
知识点列表

编号	名称	描述	级别
1	SSH 整合_方案 01	通过案例练习掌握 SSH 的整合	**
2	事务处理	通过案例掌握 Spring 管理事务的步骤及配置	**
3	SSH 整合_方案 02	通过案例练习掌握 SSH 的整合	**

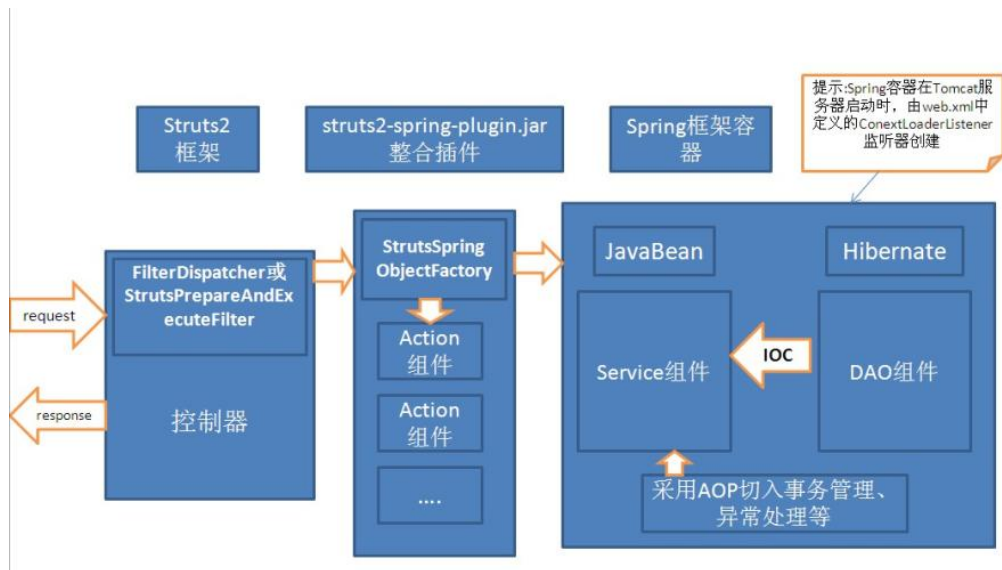
注： **"理解级别 ***"掌握级别 ****"应用级别

目录

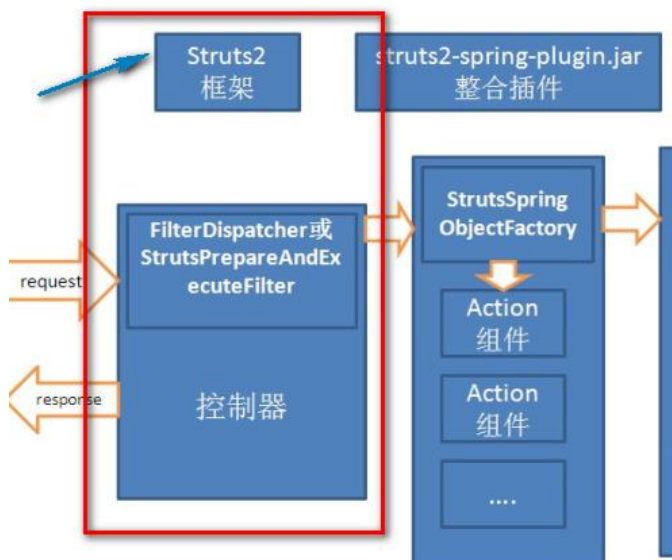
- 1. SSH 整合_方案 01 ** 3
 - 【案例 1】SSH 整合_方案 1 ** 5
- 2. 事务处理 ** 31
 - 【案例 2】SSH 整合_事务处理 ** 31
- 3. SSH 整合_方案 02 40
 - 【案例 3】SSH 整合_方案 2 ** 41

1. SSH 整合_方案 01 **

整合方案 01



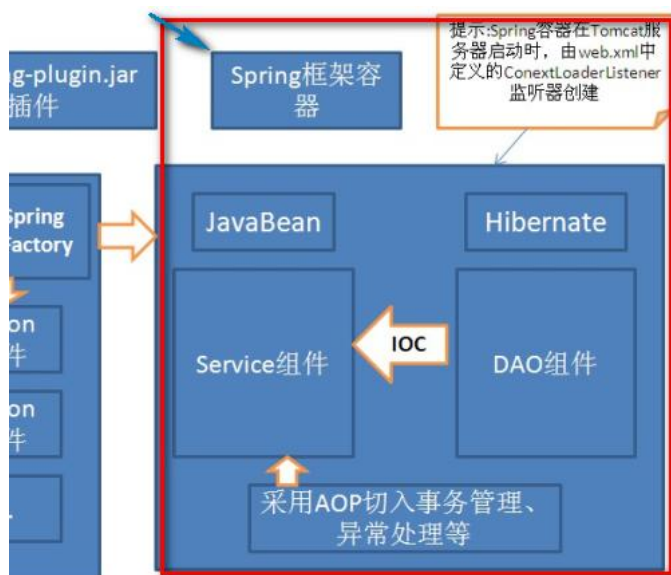
这是 Struts2 框架



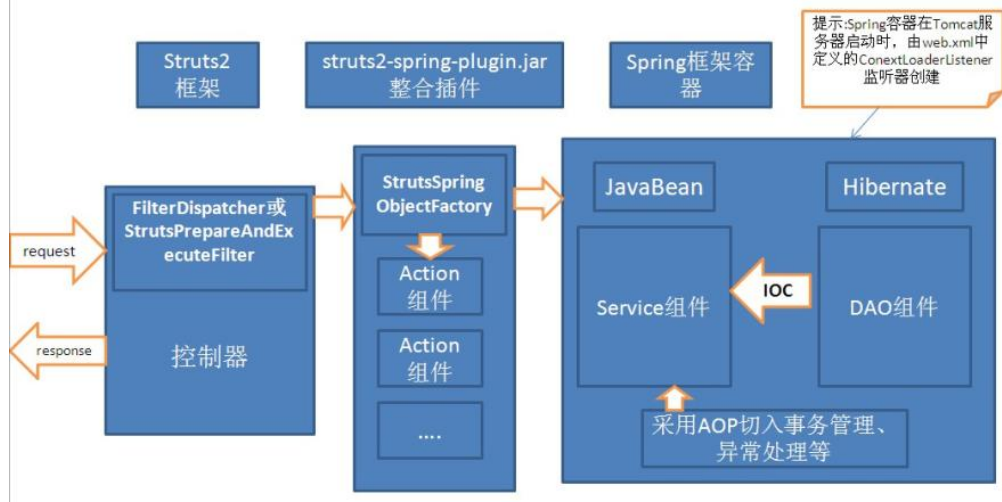
这是 Spring 框架

在 Spring 框架中整合了 Hibernate (或 JDBC 亦可)

一些业务组件 (Service 组件) 也可以放入 Spring 框架中进行管理 (昨天的例子)



如图所示



1. 请求 (request) 发出后, 该请求要调用某个 Action 进行处理
2. 拦截器 (FilterDispatcher) 照惯例拦截请求 (request) , 此时, 如果拦截器 (FilterDispatcher) 发现项目中已经引入了 struts2-spring-plugin.jar 整合插件
3. 那么接下来, 拦截器就将请求 (request) 交给 Struts2-spring-plugin.jar 整合插件来创建 Action 组件对象

4. 在插件 `struts2-spring-plugin.jar` 中有个非常重要的类：
对象工厂 `StrutsSpringObjectFactory`。
5. 对象工厂 `StrutsSpringObjectFactory` 可以创建 Action 组件并且到 Spring 框架中将
Service 组件或 DAO 组件取出，注入到 Action 中去
6. 当然，在 Spring 框架内部，就各种使用 IoC 或者 AOP，就和我们之前讲的一样。

如下所示，

整合 SSH 框架需要保证的是：在 Tomcat 启动时，Spring 容器就已经创建。

当请求（request）来时，直接就可以用，不需要临时创建了。

原理是在 `web.xml` 中配置 `ContextLoaderListener`，由它来将 Spring 容器实例化

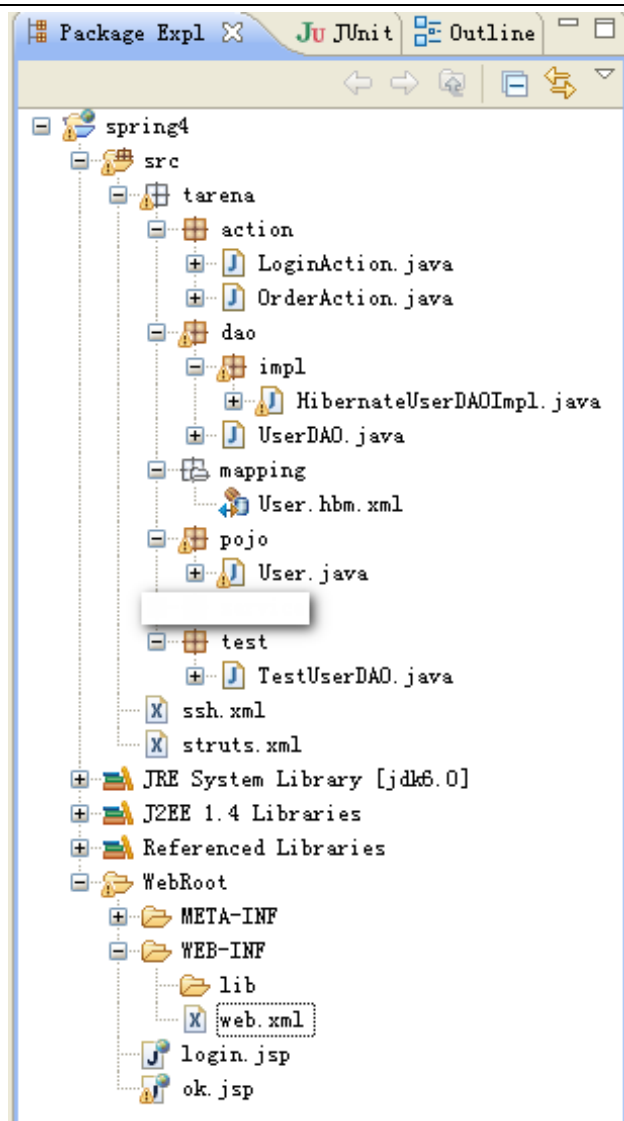


所以，整合 SSH 的难度并不大，主要是整合需要的那个 `struts2-spring-plugin.jar` 整合插件

案例如下

【案例 1】SSH 整合_方案 1 **

项目结构图



参考代码

1) 新建工程 spring4

鉴于 Struts 和 Hibernate 是由 Spring 整合的，我们先来引入 Spring 框架。（后期熟练后，先引入哪个都可以）

2) 导入 Spring 的 Jar 包

请下载 [spring_some_lib.zip](#)

