下午五点，终于搞定，高兴…今天是愚人节，刚去做了一件公益事情，自娱了一把，我们公司在一座五星写字楼里面，装修豪华，厕所的地板是橡胶瓷砖的那种，每次小便器前总有一块瓷砖上面布满了“地图”，打扫卫生的人写了一张纸贴在墙上“大家注意，往前一步，文明一大片”，我刚刚偷偷的换了一张纸，上书“尿不进去，说明你短”，哈哈…开始写这个文档，我是实习生，很菜，很多东西不会，这个东西我配置了一天，刚配好，记录下来。

首先声明，我这个事针对J2EE程序员的，不是针对Flex的，我的初衷就是在J2EE里加入Flex支持。有人说用flex没必要用Struts，但是，我说用上最好，因为你不是每个页面都用flex处理，很多很复杂的流程flex投入巨大，所以还是普通jsp搞定最好。所以加上一层action不会影响太多性能的。嘿嘿。

这个文档是建立在你已经学习了三大框架的而且已经会三大框架的整合开发的基础上的。要对整个流程熟悉，才能自己完成这个项目。其中有什么问题大家可以加入高级群27900549来讨论。也可以给我发邮件索要工程源文件。在此提醒大家一点，flex加入后的S2SH项目不能直接导入，不然会有很多问题，主要是tomcat的webapps下没有相关的配置文件，所以还是按照这个文档一步一步来吧。

Flex4的插件安装方法网上有很多的，必须MyEclipse7.5以上，但是9.0的不要尝试了。我已经试过了，失败了。我的配置如下：

* MyEclipse8.5；
* Flex4插件版；
* Tomcat 6.0.26；
* BlazeDS 4.0；

1, new🡪web project,命名为flextest，这个我想不用截图吧

2 加入S2SH三大框架的jar包，也可以使用MyEclipse导入，配置各个配置文件。这个就是三大框架集成过程，不想多啰嗦。

完成后的架构图先上一张：



这是我完全做完后的效果图

这个项目虽然很小很简单，但是我采用的是jsp🡪action🡪service🡪dao🡪数据库的标准工程做法。

其中src中的hibernateutil包可以不要，这个是动力节点老师王勇写的工具包，主要就是导出Model到数据库中，整个例子很简单。我将代码全部贴出来。到这一步你的工程里还没有flex任何东西，但是我的截图里有，这个你不用急往下看，你的SSH项目能运行起来才是关键。

User.java:

**package** com.xaut.bean;

**public** **class** User {

**private** Integer id;

**private** String username;

**private** String password;

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

}

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>

<class name=*"com.xaut.bean.User"*>

<id name=*"id"* type=*"java.lang.Integer"*>

<generator class=*"identity"* />

</id>

<property name=*"username"* type=*"java.lang.String"*/>

<property name=*"password"* type=*"java.lang.String"*/>

</class>

</hibernate-mapping>

baseDAO.java

package com.xaut.dao;

import java.io.Serializable;

import java.util.List;

@SuppressWarnings("rawtypes")

public interface BaseDAO {

/\*\* 加载指定ID的持久化对象 \*/

public Object loadById(Class clazz,Serializable id);

}

baseDAOImpl.java

package com.xaut.dao;

import java.io.Serializable;

import java.util.List;

import org.hibernate.Hibernate;

import org.hibernate.HibernateException;

import org.hibernate.Query;

import org.hibernate.Session;

import org.springframework.orm.hibernate3.HibernateCallback;

import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

@SuppressWarnings("rawtypes")

public class BaseDAOImpl extends HibernateDaoSupport implements BaseDAO {

/\*\* 加载指定ID的持久化对象 \*/

public Object loadById(Class clazz, Serializable id) {

return this.getHibernateTemplate().load(clazz, id);

}

/\*\*加载满足条件的持久化对象\*/

public Object loadObject(String hql) {

final String hql1 = hql;

Object obj = null;

List list = getHibernateTemplate().executeFind(new HibernateCallback(){

public Object doInHibernate(Session session) throws HibernateException{

Query query = session.

createQuery(hql1);

return query.list();

}

});

if(list.size()>0)obj=list.get(0);

return obj;

