课程名称：JavaEE 技术	年级：xxxx	实验日期：
姓名：　xxx	学号：xxxxx	班级：xxxx
实验名称：Struts、Hibernate 与 Spring 整合编程	实验序号：实验七	成员人数：1

一、 实验目的及要求

实验目的：

1、熟悉 Hibernate 的核心配置文件、映射配置文件；
2、熟悉 web.xml 中配置 Struts 过滤器、值栈、拦截器
3、Spring 框架核心配置文件、AOP 的思想以及 JdbcTemplate 的使用

实验原理：

1、struts2 和 Spring 进行整合
2、hibernate 和 Spring 进行整合

二、实验环境(仪器与材料)

Windows 7/10，MySQL/MariaDB，Eclipse,Tomcat

三、实验内容及完成情况

1、新建 Dynamic Web Project 项目 ssh

注意:整合时，如果使用的是 struts2.5，则使用 2.5 版本的整合包；用 struts2.3，则使用 2.3 整合包

2、创建 com.action 包，创建 UserAction 类

3、导入四个 xml catalog ： spring-aop.xsd　spring-beans.xsd　spring-context.xsd　spring-tx.xsd，如图 1 所示。

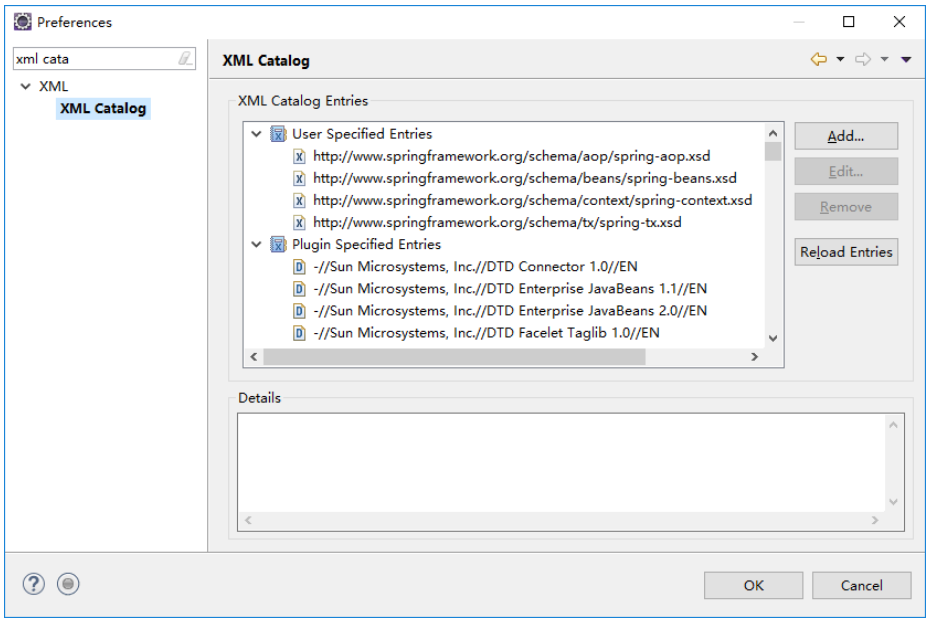

图 1 导入四个 xml catalog

4、配置 Spring　applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
</beans>
```

5、将 applicationContext.xml 切换到 design 视图，右键增加命名空间，如图 2 所示。

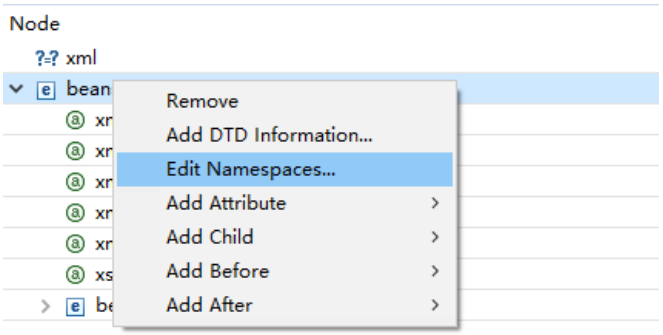

图 2 增加命名空间

6、完善 applicationContext.xml

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

7、编辑 web.xml

1、设置监听器 spring 随着 web 的启动创建　　2、设置读取 Spring 配置文件位置

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
  <display-name>ssh</display-name>
  <!-- 设置监听器 spring 随着 web 的启动创建 -->
  <listener>
   <listener-class>org.springframework.web.context.ContextLoaderListener
   </listener-class>
```

```
    </listener>
    <!-- Spring 配置文件位置 -->
    <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml</param-value>
    </context-param>
```

