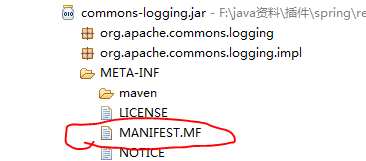
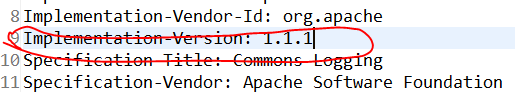
SSH三大框架整合

1. 导包：三大框架中包

把两个较为低版本的包remove，也可以直接从struts1从2008就未更新就可知，struts版本较为低

如何看包中的版本号：

1. 复制配置文件

Spring applicationContext.xml 其中包含hibernate中配置文件，这是spring管理

Struts struts\_config.xml

<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-3.0.xsd*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop-3.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd"*>

<context:component-scan base-package=*"com"*></context:component-scan>

<!-- 主键扫描 -->

<aop:aspectj-autoproxy></aop:aspectj-autoproxy>

<context:annotation-config/>

<bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"* destroy-method=*"close"*>

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

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/test"* />

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

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

</bean>

<bean id=*"mySessionFactory"* class=*"org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean"*>

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

<property name=*"packagesToScan"* value=*"com.ssh.po"*>

</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>

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

<prop key=*"hibernate.current\_session\_context\_class"*>thread</prop>

</props>

</property>

</bean>

<bean name=*"hibernateTemplate"* class=*"org.springframework.orm.hibernate3.HibernateTemplate"*>

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

</bean>

</beans>

**实验：**

1. **建log，user表，实体类**

@Entity

@Table(name="t\_log")

**public** **class** Log **implements** Serializable{

**private** **int** id;

**private** String msg;

@Id

@GeneratedValue

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

**return** id;

}

@Entity

@Table(name="t\_user2")

**public** **class** User **implements** Serializable{

**private** **int** id;

**private** String username;

**private** String password;

@Id

@GeneratedValue

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

**return** id;

}  
忽略get与set的方法 ………………

**2、普通的dao和继承DaoSuppor 的实现daoImpl类**

@Component("userDao")

**public** **class** UserDaoImpl **extends** DaoSupport **implements** UserDao {

**public** **void** addUser(User user) {

**super**.save(user);

}

}

**3.模板类实现的DaoSupport**

**public** **abstract** **class** DaoSupport **extends** HibernateTemplate {

/\*\*

\* - 【注入 SessionFactory】

\* - 注入一个sessionFactory属性,并注入到父类(HibernateDaoSupport )里

\* \*\*/

**private** SessionFactory sessionFacotry;

//注入一个bean, 默认(name = "sessionFactory"), 因此只写@Resource

@Resource

**public** **void** setSessionFacotry(SessionFactory sessionFacotry) {

**super**.setSessionFactory(sessionFacotry);

}

}

**注意：不用模板形式：**

@Component("logDao")

**public** **class** LogDao4Impl **implements** LogDao {

**private** SessionFactory sessionFactory;

**private** Session session;

@Resource(name="mySessionFactory")

**public** **void** setSessionFactory(SessionFactory sessionFactory) {

**this**.sessionFactory = sessionFactory;

session = sessionFactory.openSession();

}

**public** **void** saveLog(Log log) {

Transaction ts = session.beginTransaction();

**try** {

session.save(log);

ts.commit();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

ts.rollback();

}

}

}

**为什么需要DaoSupport：**

因为程序中需要SessionFactory，而由于spring提供的模板设计模式中的HibernateTemplate中没有注入sessionFactory，所以才需要DaoSupport

**4.只需要建User 中 service和serviceImpl**

@Service("userService")

**public** **class** UserServiceImpl **implements** UserService{

**private** UserDao userDao;

@Resource(name="userDao")

**public** **void** setUserDao(UserDao userDao) {

**this**.userDao = userDao;

}

**private** LogDao logDao;

@Resource(name="logDao")

**public** **void** setLogDao(LogDao logDao) {

**this**.logDao = logDao;

}

**public** **void** saveUser(User user) {

// **TODO** Auto-generated method stub

userDao.addUser(user);

Log log = **new** Log();

log.setMsg("save "+user.getUsername());

logDao.saveLog(log);

}

}

**5．测试类**

**public** **class** Test {

**public** **static** **void** main(String[] args) {

ApplicationContext ac = **new** ClassPathXmlApplicationContext("applicationContext.xml");

//SessionFactory sessionFactory = ac.getBean("mySessionFactory",SessionFactory.class);

UserService userService = ac.getBean("userService",UserService.**class**);

**try** {

User user = **new** User();

user.setUsername("11");

userService.saveUser(user);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

1.出现问题

当user或log中出错时，要求程序都不会进行下去这个要考虑**事务**

<prop key=*"hibernate.current\_session\_context\_class"*>thread</prop>

使其user与log方法是一个事务，但是其实还是两个事务，所以当user中出错时，log还是可以记录，但事实上，user都没有保存，何谈保存日志，所以我们必须要使用事务来解决问题。

**2.解决思路**

**提交commit除非在同一个方法中，不然，每一次调用方法都会提交，而造成数据的异常**

**但是由于service在MVC模式中是不能使用数据库中方法也就是说不能创建SessionFactroy，所以要想解决，还是等使用事务来解决**

**1.自动在dao加上事务**

@Transactional(propagation=Propagation.*REQUIRED*)

propagation=Propagation.*REQUIRED 如果这个方法上没有事务，会自动加上事务，如果有事务，就会合并事务*

@Transactional(propagation=Propagation.*REQUIRED*)

**public** **void** addUser(User user) {

Integer.*parseInt*("avc");

session.save(user);

