

一、实验步骤

1.1、实验一

第一步：编写 **UserBean**，**UserService** 和 **UserAction** 的相关代码

```
public String login(){
    UserService userService = new UserService();
    if (userService.login(loginUser)){
        return "success";
    }
    return "fail";
}

public String register(){
    UserService userService = new UserService();
    if (userService.register(loginUser).isPass()){
        return "success";
    }
    return "fail";
}
```

第二步：配置 **struts.xml** 文件

```
<struts>
  <package name="strutsBean" extends="struts-default" namespace="/">
    <action name="login" class="action.UserAction" method="login">
      <result name="success">/loginSuccess.jsp</result>
      <result name="fail">/loginFailed.jsp</result>
    </action>
    <action name="register" class="action.UserAction" method="register">
      <result name="success">/registerSuccess.jsp</result>
      <result name="fail">/registerFail.jsp</result>
    </action>
  </package>
</struts>
```

第三步：配置 web.xml 文件

第四步：登录和注册调试

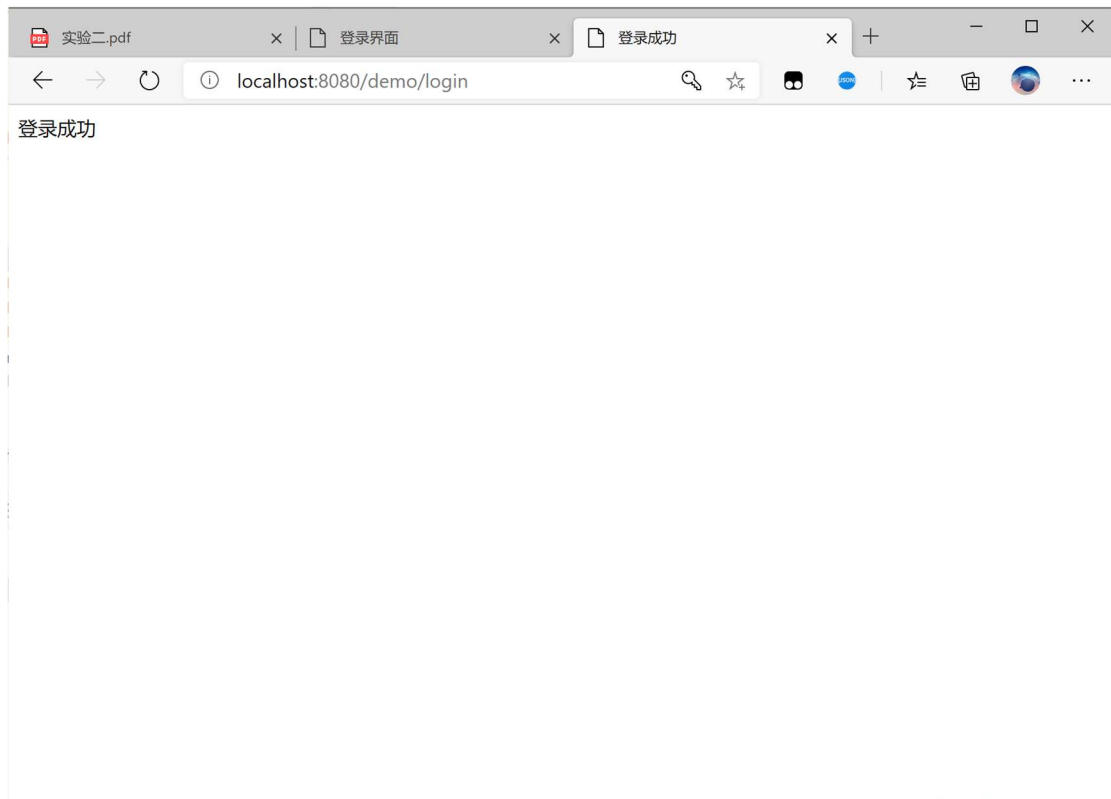



图 1 实验 1 运行结果

请输入用户名:

请输入密码:

重复输入密码:

真实姓名:

请输入生日: 

联系地址:

联系电话:

电子邮箱:

图 2 实验 1 运行结果

cht 您注册成功了!

图 3 实验 1 运行结果

第五步：测试多种 action 绑定方式

(1) 方法 1：在 struts.xml 中使用 action 绑定方法名。（传统）

(2) 方法 2：使用动态调用方式（DMI）

1. 在 struts.xml 的 struts 根内中添加如下内容

```
<constant name="struts.enable.DynamicMethodInvocation"
value="true" />
```

2. 修改 action，并修改 UserAction 类，使得四种情况下范围的结果都不同

修改后的 action 片段

```
<action name="userAction" class="action.UserAction">
  <result name="success">/loginSuccess.jsp</result>
  <result name="fail">/loginFail.jsp</result>
  <result name="registerSuccess">/registerSuccess.jsp</result>
  <result name="registerFail">/registerFail.jsp</result>
</action>
```

修改后的 UserAction 片段

```
public String login(){
    UserService userService = new UserService();
    if (userService.login(loginUser)){
        return "success";
    }
    return "fail";
}

public String register(){
    UserService userService = new UserService();
    if (userService.register(loginUser).isPass()){
        return "registerSuccess";
    }
    return "registerFail";
}
```