8、将项目部署到服务器上，启动服务器，查看启动有无异常

9、配置 Struts2 struts.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
"http://struts.apache.org/dtds/struts-2.3.dtd">
<struts>
    <package name="struts2" namespace="/" extends="struts-default">
      <action name="userAction*" class="com.action.UserAction" method="{1}">
        <result name="success">/success.jsp</result>
      </action>
    </package>
</struts>
```

10、完善 web.xml，增加 Struts2 核心过滤器

```
<!-- 配置 struts2 核心过滤器 -->
    <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>
```

11、整合 Spring 与 Struts2

配置常量，在 org.apache.struts2→static→找到 default.properties，如图 3 所示，双击打开，如图 4 所示。

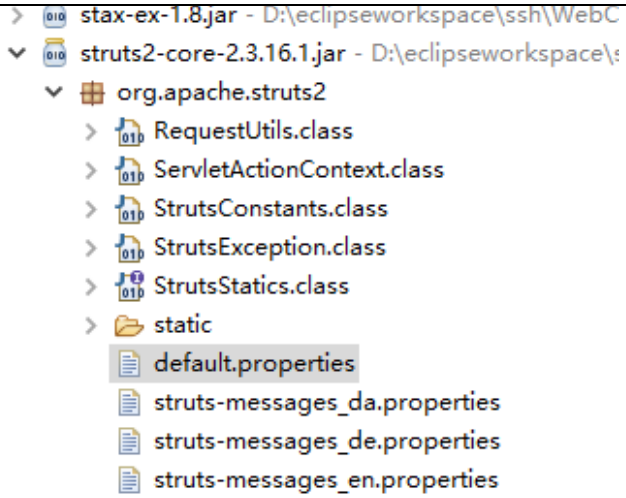

图 3 default.properties

```
# struts.objectFactory = spring

### specifies the autoWiring logic when using the SpringObjectFactory.
### valid values are: name, type, auto, and constructor (name is the default)
struts.objectFactory.spring.autoWire = name
```

```
<struts>
<!-- 将struts2中的action交给Spring创建 -->
<constant name="struts.objectFactory" value="spring"></constant>
<!-- 默认配置是打开的，可以配置，可以不配置，Spring自动装配action的依赖 -->
<constant name="struts.objectFactory.spring.autoWire" value="name"></constant>
    <package name="struts2" namespace="/" extends="struts-default">
    <!-- 整合方案1，class保持完整类名和路径 -->
        <action name="userAction_*" class="com.action.UserAction" method="{1}">
            <result name="success">/success.jsp</result>
        </action>

    </package>
```

图 4 配置信息

12、整合 Spring 和 Struts2 框架

(1)创建 com.model 包，创建 User.java 类

```
package com.model;
   public class User {
   private Integer id;
   private String username;
   private String password;
   private Integer grade;
   private String email;
      Set   get 设置省略
}
```

(2)创建 com.service、com.service.impl 包，创建 UserService 接口、UserServiceImpl 实现类

```
package com.service;

import com.model.User;

public interface UserService {

    public User getUser(User user);
}
```

```
package com.service.impl;

import com.model.User;

public class UserServiceImpl implements UserService {

    @Override
    public User getUser(User user) {
        //属于测试阶段，在控制台中打印一句话
        System.out.println("UserServiceImpl:"+user.getUsername());
        return null;
    }

}
```

(3)在 UserAction 中，增加 userService 属性，使用 set 方法。

```
package com.action;

import com.opensymphony.xwork2.ActionSupport;
import com.service.UserService;

public class UserAction extends ActionSupport {

    private UserService userService;