完成[登录功能](#)，先写[视图](#)

3) 新建视图

a. 登录页面 login.jsp

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<html>
<head>
<title>login</title>
</head>
<body style="font-size:30px;">
    <form action="" method="post">
        用户名 : <input type="text" name=""> <br/>
        密码 : <input type="text" name=""> <br/>
        <input type="submit" value="登录">
    </form>
</body>
</html>
```

b. 登录成功页面 ok.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>

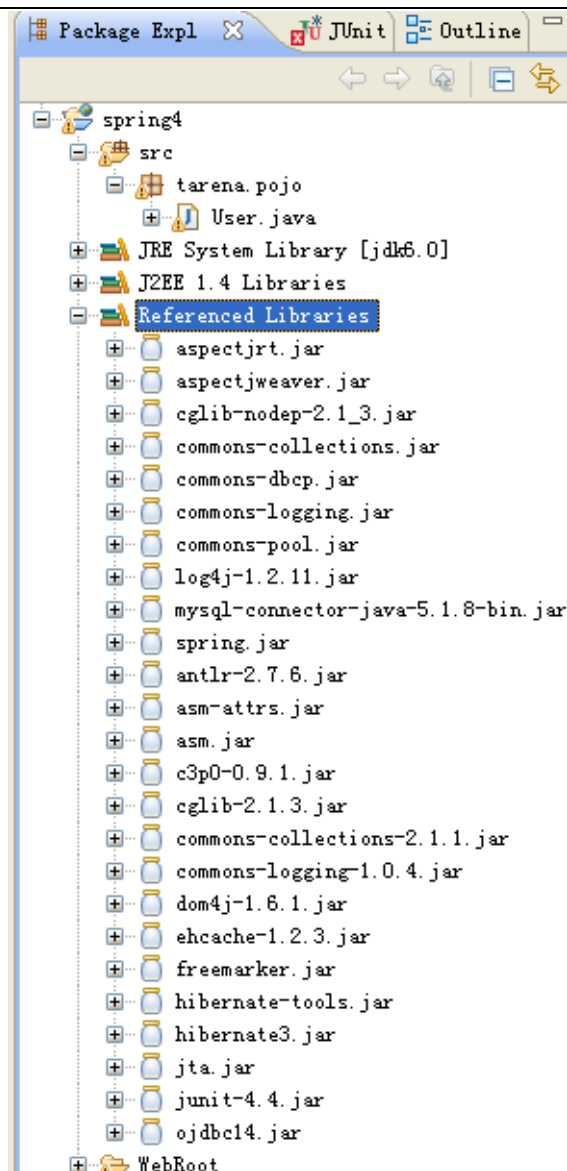
<html>
<head>
    <title>success</title>
</head>

<body>
    <h2>登录成功！</h2> <br>
</body>
</html>
```

视图写好了，接下来该写模型层 DAO，我们使用 Spring 整合 Hibernate 完成

4) 导入 hibernate 的 Jar 包

请下载 [hib_some_lib.zip](#)



5) 新建 POJO&&映射文件

a. pojo.User

```
package tarena.pojo;  
  
public class User implements java.io.Serializable {  
  
    // Fields  
    private Integer id;  
    private String email = "";
```



```

private String nickname = "";
private String password = "";
private Integer userIntegral = 0;
private boolean emailVerify = false;
private String emailVerifyCode = "";
private long lastLoginTime = 0L;
private String lastLoginIp = "";

// Constructors

/** default constructor */
public User() {
}

/** minimal constructor */
public User(String email, String password, Integer userIntegral) {
    this.email = email;
    this.password = password;
    this.userIntegral = userIntegral;
}

public boolean isEmailVerify() {return emailVerify;}
public void setEmailVerify(boolean emailVerify) {
    this.emailVerify = emailVerify;}
public Integer getId() {return this.id;}
public void setId(Integer id) {this.id = id;}
public String getEmail() {return this.email;}
public void setEmail(String email) {this.email = email;}
public String getNickname() {return this.nickname;}
public void setNickname(String nickname) {
    this.nickname = nickname;}
public String getPassword() {
    return this.password;}
public void setPassword(String password) {
    this.password = password;}
public Integer getUserIntegral() {
    return this.userIntegral;}
public void setUserIntegral(Integer userIntegral) {

```

```

        this.userIntegral = userIntegral;}
    public String getEmailVerifyCode() {
        return this.emailVerifyCode;}
    public void setEmailVerifyCode(String emailVerifyCode) {
        this.emailVerifyCode = emailVerifyCode;}
    public long getLastLoginTime() {
        return this.lastLoginTime;}
    public void setLastLoginTime(long lastLoginTime) {
        this.lastLoginTime = lastLoginTime;}
    public String getLastLoginIp() {
        return this.lastLoginIp;}
    public void setLastLoginIp(String lastLoginIp) {
        this.lastLoginIp = lastLoginIp;}
}

```

b. mapping.User.hbm.xml

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="tarena.pojo">
    <class name="User" table="d_user" catalog="test">
        <id name="id" type="integer">
            <column name="id" />
            <generator class="native"></generator>
        </id>
        <property name="email" type="string">
            <column name="email" length="50"
                not-null="true" unique="true" />
        </property>
        <property name="nickname" type="string">
            <column name="nickname" length="50" />
        </property>
        <property name="password" type="string">
            <column name="password" length="50" not-null="true" />
        </property>
        <property name="userIntegral" type="integer">

```

```

        <column name="user_integral" not-null="true" />
    </property>
    <property name="emailVerify" type="yes_no">
        <column name="is_email_verify" length="3" />
    </property>
    <property name="emailVerifyCode" type="string">
        <column name="email_verify_code" length="50" />
    </property>
    <property name="lastLoginTime" type="long">
        <column name="last_login_time" />
    </property>
    <property name="lastLoginIp" type="string">
        <column name="last_login_ip" length="15" />
    </property>
</class>
</hibernate-mapping>

```

6) 新建 DAO

a. 新建 UserDao

```

package tarena.dao;

import tarena.pojo.User;

public interface UserDao {
    public User findByEmail(String email);
}

```

b. 新建 HibernateUserDAOImpl

```

package tarena.dao.impl;

import java.util.List;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import tarena.dao.UserDAO;
import tarena.pojo.User;

public class HibernateUserDAOImpl
    extends HibernateDaoSupport implements UserDAO {

```

```

public User findByEmail(String email) {
    String hql = "from User where email=?";
    List<User> list =
        this.getHibernateTemplate().find(hql,new Object[]{email});
    User user = null;
    if(!list.isEmpty()){
        user = list.get(0);
    }
    return user;
}
}

```

7) 新建 spring 配置文件 ssh.xml

和之前一样，

首先，配置连接池 dataSource

其次，配置 SessionFactory

再次，加入 bean 组件 UserDao

最后，在 UserDao 中注入 SessionFactory

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
        http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

    <bean id="myDataSource"
        class="org.apache.commons.dbcp.BasicDataSource">

        <property name="driverClassName"
            value="com.mysql.jdbc.Driver"></property>
        <property name="url" value="jdbc:mysql:///test"></property>
        <property name="username" value="root"></property>

```

```

    <property name="password" value="root"></property>
    <property name="maxActive" value="10"></property>
    <property name="initialSize" value="2"></property>
</bean>

<bean id="mySessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="myDataSource"></property>
    <property name="mappingResources">
        <list>
            <value>tarena/mapping/User.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">
                org.hibernate.dialect.MySQL5Dialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.format_sql">true</prop>
        </props>
    </property>
</bean>

<bean id="userDao" class="tarena.dao.impl.HibernateUserDAOImpl">
    <property name="sessionFactory" ref="mySessionFactory">
    </property>
</bean>

</beans>

```

该测试了

8) 新建测试类

```

package tarena.test;

import junit.framework.Assert;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

```

```

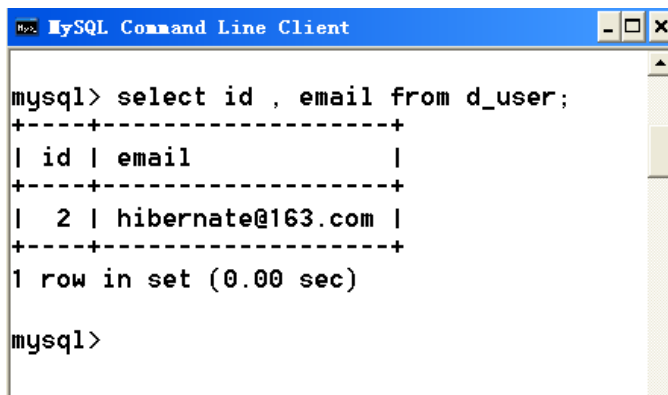
import tarena.dao.UserDAO;
import tarena.pojo.User;

public class TestUserDAO {
    @Test
    public void testFindByEmail(){
        ApplicationContext ac =
            new ClassPathXmlApplicationContext("ssh.xml");
        UserDAO userDao = (UserDAO)ac.getBean("userDao");
        User user = userDao.findByEmail("hibernate@163.com");
        Assert.assertEquals(new Integer(2), user.getId());
        Assert.assertEquals("hibernate", user.getNickname());
    }
}

```

9) 测试

a. 查询数据库



The screenshot shows a MySQL Command Line Client window. The command entered is `mysql> select id , email from d_user;`. The output displays a table with two columns: `id` and `email`. The first row shows `2` for `id` and `hibernate@163.com` for `email`. Below the table, it states `1 row in set (0.00 sec)`. The prompt `mysql>` is visible at the bottom.

