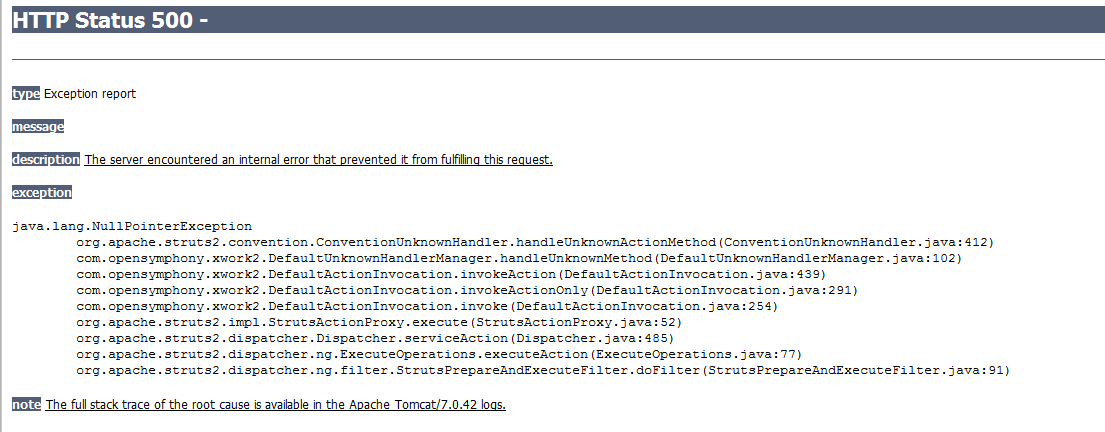
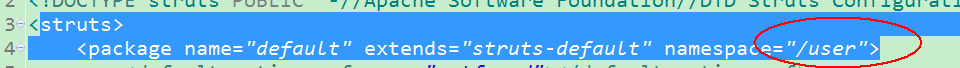
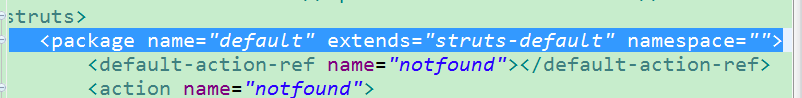
## 问题一:服务器端错误:5xxx



找不到method,控制台出错信息:

|  |
| --- |
| 19:13:03,558 DEBUG ConventionUnknownHandler:80 - Checking if there is an action named index in the namespace [/user]  19:13:03,558 DEBUG ConventionUnknownHandler:80 - Found action config  19:13:03,558 DEBUG ConventionUnknownHandler:80 - Action is not a default - redirecting  19:13:03,558 DEBUG DefaultActionInvocation:68 - Executing action method = null |

经查看发现String.xml文件配置中: 

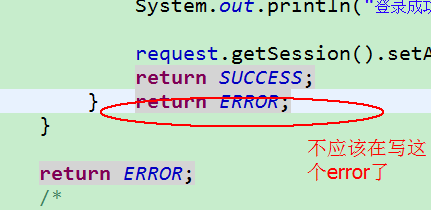
但是这样注册是能成功的但是登录不成功,改为

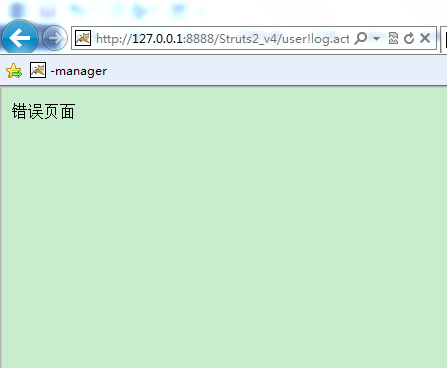
注册和登录都可以了.