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

}
```

(4)整合方案：在 struts.xml 中的 action　name 属性上，无需写上完整类名，只需把 applicationContext.xml bean 中的 name 写到 struts.xml 中即可。

注意事项：需要手动组装 action 依赖属性，还需要设置 action 作用范围为：prototype（多例）

```
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd ">
     <bean name="userAction" class="com.action.UserAction" scope="prototype">
        <property name="userService" ref="userService"></property>
    </bean>
    <bean name="userService" class="com.service.impl.UserServiceImpl"></bean>
</beans>
```

13、配置 Hibernate 框架

(1)导入 jar 包（已经完成）

(2)书写 hibernate 主配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
 <!-- 数据库驱动 -->
```

```
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
     <!-- 数据库 url -->
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/usermanager</property>
     <!-- 数据库连接用户名 -->
    <property name="hibernate.connection.username">root</property>
     <!-- 数据库连接密码 -->
    <property name="hibernate.connection.password">root</property>
    <!-- 数据库方言 -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <!-- 将 hibernate 生成的 sql 语句打印到控制台 -->
    <property name="hibernate.show_sql">true</property>
    <!-- 将 hibernate 生成的 sql 语句格式化(语法缩进) -->
    <property name="hibernate.format_sql">true</property>
    <!-- 自动建表 -->
    <property name="hibernate.hbm2ddl.auto">update</property>
    <!-- 引入实体配置文件 -->
    <mapping resource="com/model/user.hbm.xml" />
    </session-factory>
    </hibernate-configuration>
(3)书写 user.hbm.xml 配置文件
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
    <!-- 配置表与实体对象的关系 -->
    <!-- package 属性:包名性,可以直接写类名 -->
    <hibernate-mapping package="com.model" >
    <class name="User" table="tuserlogin" >
    <id name="id" column="id" type="java.lang.Integer">
       <generator class="native"></generator>
    </id>
    <property name="username" column="username" type="java.lang.String">
    </property>
    <property name="password" column="password" type="java.lang.String"></property>
    <property name="grade" column="grade" type="java.lang.Integer"></property>
    <property name="email" column="email" type="java.lang.String"></property>
    </class>
</hibernate-mapping>
(4)编写 hibernate 测试类
public class HibernateTest {
@Test
public void fun() {
Configuration cfg = new Configuration().configure();
SessionFactory sf = cfg.buildSessionFactory();
Session session = sf.openSession();
Transaction tx = session.beginTransaction();
//----------------------
User user = new User();
user.setUsername("test004");
user.setPassword("111");
```

```
user.setEmail("test@qq.com");
user.setGrade(4);
session.save(user);
//----------------------
tx.commit();
session.close();
sf.close();
}
}
```

(5)运行 junit 进行测试

14、整合 Spring 和 Hibernate

将 SessionFactory 对象交给 Spring 容器管理

在 Spring 中配置 Hibernate 信息 ，无需引入 SessionFactory 配置

```
<!-- spring整合hibernate方案2:在Spring中配置Hibernate信息 -->
<bean name="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
    <property name="hibernateProperties">
        <props>
            <!--  必选配置 -->
            <prop key="hibernate.connection.driver_class" >com.mysql.jdbc.Driver</prop>
            <prop key="hibernate.connection.url" >jdbc:mysql://localhost:3306/usermanager</prop>
            <prop key="hibernate.connection.username" >root</prop>
            <prop key="hibernate.connection.password" >root</prop>
            <prop key="hibernate.dialect" >org.hibernate.dialect.MySQLDialect</prop>
            <!--  可选配置 -->
            <prop key="hibernate.show_sql" >true</prop>
            <prop key="hibernate.format_sql" >true</prop>
            <prop key="hibernate.hbm2ddl.auto" >update</prop>
        </props>
    </property>
    <!-- 引入orm元数据,指定orm元数据所在的包路径 -->
    <property name="mappingDirectoryLocations" value="classpath:com/model" ></property>
</bean>
```

