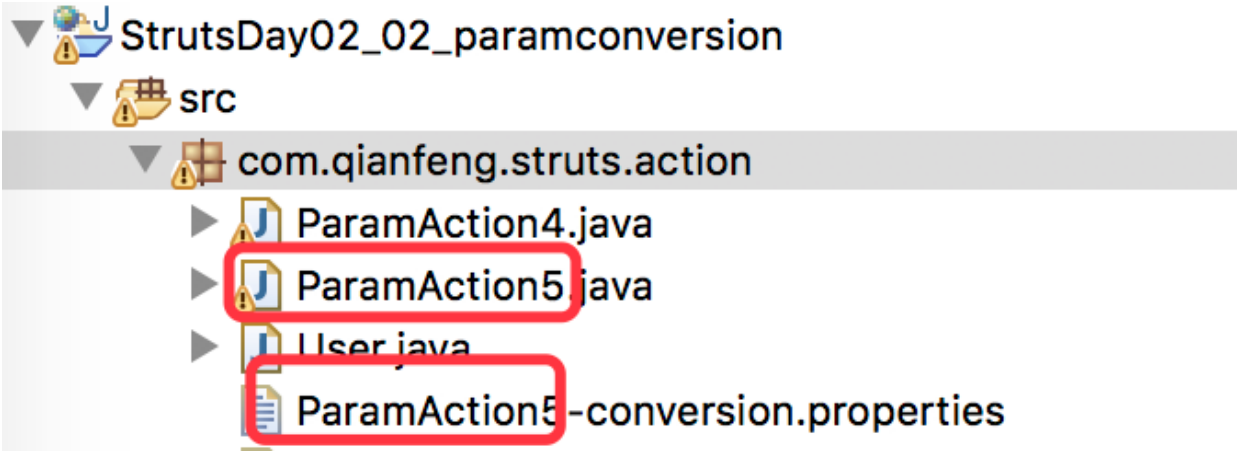
l 局部类型转换器：只为当前动作使用

```
public class ParamAction5 extends ActionSupport {

    private String name;
    private String password;
    private int age;
    private Date birthday;

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Date getBirthday() {
        return birthday;
    }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }

    public String getName() {
        return name;
    }

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

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String m1() {
        System.err.println(this);
        return NONE;
    }

    @Override
```

```
    public String toString() {
        return "ParamAction5 [name=" + name + ", password=" + password + ", age="
+ age + ", birthday=" + birthday
                + "]";
    }

}
```

1、动作类作为模型：

> 在 ParamAction5类有一个 Date类型的变量叫 birthday 需要被转换, 书写规则,在 Action 类所在的包下面编写一个 Action 类名- conversion.properties 的固定格式文件



properties 文件内部格式 Action 对象内需要被转换的属性名=用于转换的 Convert 类

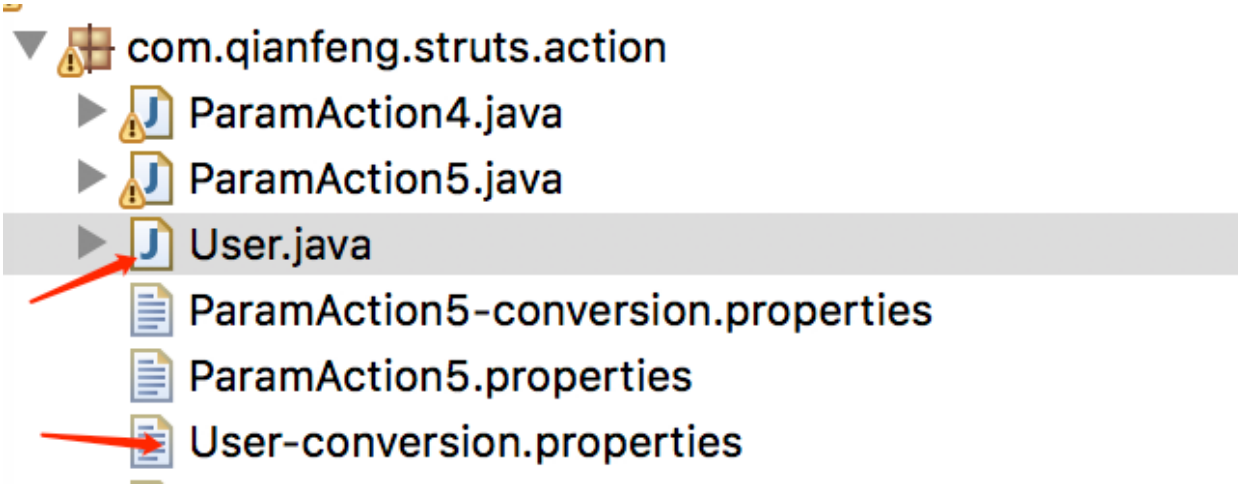
```
birthday=com.qianfeng.struts.convert.MyDateConvert
```

2、动作类和模型分开：（经常用）

```
public class User {
    private String name;
    private String password;
    private int age;
    private  Date birthday;

    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public Date getBirthday() {
        return birthday;
```

```
    }
    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    @Override
    public String toString() {
        return "User [name=" + name + ", password=" + password + ", age=" + age +
", birthday=" + birthday + "]";
    }
```

在模型所在的包中，建立以下配置文件 命名规则与上面一致



properties 内部为 User 对象中需要被转换的属性的属性名=用于转换的 Convert 类

```
birthday=com.qianfeng.struts.convert.MyDateConvert
```

l 全局类型转换器：所有动作使用

在src 目录下，建立固定名称为xwork-conversion.properties的配置文件,内部编写为 需要转换的类型=用于转换的 Convert 类

```
java.util.Date=com.qianfeng.struts.convert.MyDateConvert
```

2、类型转换失败的处理

a、转换失败，会自动转到一个name=input的逻辑视图，一般指向输入的那个页面，目的回显（建议使用struts2的表单标签）

b、错误消息提示中文版本

前提：动作类继承ActionSupport才能使用。

位置:在需要转换参数的的 Action 或者模型类所在的包下面创建对应的properties 文件, 规则 类名.properties 比如参考上面的转换 文件应该分别叫ParamAction5.properties 和 User.properties

内容 :固定格式invalid.fieldvalue.属性名=失败后的提示内容

```
invalid.fieldvalue.birthday=日期格式不合法
```

该功能是由一个叫做conversionError拦截器负责处理的。