## 问题:只能是第一个用户登录:

|  |
| --- |
| **public** String log() **throws** Exception {  String name = user.getName();  // String name = getUser().getName();  System.*out*.println(name + ":获取用户名");  // String password = getUser().getPassword();  String password = user.getPassword();  System.*out*.println("密码: " + password);  List<User> list = **null**;  User user = **null**;  list = createFactory.userDAO().findUserAll();  /\*  \* for (int i = 0; i < list.size(); i++) {  \* System.out.println(list.get(i).getName());  \* System.out.println(list.get(i).getPassword()+"密码"); if  \* (name.equals(list.get(i).getName()) &&  \* password.equals(list.get(i).getPassword())) {  \* System.out.println("登录成功");  \*  \* request.getSession().setAttribute("user", user);// 增加到session中 return  \* SUCCESS; }  \*/  **for** (User u : list) {  user = u;  System.*out*.println(user.getName() + " : " + user.getPassword());  **if** (name.equals(user.getName())  && password.equals(user.getPassword())) {  System.*out*.println("登录成功");  request.getSession().setAttribute("user", user);// 增加到session中  **return** *SUCCESS*;  }  }    **return** *ERROR*;  /\*  \* list获取的方法 Iterator iter = list.iterator();  \*  \* while (iter.hasNext()) { user = (User) iter.next();  \* System.out.println(user.getName()); System.out.println("密码:" +  \* user.getPassword());  \*  \* } return ERROR;  \*/  } |

通过研究发现注释部分(就是问题者中说的只能登录一个用户(第一个用户))

而不是注释的部分就可以 原因就是

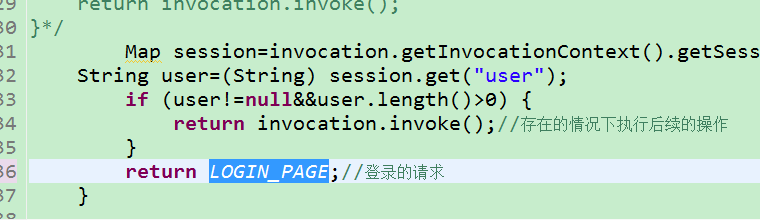
不论判断是否成功都不会在迭代了 因为 不管成功或者失败都跳转到了 相应的页面

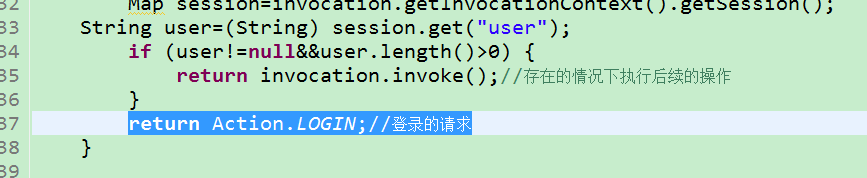
去掉之后就可以了:

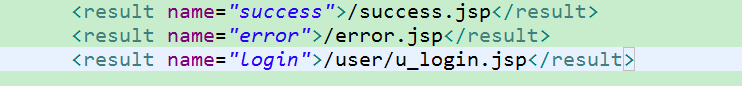


## HTTP Status 404 - No result defined for action action.addAction and result i

这个是过滤的时候找不到action中的结果实际是struts.xml文件配置出错:

这里的login\_page是: public static final String LOGIN\_PAGE="log.action";

struts.xml文件增加:



就可以了

## 页面为了安全都应该放到WEB-INF目录下:

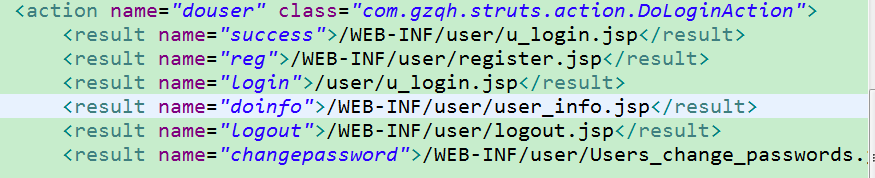
这样带来的问题是:

