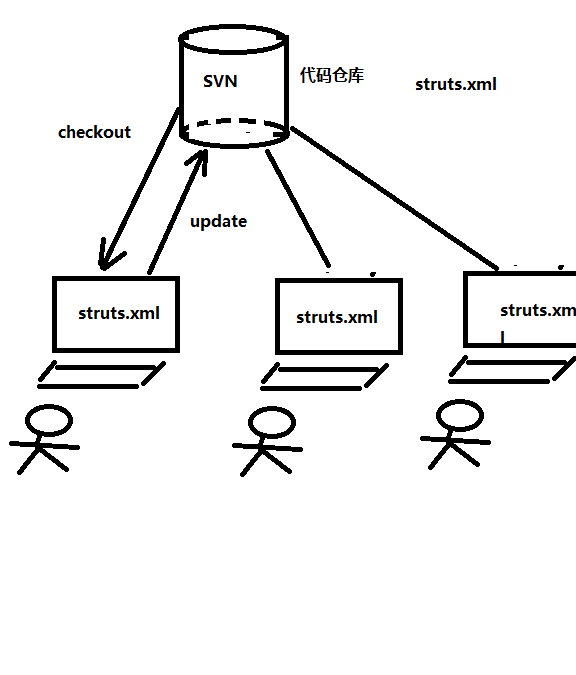
# 一、分文件编写框架配置文件

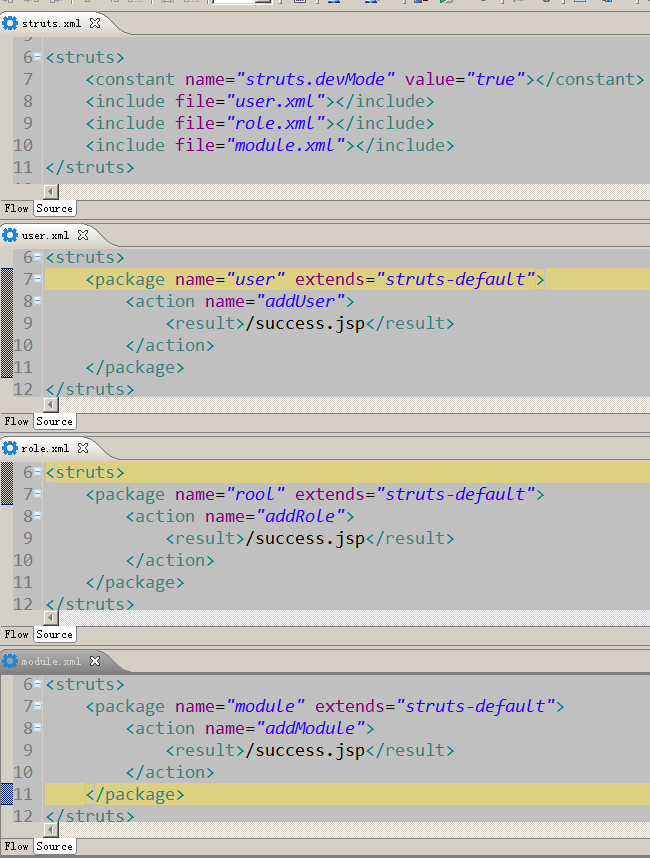
## 1、不分文件开发可能产生的问题

就类似于我们在写java类时，所有代码都写在一个类里，甚至写在一个方法里。



当3个人都checkout了struts.xml文件时，第一个人提交了，后面的人在没有更新就提交时，第一个人写的可能就白写了。

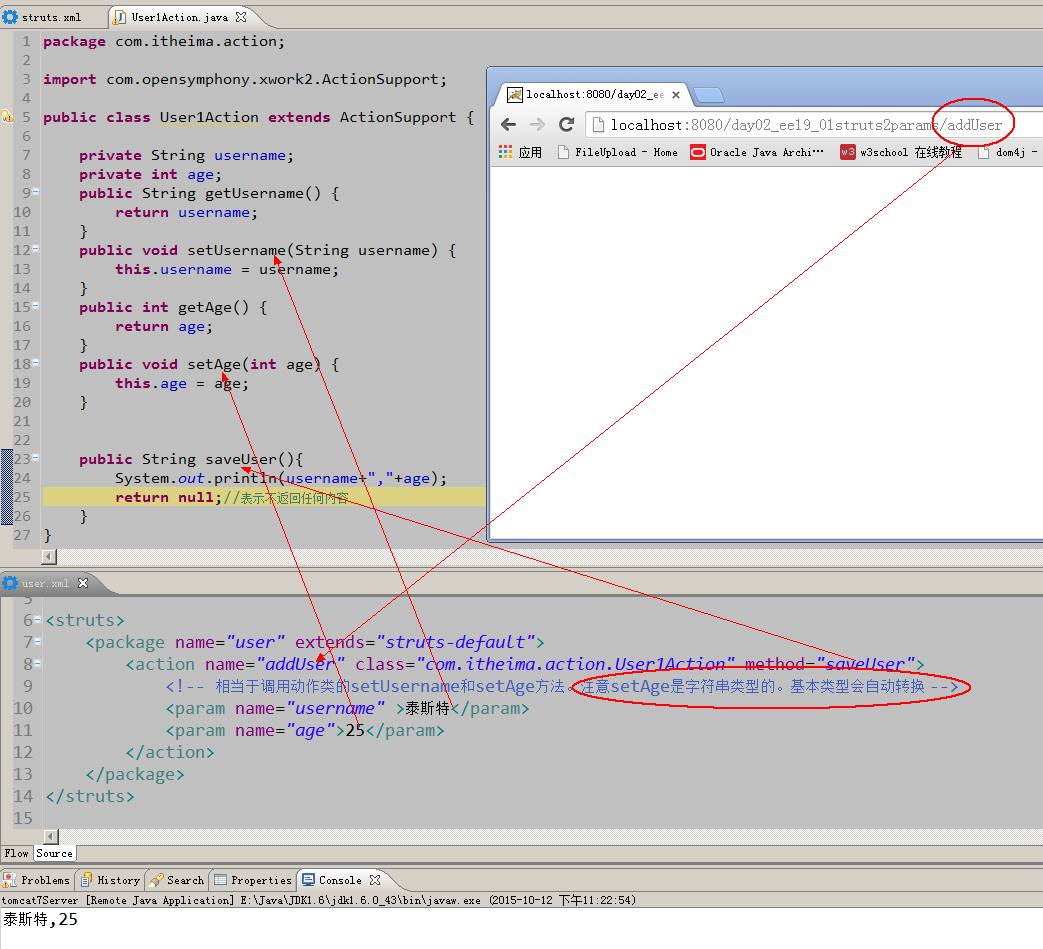
## 2、分文件编写Struts2的配置文件



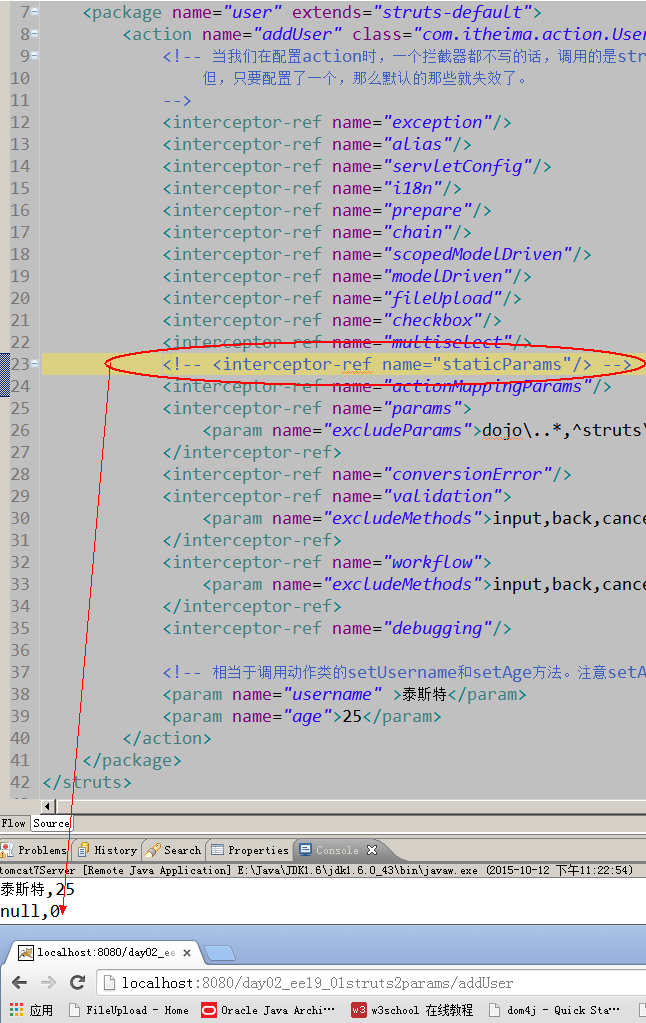
# 二、封装请求正文到对象中（非常重要）

## 1、静态参数封装

在struts.xml配置文件中，给动作类注入值。调用的是setter方法。



