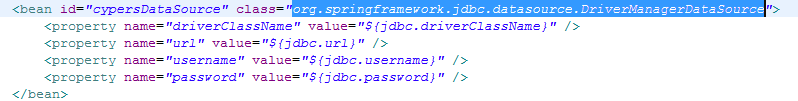
**Spring Hibernate Integration Example**

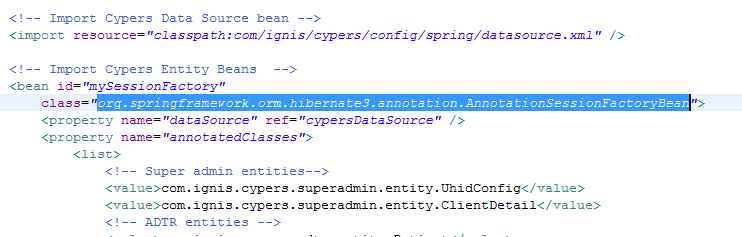
1. To use HibernateTemplate
2. To use HibernateDaoSupport
3. Use plain Hibernate with hibernate.cfg.xml (like what we have been using till date)
4. Autowire session factory as a bean in the daos

**Cypers:** **HibernateTemplate**

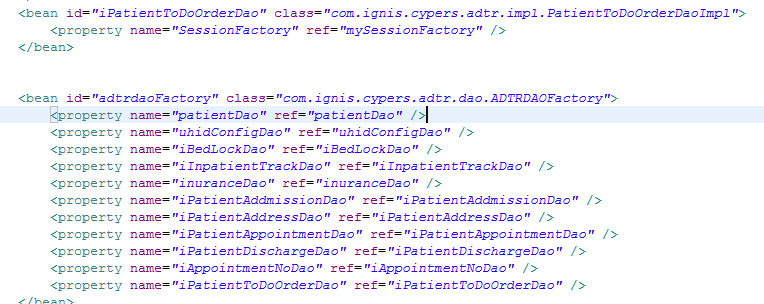
1. Create the Data Source



1. Create the Session Factory and Inject the Data source into session factory



1. Inject session Factory into Dao Layer classes



**The Pros and Cons**

1. **Using HibernateTemplate**

HibernateTemplate is a nice API from Spring for easy integration with hibernate. It mirrors all methods exposed to Hibernate session and proves to be handy! Automatic session close and transaction participation proves to be the best part of this API. Additionally this is handy not only for the nice exception translation but also helps a newbie who does not want to handle sessions or transactions by himself. But, with the advent of Hibernate 3.0.1, this API (using HibernateTemplate.find and the like  - This includes defining private HibernateTemplate in the daos and setting the hibernate template while setting the session factory in a setter method say setSessionFactory) just proves to be futile and it is highly recommended that it is better to handle the sessionFactory by ourselves rather than clinging to Spring and depending on it for handling the sessions for us! As such, this proves to be the first level of Spring and Hibernate integration and is a good option for a newbie!

1. **Using HibernateDaoSupport**

The process of writing SampleDaoImpl extends HibernateDaoSupport is not doubt a very good option. Here, we would then be using no callbacks and session  = getSession(false) rather than setting the hibernate template via the session factory. This is emphatically a very remarkable API provided by Spring for integration with hibernate and is surely better than the option 1 discussed above since this allows throwing of checked exception within the data access code.

1. **Using plain Hibernate with hibernate.cfg.xml**

While this proves to be the mos traditional way of implementation, this is just for the hibernate fans. I am not for such an implementation since there is no need for writing code as given below:

Session session = HibernateUtils.getSession();

try{

Transaction tx = session.beginTransaction();

session.save(Object o);//o is the incoming object to be persisted

tx.commit();

}catch{

(Exception e)

//code to handle exception

}

finally{

session.close();

}

1. **Using Autowiring of the SessionFactory**

As far as i see, this is the best way of integrating spring with hibernate. This is not only the latest and the updated way of integration, this proves to be the best solution for designing web application where the developer has a hold on the sessions and transactions though Spring does this in its own way

<?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"

xmlns:context="http://www.springframework.org/schema/context" xmlns:jee="http://www.springframework.org/schema/jee" 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/jee http://www.springframework.org/schema/jee/spring-jee-2.5.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<!-- Auto-detection of the DAOs -->

<context:component-scan base-package="webapp.dao" />

<context:property-placeholder location="WEB-INF/jdbc.properties" />

<!--<context:property-override location="WEB-INF/override.properties"/>-->

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close" p:driverClassName="${jdbc.driverClassName}" p:url="${jdbc.url}"

p:username="${jdbc.username}" p:password="${jdbc.password}" p:maxActive="${dbcp.maxActive}" p:maxIdle="${dbcp.maxIdle}" p:maxWait="${dbcp.maxWait}" />

