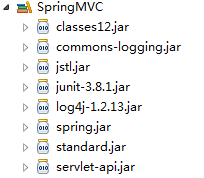
SSH整合

# 搭建Spring框架

由于这个是Web Application，所以需要在用eclipse新建project时，选择Dynamic Web Project。

## Spring2.0所需lib库

以下为SpringMVC的示例库，须看实际要求增减



## 建立Web.xml

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

<web-app id=*"WebApp\_ID"* version=*"2.4"* xmlns=*"http://java.sun.com/xml/ns/j2ee"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"*>

<display-name>SSH Integration</display-name>

<!-- Spring Context Configuration-->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/applicationContext.xml</param-value>

</context-param>

<!-- Config Spring Context Loader Listener -->

<listener>

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

</listener>

<!-- The Usual Welcome File List -->

<welcome-file-list>

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

</welcome-file-list>

</web-app>

## 建立applicationContext.xml

在/WEB-INF/spring路径下建立基本的applicationContext.xml（待填入数据）：

<?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: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-2.0.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.0.xsd*

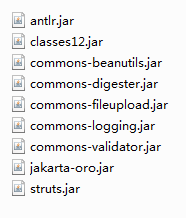
*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.0.xsd"*>

</beans>

# 整合Struts框架

## Struts所需lib库

以下为Struts的示例库，须看实际要求增减



## 修改Web.xml

增加ActionServlet和Struts Tag Library（并非必须）。

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

<web-app id=*"WebApp\_ID"* version=*"2.4"* xmlns=*"http://java.sun.com/xml/ns/j2ee"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"*>

<display-name>Struts Blank Application</display-name>

<!-- Spring Context Configuration-->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/applicationContext.xml</param-value>

</context-param>

<!-- Config Spring Context Loader Listener -->

<listener>

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

</listener>

<!-- Standard Action Servlet Configuration (with debugging) -->

<servlet>

<servlet-name>action</servlet-name>

<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>

<init-param>

<param-name>config</param-name>

<param-value>/WEB-INF/struts/struts-config.xml</param-value>

</init-param>

<init-param>

<param-name>debug</param-name>

<param-value>2</param-value>

</init-param>

<init-param>

<param-name>detail</param-name>

<param-value>2</param-value>

</init-param>

<load-on-startup>2</load-on-startup>

</servlet>

<!-- Standard Action Servlet Mapping -->

<servlet-mapping>

<servlet-name>action</servlet-name>

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

</servlet-mapping>

<!-- The Usual Welcome File List -->

<welcome-file-list>

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

</welcome-file-list>

<!-- Struts Tag Library Descriptors -->

<jsp-config>

<taglib>

<taglib-uri>/tags/struts-bean</taglib-uri>

<taglib-location>/WEB-INF/struts/struts-bean.tld</taglib-location>

</taglib>

<taglib>

<taglib-uri>/tags/struts-html</taglib-uri>

<taglib-location>/WEB-INF/struts/struts-html.tld</taglib-location>

</taglib>

<taglib>

<taglib-uri>/tags/struts-logic</taglib-uri>

<taglib-location>/WEB-INF/struts/struts-logic.tld</taglib-location>

</taglib>

<taglib>

<taglib-uri>/tags/struts-nested</taglib-uri>

<taglib-location>/WEB-INF/struts/struts-nested.tld</taglib-location>

</taglib>

<taglib>

<taglib-uri>/tags/struts-tiles</taglib-uri>

<taglib-location>/WEB-INF/struts/struts-tiles.tld</taglib-location>

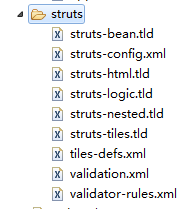
</taglib>

</jsp-config>

</web-app>

## 建立struts配置文件

在/WEB-INF/struts路径下建立一下配置文件：



### 建立struts-config.xml

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

<!DOCTYPE struts-config PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"

"http://jakarta.apache.org/struts/dtds/struts-config\_1\_2.dtd">

<struts-config>

<!-- ================================================ Form Bean Definitions -->

<form-beans>

<form-bean name=*"userRegistrationForm"* type=*"cn.hxex.tutorial.UserRegistrationForm"*/>