3. 修改调用页面的 action 以 actionName!method 的方式

```
<s:form action="UserAction!login" method="POST">
  <s:textfield name="loginUser.account"
key="login.account.lable"/>
  <s:password name="loginUser.password"
key="login.password.lable"/>
  <s:submit name="submit" key="login.submit.button"/>
</s:form>
```

4. 发现此时仍然不能以动态的方式访问 login 方法，查找资料后，发现在 struts.xml 中需要添加一个允许方法的配置，添加后的部分代码如下

```
<package name="strutsBean" extends="struts-default" namespace="/">
  <global-allowed-methods>login,register</global-allowed-methods>
  <action name="userAction" class="action.UserAction">
    <result name="success">/loginSuccess.jsp</result>
    <result name="fail">/loginFail.jsp</result>
    <result name="registerSuccess">/registerSuccess.jsp</result>
    <result name="registerFail">/registerFail.jsp</result>
  </action>
</package>
```

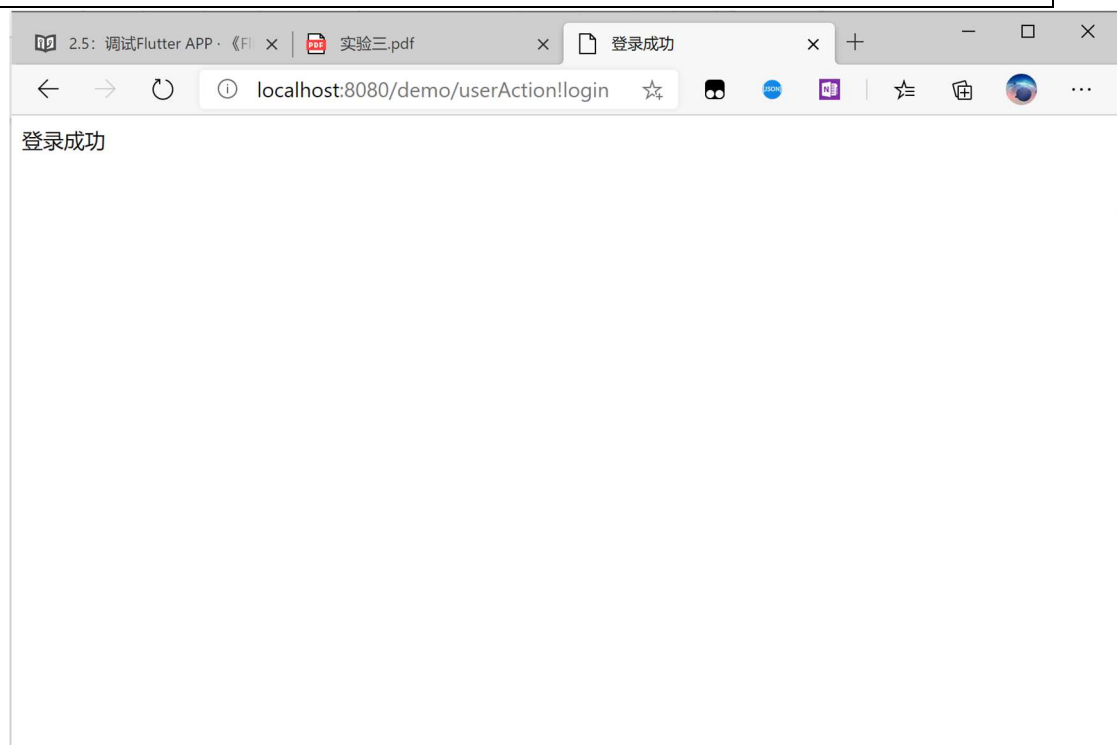


图 4 实验 1 运行结果

(3) 方法三：使用 s:submit 中的 method 属性（此时 struts.xml 的配置与方法（2）相同）

修改后的表单片段

```

<s:form action="userAction" method="POST">
  <s:textfield name="loginUser.account"
key="login.account.lable"/>
  <s:password name="loginUser.password"
key="login.password.lable"/>
  <s:submit name="submit" key="login.submit.button"
method="login"/>
</s:form>