原因：是由一个staticParams的拦截器完成注入的。



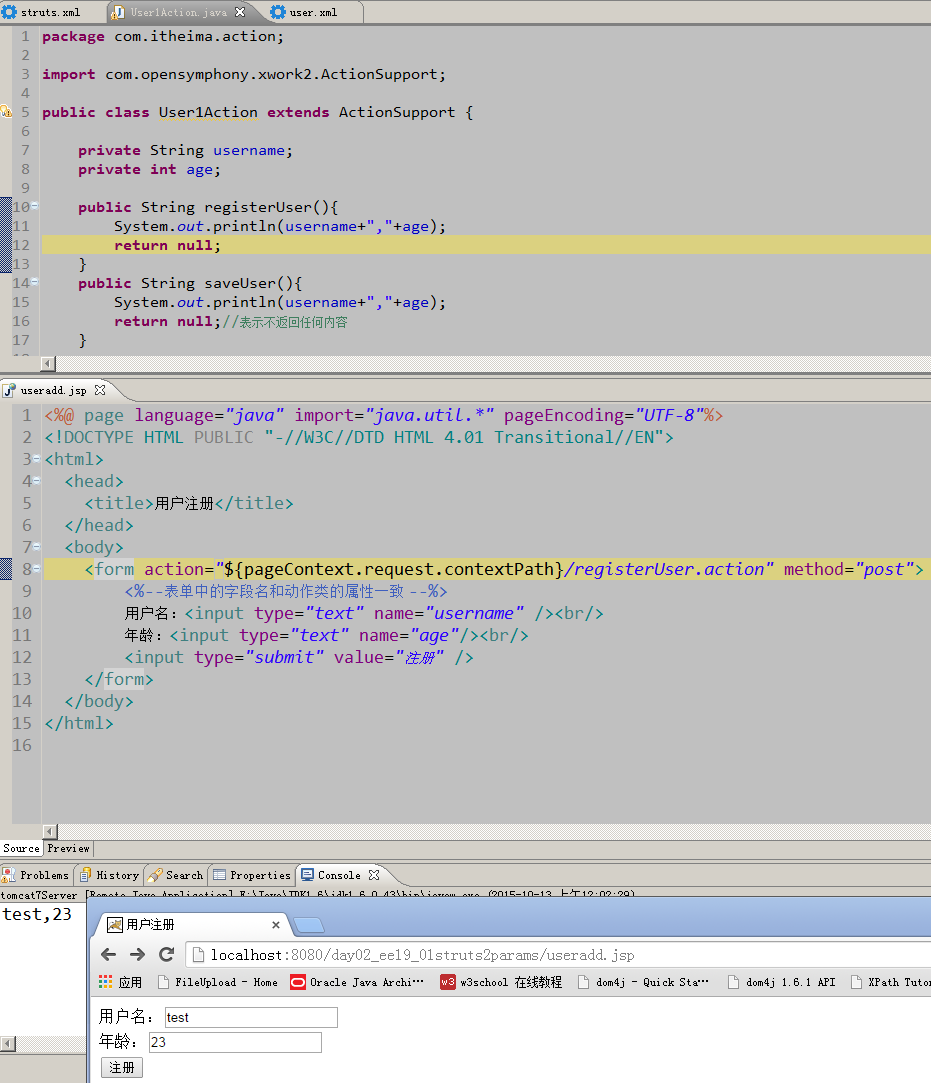
## 2、动态参数封装：开发时用到的

通过用户的表单封装请求正文参数。

### 2.1、动作类作为实体模型

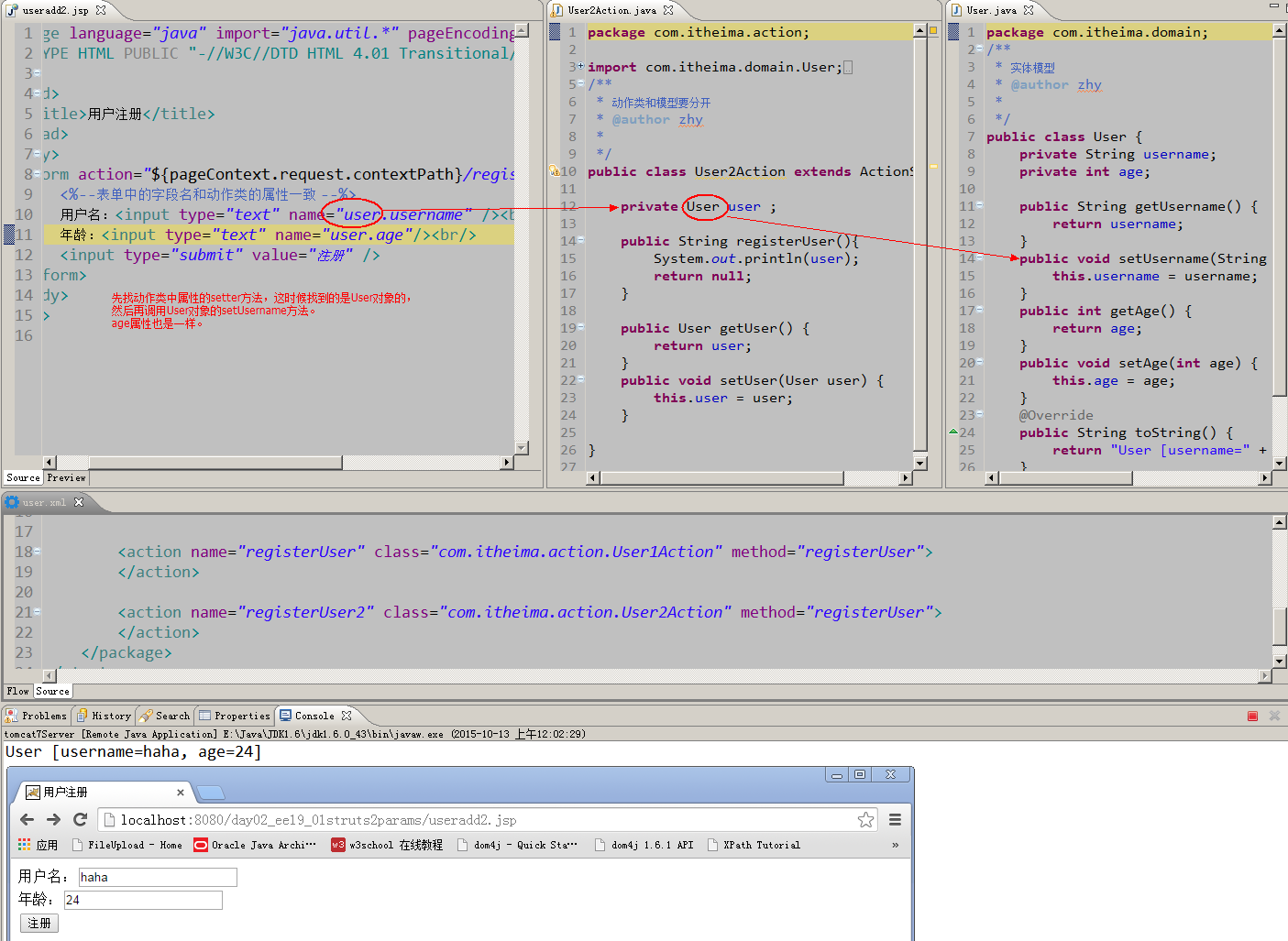
实体模型：Entity，对应数据库中表的记录（注意类对应的是表结构，而对象对应的是一条记录）





原因：是由params拦截器完成的。

### 2.2、动作类和实体模型分开

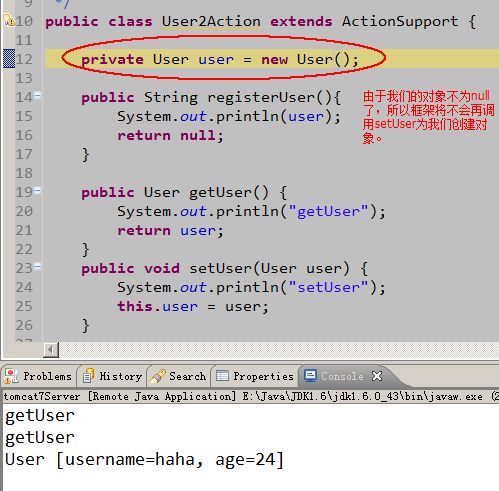
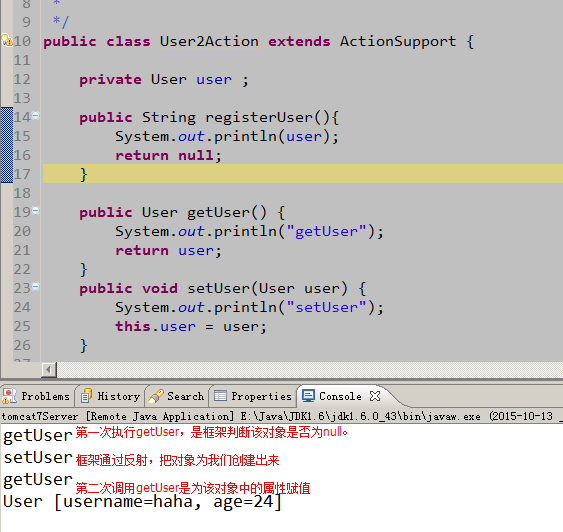


