**架构的灵活性**

**自定义一个类型**

|  |
| --- |
| package com.lwb.result;  import cn.dsna.util.images.ValidateCode; import com.opensymphony.xwork2.ActionInvocation; import org.apache.struts2.ServletActionContext; import org.apache.struts2.dispatcher.StrutsResultSupport;  import javax.servlet.http.HttpServletResponse;   */\*\*  \* 自定义结果类型  \*/* public class CAPTCHAResult extends StrutsResultSupport {  @Override  protected void doExecute(String s, ActionInvocation actionInvocation) throws Exception {  //生成验证码  ValidateCode code = new ValidateCode(200,30,4,10);   //获取response对象  HttpServletResponse response = ServletActionContext.*getResponse*();   //把图片返回给客户端  code.write(response.getOutputStream());  } } |

**然后进行配置，就可以灵活的使用各种自定义的类型，不在被框架所限制**

|  |
| --- |
| <!--配置验证码类型-->  <package name="p3" extends="struts-default">  <!--声明一个结果类型-->  <result-types>  <result-type name="captcha" class="com.lwb.result.CAPTCHAResult"></result-type>  </result-types>    <action name="checkcode">  <result type="captcha"></result>  </action>  </package> |

框架的标签就是所关系到的动作的名称，使得配置文件的可读性大大提高，起到见文知意的效果，但是，有些框架对于扩展性，做了太多的方法，比如后台验证表单数据的一个功能，使用了如下几种方法来进行验证

1.

|  |
| --- |
| @Override  public void validate() {  //判断用户名是否为空  //if(username == null && "".equals(username))  if(StringUtils.isEmpty(user.getUsername())){  addFieldError("username", "用户名不能为空");  }    //判断密码是否为空  if(StringUtils.isEmpty(user.getPassword())){  addFieldError("password", "密码不能为空");  }  } |

2.通过配置xml文件

2.1

|  |
| --- |
| <!-- 第一种校验字段方式 ：field-->  <field name="actionErrors">  <field-validator type="requiredstring">  <param name="fieldName">username</param>  <message>用户名不能为空-xml</message>  </field-validator>  </field> |

2.2

|  |
| --- |
| <!-- 第二种校验字段方式 ：field-->  <validator type="requiredstring">  <param name="fieldName">password</param>  <message>密码不能为空-xml</message>  </validator> |

两种方式配置表单验证，但是在第二种通过xml配置时，有两种方法都可以使用，个人认为有点啰嗦，两种方法目的一样，只是写法不同，感觉没有必要，个人认为是一个典型的反模式的例子