```

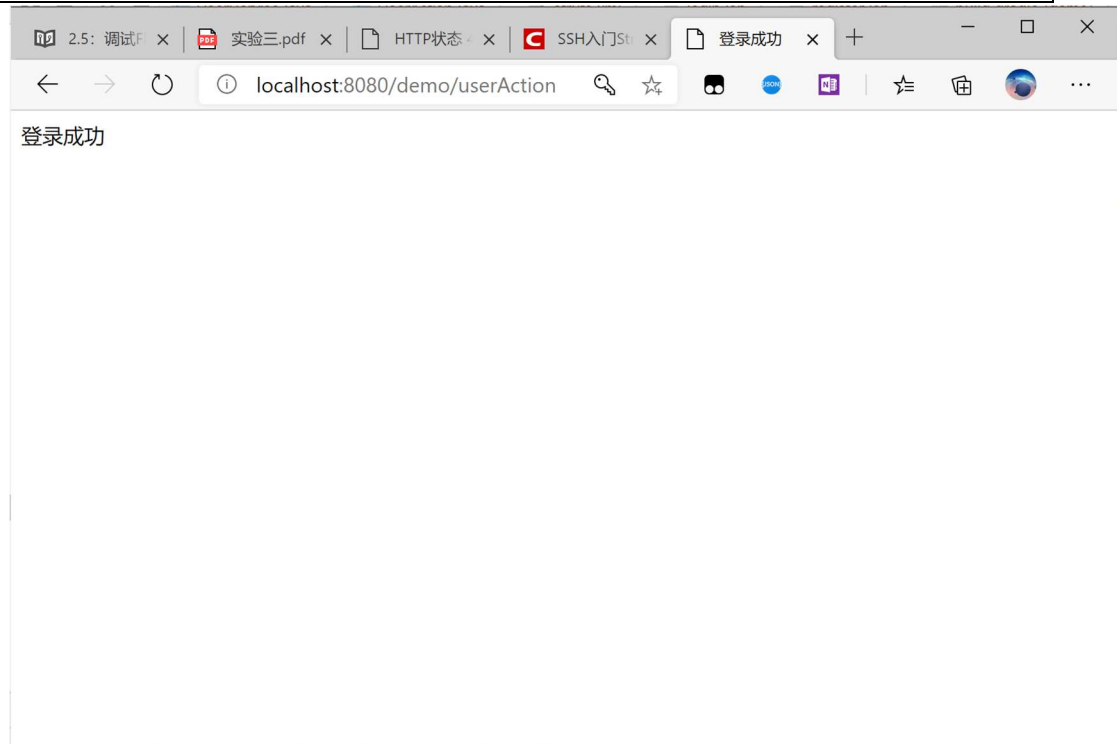


图 5 实验 1 运行结果

(4) 方法四：使用通配符配置 action

1. 修改 struts.xml

```

<constant name="struts.enable.DynamicMethodInvocation"
value="true"/>
<package name="strutsBean" extends="struts-default" namespace="/">
  <global-allowed-methods>login,register</global-allowed-methods>
  <action name="*Action" class="action.UserAction" method="{1}">
    <result name="success">/{1}Success.jsp</result>
    <result name="fail">/{1}Fail.jsp</result>
  </action>
</package>

```

其中*表示通配符，如果对应的 action 为 loginAction，则{1}=login，则会调用到 login 方法。后面的返回页面中的{1}也是同一个道理。

2. 修改 UserAction 使得返回结果为 success 或者 fail，当然，也可以在 name 属性中使用 {1}。

```
public String login() {
    UserService userService = new UserService();
    if (userService.login(loginUser)) {
        return "success";
    }
    return "fail";
}

public String register() {
    UserService userService = new UserService();
    if (userService.register(loginUser).isPass()) {
        return "success";
    }
    return "fail";
}
```

3. 修改 form 表单，action 为 loginAction 和 registerAction

```
<s:form action="loginAction" method="POST">
    <s:textfield name="loginUser.account"
key="login.account.lable"/>
    <s:password name="loginUser.password"
key="login.password.lable"/>
</s:form>
```