前台不能访问到了 ,struts提供了读取web-inf下的方式:

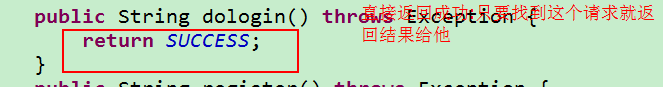
可以通过1.页面请求action;交给回台处理



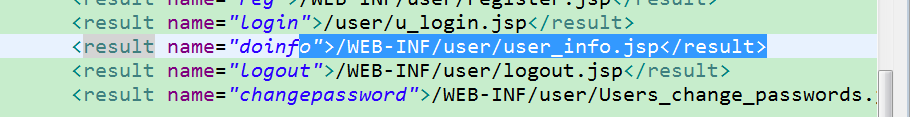
2.配置struts.xml进行后台处理



3.action处理返回结果



4.struts.xml文件交到web-inf页面;



## 所有的模块都应该分包独立:

Struts.xml文件

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">  <struts>  <include file=*"struts-user.xml"*></include>  </struts> |

直接导入xml文件就可以了:

注意命名规范:struts-xxx.xml:默认放到src下的也可以放到对应的action下但是要加路径全称.

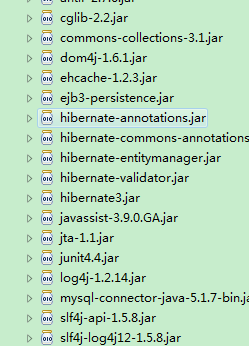
Struts-user.xml文件配置:

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">  <struts>  <package name=*"default"* extends=*"struts-default"* namespace=*""*>  <!-- <interceptors> <interceptor name="login" class="com.gzqh.struts.action.DoLoginintercept"  /> <interceptor-stack name="teamwareStack"> <interceptor-ref name="login"  /> <interceptor-ref name="defaultStack" /> </interceptor-stack> </interceptors>  <default-interceptor-ref name="teamwareStack" /> -->  <!--后续同一个package内部的所有Action执行之前都会被login拦截。 -->  <default-action-ref name=*"notfound"*></default-action-ref>  <action name=*"notfound"*>  <result>WEB-INF/notfound.jsp</result>  </action>  <action name=*"user"* class=*"com.gzqh.struts.action.UserAction"*>  <result name=*"input"*>/WEB-INF/user/success.jsp</result>  <result name=*"error"*>/WEB-INF/user/error.jsp</result>  <result name=*"login"*>/user/u\_login.jsp</result>  </action>  </package>  <package name=*"douser"* extends=*"struts-default"* namespace=*""*>  <!-- <interceptors> <interceptor name="login" class="com.gzqh.struts.action.DoLoginintercept"  /> <interceptor-stack name="teamwareStack"> <interceptor-ref name="login"  /> <interceptor-ref name="defaultStack" /> </interceptor-stack> </interceptors>  <default-interceptor-ref name="teamwareStack" /> -->  <!--后续同一个package内部的所有Action执行之前都会被login拦截。 -->  <default-action-ref name=*"notfound"*></default-action-ref>  <action name=*"notfound"*>  <result>WEB-INF/notfound.jsp</result>  </action>  <action name=*"douser"* class=*"com.gzqh.struts.action.DoLoginAction"*>  <result name=*"success"*>/WEB-INF/user/u\_login.jsp</result>  <result name=*"reg"*>/WEB-INF/user/register.jsp</result>  <result name=*"login"*>/user/u\_login.jsp</result>  <result name=*"doinfo"*>/WEB-INF/user/user\_info.jsp</result>  <result name=*"logout"*>/WEB-INF/user/logout.jsp</result>  <result name=*"changepassword"*>/WEB-INF/user/Users\_change\_passwords.jsp</result>  </action>  </package>  </struts> |