</form-beans>

<!-- ========================================= Global Exception Definitions -->

<global-exceptions>

</global-exceptions>

<!-- =========================================== Global Forward Definitions -->

<global-forwards>

<forward

name=*"welcome"*

path=*"/Welcome.do"*/>

</global-forwards>

<!-- =========================================== Action Mapping Definitions -->

<action-mappings>

<!-- Default "Welcome" action -->

<!-- Forwards to Welcome.jsp -->

<action

path=*"/Welcome"*

forward=*"/pages/Welcome.jsp"*/>

<action path=*"/userRegistration"*

name=*"userRegistrationForm"*

input=*"/userRegistration.jsp"*

type=*"cn.hxex.tutorial.UserRegistrationAction"*>

<set-property value=*"true"* property=*"cancellable"*/>

<forward name=*"success"* path=*"/regSuccess.jsp"*/>

</action>

</action-mappings>

<!-- ============================================= Controller Configuration -->

<controller processorClass=*"org.springframework.web.struts.AutowiringRequestProcessor"*/>

<!-- ======================================== Message Resources Definitions -->

<message-resources parameter=*"MessageResources"* />

<!-- =============================================== Plug Ins Configuration -->

<!-- ======================================================= Tiles plugin -->

<!-- =================================================== Validator plugin -->

<plug-in className=*"org.apache.struts.validator.ValidatorPlugIn"*>

<set-property

property=*"pathnames"*

value=*"/WEB-INF/struts/validator-rules.xml,/WEB-INF/struts/validation.xml"*/>

</plug-in>

</struts-config>

## 搭建Struts功能程序

### 建立UserRegistrationForm.java

**public** **class** UserRegistrationForm **extends** ActionForm {

/\*\*

\* serialVersionUID

\*/

**private** **static** **final** **long** ***serialVersionUID*** = -2856538722578293204L;

**private** String userName;

**private** String logName;

**private** String password;

**private** String passwordCheck;

**private** String email;

**private** String phone;

**private** String fax;

**public** String getEmail()

{

**return** email;

}

**public** **void** setEmail(String email)

{

**this**.email = email;

}

**public** String getFax()

{

**return** fax;

}

**public** **void** setFax(String fax)

{

**this**.fax = fax;

}

**public** String getLogName()

{

**return** logName;

}

**public** **void** setLogName(String logName)

{

**this**.logName = logName;

}

**public** String getPassword()

{

**return** password;

}

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

{

**this**.password = password;

}

**public** String getPasswordCheck()

{

**return** passwordCheck;

}

**public** **void** setPasswordCheck(String passwordCheck)

{

**this**.passwordCheck = passwordCheck;

}

**public** String getPhone()

{

**return** phone;

}

**public** **void** setPhone(String phone)

{

**this**.phone = phone;

}

**public** String getUserName()

{

**return** userName;

}

**public** **void** setUserName(String userName)

{

**this**.userName = userName;

}

**public** **void** reset(ActionMapping mapping,

HttpServletRequest request) {

userName=**null**;

logName=**null**;

password=**null**;

passwordCheck=**null**;

email=**null**;

phone=**null**;

fax=**null**;

}

**public** ActionErrors validate(ActionMapping mapping,

HttpServletRequest request) {

ActionErrors errors = **new** ActionErrors();

**if** (userName==**null**

|| userName.trim().equals("")){

errors.add("userName",

**new** ActionMessage(

"userRegistration.userName.problem"));

}

**return** errors;

}

}

### 建立UserRegistrationAction.java

**public** **class** UserRegistrationAction **extends** ActionSupport {

**public** ActionForward execute(ActionMapping mapping,

ActionForm form,

HttpServletRequest request,

HttpServletResponse response)