}

EportDB.java

package com.xaut.hibernateutil;

import org.hibernate.cfg.Configuration;

import org.hibernate.tool.hbm2ddl.SchemaExport;

public class ExportDB {

public static void main(String[] args) {

//默认读取hibernate.cfg.xml文件

Configuration cfg = new Configuration().configure();

SchemaExport export = new SchemaExport(cfg);

export.create(true, true);

}

}

HibernateUtils.java

package com.xaut.hibernateutil;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

public class HibernateUtils {

private static SessionFactory factory;

static{

try{

Configuration cfg = new Configuration().configure();

factory = cfg.buildSessionFactory();

}catch(Exception e){

e.printStackTrace();

}

}

public static Session getSession(){

return factory.openSession();

}

public static void closeSession(Session session){

if (session != null) {

if (session.isOpen()) {

//关闭session

session.close();

}

}

}

public static SessionFactory getSessionFactory(){

return factory;

}

}

**UserService.java**

**package** com.xaut.service;

**import** com.xaut.bean.User;

**public** **interface** UserService {

**public** User userlogin(String username,String password);

}

UserServiceImpl.java

package com.xaut.service;

import com.xaut.bean.User;

import com.xaut.dao.BaseDAO;

public class UserServiceImpl implements UserService{

/\*\* 通过依赖注入DAO组件实例 \*/

BaseDAO baseDAO;

public BaseDAO getBaseDAO() {

return baseDAO;

}

public void setBaseDAO(BaseDAO baseDAO) {

this.baseDAO = baseDAO;

}

public User userlogin(String username,String password){

String hql="from User as a where a.username='"+username+"' and a.password='"+password+"'";

return (User)baseDAO.loadObject(hql);

}

}

UserAction.java

package com.xaut.web.action;

import java.io.IOException;

import javax.servlet.http.HttpServletResponse;

import org.apache.struts2.ServletActionContext;

import com.opensymphony.xwork2.ActionSupport;

import com.opensymphony.xwork2.ModelDriven;

import com.xaut.bean.User;

import com.xaut.service.UserService;

public class UserAction extends ActionSupport implements ModelDriven<User> {

private UserService userService;

private User user=new User();

public User getModel() {

// TODO Auto-generated method stub

return user;

}

public String login() throws IOException{

HttpServletResponse response = ServletActionContext.getResponse();

User tempUser = userService.userlogin(user.getUsername(), user.getPassword());

if(tempUser!=null){

response.getWriter().write("success");

}else {

response.getWriter().write("index");

}

return null;

}

public UserService getUserService() {

return userService;

}

public void setUserService(UserService userService) {

this.userService = userService;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

}

到这一步要说明一下，login()不能像传统Action那样写一个return语句就了事。必须采用我这种方法，要不然flex获得的将是整个Action页面。这就是和flex结合时要注意的地方。

Hibernate.cfg.xml

<?xml version=*'1.0'* encoding=*'UTF-8'*?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*"hibernate.connection.url"*>jdbc:mysql://localhost:3306/flextest</property>

<property name=*"hibernate.connection.username"*>root</property>

<property name=*"hibernate.connection.password"*>111111</property>

<property name=*"hibernate.dialect"*> org.hibernate.dialect.MySQLDialect</property>

<property name=*"hibernate.show\_sql"*>true</property>

<property name=*"hibernate.format\_sql"*>false</property>

<mapping resource=*"com/xaut/bean/User.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

Struts.xml

<?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"* namespace=*"/"* extends=*"struts-default"*>

<action name=*"userLogin"* class=*"userAction"* method=*"login"*>

<result name=*"index"*>/index.jsp</result>

<result name=*"success"*>/login\_success.jsp</result>

</action>

</package>

</struts>

applicationContext.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:p=*"http://www.springframework.org/schema/p"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd"*>

<bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>

<property name=*"driverClass"* value=*"com.mysql.jdbc.Driver"*/>

<property name=*"jdbcUrl"* value=*"jdbc:mysql://localhost:3306/flextest?useUnicode=true&amp;characterEncoding=utf8"*/>