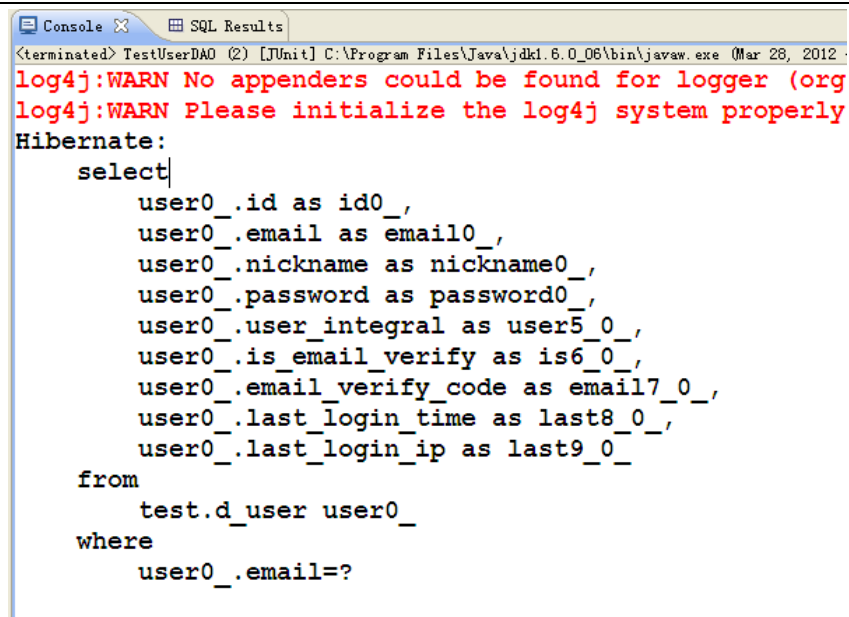
```

mysql> select id , email from d_user;
+----+-----+
| id | email          |
+----+-----+
|  2 | hibernate@163.com |
+----+-----+
1 row in set (0.00 sec)

mysql>

```

b. 运行



```
<terminated> TestUserDAO (2) [JUnit] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 28, 2012)
log4j:WARN No appenders could be found for logger (org
log4j:WARN Please initialize the log4j system properly
Hibernate:
  select
    user0_.id as id0_,
    user0_.email as email0_,
    user0_.nickname as nickname0_,
    user0_.password as password0_,
    user0_.user_integral as user5_0_,
    user0_.is_email_verify as is6_0_,
    user0_.email_verify_code as email7_0_,
    user0_.last_login_time as last8_0_,
    user0_.last_login_ip as last9_0_
  from
    test.d_user user0_
  where
    user0_.email=?
```

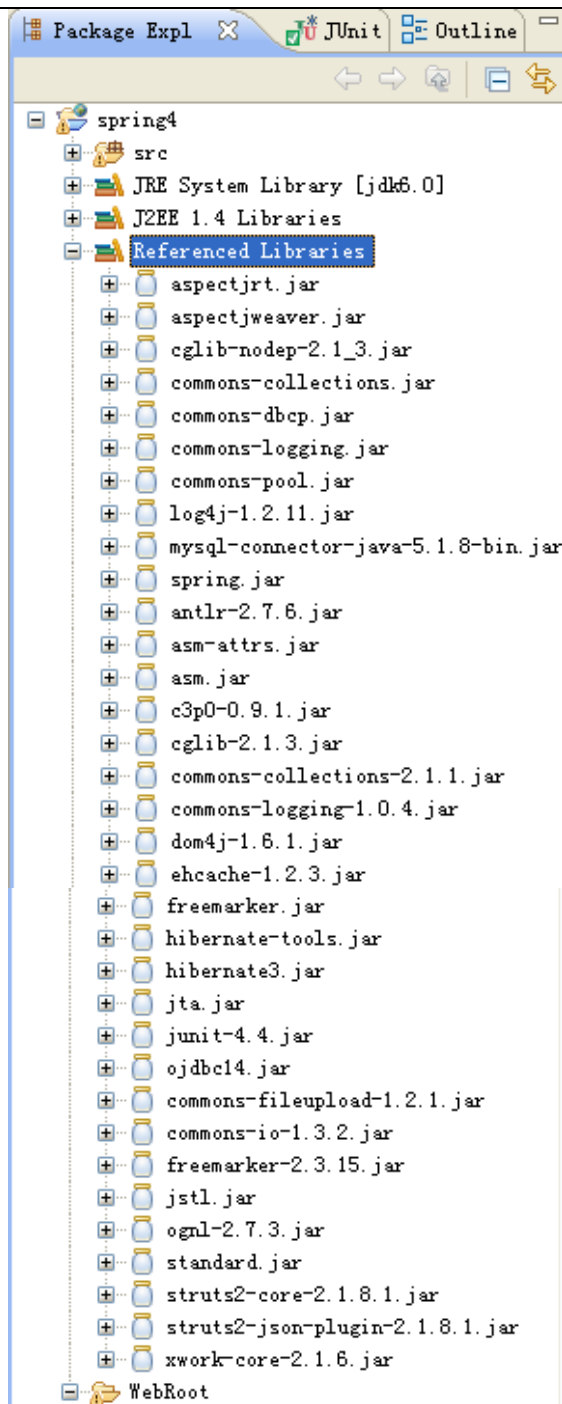
Ok, Spring 把 Hibernate 整妥帖了, 该轮到 Struts2 了

10) 导入 Struts2 的 Jar 包

请下载 [Struts2_some_lib.zip](#)

这么多的 Jar 包, 并且里面还有重复的, 只要不影响结果就不碍事, 可以删除掉, 需要注意的是, Jar 包也有版本问题, 所以在整合 SSH 时, 如果遇到莫名其妙的问题, 可能是 Jar 包版本问题导致的冲突

还有些 Jar 包不需要, 就可以删除了, 比如 JSON 需要的, jstl 需要的



11) 修改 web.xml

配置 struts2


```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  <filter>
    <filter-name>StrutsFilter</filter-name>
    <filter-class>
      org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
    </filter-class>
  </filter>

  <filter-mapping>
    <filter-name>StrutsFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

</web-app>

```

12) 新建 struts.xml

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.1.7//EN"
  "http://struts.apache.org/dtds/struts-2.1.7.dtd">

<struts>
  <package name="ssh1-demo" extends="struts-default">
    <action name="login" class="tarena.action.LoginAction">
      <result name="success">/ok.jsp</result>
      <result name="login">/login.jsp</result>
    </action>
  </package>

</struts>

```

13) 新建 LoginAction

```

package tarena.action;

import tarena.dao.UserDAO;
import tarena.pojo.User;

public class LoginAction {

    //接收表单信息的对象
    private User user;

    //默认采用名称对应规则将Spring容器中dao注入
    private UserDAO userDao;

    public String execute(){
        User usr = userDao.findByEmail(user.getEmail());
        if(usr != null){
            if(usr.getPassword().equals(user.getPassword())){
                return "success";
            }
        }
        return "login";
    }

    public UserDAO getUserDao() {return userDao;}
    public void setUserDao(UserDAO userDao) {
        this.userDao = userDao;
    }
    public User getUser() {return user;}
    public void setUser(User user) {this.user = user;}
}

```

14) 修改 login.jsp

```

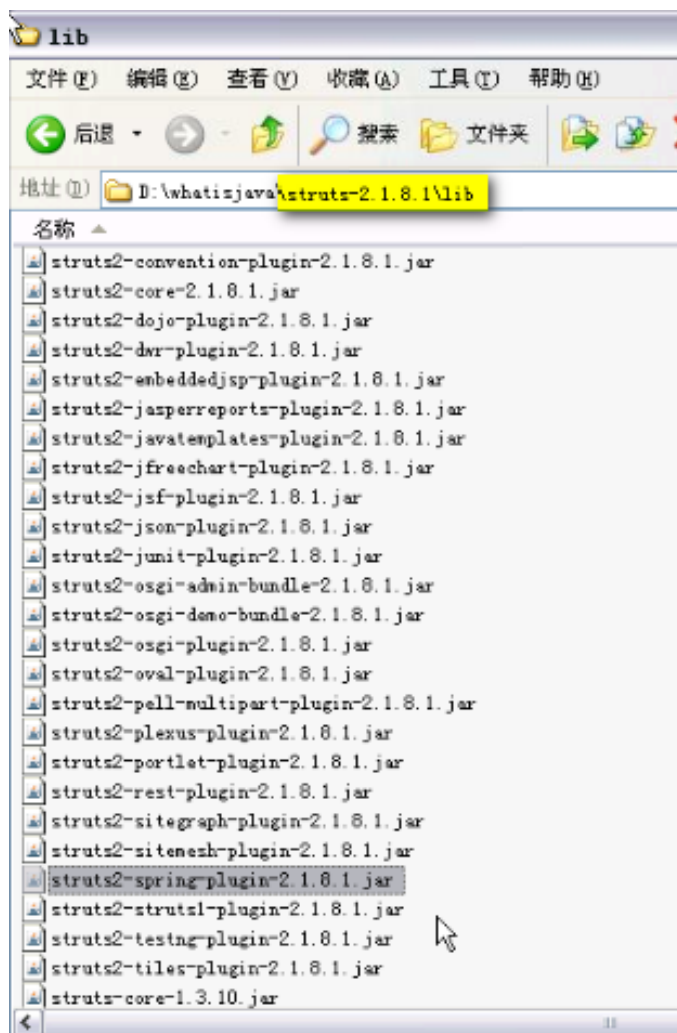
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<html>
<head>
<title>login</title>
</head>
<body style="font-size:30px;">
    <form action="login.action" method="post">

```

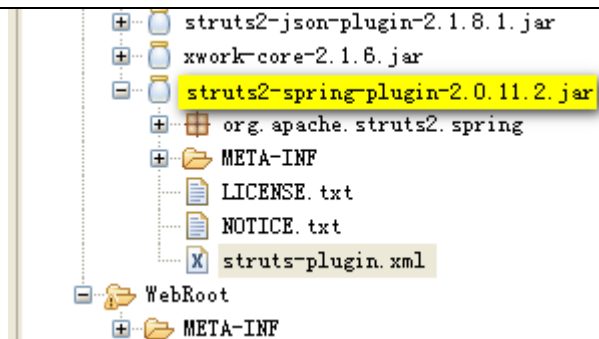
```
用户名：<input type="text" name="user.email"><br/>
密码：<input type="text" name="user.password"><br/>
<input type="submit" value="登录">
</form>
</body>
</html>
```

接下来就是整合 SSH 最重要的步骤，引入 `struts2-spring-plugin.jar` 整合插件
让它来创建 Action，和从容器中取 DAO

这个插件在 Struts2 的 Jar 包中可以找到



15) 拷贝 `struts2-spring-plugin-2.1.8.1.jar` 到 lib 目录下



在这些提供的 Jar 包中一般都会有一个 xml 文件，结构和 struts.xml 是一样的

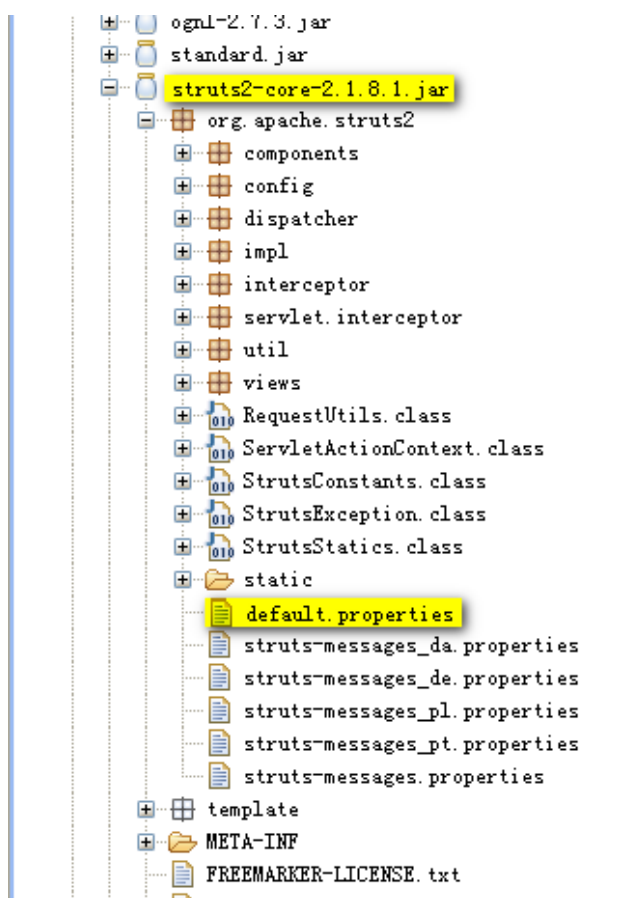
打开 struts-plugin.xml，我们查看一下

注意，其中这行代码是很关键的