}

1. service中事务

@Transactional(propagation=Propagation.*REQUIRED*)

**public** **void** saveUser(User user) {

userDao.addUser(user);

Log log = **new** Log();

log.setMsg("save "+user.getUsername());

System.*out*.println(log.getMsg());

logDao.saveLog(log);

}

合并事务

@Transactional(propagation=Propagation.*REQUIRED*)可以写在类上，也可以写在方法上，但是写在类上的是说明每一个方法都是一个事务，当不需要事务的时候，可以用

@Transactional(readOnly=**true**)

来标示该方法不需要写事务

查询方法不用写事务，效率低

**真正的三大框架整合**

1. **将以监听器的形式加载Spring**

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<context-param>

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

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

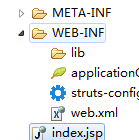
</context-param>

为什么，是因为启动服务的时候，启动spring中配置，得到spring支持，而不是在需要的时候才启动spring，效率低】

使得启动服务器时启动spring

org.springframework.web.context.ContextLoader initWebApplicationContext

启动服务器说明spring启动成功了



**Action**

**public** **class** UserAction **extends** DispatchAction {

**public** ActionForward saveUser(ActionMapping mapping, ActionForm form,

HttpServletRequest request, HttpServletResponse response) **throws** Exception {

ApplicationContext ac = WebApplicationContextUtils.*getWebApplicationContext*(request.getSession().getServletContext());

UserService userService = ac.getBean("userService",UserService.**class**);

User user = **new** User();

DynaActionForm daf = (DynaActionForm)form;

user.setUsername(daf.getString("username"));

user.setUsername(daf.getString("password"));

userService.saveUser(user);

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

}

}

**1）直接在Action中获取：**

**ApplicationContext contxt = new ClassPathXmlApplicationContext("applicationContext.xml");**

**userBO = (UserBO)contxt.getBean("UserBO");**

问题：如果有多个Action的话，会重复的读取applicationContext.xml文件，势必会造成效率的低下，而且最重要的是Spring是一个容器，多个Action会产生出来多个Spring容器，多个Spring容器如何去管理Struts和Hibernate，Struts和Hibernate又如何知道它们会被哪一个Spring容器所管理，这样势必会造成管理的混乱。对于Web项目工程来说，只需一个Spring容器就能管理Struts和 Hibernate了。因此，直接在Action中获取Spring容器是不可取的

**2）通过struts-config.xml来使Web容器加载Spring ----使用Spring容器插件的方式**

**<controller> ----使用控制器加载Spring，告知Struts将要被Spring所管理**

**<set-property property="processorClass"**

**value="org.springframework.web.struts.DelegatingRequestProcessor"/>**

**</controller>**

**<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn"> --插件方式**

**<set-property property="contextConfigLocation"**

**value="/WEB-INF/classes/applicationContext.xml" />**

**</plug-in>**

修改后Action：

@Controller("/user")

//@Scope("proto")

**public** **class** UserAction **extends** DispatchAction {

**private** UserService userService;

@Resource(name="userService")

**public** **void** setUserService(UserService userService) {

**this**.userService = userService;

}

**public** ActionForward saveUser(ActionMapping mapping, ActionForm form,

HttpServletRequest request, HttpServletResponse response) **throws** Exception {

// ApplicationContext ac = WebApplicationContextUtils.getWebApplicationContext(request.getSession().getServletContext());

// UserService userService = ac.getBean("userService",UserService.class);

User user = **new** User();

DynaActionForm daf = (DynaActionForm)form;

user.setUsername(daf.getString("username"));

System.*out*.println(user.getUsername());

user.setUsername(daf.getString("password"));

userService.saveUser(user);

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

}

}

不用多次读取配置，struts交给了spring管理

下面配置是不用注解，可以

<bean id=*"userService"* class=*"com.ssh.service.impl.UserServiceImpl"*></bean>

<!-- 注册请求动作，将请求交给spring来管理，将请求处理类中的JAVABean属性注入进去 -->

<bean name=*"/user"* class=*"com.ssh.action.UserAction"*>

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

</bean>

**乱码问题：**

<filter>

<filter-name>CharacterEncodingFilter</filter-name> <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf-8</param-value>

</init-param>

<init-param>

<param-name>forceEncoding</param-name>

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

</init-param>

</filter>

<filter-mapping>

<filter-name>CharacterEncodingFilter</filter-name>

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

</filter-mapping>

**Load延迟加载的问题：由于load是延迟加载，但是由于需要再页面上显示出数据，而load在dao中查询后关闭，所以到不到页面上，用下面的过滤器可以正确显示**

<filter>

<filter-name>hibernateFilter</filter-name>

<filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>

<init-param>

<param-name>singleSession</param-name>

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

</init-param>

</filter>

<filter-mapping>

<filter-name>hibernateFilter</filter-name>

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

</filter-mapping>

//用于解决ssh整合后带来的延迟加载session关闭问题  
//注意改过滤器要放在前面，以免不起作用  
//因为该过滤器的作用是在请求开始时打开session,在请求结束时关闭sessio

# 出现HTTP Status 500 - No bean named 'sessionFactory' is defined

**这个说明在spring配置中是**<bean id=*"mySessionFactory"* class=*"org.spri*

*需要改成sessionFactory*

*Dao整合：*

1. 介绍泛型 <T>

**public** **class** Test {

**public** **static** **void** main(String[] args) {

List<Integer> list = **new** ArrayList<Integer>();

Ts.*m*(list);

}

}

**class** Ts{

**public** **static** <T> **void** m(List<T> list){

T t = (T) **new** Object();

}

}