问题：

由于我们没有初始化user对象，默认为null，一调用setUser方法，就空指针异常了。但是框架却封装进去值了。

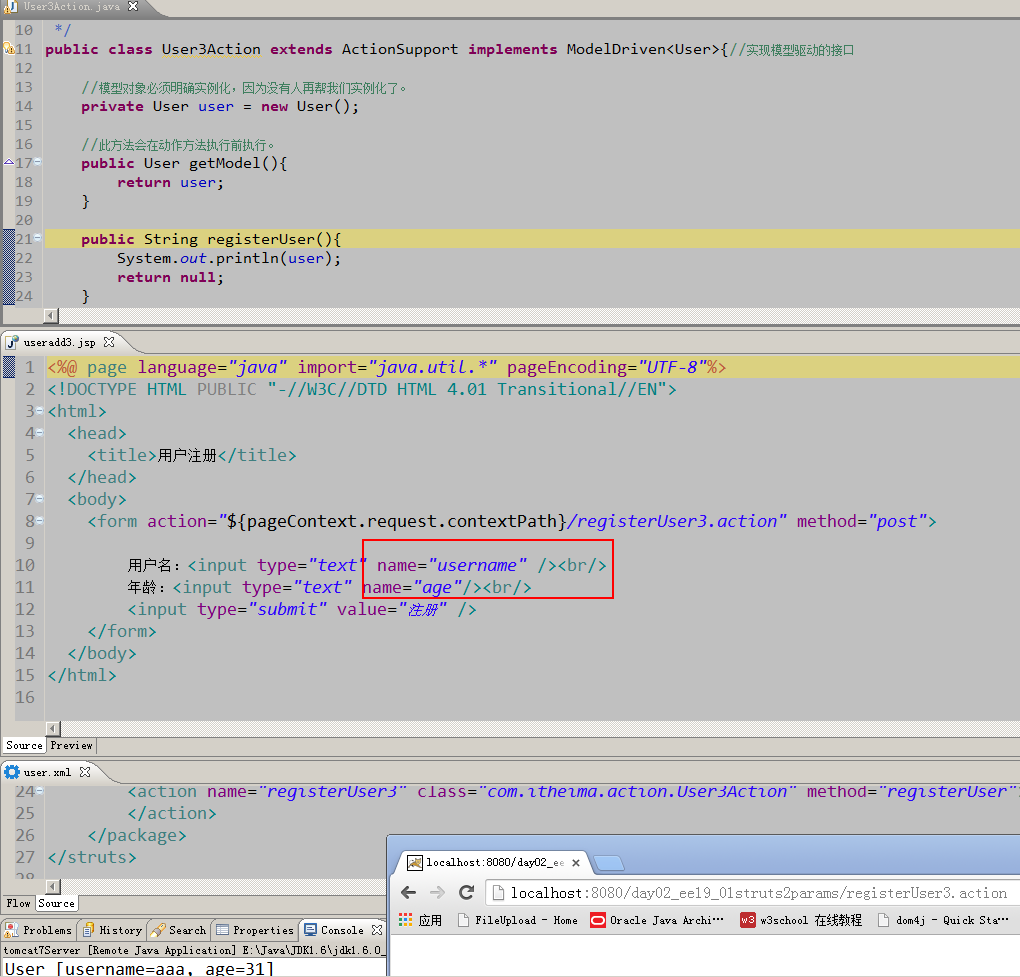
原因：

通过执行过程：



### 2.3、模型驱动：建立动作类和模型分开的前提下（开发中采用的方式）

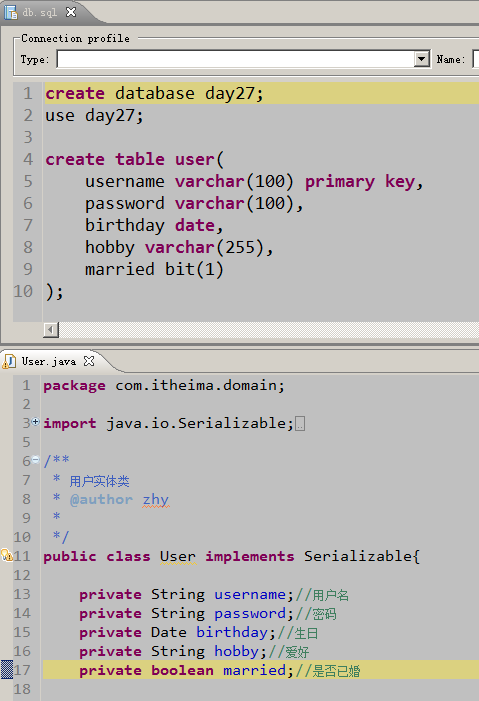
**此处的学习目标：目前先记住怎么写，要想理解，必须等讲完OGNL表达式之后。**