15、配置 Spring c3p0 连接池

(1)在 src 中加入 db.properties 配置文件

(2)在 applicationContext.xml 中，加入

```
<!-- 读取 db.properties 文件 -->
<context:property-placeholder location="classpath:db.properties" />
<!-- 配置 c3p0 连接池 -->
<bean name="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
<property name="jdbcUrl" value="${jdbc.jdbcUrl}" ></property>
<property name="driverClass" value="${jdbc.driverClass}" ></property>
<property name="user" value="${jdbc.user}" ></property>
<property name="password" value="${jdbc.password}" ></property>
</bean>
```

```
1 jdbc.jdbcUrl=jdbc:mysql://localhost:3306/usermanager
2 jdbc.driverClass=com.mysql.jdbc.Driver
3 jdbc.user=root
4 jdbc.password=root
```

(3)在 applicationContext.xml 中，将连接池注入到<bean name=“sessionFactory“>中，同时注释掉四行配置信息

```
<bean name="sessionFactory"
```

```
class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
<!-- 将连接池注入到 sessionFactory, hibernate 会通过连接池获得连接 -->
<property name="dataSource" ref="dataSource" ></property>
<property name="hibernateProperties">
<props>      <!--  必选配置 -->
<!—  以下四行注释掉，因为连接数据的选项，在连接池中已经配置好
<prop key="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prop>
<prop key="hibernate.connection.url" >jdbc:mysql://localhost:3306/usermanager</prop>
<prop key="hibernate.connection.username" >root</prop>
<prop key="hibernate.connection.password" >root</prop>
-->
<prop key="hibernate.dialect" >org.hibernate.dialect.MySQLDialect</prop>
<!--  可选配置 -->
<prop key="hibernate.show_sql" >true</prop>
<prop key="hibernate.format_sql" >true</prop>
<prop key="hibernate.hbm2ddl.auto" >update</prop>
</props>
</property>
```

16、配置 HibernateTemplate 操纵 dao

(1)增加 com.dao 包，增加接口 UserDao

```
package com.dao;
import com.model.User;
public interface UserDao {
        public User getUserById(Integer id);
}
```

(2)增加 com.dao.impl 包，增加接口的实现类 UserDaoImpl

```
public class UserDaoImpl extends HibernateDaoSupport implements UserDao {
@Override
public User getUserById(Integer id) {
//等同于 HibernateTemplate ht
      User user = getHibernateTemplate().get(User.class,1);
      System.out.println(user.getUsername());
      return user;
      }
}
```

(3)修改 applicationContext.xml，增加 bean，如下：

```
<!-- 配置 dao -->
      <bean id="userDao" class="com.dao.impl.UserDaoImpl">
         <property name="sessionFactory" ref="sessionFactory"></property>
      </bean>
```

17、测试类中增加 fun2()方法，进行测试

```
@Resource(name="userDao")
private UserDao userDao;
@Test
public void fun2() {
        userDao.getUserById(31);
}
```

18、在 Spring 中配置事务，使用注解方式

(1)配置 applicationContex.xml，增加核心事务管理器配置，增加注解事务配置

```
<!-- 配置核心事务管理器-->
<bean name="transactionManager"
class="org.springframework.orm.hibernate5.HibernateTransactionManager">
<property name="sessionFactory" ref="sessionFactory"></property>
</bean>
<!-- 注解事务 -->
<tx:annotation-driven transaction-manager="transactionManager"/>
```

(2)在 applicationContext.xml 中增加 userService bean 配置，并且在 userService 中注入 userDao

```
<!-- 配置 service -->
<bean name="userService" class="com.service.impl.UserServiceImpl">
<property name="userDao" ref="userDao"></property>
</bean>
<!-- 配置 dao -->
    <bean id="userDao" class="com.dao.impl.UserDaoImpl">
      <property name="sessionFactory" ref="sessionFactory"></property>
    </bean>
```

19、在 UserDao 中增加 saveUser 方法，在 UserDaoImpl 中实现该方法。

UserDao:

```
public interface UserDao {
        public User getUserById(Integer id);
        public void saveUser(User user);
}
```

UserDaoImpl:

```
public class UserDaoImpl extends HibernateDaoSupport implements UserDao {
@Override
public void saveUser(User user) {
        getHibernateTemplate().save(user);
}
```

20、在 UserService 接口中增加 saveUser 方法，在 UserServiceImpl 类中增加 saveUser 方法实现

```
public interface UserService {
public User getUser(User user);
public void saveUser(User user);
}
```