```
29 <struts>
30   <bean type="com.opensymphony.xwork2.ObjectFactory"
31     name="spring"
32     class=
33       "org.apache.struts2.spring.
34         StrutsSpringObjectFactory" />
35
36   <!-- Make the Spring object factory
37     the automatic default -->
38   <constant name="struts.objectFactory"
39     value="spring" />
40
41   <package name="spring-default">
42     <interceptors>
43       <interceptor name="autowiring" class="com.o
44       <interceptor name="sessionAutowiring" clas
45     </interceptors>
46   </package>
47 </struts>
```

为什么我们只要导入 `struts2-spring-plugin-2.1.8.1.jar` 包，Struts2 框架就知道该将创建 Action 的功能交给插件来做，不用自己创建了？

`<constant name="struts.objectFactory" value="spring" />` 这个标签可以指定覆盖 struts2 中原有的 `default.properties`



打开default.properties，在其中定义了[字符处理编码](#)

```
28### one could extend org.apache.struts2.config.Config
29### to build one's customize way of getting the config
30# struts.configuration=org.apache.struts2.config.Default
31
32### This can be used to set your default locale and e
33# struts.locale=en_US
34struts.i18n.encoding=UTF-8
35
```

[上传文件的一些参数设置](#)

```

65#### Parser to handle HTTP POST requests, encoded u
66# struts.multipart.parser=cos
67# struts.multipart.parser=pell
68struts.multipart.parser=jakarta
69# uses javax.servlet.context.tempdir by default
70struts.multipart.saveDir=
71struts.multipart.maxSize=2097152
72

```

还有，将扩展名设置为*.action

```

79#### Used by the DefaultActionMapper
80#### You may provide a comma separated list, e.g. str
81#### The blank extension allows you to match directory
82#### without interfering with static resources.
83struts.action.extension=action,
84

```

所以，在default.properties文件中定义了一些struts2的系统参数
其中有一项非常重要的参数设置

```

37#### Note: short-hand notation is supported in some c
38#### Alternatively, you can provide a com.opensys
39# struts.objectFactory = spring
40

```

此时是注释的，如果取消注释，那么即为告诉Struts2框架Action的创建和维护交由Spring插件管理。

我们再回顾 struts2-spring-plugin-2.1.8.1.jar 包的 struts-plugin.xml 内容

<constant name="struts.objectFactory" value="spring" />的作用就相当于取消 default.properties 文件中 struts.objectFactory = spring 的注释。

```

30
31 <!-- Make the Spring object factory the automatic
32 <constant name="struts.objectFactory" value="spring"
33
34 <constant name="struts.class reloading.watchList" valu
35 <constant name="struts.class reloading.acceptClasses"
36 <constant name="struts.class reloading.reloadConfig" v
37

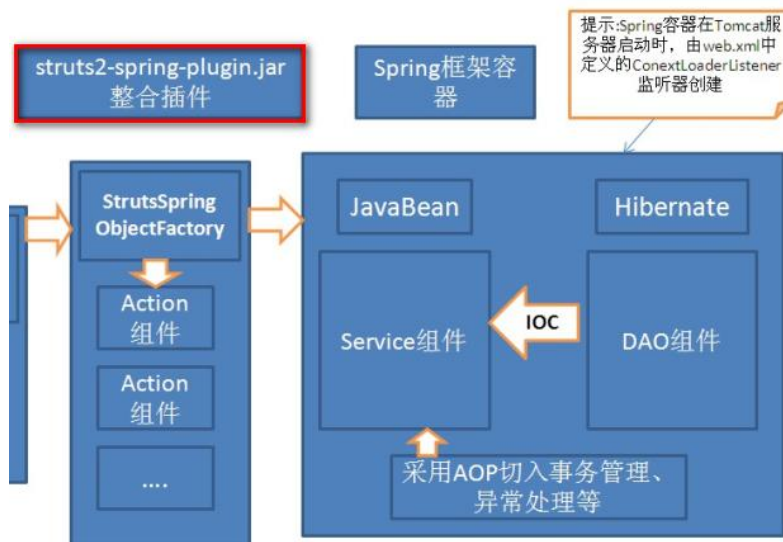
```

如上是 Spring 整合过程中需要理解的关键点

还需要注意的是此项是打开的

```
41#### specifies the autoWiring logic when using the Spr  
42#### valid values are: name, type, auto, and construct  
43struts.objectFactory.spring.autoWire = nameI  
44
```

此项表达的意思是什么？我们先看图示



Struts2-spring-plugin.jar 的作用是可以从 Spring 容器中找到 DAO 或者 Service , 注入给 Action , 然而, 我们在之前学习注解的时候讲到过, 注入的方式分两种, 按照名称或者按照类型。

所以, 此处表达的意思就是说, 为 Action 注入 Service 或者 DAO 时, 按照 name 注入才可以。

```
41#### specifies the autoWiring logic when using the Spr  
42#### valid values are: name, type, auto, and construct  
43struts.objectFactory.spring.autoWire = nameI  
44
```

也就是说, 这样是不行的

在 loginAction 中, 我们注入 DAO 的名字是 userDao

```

7 //接收表单
8 private User user;
9 //注入
10 private UserDao userDao;
11
12 public String execute(){
13     User usr = userDao.findByEmail(user.getEmail());
14     if(usr != null){
15         if(usr.getPassword().equals(user.getPassword())

```

在 ssh.xml 中为 DAO 起名为 hibernateUserDao

```

30     </props>
31     </property>
32 </bean>
33
34 <bean id="hibernateUserDao" class="tarena.dao.impl.H
35     <property name="sessionFactory" ref="mySessioni
36 </bean>
37

```

按照默认的方式，这样不行。

如何修改？

方法 1：让两个名字相同，

方法 2：为按照类型 type 注入

```

41### specifies the autoWiring logic when using the Spring
42### valid values are: name, type, auto, and constructor
43struts.objectFactory.spring.autoWire = name

```

我们在 struts.xml 中添加这样一句就可以了

```

<struts>
    <package name="ssh1-demo" extends="struts-default">
        <action name="login" class="tarena.action.LoginActi
            <result name="success">/ok.jsp</result>
            <result name="login">/login.jsp</result>
        </action>
    </package>

    <constant name="struts.objectFactory.spring.autoWire"
        value="type" />
</struts>

```

在 web.xml 中进行配置

16) 修改 web.xml

加入加载 Spring 配置文件，启动 spring 容器实例的监听器

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <!--用于指定spring配置文件的位置-->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:ssh.xml</param-value>
    </context-param>

    <!--用于加载指定的spring配置文件，配置文件的位置在<context-param>
        中指定-->
    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>

    <filter>
        <filter-name>StrutsFilter</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>StrutsFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

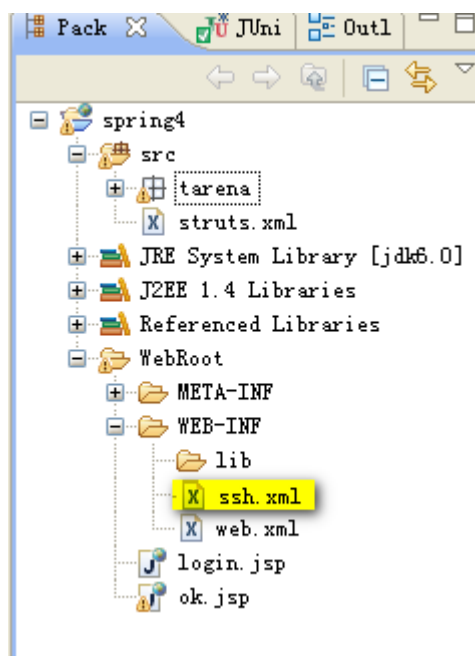
</web-app>
```