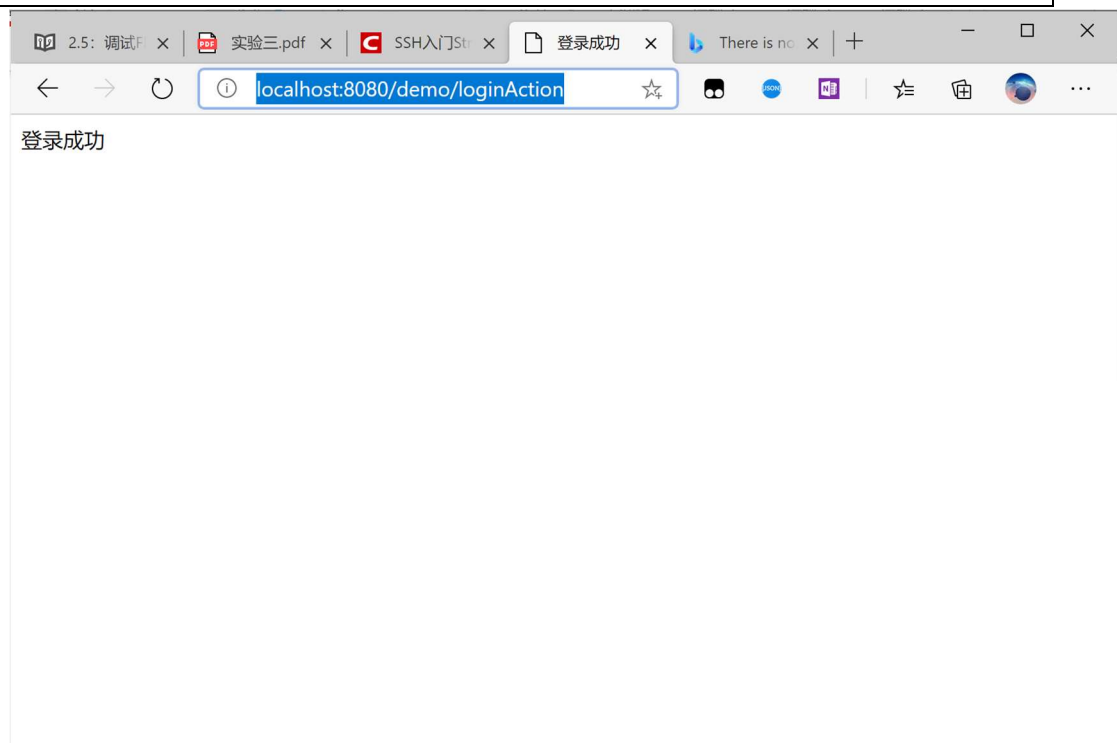


图 6 实验 1 运行结果

第六步：测试 action 的生命周期。

无论刷新多少次，count 的值都为 1，这表明每一次请求都会新建一个 UserAction 实例。

1					
编号	名称	说明	单价	数量	
book001	JavaEE技术实验指导课程	WEB 程序设计知识回顾、轻量级 JAVAEE 应用框架、企业级 EJB 组件编程技术、JAVAEE 综合应用开发.	19.95	2	

图 7 实验 1 运行结果

创建了一个UserAction类对象
创建了一个UserAction类对象
创建了一个UserAction类对象

图 8 实验 1 运行结果

将 type 改成 redirect 后，无论刷新多少次，都不显示 count 的值，此时 count 为 null。

图 9 实验 1 运行结果

编号	名称	说明	单价	数量
book001	JavaEE技术实验指导课程	WEB 程序设计知识回顾、轻量级 JAVAEE 应用框架、企业级 EJB 组件编程技术、JAVAEE 综合应用开发.	19.95	2

1.2、实验二

第一步：添加 validate 方法和在 struts 中配置 input 的 result

```
public void validateLogin() {  
    String account = loginUser.getAccount();  
    String password = loginUser.getPassword();  
    if (account == null || account.equals("")) {  
        this.addFieldError("loginUser.account",  
            getText("login.account.error"));  
    }  
    if (password == null || password.equals("")) {  
        this.addFieldError("loginUser.password",  
            getText("login.password.error"));  
    }  
}
```

```
}  
}
```

```
<action name="*Action" class="action.UserAction" method="{1}">  
  <result name="success">/{1}Success.jsp</result>  
  <result name="fail">/{1}Fail.jsp</result>  
  <result name="input">/{1}.jsp</result>  
</action>
```

请输入您的用户名!

请输入用户名:

请输入您的密码!

请输入密码:

提交

第二步：添加<s:fielderror>标签

- 请输入您的用户名!
- 请输入您的密码!

请输入您的用户名!

请输入用户名:

请输入您的密码!

请输入密码:

提交

第三步：修改 UserAction 的 login 方法，并添加<s:actionerror>，

<s:actionmessage>标签

UserAction.java 的部分代码

```
public String login() {  
    UserService userService = new UserService();  
    if (userService.login(loginUser)) {  
        this.addActionMessage(getText("login.action.success"));  
        return "success";  
    }  
    this.addActionError(getText("login.action.fail"));  
    return "fail";  
}
```


struts.xml 的部分代码

```
<action name="*Action" class="action.UserAction" method="{1}">
  <result name="success">/{1}Success.jsp</result>
  <result name="fail">/{1}.jsp</result>
  <result name="input">/{1}.jsp</result>
</action>
```

login.jsp 的部分代码

```
<s:actionerror/>
<s:fielderror/>
<s:form action="loginAction" method="POST">
  <s:textfield name="loginUser.account"
key="login.account.lable"/>
  <s:password name="loginUser.password"
key="login.password.lable"/>
  <s:submit name="submit" key="login.submit.button"/>
</s:form>
```