**throws** Exception {

**if** (isCancelled(request)) {

**return** mapping.findForward("welcome");

}

**return** mapping.findForward("success");

}

}

## 建立JSP页面

### 建立userRegistration.jsp页面

<%@ page contentType=*"text/html;charset=UTF-8"* language=*"java"* %>

<%@ taglib uri=*"/tags/struts-html"* prefix=*"html"*%>

<%@ taglib uri=*"/tags/struts-bean"* prefix=*"bean"*%>

<html>

<head>

<title><bean:message key=*"userRegistration.title"*/></title>

</head>

<body>

<h1><bean:message key=*"userRegistration.title"*/></h1>

<html:errors/>

<table>

<html:form action=*"userRegistration"*>

<tr>

<td>

<bean:message key=*"userRegistration.userName"* />\*

</td>

<td>

<html:text property=*"userName"* />

</td>

</tr>

<tr>

<td>

<bean:message key=*"userRegistration.logName"* />\*

</td>

<td>

<html:text property=*"logName"* />

</td>

</tr>

<tr>

<td>

<bean:message key=*"userRegistration.password"* />\*

</td>

<td>

<html:password property=*"password"* />

</td>

</tr>

<tr>

<td>

<bean:message key=*"userRegistration.password"* />\*

</td>

<td>

<html:password property=*"passwordCheck"* />

</td>

</tr>

<tr>

<td>

<bean:message key=*"userRegistration.email"* />\*

</td>

<td>

<html:text property=*"email"* />

</td>

</tr>

<tr>

<td>

<bean:message key=*"userRegistration.phone"* />

</td>

<td>

<html:text property=*"phone"* />

</td>

</tr>

<tr>

<td>

<bean:message key=*"userRegistration.fax"* />

</td>

<td>

<html:text property=*"fax"* />

</td>

</tr>

<tr>

<td>

<html:submit />

</td>

<td>

<html:cancel />

</td>

</tr>

</html:form>

</table>

</body>

</html>

### 建立regSuccess.jsp页面

<%@ page contentType=*"text/html;charset=UTF-8"* language=*"java"* %>

<%@ taglib uri=*"/tags/struts-bean"* prefix=*"bean"* %>

<html>

<head>

<title><bean:message key=*"registration.success"* /></title>

</head>

<body>

<h1><bean:message key=*"registration.success"* /></h1>

</body>

</html>

## 添加MessageResources属性文件

此处只列出MessageResources.properties的内容

registration.success=User Registration Was Successful!