<property name=*"user"* value=*"root"*/>

<property name=*"password"* value=*"111111"*/>

<property name=*"maxPoolSize"* value=*"20"*/>

<property name=*"minPoolSize"* value=*"2"*/>

<property name=*"initialPoolSize"* value=*"2"*/>

<property name=*"maxIdleTime"* value=*"20"*/>

</bean>

<bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate3.LocalSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSource"*/>

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</prop>

<prop key=*"hibernate.show\_sql"*>true</prop>

</props>

</property>

<property name=*"mappingResources"*>

<list>

<value>com/xaut/bean/User.hbm.xml</value>

</list>

</property>

</bean>

<bean id=*"transactionManager"* class=*"org.springframework.orm.hibernate3.HibernateTransactionManager"*>

<property name=*"sessionFactory"* ref=*"sessionFactory"*/>

</bean>

<bean id=*"transactionInterceptor"* class=*"org.springframework.transaction.interceptor.TransactionInterceptor"*>

<property name=*"transactionManager"* ref=*"transactionManager"*/>

<property name=*"transactionAttributes"*>

<props>

<prop key=*"\*"*>PROPAGATION\_REQUIRED</prop>

</props>

</property>

</bean>

<bean class=*"org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator"*>

<property name=*"beanNames"*>

<list>

<value>UserService</value>

</list>

</property>

<property name=*"proxyTargetClass"*>

<value>true</value>

</property>

<property name=*"interceptorNames"*>

<list>

<value>transactionInterceptor</value>

</list>

</property>

</bean>

<bean id=*"baseDAO"* class=*"com.xaut.dao.BaseDAOImpl"*>

<property name=*"sessionFactory"* ref=*"sessionFactory"*></property>

</bean>

<bean id=*"userService"* class=*"com.xaut.service.UserServiceImpl"*>

<property name=*"baseDAO"* ref=*"baseDAO"*/>

</bean>

<bean id=*"userAction"* class=*"com.xaut.web.action.UserAction"* scope=*"prototype"*>

<property name=*"userService"* ref=*"userService"*/>

</bean>

</beans>

Web.xml

<?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">

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<filter>

<filter-name>struts2</filter-name>

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

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<display-name>BlazeDS</display-name>

<description>BlazeDS Application</description>

<!-- Http Flex Session attribute and binding listener support -->

<listener>

<listener-class>flex.messaging.HttpFlexSession</listener-class>

</listener>

<!-- MessageBroker Servlet -->

<servlet>

<servlet-name>MessageBrokerServlet</servlet-name>

<display-name>MessageBrokerServlet</display-name>

<servlet-class>flex.messaging.MessageBrokerServlet</servlet-class>

<init-param>

<param-name>services.configuration.file</param-name>

<param-value>/WEB-INF/flex/services-config.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<!-- begin rds

<servlet>

<servlet-name>RDSDispatchServlet</servlet-name>

<display-name>RDSDispatchServlet</display-name>

<servlet-class>flex.rds.server.servlet.FrontEndServlet</servlet-class>

<init-param>

<param-name>useAppserverSecurity</param-name>

<param-value>true</param-value>

</init-param>

<load-on-startup>10</load-on-startup>

</servlet>

<servlet-mapping id="RDS\_DISPATCH\_MAPPING">

<servlet-name>RDSDispatchServlet</servlet-name>

<url-pattern>/CFIDE/main/ide.cfm</url-pattern>

</servlet-mapping>

end rds -->

<servlet-mapping>

<servlet-name>MessageBrokerServlet</servlet-name>

<url-pattern>/messagebroker/\*</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

</welcome-file-list>

<!-- for WebSphere deployment, please uncomment -->

<!--

<resource-ref>

<description>Flex Messaging WorkManager</description>

<res-ref-name>wm/MessagingWorkManager</res-ref-name>

<res-type>com.ibm.websphere.asynchbeans.WorkManager</res-type>

<res-auth>Container</res-auth>

<res-sharing-scope>Shareable</res-sharing-scope>

</resource-ref>

-->

</web-app>

建好工程后建好数据库就可以运行ExportDB.java来导出表了，很方便。王勇老师很强大，建议大家有时间多看看他的视频教程。

至此你的j2ee项目应该能运行了吧。

添加flex支持：

首先得有BlazeDS，解压开里面的blazeds.war。打开WEB-INF找到flex和lib文件夹，复制到项目的WEB-INF文件夹下，有充满文件选择yes-to-all.