****

**原因：是由一个modelDriven拦截器做的。**

# 三、用户注册案例（重点）

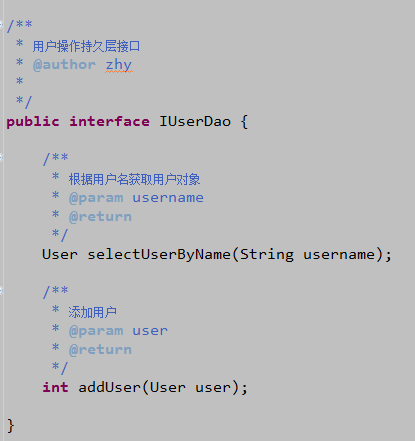
## 1、数据建模（实体模型和数据库）



## 2、建立业务层接口

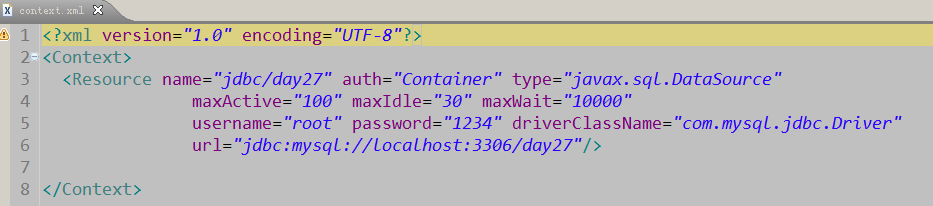


## 3、建立持久层接口



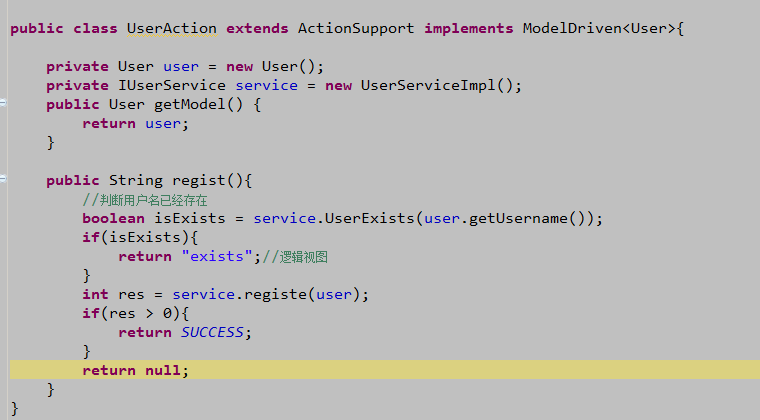
### 4、数据源工具类



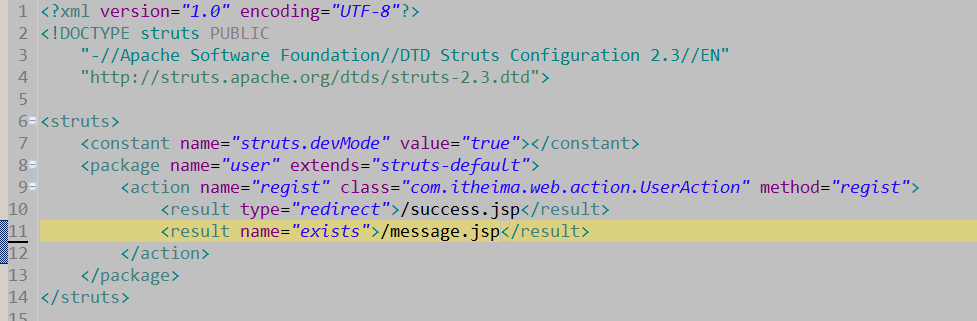


## 5、表现层使用Struts2框架实现

### 5.1、动作类：



### 5.2、配置文件

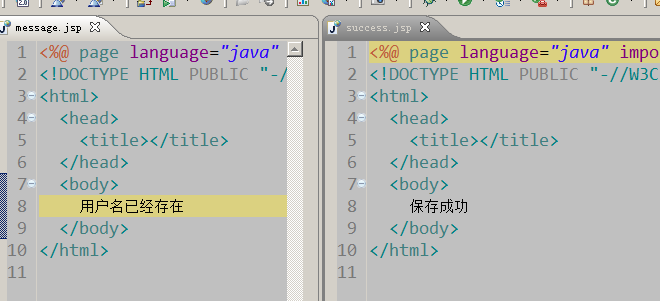


### 5.3、注册界面和结果视图

**注册界面：**



**结果视图：**

****

# 四、数据类型的转换（明白原理，实际开发中几乎不用）

## 1、开发中的情况：

实际开发中用户通过浏览器输入的数据都是String或者String[]。

String/String[]————填充模型(set方法)————>POJO(plain old java object) pojo中有java的数据类型。

POJO————————获取(get方法)————>页面展示：String

## 2、类型转换情况

写数据：（增，删，改）都是String或String[]数组转换为其他类型。