loginSuccess.jsp 的部分代码

```
<s:actionmessage/>
```

- 用户名或密码错误，请重新输入！

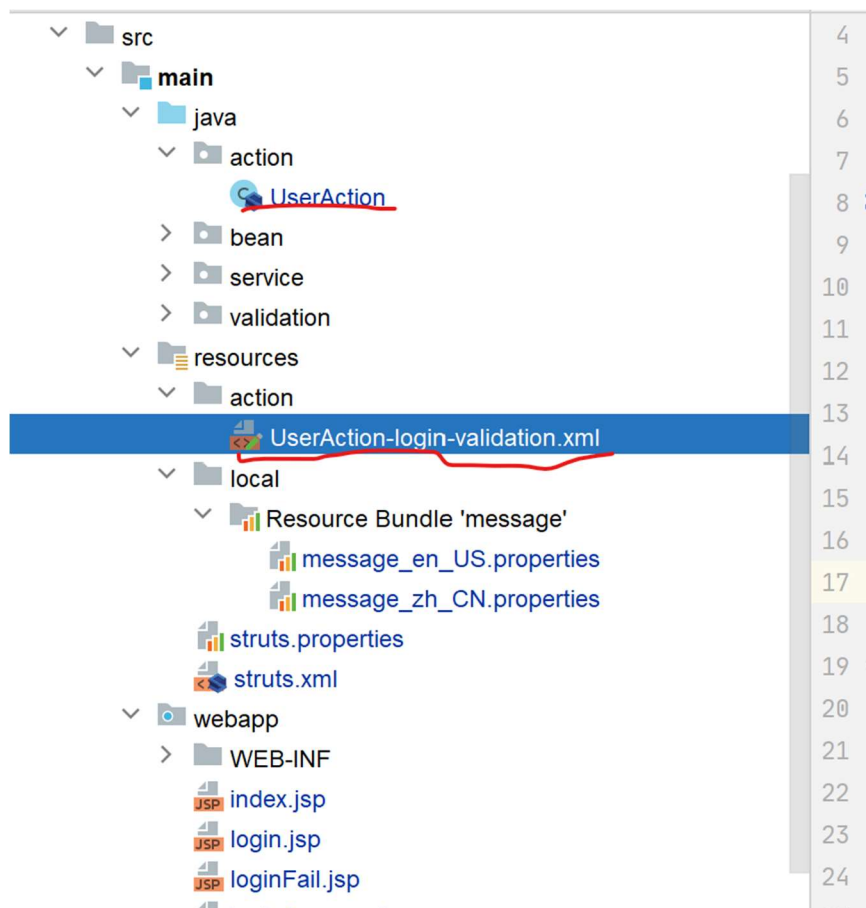
请输入用户名:

请输入密码:

- 登录成功！

第四步：编写 UserAction-login-validation.xml 文件

问题：注意放在 java 文件夹下是不会将此拷贝到生成目录下的，需要放在 resources 中的对应目录。如下图所示。



- 用户名不能为空
- 密码不能为空

用户名不能为空

请输入用户名:

密码不能为空

请输入密码:

第五步：对校验器和 **UserAction** 进行国际化

UserAction-login-validation.xml 中的代码

```
<validators>
<field name="loginUser.account">
  <field-validator type="requiredstring">
    <param name="trim">true</param>
    <message key="login.account.error"/>
  </field-validator>
</field>
<field name="loginUser.password">
```

```

        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message key="login.password.error"/>
        </field-validator>
    </field>
</validators>

```

UserAction 中的代码

```

public String login() {
    UserService userService = new UserService();
    if (userService.login(loginUser)) {
        this.addActionMessage(getText("login.action.success"));
        return "success";
    }
    this.addActionError(getText("login.action.fail"));
    return "fail";
}

```

- 请输入您的用户名!
- 请输入您的密码!

请输入您的用户名!

请输入用户名:

请输入您的密码!

请输入密码:

第六步：将 UserBean 中的 birthday 字段改为 java.util.Date 类型

```

private String repassword;
private String name;
private String sex;
private Date birthday;
private String address;
private String phone;
private String email;

```

请输入用户名:

请输入密码:

请重新输入密码:

真实姓名:

真实性别: 男

真实生日:

联系地址:

联系电话:

电子邮箱:

第七步：添加类型转换的校验信息

请输入用户名:

请输入密码:

请重新输入密码:

真实姓名:

真实性别: 男

真实生日:

联系地址:

联系电话:

电子邮箱:

第八步：添加 **register** 的输入校验

UserAction-register-validation.xml 的内容

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE validators PUBLIC
    "-//Apache Struts//XWork Validator 1.0.2//EN"
    "http://struts.apache.org/dtds/xwork-validator-1.0.2.dtd">
<validators>
    <field name="loginUser.account">
        <field-validator type="requiredstring">
            <message key="login.account.error"/>
        </field-validator>
    </field>
```

```
<field name="loginUser.password">
  <field-validator type="requiredstring">
    <message key="login.password.error"/>
  </field-validator>
</field>
<field name="loginUser.repassword">
  <field-validator type="fieldexpression">
    <param
name="expression"><![CDATA[password==repassword]]></param>
    <message key="login.repassword.error"/>
  </field-validator>
</field>
<field name="loginUser.name">
  <field-validator type="requiredstring">
    <message key="login.name.error"/>
  </field-validator>
</field>
<field name="loginUser.address">
  <field-validator type="requiredstring">
    <message key="login.address.error"/>
  </field-validator>
</field>
<field name="loginUser.phone">
  <field-validator type="stringlength">
    <param name="minLength">5</param>
    <param name="maxLength">15</param>
    <message key="login.phone.error"/>
  </field-validator>
</field>
<field name="loginUser.email">
  <field-validator type="email">
    <message key="login.email.error"/>
  </field-validator>
</field>
</validators>
```

- 请输入您的用户名!
- 请输入您的密码!
- 密码和确认密码需要相同。
- 真实姓名不能为空
- 联系地址不能为空
- 联系电话的长度应为5-15位
- 电子邮箱应当符合对应的格式。

请输入您的用户名!

请输入用户名:

请输入您的密码!

请输入密码:

密码和确认密码需要相同。

请重新输入密码:

真实姓名不能为空

真实姓名:

真实性别: 男

真实生日:

联系地址不能为空

联系地址:

联系电话的长度应为5-15位

联系电话:

电子邮箱应当符合对应的格式。

电子邮箱:

第九步：添加 **regiter** 的手动校验