userRegistration.title=User Registration

userRegistration.userName=User Name

userRegistration.logName=Logon Name

userRegistration.password=Password

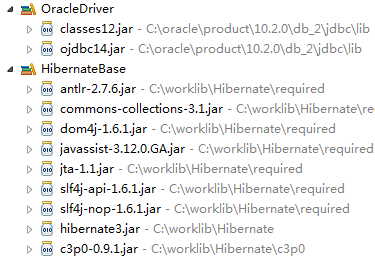
userRegistration.email=Email

userRegistration.phone=Phone

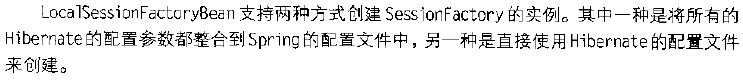
userRegistration.fax=Fax

# 整合Hibernate框架

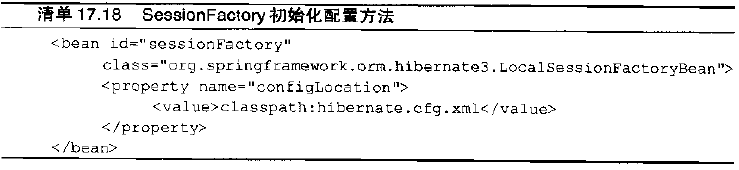
## Hibernate所需lib库



## 利用Spring的IoC容器创建SessionFactory



### 直接使用Hibernate的配置文件



### 在Spring的配置文件中配置Hibernate

<!-- 配置Hibernate的sessionFactory -->

<bean id=*"sessionFactory"*

class=*"org.springframework.orm.hibernate3.LocalSessionFactoryBean"*>

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

<property name=*"mappingLocations"*>

<list>

<value>

classpath\*:/cn/hxex/tutorial/vo/\*.hbm.xml

</value>

</list>

</property>

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.Oracle10gDialect</prop>

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

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

<prop key=*"hibernate.hbm2ddl.auto"*>create</prop>

<prop key=*"hibernate.jdbc.batch\_size"*>1000</prop>

<!-- 指定连接池里最大缓存多少个Statement对象 -->

<prop key=*"hibernate.c3p0.max\_statements"*>100</prop>

<prop key=*"hibernate.c3p0.idle\_test\_period"*>3000</prop>

<prop key=*"hibernate.c3p0.acquire\_increment"*>2</prop>

<prop key=*"hibernate.c3p0.validate"*>true</prop>

<!-- 指定连接池里最大连接数 -->

<prop key=*"hibernate.c3p0.max\_size"*>20</prop>

<!-- 指定连接池里最小连接数 -->

<prop key=*"hibernate.c3p0.min\_size"*>1</prop>

<!-- 指定连接池里连接的超时时长 -->

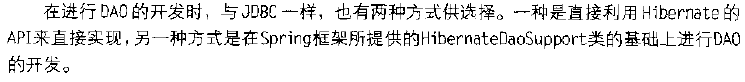
<prop key=*"hibernate.c3p0.timeout"*>5000</prop>