读数据：（查）其他类型转换为String。

## 3、Struts2提供的常用类型转换

a.基本数据类型自动转换。

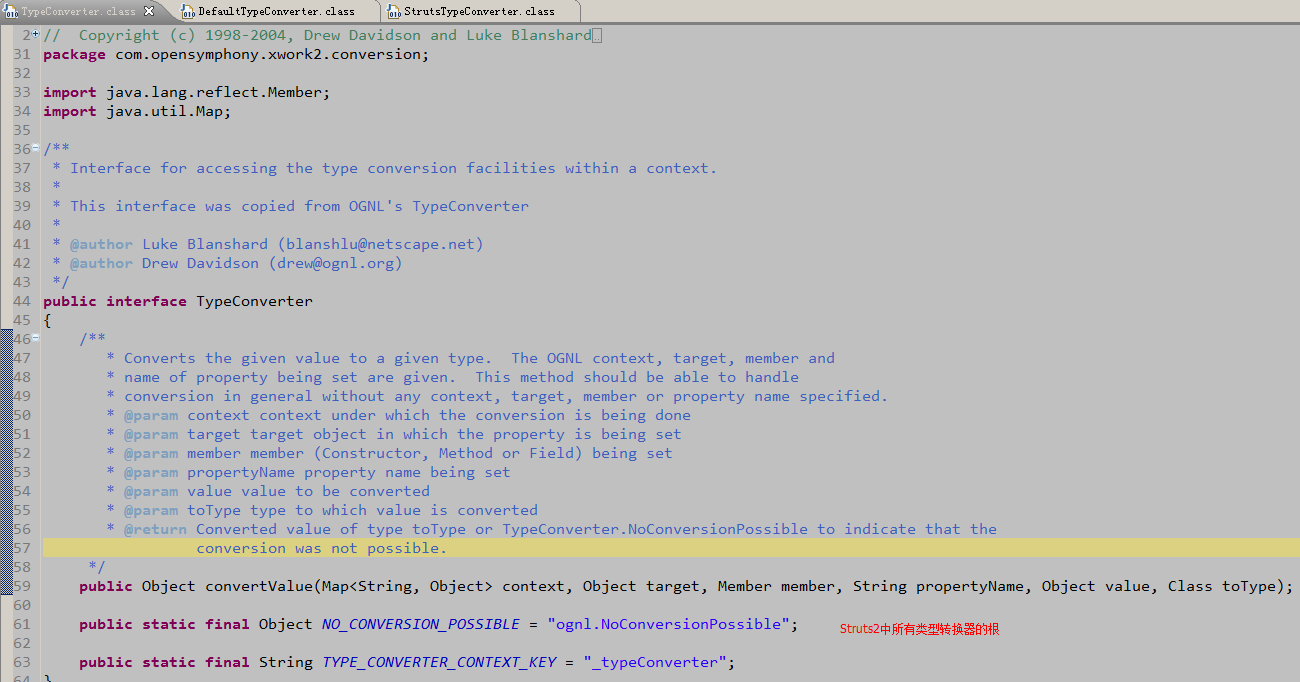
b.日期类型：默认按照本地日期格式转换（yyyy-MM-dd）。

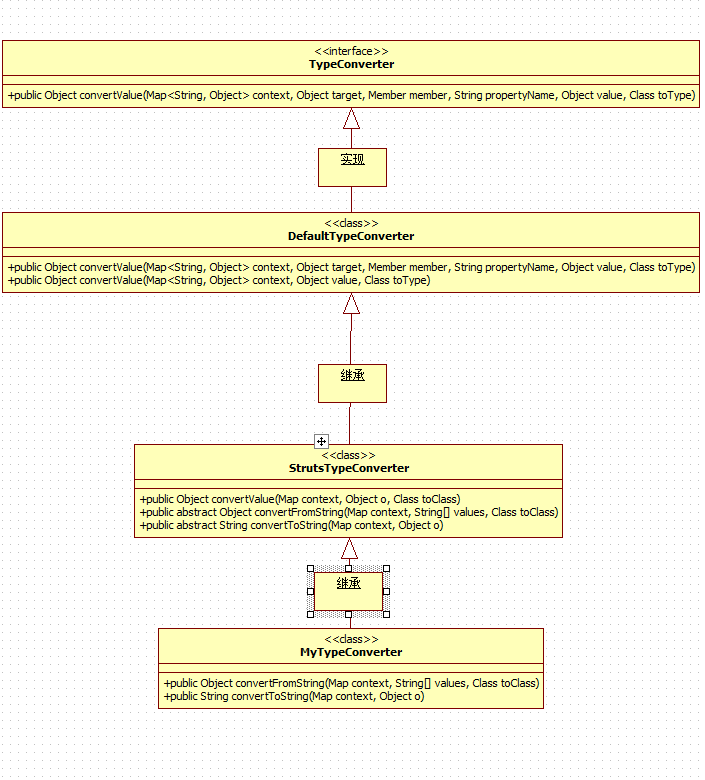
c.字符串数组：默认用逗号+空格，连接成一个字符串。

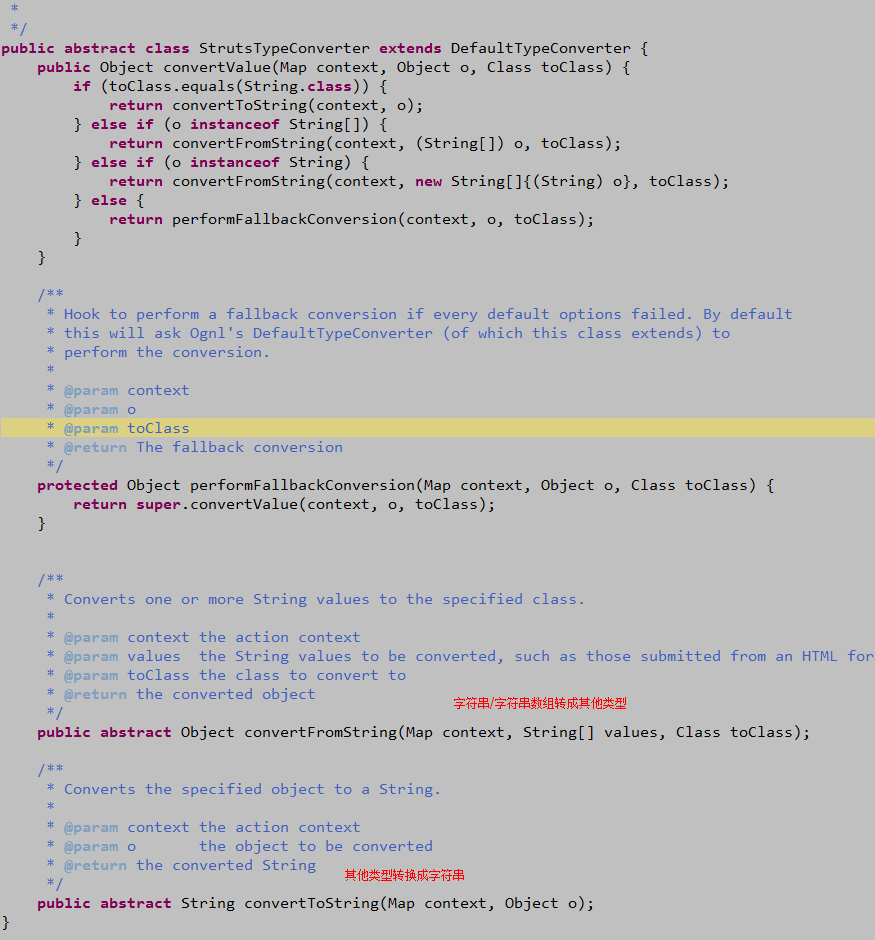
## 4、自定义类型转换器（知道）

示例：把日期格式按照 MM/dd/yyyy的格式转换

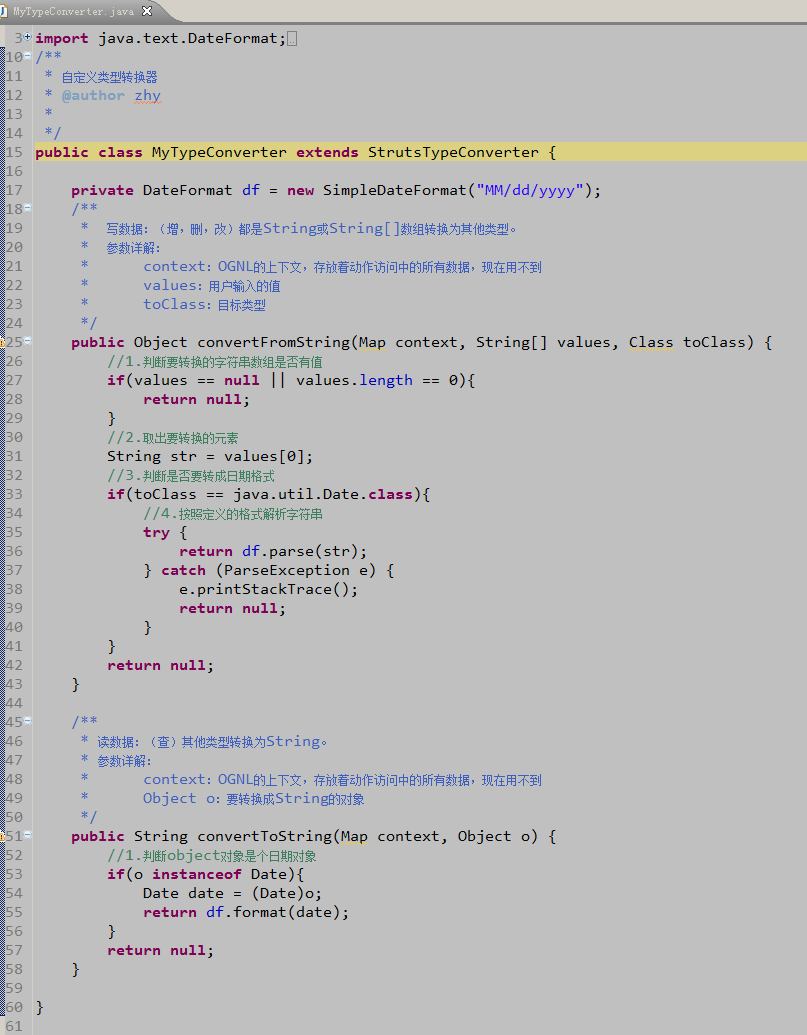
### 4.1、Struts2中的类型转换器结构：







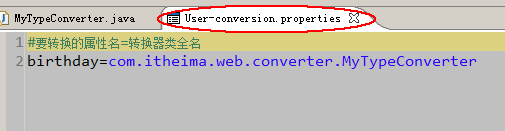
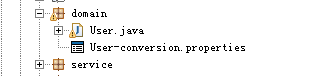
### 4.2、编写类型转换器（编写一个类继承StrutsTypeConverter，实现抽象方法）



### 4.3、注册类型转换器

**局部类型转换器：只能指定javabean中的属性用**

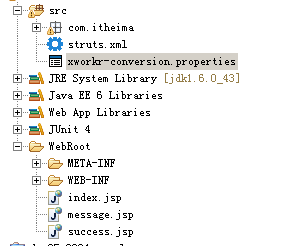
按照属性来注册。在属性所属的javabean的包下建立一个.properties文件。文件名称：javabean名称-conversion.properties