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean" p:dataSource-ref="dataSource"

p:configurationClass="org.hibernate.cfg.AnnotationConfiguration" p:packagesToScan="webapp.model">

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect">${hibernate.dialect}</prop>

<prop key="hibernate.show\_sql">${hibernate.show\_sql}</prop>

<prop key="hibernate.format\_sql">${hibernate.format\_sql}</prop>

<prop key="hibernate.generate\_statistics">${hibernate.generate\_statistics}</prop>

</props>

</property>

</bean>

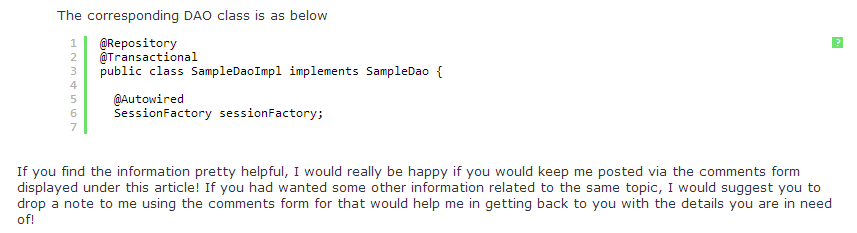
<tx:annotation-driven transaction-manager="txnManager"/>

<bean id="txnManager"

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

p:sessionFactory-ref="sessionFactory"/>

</beans>



To integrate spring with hibernate we need to configure our application.xml. We have to configure session factory by the class AnnotationSessionFactoryBean.

Define HibernateTemplate passing session factory. For the database transaction define HibernateTransactionManager and with the transaction manager configure AOP. Configure all the DAOs.

**Step 1. Create SessionFactory**

Spring allows you to define resources like a JDBC DataSource or a Hibernate SessionFactory as beans in an application context. Set up a JDBC DataSource and a Hibernate SessionFactory on top of it.

|  |
| --- |
| <beans> <bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close"> <property name="driverClassName" value="org.hsqldb.jdbcDriver"/> <property name="url" value="jdbc:hsqldb:hsql://localhost:9001"/> <property name="username" value="sa"/> <property name="password" value="sa"/> </bean> <bean id="mySessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean"> <property name="dataSource" ref="myDataSource"/> <property name="mappingResources"> <list> <value>emp.hbm.xml</value> </list> </property> <property name="hibernateProperties"> <value> hibernate.dialect=org.hibernate.dialect.HSQLDialect </value> </property> </bean> </beans> |

**Step 2. Define HibernateTemplate and Create HibernateTemplate**

The HibernateTemplate class provides many methods that mirror the methods exposed on the Hibernate Session interface.  
Define DAO object and inject Session Factory.

|  |
| --- |
| <beans> <bean id="empDao" class="com.techfaq.EmpDAO"> <property name="sessionFactory" ref="mySessionFactory"/> </bean> </beans> |

|  |
| --- |
| ublic class EmpDAO {  private HibernateTemplate hibernateTemplate;  public void setSessionFactory(SessionFactory sessionFactory) { this.hibernateTemplate = new HibernateTemplate(sessionFactory); }  public Collection getEmpByDept(String dept) throws DataAccessException { return this.hibernateTemplate.find("from com.bean.Emp e where e.dept=?", dept); } } |

OR You can use

**Plain Hibernate to access data base.**

Define DAO object and inject Session Factory.

|  |
| --- |
| <beans> <bean id="empDao" class="com.techfaq.EmpDAO"> <property name="sessionFactory" ref="mySessionFactory"/> </bean> </beans> |

EmpDAO.java class

|  |
| --- |
| public class EmpDAO{  private SessionFactory sessionFactory;  public void setSessionFactory(SessionFactory sessionFactory) { this.sessionFactory = sessionFactory; }  public Collection getEmpByDept(String dept) { return this.sessionFactory.getCurrentSession() .createQuery("from com.bean.Emp e where e.dept=?") .setParameter(0, dept) .list(); } } |

OR You can use

**HibernateDaoSupport.**

HibernateDaoSupport base class offers methods to access the current transactional Session. EmpDAO.java class

|  |
| --- |
| public class EmpDAO extends HibernateDaoSupport {  public Collection getEmpByDept(String dept) throws DataAccessException, MyException { Session session = getSession(false); try { Query query = session.createQuery("from com.bean.Emp e where e.dept=?"); query.setString(0, dept); List result = query.list(); if (result == null) { throw new MyException("No results."); } return result; } catch (HibernateException ex) { throw convertHibernateAccessException(ex); } } } |