</props>

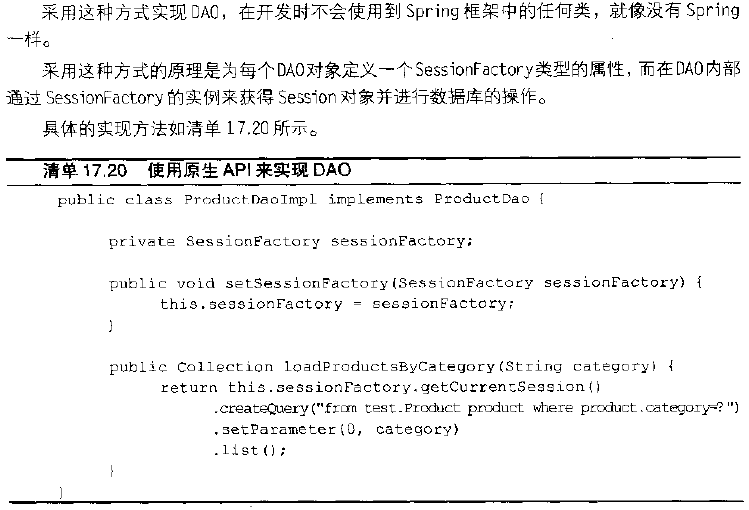
</property>

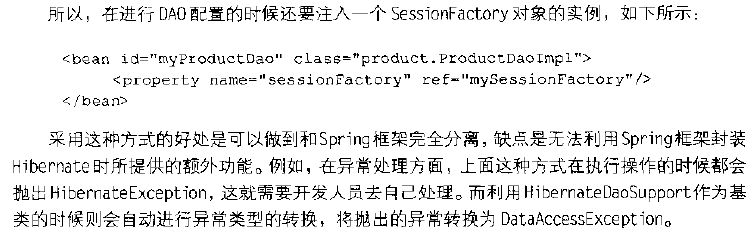
</bean>

## DAO的开发

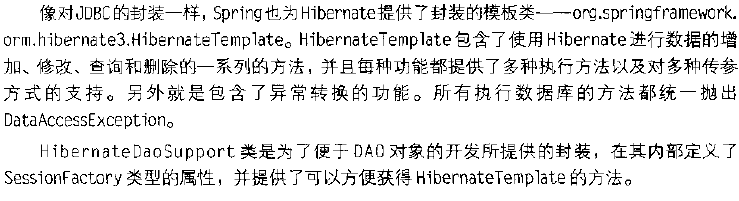


### 使用原生API实现DAO





### 在HibernateDaoSupport的基础上开发DAO



**BaseDAO.java**

**public** **class** BaseDAO **extends** HibernateDaoSupport {

**protected** Serializable save(Object entity) **throws** DataAccessException {

**return** getHibernateTemplate().save(entity);

}

}

**UserDAO.java**

**public** **class** UserDAO **extends** BaseDAO {

**public** **void** createUser(UserInfo user) {

**this**.save(user);

}

}

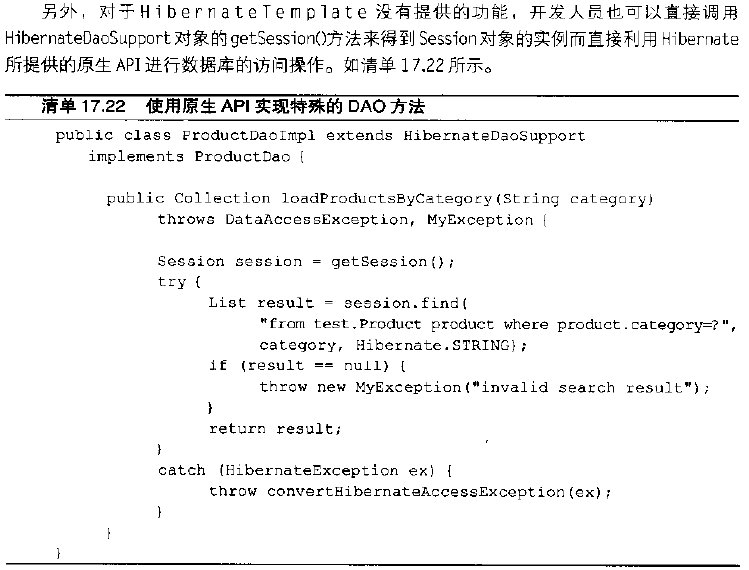
**在applicationContext.xml中配置bean：**

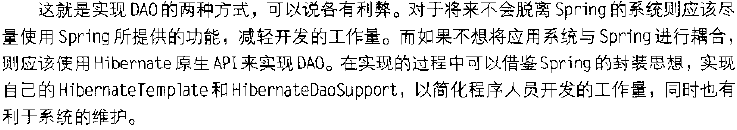
<bean id=*"baseDAO"* class=*"cn.hxex.tutorial.dao.BaseDAO"* abstract=*"true"*>

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

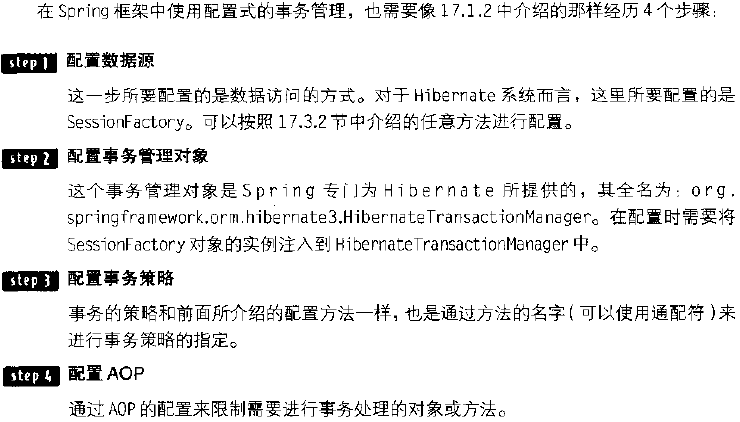
</bean>

<bean id=*"userDAO"* class=*"cn.hxex.tutorial.dao.UserDAO"* parent=*"baseDAO"*/>





## Hibernate的事务管理



<!-- 配置数据源 -->

<bean id=*"dataSource"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name=*"driverClassName"*><value>oracle.jdbc.driver.OracleDriver</value></property>

<property name=*"url"*>

<value>jdbc:oracle:thin:@localhost:1521:LEON</value>

</property>

<property name=*"username"*><value>Leon</value></property>

<property name=*"password"*><value>Leonhard8</value></property>

</bean>

<!-- 配置TransactionManager -->

<bean id=*"transactionManager"*

class=*"org.springframework.orm.hibernate3.HibernateTransactionManager"*>

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

</bean>

<!-- 配置事务策略 -->

<tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<tx:method name=*"create\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>

</tx:attributes>

</tx:advice>

<!-- 配置AOP -->

<aop:config>

<aop:pointcut id=*"userServicePointCut"*

expression=*"execution(\* cn.hxex.tutorial.bo.\*Service.\*(..))"*/>

<aop:advisor pointcut-ref=*"userServicePointCut"* advice-ref=*"txAdvice"*/>

</aop:config>

## VO与BO的开发

### VO的开发

**UserInfo.java**

**public** **class** UserInfo {

**int** id;

String email;