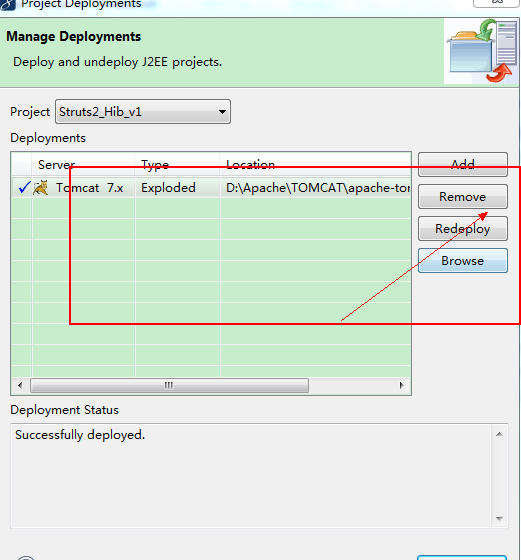
注意:这里说明要导入的xml文件应该和源文件一样都有完整的结构包括声明

## 关于包冲突的处理:

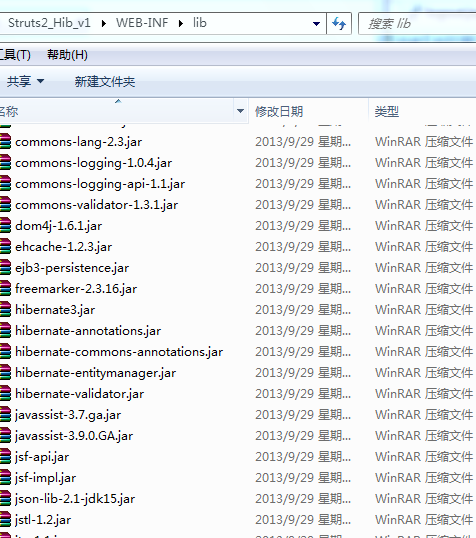
问题说明:有时候在SSH整合的时候会存在包冲突的情况:



很多时候我们remove掉了 但是实际上通过



但是这里:

还是会有的:

正确的做法是:

1. copy走上面的整个lib目录
2. 把整个项目中的jar包remove然后把服务器上的jar都删除
3. 把我们备份的lib目录放到项目中的lib中

## 解决相对路径找不到文件:

<base> 标签为页面上的所有链接规定默认地址或默认目标。

通常情况下，浏览器会从当前文档的 URL 中提取相应的元素来填写相对 URL 中的空白。

使用 <base> 标签可以改变这一点。浏览器随后将不再使用当前文档的 URL，而使用指定的基本 URL 来解析所有的相对 URL。这其中包括 <a>、<img>、<link>、<form> 标签中的 URL。

实例

<head>

<base href="${basePath}" />

<base target="\_blank" />

</head>

<body>

<img src="eg\_smile.gif" />

<a href="http://www.w3school.com.cn">W3School</a>

</body>

1. jsp页面中:

|  |
| --- |
| <%  //用于获取servlet的路径（验证码）  String path = request.getContextPath();  String basePath = request.getScheme() + "://"  + request.getServerName() + ":" + request.getServerPort()  + path + "/";  %> |

2.jsp页面中:

<base href=*"*${basePath}*"* />

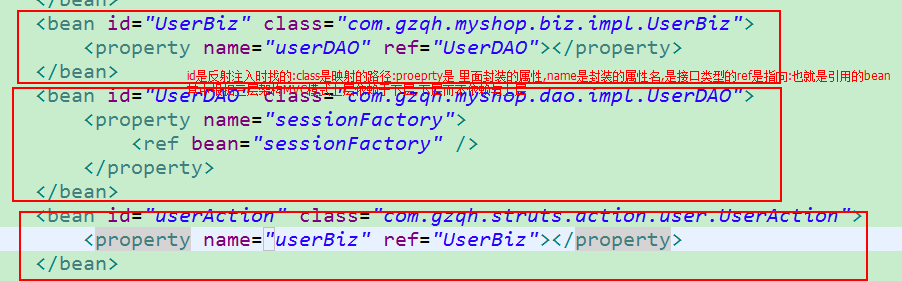
<base target=*"\_blank"* />

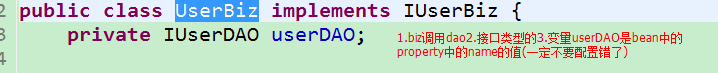
注意base标签一定要加在head标签中.