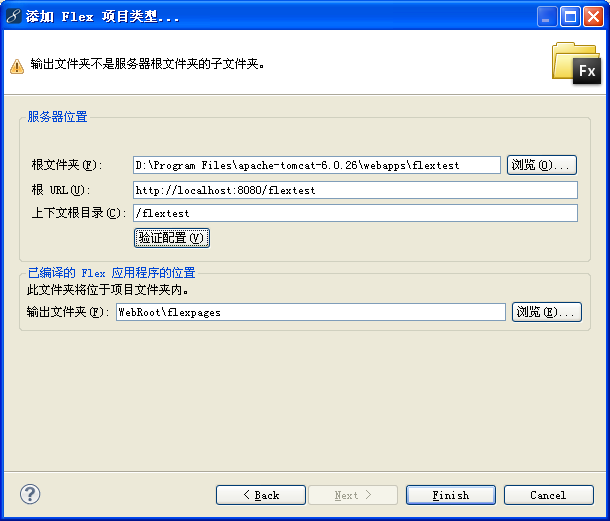
3 在WebRoot下新建一个文件夹flexpages准备保存flex文件。到这一步先不要急，将项目部署到tomcat下，并启动tomcat，一定要启动，不然下面加入flex时Myeclipse无法获取flex配置文件services-config.xml，导致不能继续下去。

4 右键工程名，选择“添加/更改项目类型”，选择“添加Flex项目类型”

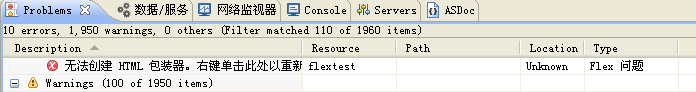


这一步具体选择应该和图中保持一致。

5，Next：这一步如果你没有错误的话根文件夹就会自动获取到信息。只需要自己改改就OK了。



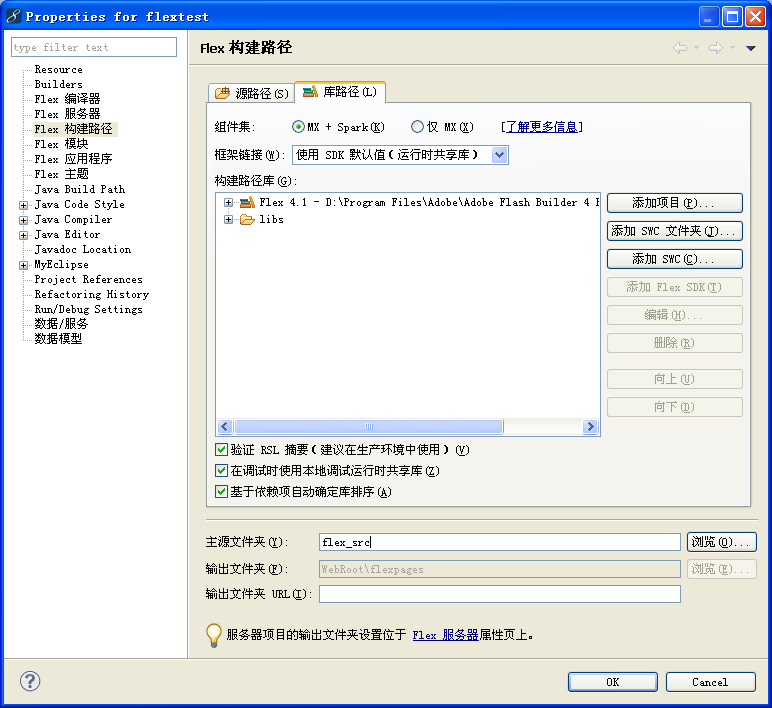
6：点击验证配置，Finish后，会发现项目上有一个红叉，打开Problem视图：