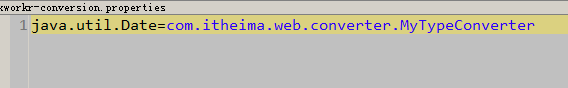
 

全局类型转换器：（推荐）

按照要转换的数据类型来注册。

at the top op classpath，建立一个固定名称xwork-conversion.properties的属性文件。



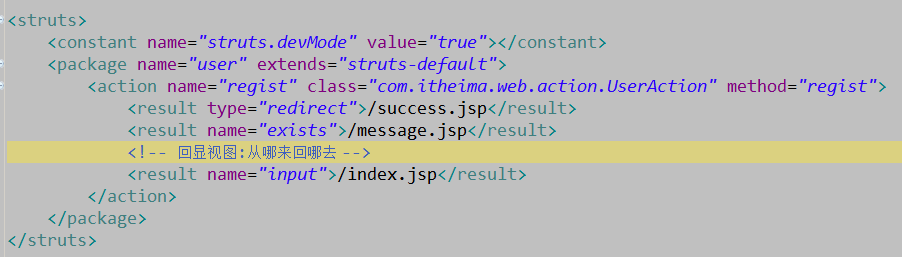


## 5、转换失败后的处理（需要掌握）

当转换失败后，页面提示：



解决办法：配置回显结果视图

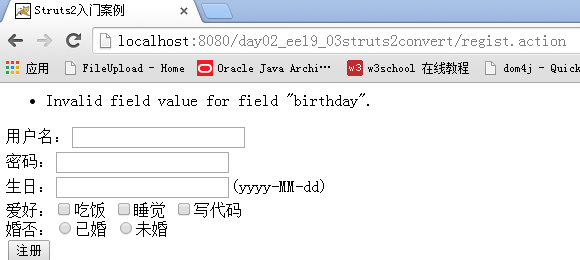


问题：

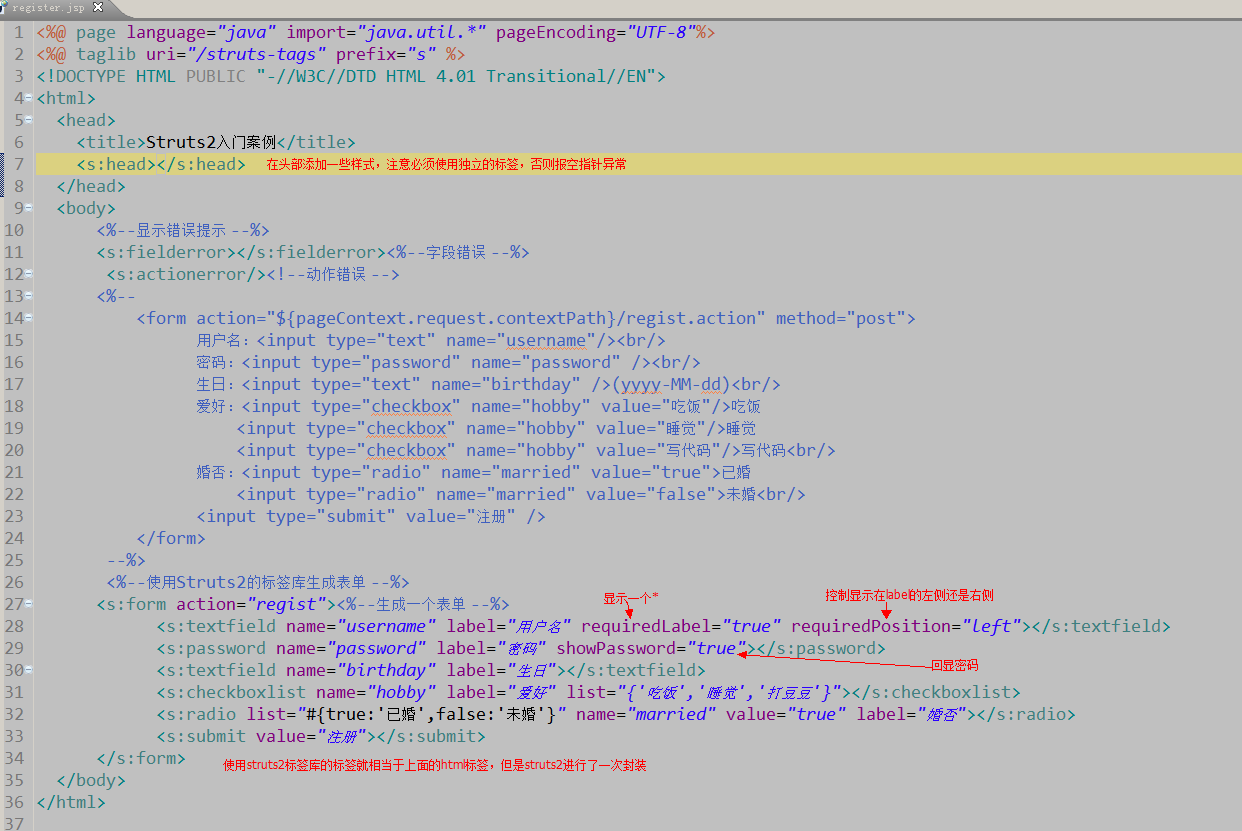
配置了回显视图后，当转换失败时，可以回到请求页面，但是表单数据都没了？

显示错误提示：借助Struts2的标签库。

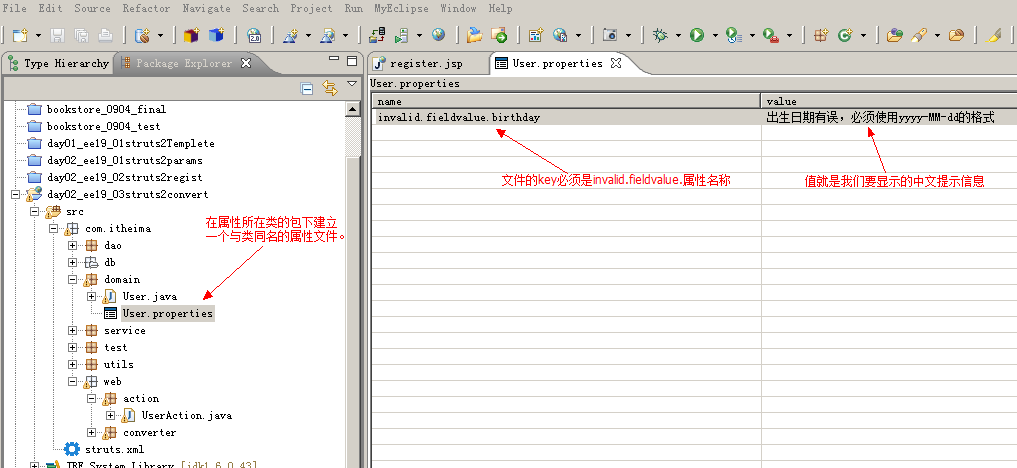


回显数据：使用struts2的标签库生成表单。（建议使用）





错误信息中文提示：使用的是struts2的国际化。





问题：

类型转换器当转换失败后，如何进入input视图的？

原因：

是由一个叫做conversionError的拦截器完成的。

注意：

要想使用类型转换中的错误处理，在定义Action时必须继承ActionSupport

# 五、数据验证

**用户的输入验证，必须做，且工作量巨大。**

## 1、验证的方式

客户端验证：javascript

服务端验证：逻辑验证（我们的代码）

注意：如果客户端和服务端二选一的话，服务器端的不能省。

实际开发中：客户端+服务端

## 2、Struts2的服务端验证

### 2.1、编程式验证

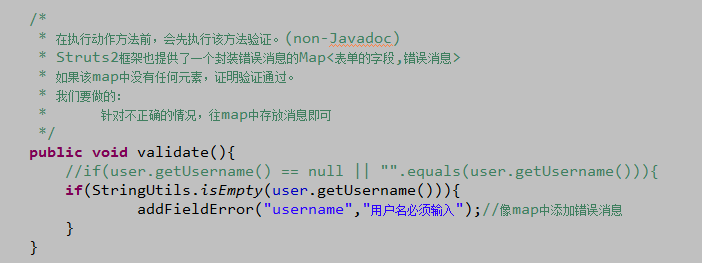
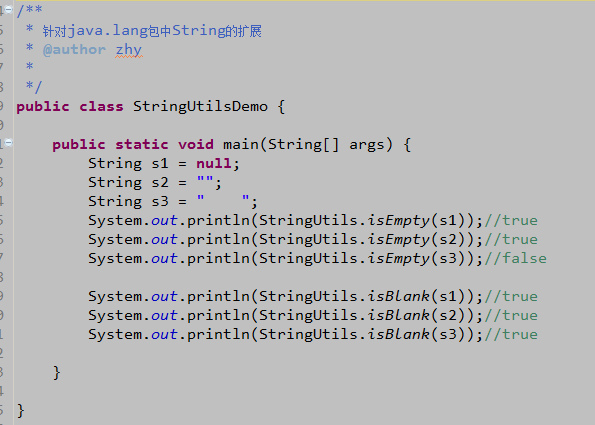
**前提：**

**动作类必须继承ActionSupport**

**在代码中编写验证规则。**

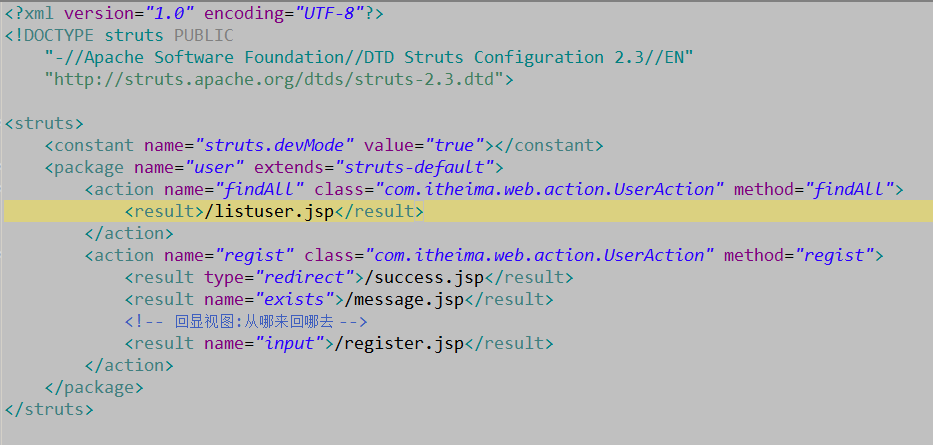
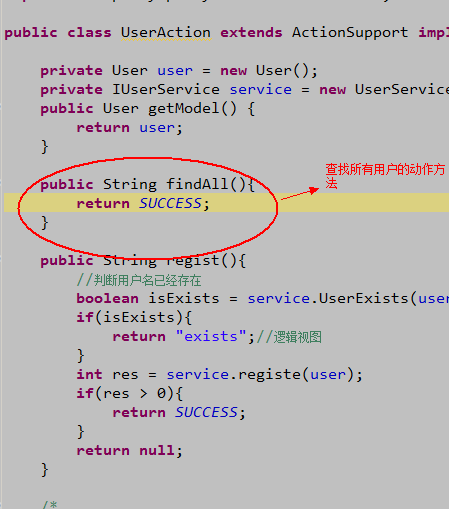
a、针对动作类中的所有动作方法进行验证：

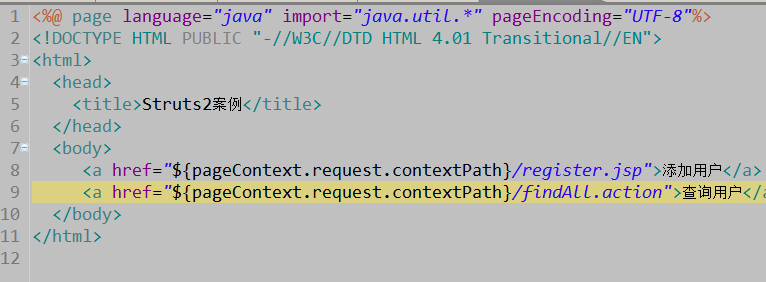
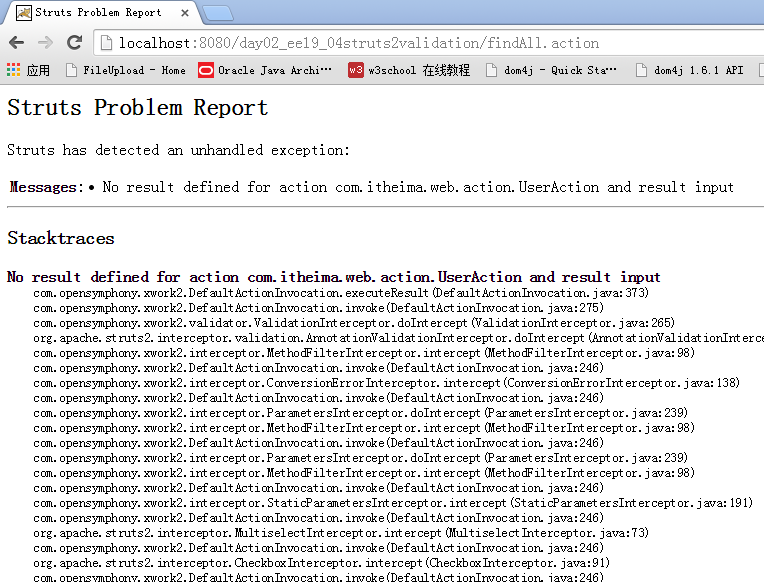
在动作类中覆盖public void validate()方法。