# SSH整合说明

反射注入:和AOP是Spring的核心;

项目中的spring配置文件的配置很重要:所有的对象的创建都是有配置文件=进行管理在运行时有选择的提供,这也是依赖注入:





## 读取spring配置文件的方法:

1).利用ClassPathXmlApplicationContext，这种方式配置文件应该放在类路径下，否则Spring将找不到该文件。

Java代码：

ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");

ctx.getBean("");

2).1利用FileSystemXmlApplicationContext,这种方法可以将配置文件放在工程的直接目录下

Java代码：

ApplicationContext ctx=new FileSystemXmlApplicationContext("applicationContext.xml");

ctx.getBean("");

2.显式给出配置文件的绝对路径，我们假定配置文件位于工程主目录中的conf文件夹中

String path=System.getProperty("user.dir");

ApplicationContext ctx=new FileSystemXmlApplicationContext(path+"//conf//applicationContext.xml");

ctx.getBean("");

3.利用FileSystemResource，这个方法跟随2)有点相似，可以将配置文件放在工程直接目录下

Resource r=new FileSystemResource("applicationContext.xml");

BeanFactory factory=new XmlBeanFactory(rs);

factory.getBean("");

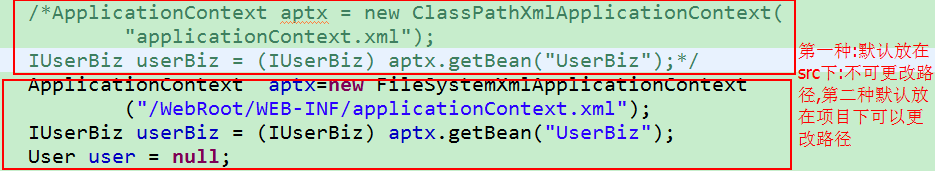
显式给出配置文件的绝对路径，我们假定配置文件位于工程主目录中的conf文件夹中

String path=System.getProperty("user.dir");

Resource rs=new FileSystemResource(path+"//conf//applicationContext.xml");

BeanFactory factory=new XmlBeanFactory(rs);

factory.getBean("");



这里读取的bean是biz层的.

## SSH2 整合在页面显示 Unable to instantiate Action 控制台没报错。

Unable to instantiate Action, ....action.CertAction, defined for '...' in namespace '/...'null

原因是变量定义：

private UserUtil userUtil = (UserUtil)ServletActionContext.getContext().getSession().get("userUtil");

关键是这里,这个uerUtil成员变量使用了 action 的上下文"ServletActionContext.getContext()".

了解webwork中ServletDispatcher原理的人可能知道原因了, 因为ServletDispatcher接受客户端的HTTP请求，将JavaServlet的很多相关对象进行包装，再传给我们的XWork框架，由我 们的XWork 框架去解析我们的xwork.xml配置文件，根据配置文件的信息，创建对应的Action，组装并调用相应的拦截器，执行Action，返回执行结果。

因此每次客户端的请求都将调用ServletDispatcher 的 service（）方法,

该方法执行顺序如下:

1、通过request请求取得action的命名空间

2、根据servlet请求的Path,解析出要调用该请求的Action的名字（actionName）

3、创建Action上下文（extraContext）

4、根据前面获得的namespace、actionName、extraContext，创建一个ActonProxy

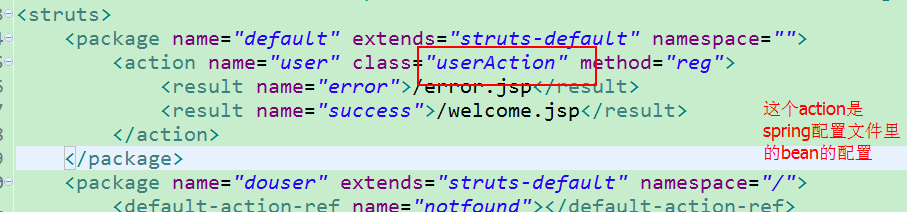
5、执行proxy的execute()方法,根据Action执行返回的值去调用执行相应的Result（返回结果处理）的方法。

现在应该能明白，原来是Action对象实例在ActionContext对象实例之前创建的，所有这样取得ActionContext容器对象就有可能会返回null.

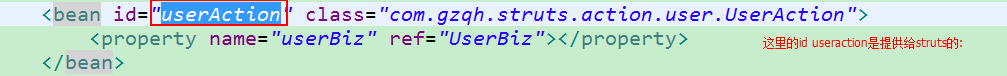
因此解决次问题方法很简单,就是最好不要在成员变量或者是构造方法里调用ActionContext,而把它放在其它方法里.

## Userbiz为空:

Struts里面的action是交给sping



Spring的配置:



## 一个异常:tomcat启动的时候抱的:org.apache.catalina.core.StandardContext filterStart

启动服务器的时候抱错:

检查是因为jar包冲突:但是jar不好找,所以重新使用工具建立一个新的SSH工程部署到tomcat服务器上然后要新工程里的web-inf下的lib里的所有jar包copy到报错的工厂下:所有的操作都是在tomcat服务器上,也就是通过部署项目时候的brower来找到的

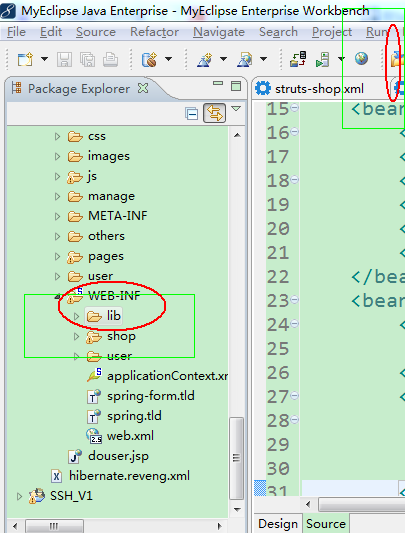
## 关于冲突jar包的删除

解决jar包冲突要到服务器项目下的web-inf下的lib目录:

1. 先备份lib目录所有
2. 删除lib目录所有
3. 把备份的lib所有copy到myeclipse项目下的lib下

以上是jar包冲突的解决:

但是删除jar包的时候会不让删

1. 点击myeclipse工具的找到lib所在的地方
2. 关闭myeclipse工具(不关会提示占用)
3. 删除lib下的哪个都可以
4. 打开myeclipse工具
5. 刷新工程(不刷新是看不到效果的)

## 关于spring管理的xxx.hbn.xml文件的说明:

|  |
| --- |
| <!-- 在spring配置文件中如果有多个\*.hbm.xml文件的话，无论是项目开发过程中还是维护过程中修改起来都会很麻烦切容易出错 这时我们可以把name="mappingResources"改为name="mappingDirectoryLocations"，然后只需要配置\*.hbm.xml文件所在目录就行了 -->  <!-- <property name="mappingResources"> <list> <value>com/gzqh/myshop/pojo/User.hbm.xml</value> </list> </property> -->  <property name=*"mappingDirectoryLocations"*>  <list>  <value>classpath:/com/gzqh/myshop/pojo</value>  </list>  </property> |