String userName;

String logName;

String password;

String phone;

String fax;

**public** **int** getId() {

**return** id;

}

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

**this**.id = id;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getUserName() {

**return** userName;

}

**public** **void** setUserName(String userName) {

**this**.userName = userName;

}

**public** String getLogName() {

**return** logName;

}

**public** **void** setLogName(String logName) {

**this**.logName = logName;

}

**public** String getPassword() {

**return** password;

}

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

**this**.password = password;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

**public** String getFax() {

**return** fax;

}

**public** **void** setFax(String fax) {

**this**.fax = fax;

}

}

**UserInfo.hbm.xml**

<?xml version=*"1.0"*?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

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

<hibernate-mapping package=*"cn.hxex.tutorial.vo"*>

<class name=*"UserInfo"* table=*"USERINFO"*>

<id name=*"id"* column=*"id"*>

<generator class=*"increment"*/>

</id>

<property name=*"email"* column=*"email"* type=*"string"* length=*"80"*/>

<property name=*"userName"* column=*"user\_name"* type=*"string"* length=*"80"*/>

<property name=*"logName"* column=*"log\_name"* type=*"string"* length=*"80"*/>

<property name=*"password"* column=*"password"* type=*"string"* length=*"11"*/>

<property name=*"phone"* column=*"phone"* type=*"string"* length=*"11"*/>

<property name=*"fax"* column=*"fax"* type=*"string"* length=*"11"*/>

</class>

</hibernate-mapping>

### BO的开发

**UserService.java**

**public** **class** UserService {

**private** UserDAO userDAO;

**public** **void** createClient(UserInfo user) {

userDAO.createUser(user);

}

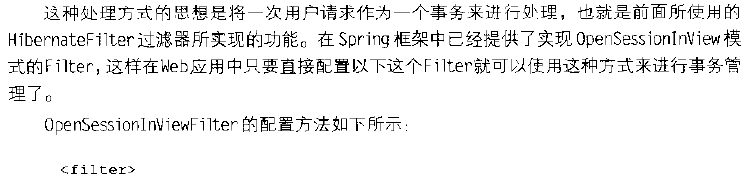
**public** **void** setUserDAO(UserDAO userDAO) {

**this**.userDAO = userDAO;

}

}

## OpenSessionInView模式





什么是OpenSessionInView

在hibernate中使用load方法时，并未把数据真正获取时就关闭了session，当我们真正想获取数据时会迫使load加载数据，而此时session已关闭，所以就会出现异常。 比较典型的是在MVC模式中，我们在M层调用持久层获取数据时(持久层用的是load方法加载数据)，当这一调用结束时，session随之关闭，而我们希望在V层使用这些数据，这时才会迫使load加载数据，我们就希望这时的session是open着得，这就是所谓的Open Session In view 。 我们可以filter来达到此目的。

OpenSessionInViewFilter是Spring提供的一个针对Hibernate的一个支持类，其主要意思是在发起一个页面请求时打开Hibernate的Session，一直保持这个Session，直到这个请求结束，具体是通过一个Filter来实现的。

由于Hibernate引入了Lazy Load特性，使得脱离Hibernate的Session周期的对象如果再想通过getter方法取到其关联对象的值，Hibernate会抛出一个LazyLoad的Exception。所以为了解决这个问题，Spring引入了这个Filter，使得Hibernate的Session的生命周期变长。

首先分析一下它的源码，可以发现，它所实现的功能其实比较简单：

Java代码

SessionFactory sessionFactory = lookupSessionFactory(request);

Session session = null;

boolean participate = false;

if (isSingleSession()) {

// single session mode

if (TransactionSynchronizationManager.hasResource(sessionFactory)) {

// Do not modify the Session: just set the participate flag.

participate = true;

} else {

logger.debug("Opening single Hibernate Session in OpenSessionInViewFilter");

session = getSession(sessionFactory);