<param-value>classpath:ssh.xml</param-value>

其中 classpath 表示在类路径下找 ssh.xml 文件，

如果是 WEB-INF 目录下，直接写就可以了，写 WEB-INF/ssh.xml

文件位置放到这里



配置文件这样写

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="2.5"
3     xmlns="http://java.sun.com/xml/ns/javaee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema"
5     xsi:schemaLocation="http://java.sun.com/xml
6     http://java.sun.com/xml/ns/javaee/web-app_
7
8 <context-param>
9     <param-name>contextConfigLocation</param-na
10    <param-value>WEB-INF/ssh.xml</param-value>
11 </context-param>
12
13 <listener>
```

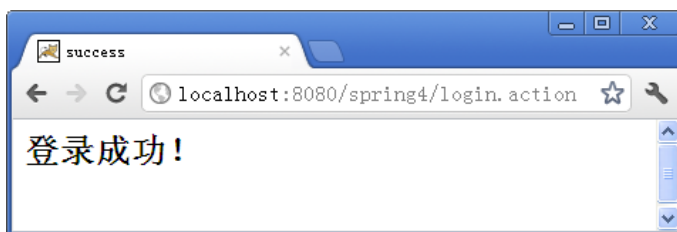
17) 部署项目，启动 Tomcat

18) 访问 <http://localhost:8080/spring4/login.jsp>

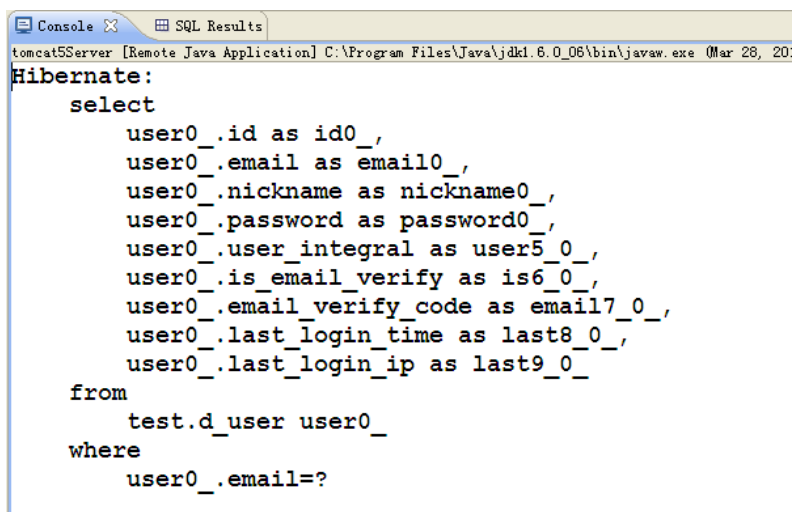
输入 "hibernate@163.com" "1111" , 点击 "登录"



成功页面



控制台打印



全部代码工作完成，让我们再顺一下[程序调用流程](#)

19) 程序调用流程

首先，[tomcat 启动](#)，加载 [web.xml](#)，指定 web.xml 中的配置

在 web.xml 中，配置了 [Struts2 控制器](#)，创建出来

```
16 <filter>
17   <filter-name>StrutsFilter</filter-name>
18   <filter-class>org.apache.struts2.dispatcher.ng.filter
19 </filter>
20 <filter-mapping>
21   <filter-name>StrutsFilter</filter-name>
22   <url-pattern>/*</url-pattern>
23 </filter-mapping>
```

其次，通过类 [contextConfigLocation](#) 找到指定的 Spring 配置文件 [ssh1.xml](#)，
创建出 [Spring 容器的实例](#)

```
7 <context-param>
8   <param-name>contextConfigLocation</param-name>
9   <param-value>classpath:ssh1.xml</param-value>
10 </context-param>
11
12 <listener>
13   <listener-class>org.springframework.web.context.Cor
14 </listener>
```

启动 tomcat 服务器后，主要完成了如上两项工作

接下来，

当用户访问 [login.jsp](#)，点击了“登录”按钮，发出了“[login.action](#)”请求，
“[login.action](#)”请求带着用户提交的 2 个参数 [user.email](#) 和 [user.password](#)

```
7 body style="font-size:30px;"
8 <form action="login.action" method="post">
9   用户名: <input type="text" name="user.email"> <br
10   密码: <input type="text" name="user.password"> <
11   <input type="submit" value="登录">
12 </form>
13 /body>
```

此时，“[login.action](#)”请求会根据 web.xml 中对 struts2 的配置找到 [struts.xml](#)

```

6 <struts>
7 <package name="ssh1-demo" extends="struts-default">
8   <action name="login" class="tarena.action.LoginAction"
9     <result name="success">/ok.jsp</result>
10    <result name="login">/login.jsp</result>
11  </action>
12 </package>

```

根据 struts.xml 文件中的配置，“login.action”请求发现 name=login 的 action 匹配，接着找到具体的 Action（LoginAction）进行处理

但是，因为导入的插件 struts2-spring-plugin.jar 的缘故，在该插件中的 struts-plugin.xml 中指定了对象工厂由 spring 来维护，

```

28 <struts>
29   <bean type="com.opensymphony.xwork2.ObjectFactory"
30
31   <!-- Make the Spring object factory the automatic
32   <constant name="struts.objectFactory" value="spring"
33
34   <constant name="struts.class.reloading.watchList" value=""
35   <constant name="struts.class.reloading.acceptClasses" value=""
36   <constant name="struts.class.reloading.reloadConfig" value=""
37

```

所以由插件 struts2-spring-plugin.jar 来创建 Action 实例，并且按照“名称对应”的规则，将 spring 容器中配置的 UserDao 对象注入到 LoginAction 中，

```

5
6 public class LoginAction {
7   //接收表单
8   private User user;
9   //默认采用名称对应规则将Spring容器中dao注入
10  private UserDao userDao;
11
12  public String execute(){
13    User usr = userDao.findByEmail(user.getEmail());
14

```

在 ssh1.xml 中对应的 bean

```

33
34 <bean id="userDao" class="tarena.dao.impl.HibernateL
35     <property name="sessionFactory" ref="mySession
36 </bean>
37

```

创建的方法就是调用 `dataSource`，通过 `SessionFactory` 等等一系列操作完成。

```

<bean id="myDataSource" class="org.apache.common
    <property name="driverClassName" value="com.
    <property name="url" value="jdbc:mysql:///te
    <property name="username" value="root">/pro
    <property name="password" value="root">/pro
    <property name="maxActive" value="10">/prop
    <property name="initialSize" value="2">/pro
</bean>

<bean id="mySessionFactory"
    class="org.springframework.orm.hibernate3.Local
    <property name="dataSource" ref="myDataSourc
    <property name="mappingResources">
        <list>
            <value>tarena/mapping/User.hbm.xml</

```

此时，`LoginAction` 实例被创建，执行 `execute` 方法，

在其中调用 `userDao.findByEmail` 方法，执行结果被返回后，交给 `Struts2` 控制器，

```

12 public String execute(){
13     User usr = userDao.findByEmail(user.getEmail());
14     if(usr != null){
15         if(usr.getPassword().equals(user.getPassword())
16             return "success";
17     }
18 }
19 return "login";
20 }
21

```

`Struts2` 控制器根据返回值 `"login"`，交给 `Result` 来处理

```

6 <struts>
7   <package name="ssh1-demo" extends="struts-default"
8     <action name="login" class="tarena.action.LoginAct
9       <result name="success">/ok.jsp</result>
10      <result name="login">/login.jsp</result>
11    </action>
12  </package>
13

```

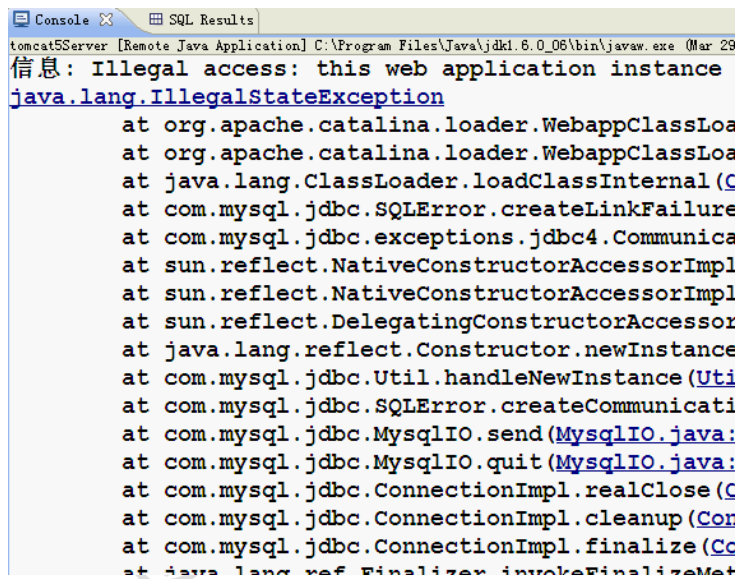
建议：

整合 SSH，建议一步一步来，首先整合好 Spring 和 Hibernate，测试成功后再加入 Struts2

提示：

如果出现这样的异常，将项目重新部署（或重新安装 Tomcat）

有可能是 Jar 包冲突的原因，限于时间，无法提供更好的解决方法。



```

tomcat5Server [Remote Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 29)
信息: Illegal access: this web application instance
java.lang.IllegalStateException
    at org.apache.catalina.loader.WebappClassLoa
    at org.apache.catalina.loader.WebappClassLoa
    at java.lang.ClassLoader.loadClassInternal(C
    at com.mysql.jdbc.SQLException.createLinkFailure
    at com.mysql.jdbc.exceptions.jdbc4.Communica
    at sun.reflect.NativeConstructorAccessorImpl
    at sun.reflect.NativeConstructorAccessorImpl
    at sun.reflect.DelegatingConstructorAccessor
    at java.lang.reflect.Constructor.newInstance
    at com.mysql.jdbc.Util.handleNewInstance(Uti
    at com.mysql.jdbc.SQLException.createCommunicati
    at com.mysql.jdbc.MySqlIO.send(MySqlIO.java:
    at com.mysql.jdbc.MySqlIO.quit(MySqlIO.java:
    at com.mysql.jdbc.ConnectionImpl.realClose(C
    at com.mysql.jdbc.ConnectionImpl.cleanup(Cor
    at com.mysql.jdbc.ConnectionImpl.finalize(Cc
    at java.lang.ref.Finalizer.invokeFinalizeMet

```

（案例结束）

2. 事务处理 **

【案例 2】SSH 整合_事务处理 **

案例描述

通过完成生成订单业务，掌握事务处理。

需要 d_order 表和 d_item 表

Tables_in_dangdang
d_book
d_category
d_category_product
d_item
d_order
d_product
d_receive_address
d_user

订单生成时的业务逻辑：向 d_order 插入 1 条数据的同时，向 t_item 中插入若干条数据

这就是一个独立的事务，

```
12 public User findByEmail(String email) {
13     String hql = "from User where email=?";
14     List<User> list = this.getHibernateTemplate().find(hql, email);
15     User user = null;
16     if(!list.isEmpty()){
17         user = list.get(0);
18     }
19     return user;
20 }
```

我们之前做的是单表操作，使用默认事务即可，但是涉及到稍复杂的多表操作时，我们就需要做事务处理。

如果我们按之前的方式，在 Action 中调用 DAO，是没有办法将两个 DAO 操作封装为一个事务的。

```
OrderAction.java
1 package tarena.action;
2
3 public class OrderAction {
4     public String execute() {
5         //1.调用OrderDao.save();保存
6         //2.调用ItemDao.save();保存
7         return "success";
8     }
9 }
10
```

为此，我们需要再分层，提出 Service，在 service 中进行事务控制。

参考代码

20) 使用工程 spring4

请下载 [spring4.zip](#)

首先，我们先将 UserService 抽取出来。

[重构登录功能](#)

21) 新建 UserService

```
package tarena.service;

import tarena.pojo.User;

public interface UserService {
    public boolean findLogin(User user);
}
```

22) 新建 UserServiceImpl