显示有一个问题，右键🡪重新创建HTML模板；OK；

7：新的问题来了，你会发现flex源文件和java源文件在一个文件夹中，这样是不是很不爽，想要实现项目有效管理，就应该将两者分开；

在项目名上右击🡪Properties



在下面的主源文件夹，将src改成flex\_src;OK！输出文件夹不用管。

8：将src里的mxml文件拖到刚建好的flex\_src下。OK基本已经成型。

9：在web.xml.中添加以下内容

<!-- Http Flex Session attribute and binding listener support -->

<listener>

<listener-class>flex.messaging.HttpFlexSession</listener-class>

</listener>

<!-- MessageBroker Servlet -->

<servlet>

<servlet-name>MessageBrokerServlet</servlet-name>

<servlet-class>flex.messaging.MessageBrokerServlet</servlet-class>

<init-param>

<param-name>services.configuration.file</param-name>

<param-value>/WEB-INF/flex/services-config.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>MessageBrokerServlet</servlet-name>

<url-pattern>/messagebroker/\*</url-pattern>

</servlet-mapping>

最终的web.xml如下所示；

<?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"*>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- Http Flex Session attribute and binding listener support -->

<listener>

<listener-class>flex.messaging.HttpFlexSession</listener-class>

</listener>

<!-- MessageBroker Servlet -->

<servlet>

<servlet-name>MessageBrokerServlet</servlet-name>

<servlet-class>flex.messaging.MessageBrokerServlet</servlet-class>

<init-param>

<param-name>services.configuration.file</param-name>

<param-value>/WEB-INF/flex/services-config.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>MessageBrokerServlet</servlet-name>

<url-pattern>/messagebroker/\*</url-pattern>

</servlet-mapping>

</web-app>

9：编写flextest.mxml;内容如下

<?xml version="1.0" encoding="utf-8"?>

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">

<mx:Script>

<![CDATA[

**import** flash.net.navigateToURL;

**import** mx.controls.Alert;

**import** mx.rpc.events.ResultEvent;

**private** **function** loginHandle():**void** {

doLogin.send();

}

**private** **function** loginResult(evt:ResultEvent):**void** {

**if**(evt.result.toString()==**"success"**)

{

loginSuccess();

}**else** {

Alert.show(**"用户名或密码错误！"**);

}

}

**private** **function** loginSuccess():**void** {

**var** successful:String = **"login\_success.jsp"**;

**var** request:URLRequest = **new** URLRequest(successful);

navigateToURL(request,**"\_self"**);

}

]]>

</mx:Script>

<mx:HTTPService id="doLogin" result="loginResult(event)" resultFormat="text"

showBusyCursor="true" method="POST" url="userLogin.action" >

<mx:request xmlns="">

<username>

{username.text}

</username>

<password>

{password.text}

</password>

</mx:request>

</mx:HTTPService>

<mx:Panel x="85" y="53" width="412" height="256" layout="absolute">

<mx:Label x="31" y="49" text="用户名" width="72"/>

<mx:Label x="31" y="96" text="密 码" width="43"/>

<mx:TextInput x="82" y="47" id="username"/>

<mx:TextInput x="82" y="94" id="password" displayAsPassword="true"/>

<mx:Button x="172" y="163" label="登录" click="loginHandle()" />

</mx:Panel>

</mx:Application>

在WEB-INF下flex文件夹中romoting-config.xml中添加如下内容

<destination id=*"doLogin"*>

<properties>

<source>com.xaut.web.action.UserAction</source>

</properties>

</destination>

在flexpages下新建一个jsp页面命名login\_success.jsp；里面内容随意修改，登录成功后显示。你也可以使用flex编写。至此所有配置全部搞完。

重新部署项目，启动服务器运行<http://localhost:8080/flextest/flexpages/flextest.html>;看看我的效果图；



登录成功



这个项目是我用flexlogin做的，所以运行时是flexlgoin.html,我为了给大家截图，新建了flextest这个工程，其实命名无所谓。关键是你自己心里要清楚。

总结：其实flextest.mxml文件可以很简单的将数据传给Action，让java去处理，这样的更简单，但是只是让flex做静态页面，失去了flex的作用。我这样写可以让flex和用户交互更丰富一些。