TransactionSynchronizationManager.bindResource(sessionFactory, new SessionHolder(session));

}

} else {

// deferred close mode

if (SessionFactoryUtils.isDeferredCloseActive(sessionFactory)) {

// Do not modify deferred close: just set the participate flag.

participate = true;

} else {

SessionFactoryUtils.initDeferredClose(sessionFactory);

}

}

try {

filterChain.doFilter(request, response);

} finally {

if (!participate) {

if (isSingleSession()) {

// single session mode

TransactionSynchronizationManager.unbindResource(sessionFactory);

logger.debug("Closing single Hibernate Session in OpenSessionInViewFilter");

closeSession(session, sessionFactory);

}else {

// deferred close mode

SessionFactoryUtils.processDeferredClose(sessionFactory);

}

}

}

在上述代码中，首先获得SessionFactory，然后通过SessionFactory获得一个Session。然后执行真正的Action代码，最后根据情况将Hibernate的Session进行关闭。整个思路比较清晰。

注意，在这里有个2个Tips：

1）通过getSession()获得的这个Session做了一次

session.setFlushMode(FlushMode.NEVER); 有关FlushMode可以参考一下这篇文章。http://www2.matrix.org.cn/resource/article/2006-10-08/Hibernate+FlushMode+NEVER\_312bca85-5699-11db-91a0-d98dff0aec60.html

2）Spring对拿到的Session做了一次绑定到当前线程的做法，使得这个Session是线程安全的。

从上述代码其实可以得到一些对我们的开发有帮助的结论：

1）如果使用了OpenSessionInView模式，那么Spring会帮助你管理Session的开和关，从而你在你的DAO中通过HibernateDaoSupport拿到的getSession()方法，都是绑定到当前线程的线程安全的Session，即拿即用，最后会由Filter统一关闭。

2）由于拿到的Hibernate的Session被设置了session.setFlushMode(FlushMode.NEVER); 所以，除非你直接调用session.flush()，否则Hibernate session无论何时也不会flush任何的状态变化到数据库。因此，数据库事务的配置非常重要。（我们知道，在调用org.hibernate.Transaction.commit()的时候会触发session.flush()）我曾经见过很多人在使用OpenSessionInView模式时，都因为没有正确配置事务，导致了底层会抛出有关FlushMode.NEVER的异常。

OpenSessionInView这个模式使用比较简单，也成为了大家在Web开发中经常使用的方法，不过它有时候会带来一些意想不到的问题，这也是在开发中需要注意的。

1. 在Struts＋Spring＋Hibernate环境中，由于配置的问题导致的模式失效

这个问题以前论坛曾经讨论过，可以参考一下下面这个帖子：

http://www.iteye.com/topic/15057

2. OpenSessionInView的效率问题

这个问题也有人在论坛提出过，Robbin曾经做过具体的测试，可以具体参考一下下面这个帖子：

http://www.iteye.com/topic/17501

3. 由于使用了OpenSessionInView模式后造成了内存和数据库连接问题

这个问题是我在生产环境中碰到的一个问题。由于使用了OpenSessionInView模式，Session的生命周期变得非常长。虽然解决了Lazy Load的问题，但是带来的问题就是Hibernate的一级缓存，也就是Session级别的缓存的生命周期会变得非常长，那么如果你在你的Service层做大批量的数据操作时，其实这些数据会在缓存中保留一份，这是非常耗费内存的。还有一个数据库连接的问题，存在的原因在于由于数据库的Connection是和Session绑在一起的，所以，Connection也会得不到及时的释放。因而当系统出现业务非常繁忙，而计算量又非常大的时候，往往数据连接池的连接数会不够。这个问题我至今非常头痛，因为有很多客户对数据连接池的数量会有限制，不会给你无限制的增加下去。

4. 使用了OpenSessionInView模式以后取数据的事务问题

在使用了OpenSessionInView以后，其实事务的生命周期比Session的生命周期来得短，就以为着，其实有相当一部分的查询是不被纳入到事务的范围内的，此时是否会读到脏数据？这个问题我至今不敢确认，有经验的朋友请指教一下。