```
package tarena.service.impl;

import tarena.dao.UserDAO;
import tarena.pojo.User;
import tarena.service.UserService;

public class UserServiceImpl implements UserService {
    //默认采用名称对应规则将Spring容器中dao注入
    private UserDAO userDao;

    public UserDAO getUserDao() {return userDao;}
    public void setUserDao(UserDAO userDao) {
        this.userDao = userDao;
    }

    public boolean findLogin(User user) {

        User usr = userDao.findByEmail(user.getEmail());
```

```

        if(usr != null){
            if(usr.getPassword().equals(user.getPassword())){
                return true;
            }
        }
        return false;
    }
}

```

23) 修改 LoginAction

```

package tarena.action;

import tarena.pojo.User;
import tarena.service.UserService;

public class LoginAction {

    //接收表单信息的对象
    private User user;

    //默认采用名称对应规则将Spring容器中dao注入
    // private UserDAO userDao;
    // public UserDAO getUserDao() {return userDao;}
    // public void setUserDao(UserDAO userDao) {
    //     this.userDao = userDao;}
    //
    // public String execute(){
    //     User usr = userDao.findByEmail(user.getEmail());
    //     if(usr != null){
    //         if(usr.getPassword().equals(user.getPassword())){
    //             return "success";
    //         }
    //     }
    //     return "login";
    // }

    private UserService userService;

```

```

    public UserService getUserService() {return userService;}
    public void setUserService(UserService userService) {
        this.userService = userService;
    }

    public String execute(){
        if(userService.findLogin(user)){
            return "success";
        }
        return "login";
    }

    public User getUser() {return user;}
    public void setUser(User user) {this.user = user;}
}

```

24) 修改 ssh.xml

```

<bean id="myDataSource"
    class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName"
        value="com.mysql.jdbc.Driver"> </property>
    <property name="url" value="jdbc:mysql:///test"> </property>
    <property name="username" value="root"> </property>
    <property name="password" value="root"> </property>
    <property name="maxActive" value="10"> </property>
    <property name="initialSize" value="2"> </property>
</bean>

<bean id="mySessionFactory"
    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="myDataSource"> </property>
    <property name="mappingResources">
        <list>
            <value>tarena/mapping/User.hbm.xml</value>
        </list>
    </property>

```

```
<property name="hibernateProperties">
    <props>
        <prop key="hibernate.dialect">
            org.hibernate.dialect.MySQL5Dialect</prop>
        <prop key="hibernate.show_sql">true</prop>
        <prop key="hibernate.format_sql">true</prop>
    </props>
</property>
</bean>

<bean id="userDao" class="tarena.dao.impl.HibernateUserDAOImpl">
    <property name="sessionFactory" ref="mySessionFactory">
    </property>
</bean>

<bean id="userService" class="tarena.service.impl.UserServiceImpl">
    <property name="userDao" ref="userDao"></property>
</bean>

</beans>
```

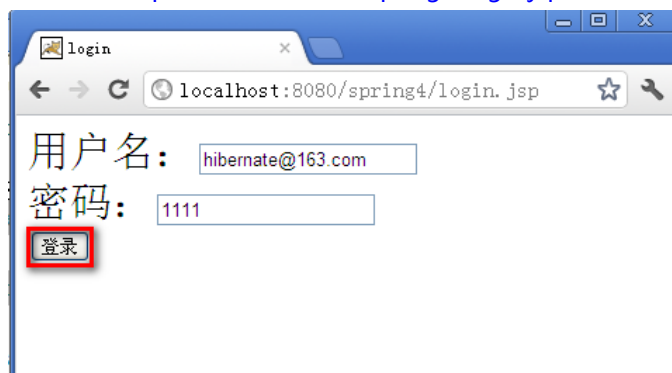
25) 部署项目

注意：部署项目的时候可能遇到这个异常，可以先忽略

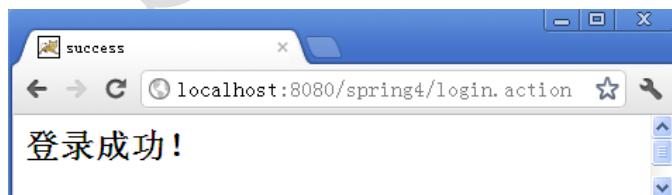
```
Console  SQL Results
tomcat5Server [Remote Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 29
信息: Illegal access: this web application instance
java.lang.IllegalStateException
    at org.apache.catalina.loader.WebappClassLoader
    at org.apache.catalina.loader.WebappClassLoader
    at java.lang.ClassLoader.loadClassInternal(C
    at com.mysql.jdbc.SQLException.createLinkFailure
    at com.mysql.jdbc.exceptions.jdbc4.Communicat
    at sun.reflect.NativeConstructorAccessorImpl
    at sun.reflect.NativeConstructorAccessorImpl
    at sun.reflect.DelegatingConstructorAccessor
    at java.lang.reflect.Constructor.newInstance
    at com.mysql.jdbc.Util.handleNewInstance(Uti
    at com.mysql.jdbc.SQLException.createCommunicati
    at com.mysql.jdbc.MysqlIO.send(MysqlIO.java:
    at com.mysql.jdbc.MysqlIO.quit(MysqlIO.java:
    at com.mysql.jdbc.ConnectionImpl.realClose(C
    at com.mysql.jdbc.ConnectionImpl.cleanup(Con
    at com.mysql.jdbc.ConnectionImpl.finalize(Cc
    at java.lang.ref.Finalizer.invokeFinalizeMet
    at java.lang.ref.Finalizer.runFinalizer(Fina
    at java.lang.ref.Finalizer.access$100(Finali
```

26) 测试

a. 访问 <http://localhost:8080/spring4/login.jsp>



b. 点击登录



测试成功

如上所示，

如果想管理事务的话，就需要抽取业务层 Service（由 Service 调用 DAO 的方式）

接下来，我们进行事务控制。

事务控制有两种：

一种是[编程式事务控制](#)（通过代码方式控制事务逻辑），

一种是配置型的，我们称为[声明式事务控制](#)，

如下我们使用配置型的，交由 Spring 来控制

27) 修改 ssh.xml

加入声明式事务控制

所有以 save 开头的方法，声明事务管理策略为“REQUIRED”，表示必须进行事务控制
update*/delete*/find*意思一样

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                           http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
                           http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
                           http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

    <bean id="myDataSource"
class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
value="com.mysql.jdbc.Driver"></property>
        <property name="url" value="jdbc:mysql:///test"></property>
        <property name="username" value="root"></property>
        <property name="password" value="root"></property>
        <property name="maxActive" value="10"></property>
        <property name="initialSize" value="2"></property>
    </bean>

    <bean id="mySessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
```

```

    <property name="dataSource" ref="myDataSource"></property>
    <property name="mappingResources">
        <list>
            <value>tarena/mapping/User.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">
                org.hibernate.dialect.MySQL5Dialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.format_sql">true</prop>
        </props>
    </property>
</bean>

<bean id="userDao" class="tarena.dao.impl.HibernateUserDAOImpl">
    <property name="sessionFactory" ref="mySessionFactory">
    </property>
</bean>

<bean id="userService" class="tarena.service.impl.UserServiceImpl">
    <property name="userDao" ref="userDao"></property>
</bean>

<!-- 声明式事务控制 -->
<bean id="txManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="mySessionFactory">
    </property>
</bean>

<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="save*" propagation="REQUIRED"/>
        <tx:method name="update*" propagation="REQUIRED"/>
        <tx:method name="delete*" propagation="REQUIRED"/>
        <tx:method name="find*" read-only="true"

```

```

        propagation="NOT_SUPPORTED"/>
        <tx:method name="*" propagation="REQUIRED"/>
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:pointcut expression="within(tarena.service..*)"
        id="servicePointcut"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="servicePointcut"/>
</aop:config>

</beans>

```

- ✓ **注意：**使用 Spring 管理事务，需要引入命名空间
- ✓ propagation 属性用来指明事务管理策略
- ✓ propagation="NOT_SUPPORTED" 表示不使用事务管理策略
- ✓ 我们使用<aop:advisor>引用<tx:advice>

如上所示，使用 **Spring 的优点**就在于：

首先，可以使用 IoC 方式进行注入

其次，可以使用 AOP 的思想进行切面编程

再次，就是控制事务相对简单。

需要**注意**的是，spring 在底层对异常处理的很干净，所以出现异常后，控制台基本看不到哪里出错了，我们需要**引入 log4j.jar**，借助于 log4j 可以查看到错误源。