```
public void validateRegister() {
    String emailRegular = "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-9-]+)*@[A-Za-z0-9-9-]+(\\.[A-Za-z0-9-9-]+)* (\\.[A-Za-z]{2,})$";
    //begins.. account
    String account = loginUser.getAccount();
    if (account == null || account.equals("")) {
        this.addFieldError("loginUser.account",
            getText("login.account.error"));
    }
    String password = loginUser.getPassword();
    if (password == null || password.equals("")) {
```

```

        this.addFieldError("loginUser.password",
getText("login.password.error"));
    }
    String repassword = loginUser.getRepassword();
    if(repassword == null || !repassword.equals(password)) {
        this.addFieldError("loginUser.repassword",
getText("login.repassword.error"));
    }
    String name = loginUser.getName();
    if (name == null || name.equals("")) {
        this.addFieldError("loginUser.name",
getText("login.name.error"));
    }
    String address = loginUser.getAddress();
    if (address == null || address.equals("")) {
        this.addFieldError("loginUser.address",
getText("login.address.error"));
    }
    String phone = loginUser.getPhone();
    if( phone == null || phone.length() < 5 || phone.length() >
15){
        this.addFieldError("loginUser.phone",
getText("login.phone.error"));
    }

    String email = loginUser.getEmail();
    if(email == null || !email.matches(emailRegular) {
        this.addFieldError("loginUser.email",
getText("login.email.error"));
    }
}

```

- 请输入您的用户名!
- 请输入您的密码!
- 密码和确认密码需要相同。
- 真实姓名不能为空
- 联系地址不能为空
- 联系电话的长度应为5-15位
- 电子邮箱应当符合对应的格式。

请输入您的用户名!
 请输入用户名:

请输入您的密码!
 请输入密码:

密码和确认密码需要相同。
 请重新输入密码:

真实姓名不能为空
 真实姓名:

真实性别: ▼

真实生日: 

联系地址不能为空
 联系地址:

联系电话的长度应为5-15位
 联系电话:

电子邮箱应当符合对应的格式。
 电子邮箱:

1.3、实验三

第一步：使用域 map（UserAction.java 的部分代码）

```
private final static String REQUEST = "request";
private final static String COUNTER = "counter";
private final static String FAIL = "fail";
private final static String USER = "user";
private final static String TIP = "tip";

@SuppressWarnings({"unchecked"})
public String login() {
    // 获取 ActionContext 对象
```



```

ApplicationContext applicationContext = ApplicationContext.getContext();
request = (Map<String, Object>) applicationContext.get(REQUEST);
session = applicationContext.getSession();
application = applicationContext.getApplication();

// 访问 application 范围的属性值
Integer counter = (Integer) application.get(COUNTER);
if (counter == null) {
    counter = 1;
} else {
    ++counter;
}

// 设置 application 范围的属性
application.put(COUNTER, counter);

UserService userService = new UserService();
if (userService.login(loginUser)) {
    this.addActionMessage(getText("login.action.success"));
    session.put(USER, loginUser.getAccount());
    request.put(TIP, "您已登录成功");
    return SUCCESS;
}
this.addActionError(getText("login.action.fail"));
return FAIL;
}

```

loginSuccess.jsp 的部分片段

```

<s:actionmessage/>
本站访问次数为: <s:property value="#application.counter"/><br/>
<s:property value="#session.user"/>,
<s:property value="#request.tip"/>

```

- 登录成功!

本站访问次数为: 1
12, 您已登录成功

第二步: 使用 ApplicationAware, RequestAware, SessionAware

```

@Override
public void setApplication(Map<String, Object> application) {
    this.application = application;
}

```

```

}

@Override
public void setRequest(Map<String, Object> request) {
    this.request = request;
}

@Override
public void setSession(Map<String, Object> session) {
    this.session = session;
}

```

- 登录成功!

本站访问次数为: 1
12, 您已登录成功

第三步: 使用 ServletContextAware, ServletResponseAware, ServletRequestAware

UserAction 的部分代码

```

private HttpServletRequest request;
private HttpServletResponse response;
private ServletContext context;

@SuppressWarnings({"unchecked"})
public String login() {
    // 访问 application 范围的属性值
    Integer counter = (Integer) context.getAttribute(COUNTER);
    if (counter == null) {
        counter = 1;
    } else {
        ++counter;
    }
    // 设置 application 范围的属性
    context.setAttribute(COUNTER, counter);

    HttpSession session = request.getSession();

    UserService userService = new UserService();
    if (userService.login(loginUser)) {
        this.addActionMessage(getText("login.action.success"));
        session.setAttribute(USER, loginUser.getAccount());
    }
}

```

```

        request.setAttribute(TIP, "您已登录成功");
        return SUCCESS;
    }
    this.addActionError(getText("login.action.fail"));
    return FAIL;
}

@Override
public void setServletRequest(HttpServletRequest request) {
    this.request = request;
}

@Override
public void setServletResponse(HttpServletResponse response) {
    this.response = response;
}