最后提一下OpenSessionInView模式的一些替代方案，可以使用OpenSessionInViewInterceptor来代替这个Filter，此时可以使用Spring的AOP配置，将这个Interceptor配置到你所需要的层次上去。另外就是只能使用最古老的Hibernate.initialize()方法进行初始化了。

# 附录

## 使用AutowiringRequestProcessor整合struts和Spring2.0

下面着重谈一下Spring2.0新增的一个整个方式，我感觉挺不错，可以完全将Struts的配置和Spring的配置分离。具体步骤如下：   
**1. 编写Spring的配置文件applicationContext.xml**，简单起见，仅仅定义一个Service对象。

引用

<?xml version="1.0" encoding="UTF-8"?>   
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">   
  
<beans>   
  
<bean id="userService" class="com.bearingpoint.gdc.zero.service.impl.UserServiceImpl" />   
  
</beans>

这看上去就和普通的Spring配置文件没有任何分别。   
**2. 编写Struts的配置文件struts-config.xml，注意其中的controller的配置，用到了Spring2.0的新特性。**

引用

<?xml version="1.0" encoding="UTF-8"?>   
<!DOCTYPE struts-config   
PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"   
"http://jakarta.apache.org/struts/dtds/struts-config\_1\_2.dtd"   
>   
  
<struts-config>   
  
<action-mappings>   
  
<action path="/addUser"   
type="com.bearingpoint.gdc.zero.action.user.AddUser"   
scope="request"   
>   
<forward name="success" path="/index.jsp" />   
</action>   
  
</action-mappings>   
  
**<controller processorClass="org.springframework.web.struts.AutowiringRequestProcessor" />**   
  
</struts-config>

3. 然后为你的Struts的Action注入你需要的Service

引用

private UserService userService;   
  
public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws Exception {   
 User user = new User();   
 userService.addUser(user);   
 return mapping.findForward("success");   
}   
  
/\*\*   
\* @param userService   
\*            The userService to set.   
\*/   
public void setUserService(UserService userService) {   
 this.userService = userService;   
}

看上去你好像啥都没做，而事实上，注入工作已经由AutowiringRequestProcessor自动完成。   
  
4. 编写web.xml进行测试。

引用

?xml version="1.0" ?>   
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app\_2\_3.dtd">   
  
<web-app>   
  
    <context-param>   
        <param-name>contextConfigLocation</param-name>   
        <param-value>/WEB-INF/classes/applicationContext.xml</param-value>   
    </context-param>   
  
<listener>   
<listener-class>**org.springframework.web.context.ContextLoaderListener**</listener-class>   
</listener>   
  
<servlet>   
<servlet-name>struts</servlet-name>   
<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>   
<init-param>   
<param-name>config</param-name>   
<param-value>/WEB-INF/classes/struts-config.xml</param-value>   
</init-param>   
<init-param>   
<param-name>detail</param-name>   
<param-value>2</param-value>   
</init-param>   
<init-param>   
<param-name>validate</param-name>   
<param-value>true</param-value>   
</init-param>   
<load-on-startup>2</load-on-startup>   
</servlet>   
  
<servlet-mapping>   
<servlet-name>struts</servlet-name>   
<url-pattern>\*.do</url-pattern>   
</servlet-mapping>   
  
    <welcome-file-list>   
        <welcome-file>index.jsp</welcome-file>   
    </welcome-file-list>   
</web-app>

最后，启动Jetty进行测试，顺利运行通过！   
  
看上去如此简单，配置起来也没有什么很特别的地方，只是按照常规来写你的Spring和Struts的配置文件就好了。   
  
不过在这里还是说一下其中的要注意两个问题：   
1. 这种autowire的注入支持两种不同的方式，分别是byName和byType，默认是byType。我想这对于绝大多数开发者来说是够了。   
2. 鉴于在http://www.javaeye.com/topic/15057中所提到的OpenSessionInView模式的失效的问题。我仔细看了一下Spring的源码。对于这种autowire的整合方式，不推荐在struts-config.xml文件中配置ContextLoaderPlugIn，而是采用web.xml中的ContextLoaderListener来加载Spring的初始化配置。否则，你的OpenSessionInView模式可能会失效。