但是当我们再写一个动作方法时：

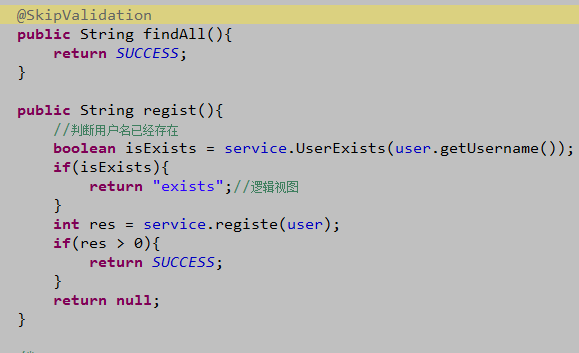
 

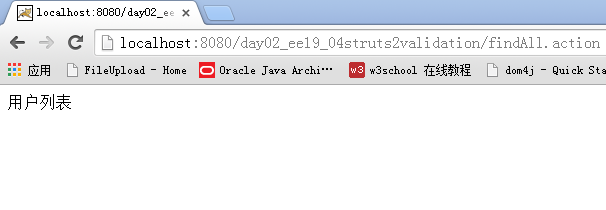
由此可知，该验证方法会对动作类中的所有动作方法进行验证。

b、针对动作类中的某个动作方法进行验证

针对上面的问题，解决办法1：给不需要验证的动作方法添加一个@SkipValidation注解。

解决办法2：validation方法遵守书写规范。

解决办法1和解决办法2的使用时机：需要验证的动作方法少，用解决办法2。需要验证的方法多，用解决方式1。（简单一点：挑少的写）

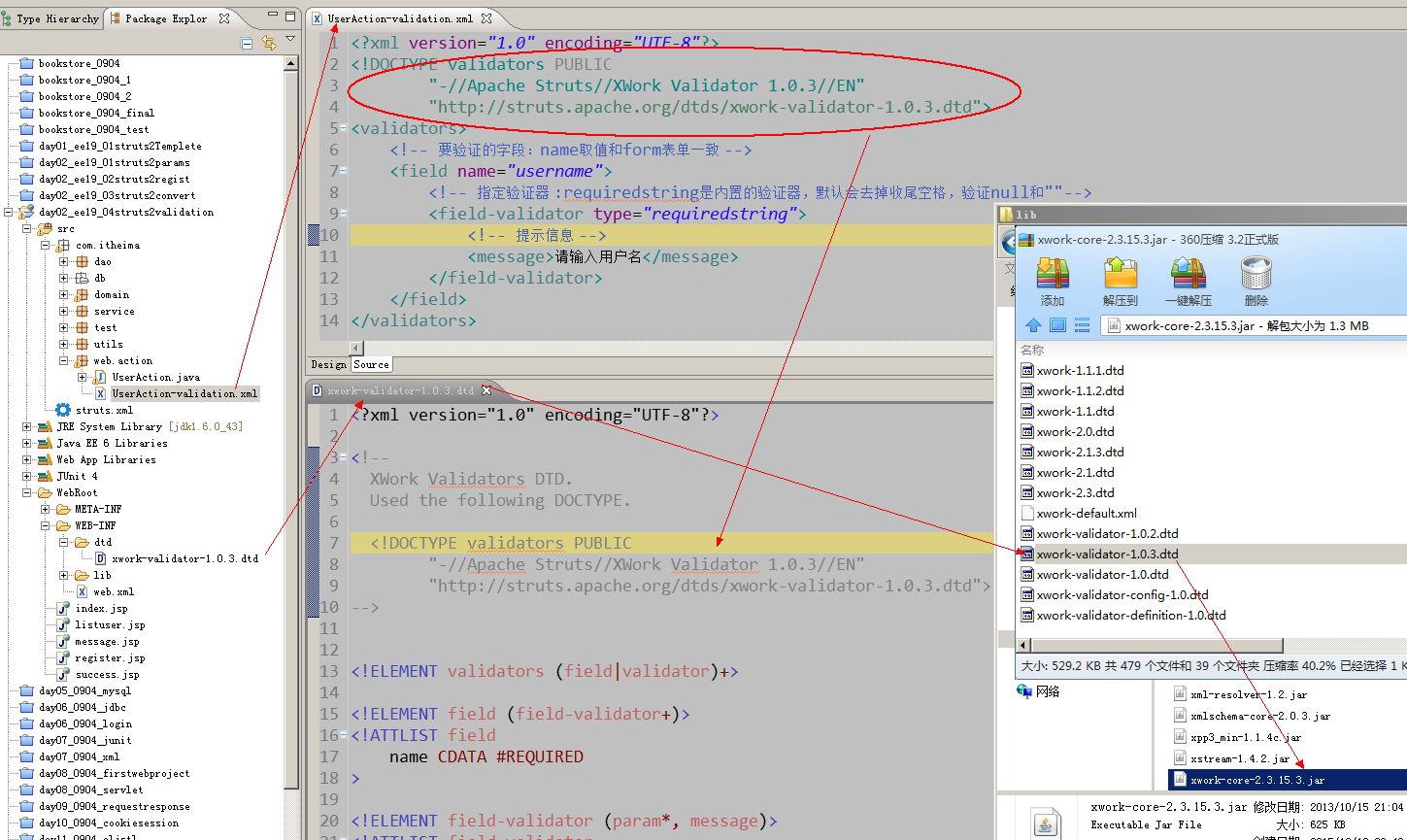
**所有编程式验证的弊端：硬编码。**

### 2.2、声明式验证（推荐）

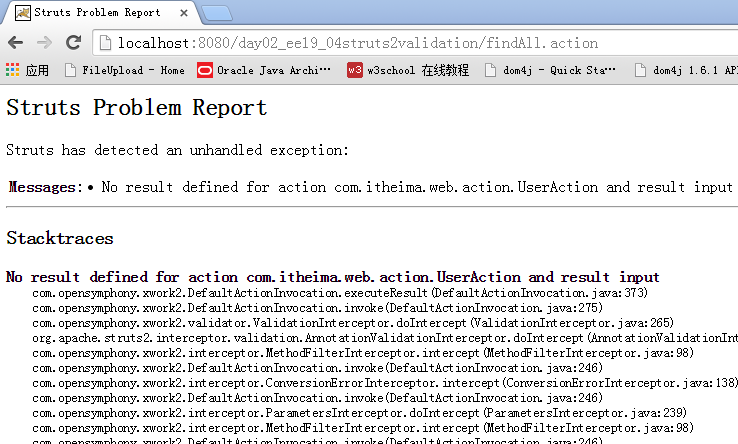
**通过编写验证规则的xml文件。需要验证时，编写xml文件，不要验证，就不写。**

**优势：解决了2.1编程式验证的弊端**

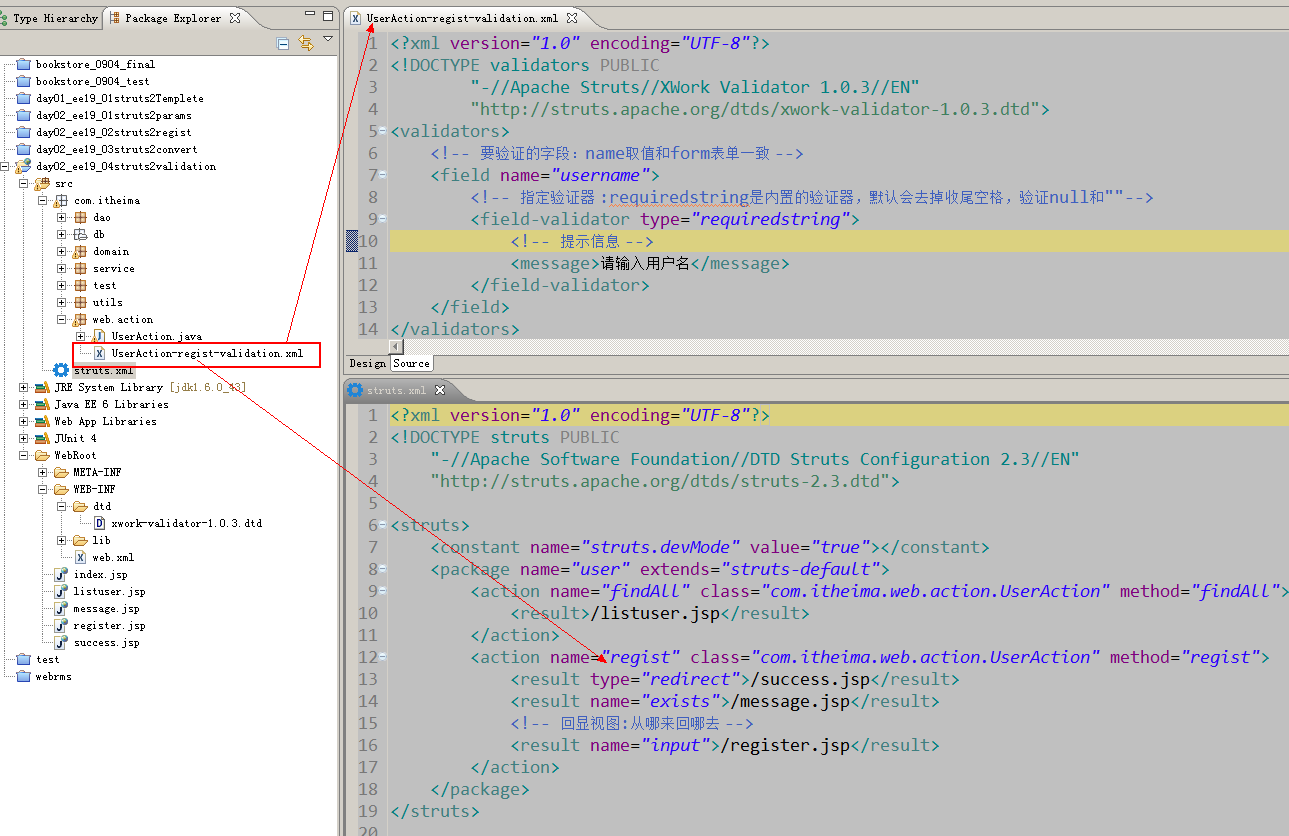
a、针对动作类中的所有动作进行验证：在动作类所在的包中，建立一个**ActionClassName-validation.xml**的文件，内容如下：



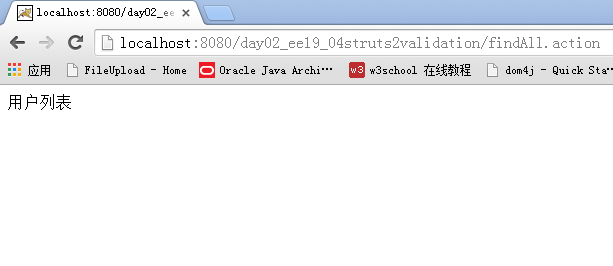
注意：它是针对动作类中的所有动作方法。

b、针对动作类中的某个动作进行验证：在动作类所在的包中建立一个xml文件，名称为**ActionClassName-ActionName-validation.xml**。内容如下：