@Override
public void setServletContext(ServletContext context) {
    this.context = context;
}

```

- 登录成功!

本站访问次数为: 1
123, 您已登录成功

第四步：使用 ServletActionContext 工具类获取 ServletContext，

HttpServletRequest 对象

```

context = ServletActionContext.getServletContext();
request = ServletActionContext.getRequest();

```

- 登录成功!

本站访问次数为: 2
123, 您已登录成功

第五步：购物车对象存储在 Session 中

UserAction.login()的部分片段

```
if (userService.login(loginUser)) {  
    this.addActionMessage(getText("login.action.success"));  
    ShoppingCart shoppingCart = new ShoppingCart();  
    session.setAttribute("shoppingCart", shoppingCart);  
    return SUCCESS;  
}
```

loginSuccess.jsp 的部分片段

```
<table style="border: 1px">  
  <tr>  
    <th>编号</th>  
    <th>名称</th>  
    <th>说明</th>  
    <th>单价</th>  
    <th>数量</th>  
  </tr>  
  <s:iterator value="#session.shoppingCart.itemOrderList"  
var="item">  
    <tr>  
      <td><s:property value="item.itemId"/> </td>  
      <td><s:property value="item.name"/> </td>  
      <td><s:property value="item.description"/> </td>  
      <td><s:property value="item.cost"/> </td>  
      <td><s:property value="numItems"/> </td>  
    </tr>  
  </s:iterator>  
</table>
```

编号	名称	说明	单价	数量
book001	JavaEE技术实验指导课程	WEB 程序设计知识回顾、轻量级 JAVAEE 应用框架、企业级 EJB 组件编程技术、JAVAEE 综合应用开发。	19.95	2

二、实验结果及分析

2.1、实验一

(3) Action 的实例化情况

Action 实例将在每次调用到该 Action 时会创建一个新的实例，因此每次访问 login()时，都会创建一个 UserAction 实例，这将会导致每次 count 的初始值为 0，执行++count 操作后，count 始终为 1。但是 Servlet 会在 Web 容器初始化时初

始化，然后装入 Web 容器。

(4) Jsp 文件中获取 Action 属性的主要过程。

创建 Action 实例时，其所有的实例变量都会加入值栈，因此，在<s:property>中，就能获取到 Action 的实例变量。

(5) type=redirect 的区别

forward 表示先前传递，将会保留 Request, Session, Application 和 Action 域的变量，而 redirect 表示重定向，将不会保留 Request 和 Action 域内的变量，因此使用 type=redirect 时，将不会读取到 count 的值，因此不显示。

(6) result 类型

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

(7) 碰到的问题及解决方案或思考

出现于实验 1 第 10 题（使用 DMI 的方式进行动态访问），发现按照实验的步骤进行配置后出现错误“无法调用 UserAction.login 方法”，经检查没有出现拼

写问题后，尝试网络搜索，发现少了一个配置项。

```
<constant                name="struts.enable.DynamicMethodInvocation"
value="true"/>

<package name="strutsBean" extends="struts-default" namespace="/">

    <global-allowed-methods>login,register</global-allowed-methods>

    <action name="*Action" class="action.UserAction" method="{1}">

        <result name="success">/{1}Success.jsp</result>

        <result name="fail">/{1}Fail.jsp</result>

    </action>

</package>
```

需要在 action 内添加 global-allowed-methods 的选项，但是实验操作说明书中并没有出现这一问题。

继续查找资料和询问同学后发现了问题：

```
dependencies {
    compileOnly('javax:javaee-web-api:8.0.1')

    testImplementation("org.junit.jupiter:junit-jupiter-api:${junitVersion}")
    testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:${junitVersion}")

    // 加入struts2-core核心库
    compile group: 'org.apache.struts', name: 'struts2-core', version: '2.5.25'
    // 使用dojo扩展库
    compile group: 'org.apache.struts', name: 'struts2-dojo-plugin', version: '2.3.37'
}
```

此项目使用的 struts2-core 库的版本 2.5.25，而实验的版本是 2.3.15。2.5 版本为了动态访问的安全性，添加了 strict-mode，而且默认是 open 模式，因此需要添加 global-allowed-methods 选项。

或者修改成下列的情况也能解决问题。