28) 导入 log4j.jar

29) 拷贝 log4j.properties 到 src 目录下

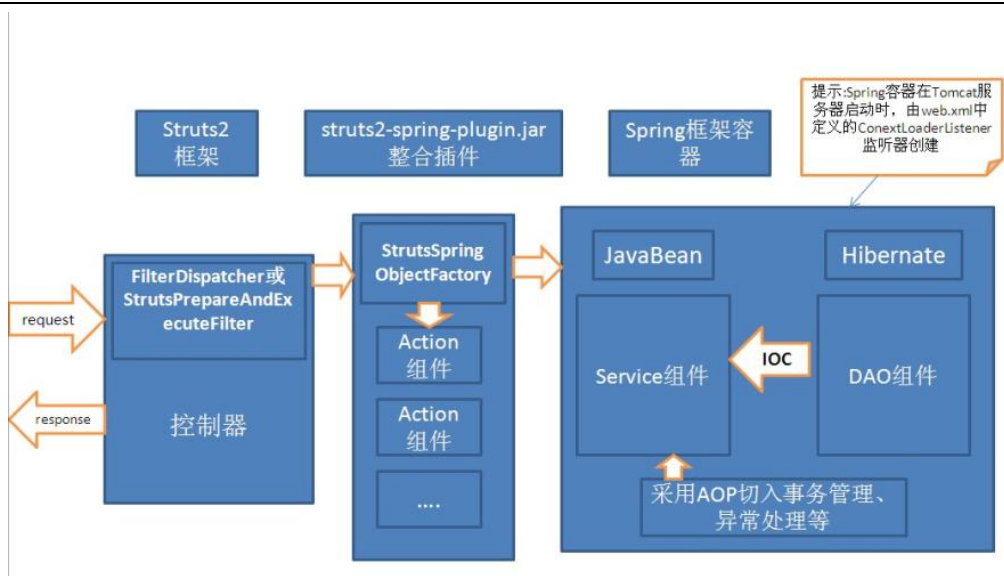
30) 测试（略）

（案例结束）

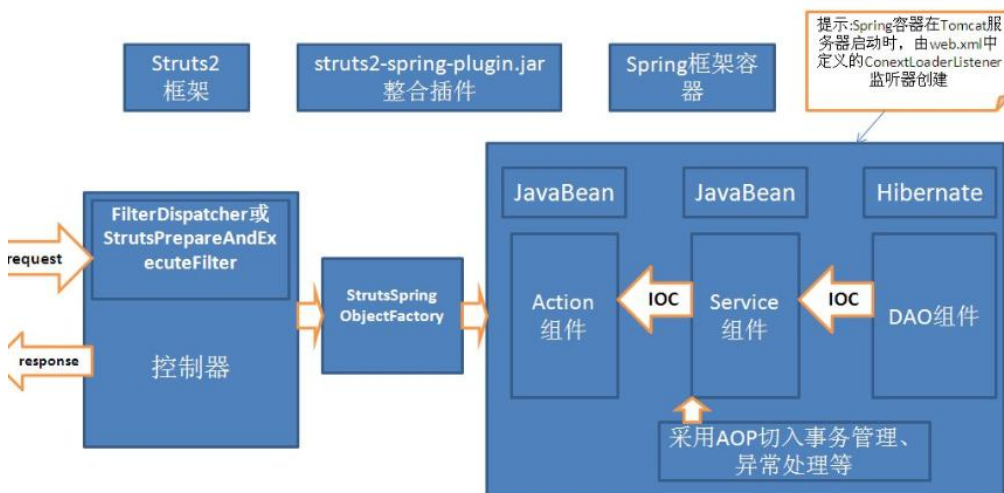
3. SSH 整合_方案 02

我们先回顾一下

整合方案 01



整合方案 02



对比方案 1 和方案 2

方案 2 中, Action 组件不再由整合插件创建了, Action 组件也被纳入到 Spring 容器当中;
整合插件虽然不再创建 Action 对象, 但是我们仍然需要整合插件来访问 Spring 容器

【案例 3】SSH 整合_方案 2 **

案例描述

两个知识点的演示

其一，SSH 整合的第二个方案

其二，Spring+JDBC+Struts2

参考代码

31) 使用工程 spring4

32) 修改 ssh.xml

增加<bean name="loginAction">

```
<bean id="myDataSource"
    class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver">
</property>
    <property name="url" value="jdbc:mysql:///test"></property>
    <property name="username" value="root"></property>
    <property name="password" value="root"></property>
    <property name="maxActive" value="10"></property>
    <property name="initialSize" value="2"></property>
</bean>

<bean id="mySessionFactory"
    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="myDataSource"></property>
    <property name="mappingResources">
        <list>
            <value>tarena/mapping/User.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">
                org.hibernate.dialect.MySQL5Dialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.format_sql">true</prop>
        </props>
    </property>
</bean>
```

```

<bean id="userDao" class="tarena.dao.impl.HibernateUserDAOImpl">
    <property name="sessionFactory" ref="mySessionFactory">
    </property>
</bean>

<bean id="userService" class="tarena.service.impl.UserServiceImpl">
    <property name="userDao" ref="userDao"> </property>
</bean>

<bean id="loginAction" class="tarena.action.LoginAction">
    <property name="userService" ref="userService"> </property>
</bean>

<!-- 声明式事务控制 -->
<bean id="txManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="mySessionFactory">
    </property>
</bean>

<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="save*" propagation="REQUIRED"/>
        <tx:method name="update*" propagation="REQUIRED"/>
        <tx:method name="delete*" propagation="REQUIRED"/>
        <tx:method name="find*" read-only="true"
            propagation="NOT_SUPPORTED"/>
        <tx:method name="*" propagation="REQUIRED"/>
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:pointcut expression="within(tarena.service..*)"
        id="servicePointcut"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="servicePointcut"/>
</aop:config>

```

```
</beans>
```

33) 修改 struts.xml

与方案 1 相比，此处我们只要调用 Spring 容器中管理的 bean 组件即可

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.1.7//EN"
    "http://struts.apache.org/dtds/struts-2.1.7.dtd">

<struts>
    <package name="ssh1-demo" extends="struts-default">
        <action name="login" class="loginAction">
            <result name="success">/ok.jsp</result>
            <result name="login">/login.jsp</result>
        </action>
    </package>
</struts>
```

34) 测试 (略)

如上，两种 SSH 整合的方案就结束了。

此时，我们再提新的需求，老板一句话“不喜欢 Hibernate，用 JDBC”，

Spring 整合 JDBC、整合 Struts2

还好使用 Spring 框架进行了解耦，我们只需要增加实现类 JDBCUserDAOImpl，并且修改配置文件 ssh.xml 即可

35) 新建 JdbcUserDAOImpl

```
package tarena.dao.impl;

import java.util.List;
import org.springframework.jdbc.core.support.JdbcDaoSupport;
import tarena.dao.UserDAO;
import tarena.pojo.User;
import tarena.pojo.UserMapper;

public class JdbcUserDAOImpl
    extends JdbcDaoSupport implements UserDAO {
```

```

public User findByEmail(String email) {
    String sql = "select * from d_user where email=?";
    List<User> list =
        this.getJdbcTemplate().query(
            sql, new Object[]{email}, new UserMapper());
    User user = null;
    if(!list.isEmpty()){
        user = list.get(0);
    }
    return user;
}
}

```

36) 新建 UserMapper

```

package tarena.pojo;

import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;

public class UserMapper implements RowMapper {
    private static final String ID = "id";
    public Object mapRow(ResultSet rs, int index)
        throws SQLException {
        User user = new User();
        user.setId(rs.getInt(ID));
        user.setPassword(rs.getString("password"));
        if(rs.getString("email") != null){
            user.setEmail(rs.getString("email"));
        }
        user.setNickname(rs.getString("nickname"));
        if(rs.getString("is_email_verify").equals("Y")){
            user.setEmailVerify(true);
        }else{
            user.setEmailVerify(false);
        }
        user.setEmailVerifyCode(rs.getString("email_verify_code"));
    }
}

```

```

        user.setLastLoginTime(rs.getLong("last_login_time"));
        return user;
    }
}

```

37) 修改 ssh.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:aop="http://www.springframework.org/schema/aop"
        xmlns:tx="http://www.springframework.org/schema/tx"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
        http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
        http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

    <bean id="myDataSource"
class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
value="com.mysql.jdbc.Driver"></property>
        <property name="url" value="jdbc:mysql:///test"></property>
        <property name="username" value="root"></property>
        <property name="password" value="root"></property>
        <property name="maxActive" value="10"></property>
        <property name="initialSize" value="2"></property>
    </bean>

    <!--注释掉Hibernate配置的bean SessionFactory-->
    <!--
    <bean id="mySessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

```

```

    <property name="dataSource" ref="myDataSource"></property>
    <property name="mappingResources">
        <list>
            <value>tarena/mapping/User.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">
                org.hibernate.dialect.MySQL5Dialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.format_sql">true</prop>
        </props>
    </property>
</bean>

<bean id="userDao" class="tarena.dao.impl.HibernateUserDAOImpl">
    <property name="sessionFactory"
ref="mySessionFactory"></property>
</bean>

<bean id="userService" class="tarena.service.impl.UserServiceImpl">
    <property name="userDao" ref="userDao"></property>
</bean>
-->

<bean id="jdbcUserDao" class="tarena.dao.impl.JdbcUserDAOImpl">
    <property name="dataSource" ref="myDataSource"></property>
</bean>

<bean id="userService" class="tarena.service.impl.UserServiceImpl">
    <property name="userDao" ref="jdbcUserDao"></property>
</bean>

<bean id="loginAction" class="tarena.action.LoginAction">
    <property name="userService" ref="userService"></property>
</bean>

```

```

<!-- 声明式事务控制 -->
<!--
<bean id="txManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory"
ref="mySessionFactory"></property>
</bean>
-->

<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="myDataSource"></property>
</bean>

<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="save*" propagation="REQUIRED"/>
        <tx:method name="update*" propagation="REQUIRED"/>
        <tx:method name="delete*" propagation="REQUIRED"/>
        <tx:method name="find*" read-only="true"
            propagation="NOT_SUPPORTED"/>
        <tx:method name="*" propagation="REQUIRED"/>
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:pointcut expression="within(tarena.service..*)"
        id="servicePointcut"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="servicePointcut"/>
</aop:config>

</beans>

```

38) 测试 (略)

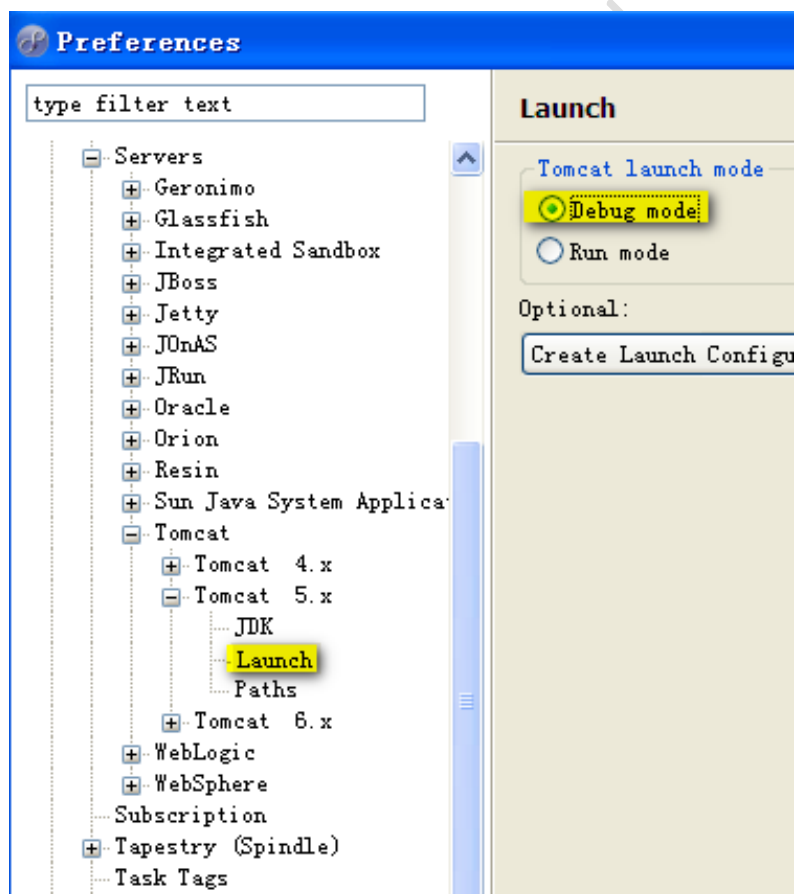
在编写程序的过程中，会调试大量的 Exception，使用 System.out.println() 是一种简便的方法。另外，MyEclipse 还提供了调试程序的 [Debug 工具](#)