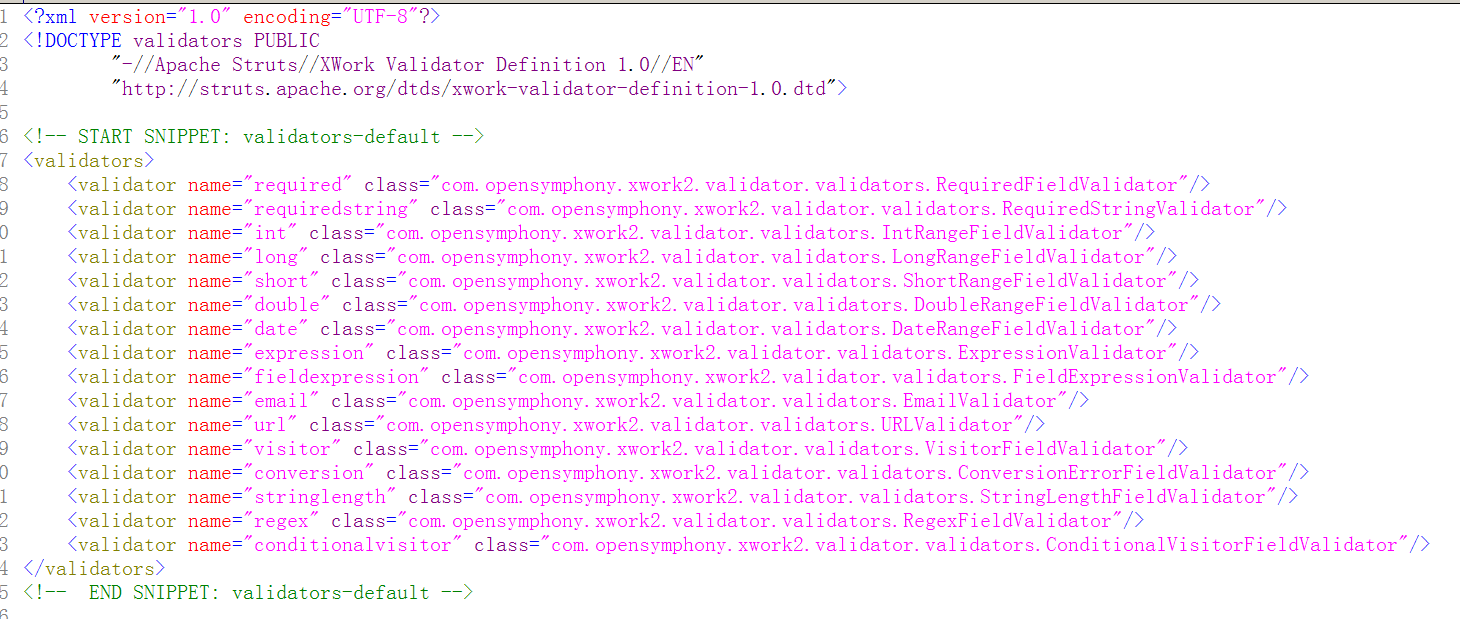
它是针对指定动作方法进行验证：

### 2.3、Struts2内置的常用声明式验证器

#### 2.3.1位置：

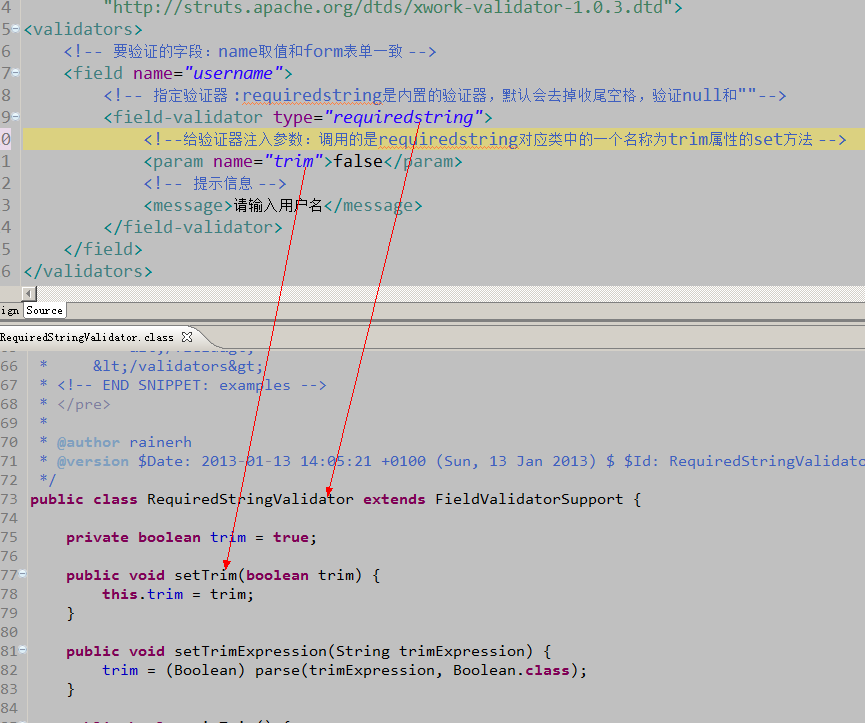
**xwork-core-2.3.15.3.jar\com\opensymphony\xwork2\validator\validator\default.xml**

****

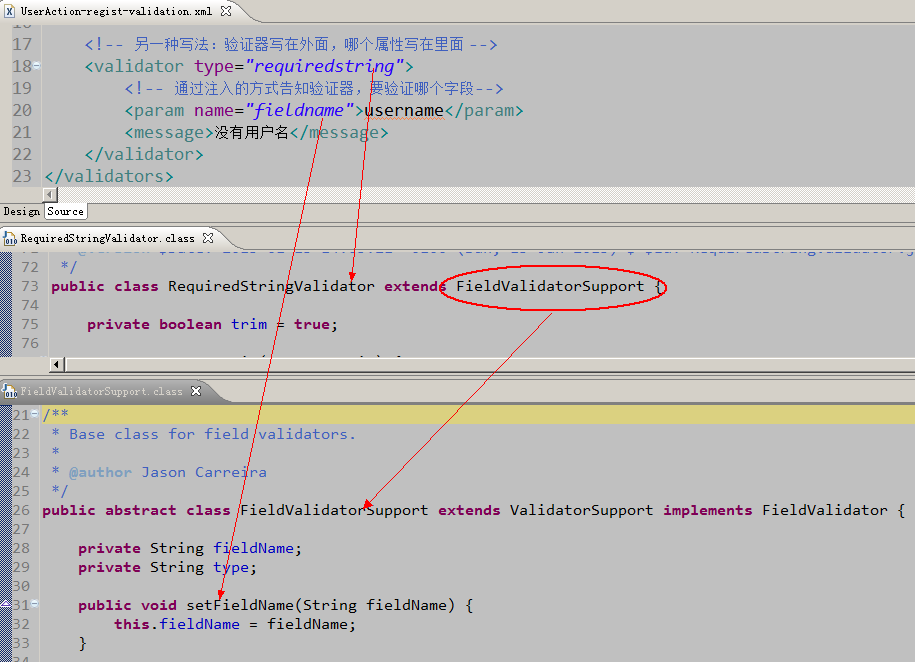
#### 2.3.2、验证器注入参数

例如：我们使用requiredstring，默认是去空格，当我们不想去空格时，就可以给验证器注入参数。

**基于字段的：**

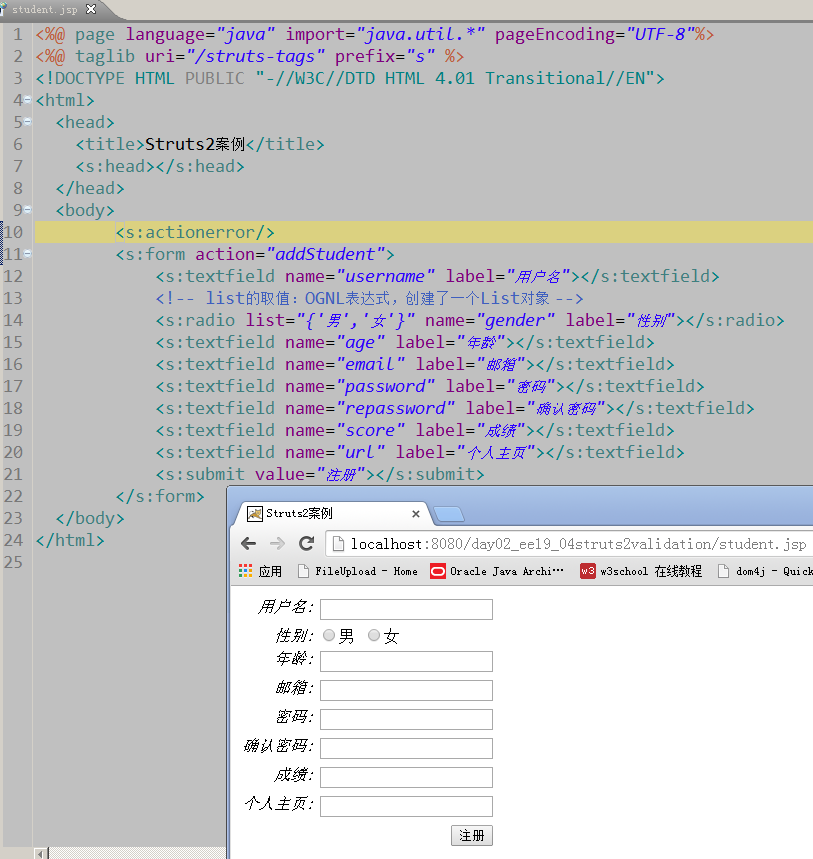


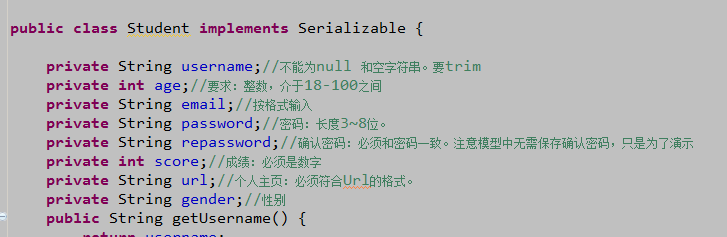
**另一种基于验证器的：**

****

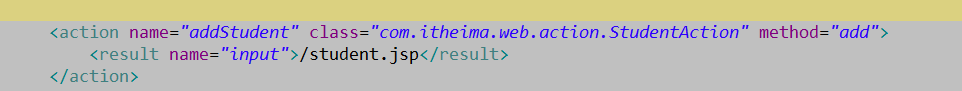
****

## 3、常用验证器示例







运行结果：