```
<package name="strutsBean" extends="struts-
default" namespace="/" strict-method-invocation="false">
```

（8）实验收获

通过第一个实验，了解了 struts 框架的运行流程和运行流程，掌握了多种 action 的访问方式，对于 MVC 框架的设计理念也有了一定的了解。通过实验区分了 Action 和 Servlet 生命周期模型的不同特征，以及 struts2 框架是一个非侵入式框架的概念。

2.2、实验二

(3) 校验规则文件的作用

各种校验规则及其用法

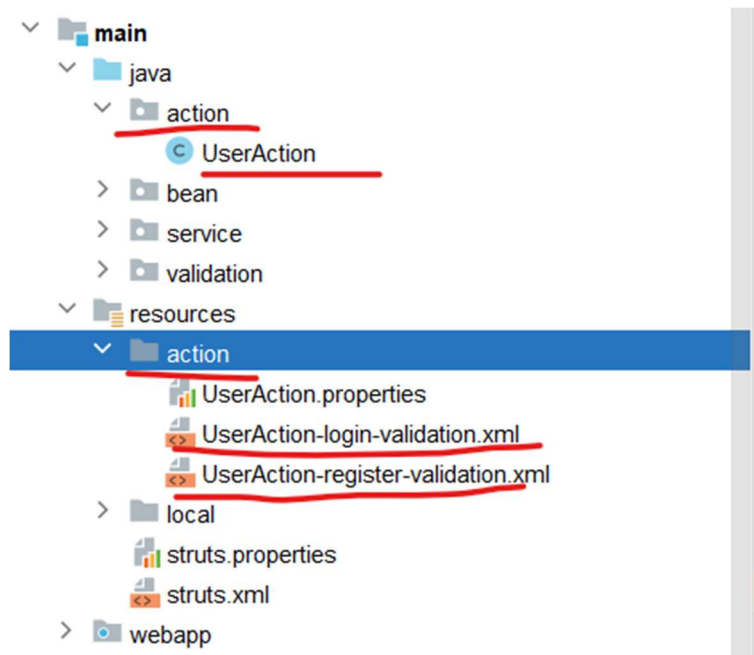
```
<validators>
  <validator name="required"
class="com.opensymphony.xwork2.validator.validators.RequiredFieldValidator"/>
  <validator name="requiredstring"
class="com.opensymphony.xwork2.validator.validators.RequiredStringValidator"/>
  <validator name="int"
class="com.opensymphony.xwork2.validator.validators.IntRangeFieldValidator"/>
  <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="creditcard"
class="com.opensymphony.xwork2.validator.validators.CreditCardValidator"/>
  <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.ConditionalVisitorFieldValidator"/>
</validators>
```

类型	说明	用法
----	----	----

required	必填	
requiredstring	必须包含非空字符	
int	整型的范围值	min~max/minExpression~max Expression
long	长整型的范围值	min~max
short	短整型的范围值	min~max
double	双精度浮点数的范围值	min~max
date	日期的范围值	min~max
expression	正则表达式	expression, message
field	与字段比较	expression, message
email	邮箱	
creditcard	不适用与中国	
url	一个资源定位符	
vistor	复合校验器	context,appendPrefix
conversion	转换校验器	
stringlength	字符串长度	minLength, maxLength, trim
regex	正则表达式的复杂形式	regex
conditionalvistor	ognl 表达式验证	expression

配置方法：

在 resouce 的文件夹和 action 的同级目录下创建 xxxAction-xxxmethod-validation.xml 文件，然后按照格式编写验证器，如果要写公共的验证器，则可以编写 xxxAction-xxxmethod-validation.xml 文件



(4) 在 Action 中使用国际化资源文件的步骤及方法。

首先该 action 类必须继承 ActionSupport 积累，然后使用 getText()来访问对应的国际化资源

```
UserService userService = new UserService();
if (userService.login(loginUser)) {
    this.addActionMessage(getText(aTextName: "login.action.success"));
    ShoppingCart shoppingCart = new ShoppingCart();
    session.setAttribute(name: "shoppingCart", shoppingCart);
    return SUCCESS;
}
this.addActionError(getText(aTextName: "login.action.fail"));
return FAIL;
```

(5) struts2 常用的内置类型转换器及其使用方法

struts 默认支持 String 到下列类型的内置类型转换

String, Boolean, Character...Float..., Date, arrays, collections

只要在对应的 model 类中声明对应的类型，变为使用内置类型转换。

(6) 碰到的问题及解决方案或思考

配置 UserAction-login-validation.xml 时无法生效的问题

原因：因为此项目的文件被分为 java、resources、webapp 三部分，而 IDE 编译的行为时不编译 java 文件夹之下的非 java 文件，因此.xml 并不会拷贝到对应的 build 目录。而应当在 resources 文件夹下创建相同的目录并放置此类资源文

件。

（7）实验收获

我们之前写纯 Servlet 项目时的字段校验和类型转换都是写在对应的业务逻辑中的，这也经常导致业务无关代码占据了较大的量，而且这些都是重复性的代码，structs 使用组件化和 AOP 的概念，将字段校验和类型转换单独分离出来，构成自己的逻辑，从而提高了代码的可阅读性。

2.3、实验三

（2）访问 Servlet API

方法	返回类型	特点
使用 ActionContext 工具类	ActionContext Map<String,Object>	非侵入式
实现 ApplicationAware 等接口	Map<String,Object>	非侵入式
使用 ServletActionContext 工具类	ServletActionContext HttpServletRequest HttpServletResponse	侵入式
实现 ServletApplicationAware 等接口	ServletActionContext HttpServletRequest HttpServletResponse	侵入式

（4）碰到的问题及解决方案或思考

没有碰到复杂的问题

（5）实验收获

学习了在 Action 实例中保存持久化对象的方法，因为 Action 实例是每次请求时单独创建的，因此 Action 中的实例变量的作用范围和 requestAttributes 是等效的，因此在 Action 中要存储在应用中不变或者在一次会话中不变的对象，需要借助 SessionAware 和 ApplicationAware 接口或者对应的 ActionContext 工具类。