Debug 工具演示

假设，我们现在想跟踪 JdbcUserDAOImpl 中的实现方法 findByEmail，
首先，设置调试断点，
在需要跟踪的代码处双击“序号前面”，将出现一个蓝色小点，

```
9 public class JdbcUserDAOImpl extends JdbcDaoSupport
10
11 public User findByEmail(String email) {
12     String sql = "select * from d_user where email=?";
13     User user = (User)this.getJdbcTemplate().queryF
14     return user;
15 }
16
```

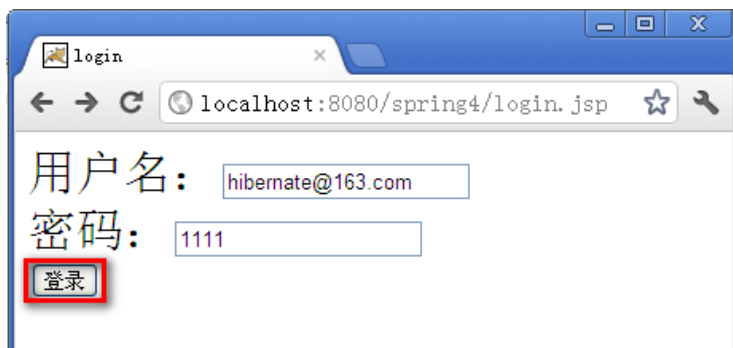
其次，设置 tomcat 为 debug 模式



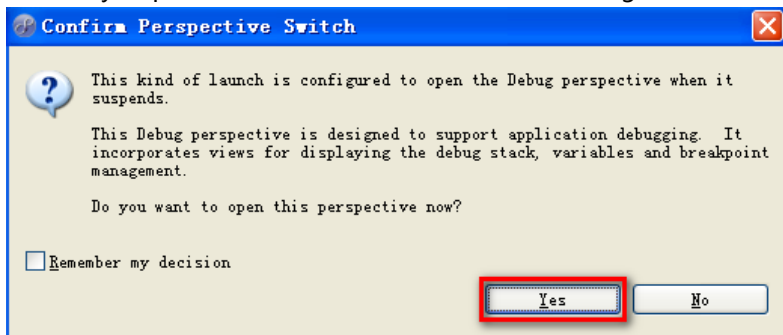
开始调试

访问 <http://localhost:8080/spring4/login.jsp>

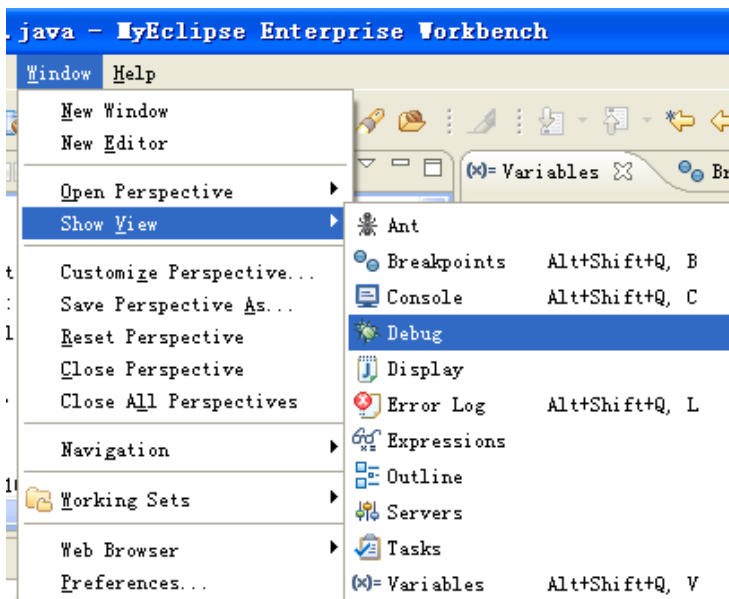
点击 “登录” 按钮



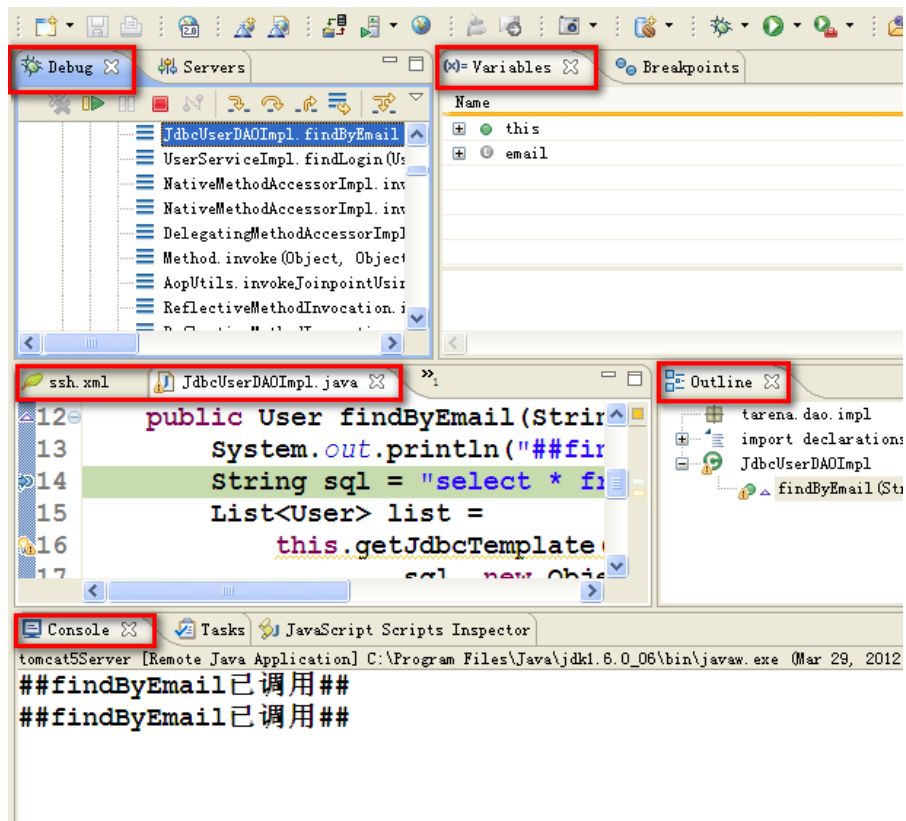
此时，MyEclipse 工具将弹出对话框询问是否打开 “Debug” 视图，选择 “Yes”



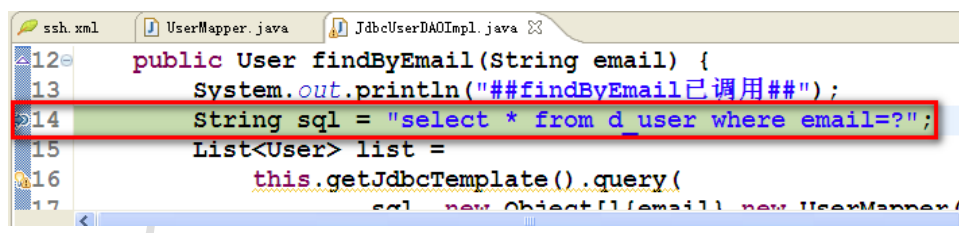
从 “show view” 中打开也一样



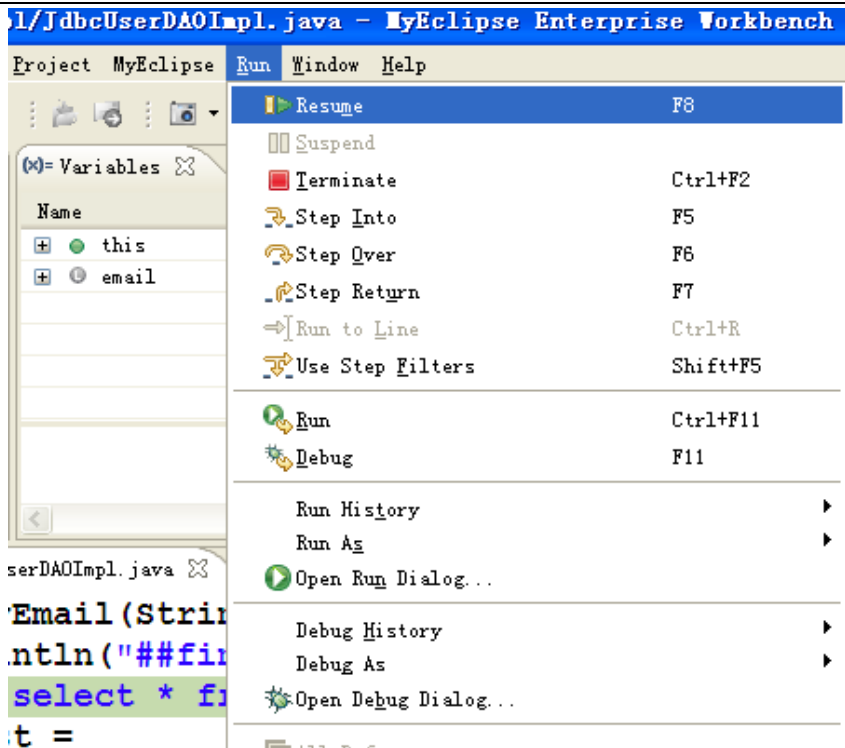
整个 Debug 视图由如下几部分组成



此时，程序停在了这里



点击 “Run” 选项，就有许多快捷键



使用说明

按键	快捷键	功能
Resume	F8	表示从当前断点处，继续向下执行，即放弃当前断点，直接向下执行（如果有多个断点，则停在下一个断点处）
Step Over	F6	表示从当前设置断点处的代码向下，一行一行执行
Step Into	F5	如果当前设置断点代码处，调用了子方法，比如 <code>String s = foo.findAll();</code> 当想进入到该子方法 <code>findAll()</code> 当中去时，就可以点击 F5
Step Return	F7	万一不小心跳入到别的方法中，该快捷点可以再跳出来，回到上一级

一般情况下，如果设置断点处，有自定义的方法，那么就可以按 F5 Step Into 跟进，如果是 JDK 或者框架提供的方法，那就按 F6，直接执行下一行代码即可。

当断点调试程序结束，如果想取消设置的断点，可以在断点的蓝色小点上再双击一下就取消了。也可以打开 BreakPoint 视图，将 checkBox 前的对勾去掉即可（如果设置了多个断点时，好用）