21、在 UserServiceImpl 类中增加 saveUser 方法实现代码，同时加上@Transactional 注解

```
@Transactional(isolation=Isolation.REPEATABLE_READ,propagation=Propagation.REQUIRED,rea
dOnly=true) //写在类上的注解，对整个类生效，readOnly=true 表明事务只能处理只读
public class UserServiceImpl implements UserService {
        private UserDao userDao;
        public void setUserDao(UserDao userDao) {
                this.userDao = userDao;
        }
@Override
```

```
@Transactional(isolation=Isolation.REPEATABLE_READ,propagation=Propagation.REQUIRED,readOnly=false) //saveUser 方法是保存对象，所以需要在方法上配置 readOnly=false
        public void saveUser(User user) {
        userDao.saveUser(user);
        }
}
```

22、在 hibernate 测试类中增加对事务进行测试代码

```
   @RunWith(SpringJUnit4ClassRunner.class)
   @ContextConfiguration("classpath:applicationContext.xml")
   public class Spring_Hibernate_Test{
   //测试 aop 事务
   @Resource(name="userService")
   private UserService userService;
   @Test
   public void fun_tx() {
      User user = new User();
      user.setUsername("tx");
      user.setPassword("tx");
      user.setEmail("tx@qq.org");
      user.setGrade(111);
      userService.saveUser(user);
   }
}
```

23、运行 junit 进行测试

24、增加 Session 作用范围

```
<!-- 配置 open   filter,扩大 Session 范围 -->
   <filter>
      <filter-name>openSessionInView</filter-name>
      <filter-class>org.springframework.orm.hibernate5.support.
                        OpenSessionInViewFilter
      </filter-class>
   </filter>
   <filter-mapping>
      <filter-name>openSessionInView</filter-name>
      <url-pattern>/*</url-pattern>
   </filter-mapping>
```

25、以登录为例，代码整合 SSH 框架

(1)首先编写 login.jsp 页面

```
<div class="col-md-3 col-md-offset-3" id="loginbody">
<!-- 居中对齐 -->
<p class="text-center" style="font-size:30px">登录</p>
<form action="userAction_login" method="post">
<label for="exampleInputEmail1">用户名</label>
      <input type="text" class="form-control" name="username" placeholder="用户名">
```

```
    <label for="exampleInputEmail1">密码</label>
<input type="password" class="form-control" name="password" placeholder="密码">
<br/><br/>
<button type="submit" class="btn btn-success">登录</button>
<button type="button" class="btn btn-info">重置</button>
</form>
</div>
```

(2)在 UserDao 中增加方法

```
public User getUserByNameAndPassword(User user);
```

(3)在 UserDaoImpl 中实现该方法

```
@Override
//Criteria
public User getUserByNameAndPassword(User user) {
   DetachedCriteria dc = DetachedCriteria.forClass(User.class);
   dc.add(Restrictions.eq("username", user.getUsername()));
   dc.add(Restrictions.eq("password", user.getPassword()));
   List<User> list = (List<User>) getHibernateTemplate().findByCriteria(dc);
   if(list != null && list.size()>0){
      return list.get(0);
   }else{
      return null;
   }
}
```

(4)在 UserService 中增加方法：

```
public User getUserByNameAndPassword(User user);
```

(5)在 UserServiceImpl 中增加该方法的实现：

```
@Override
public User getUserByNameAndPassword(User user) {
         return userDao.getUserByNameAndPassword(user);
}
```

(6)在 UserAction 中增加代码：

```
public String login() {
   User tmp_user = userService.getUserByNameAndPassword(user);
   if(null!=tmp_user) {
      ActionContext ac = ActionContext.getContext();
      ac.getSession().put("tmp_user", tmp_user);
      return SUCCESS;
      }else {
         return INPUT;
      }
```

}

(7)修改 struts.xml 配置文件，增加代码：

```
<action name="userAction_*" class="userAction" method="{1}">
    <result name="success">/success.jsp</result>
    <result name="input">/login.html</result>
</action>
```

26、测试

在 url 地址栏输入：　http://localhost:8080/ssh/login.html

四、 出现的问题及解决方案

1、