Hibernate and Spring Integration

We can simply integrate **hibernate application with spring application**.

In hibernate framework, we provide all the database information hibernate.cfg.xml file.

But if we are going to integrate the hibernate application with spring, we don't need to create the hibernate.cfg.xml file. We can provide all the information in the applicationContext.xml file.

Advantage of Spring framework with hibernate

The Spring framework provides **HibernateTemplate** class, so you don't need to follow so many steps like create Configuration, BuildSessionFactory, Session, beginning and committing transaction etc.

So **it saves a lot of code**.

Understanding problem without using spring:

Let's understand it by the code of hibernate given below:

```
//creating configuration
Configuration cfg=new Configuration();
cfg.configure("hibernate.cfg.xml");

//creating seession factory object
SessionFactory factory=cfg.buildSessionFactory();

//creating session object
Session session=factory.openSession();

//creating transaction object
Transaction t=session.beginTransaction();

Employee e1=new Employee(111,"arun",40000);
session.persist(e1);//persisting the object

t.commit();//transaction is commited
session.close();
```

As you can see in the code of sole hibernate, you have to follow so many steps.

Solution by using HibernateTemplate class of Spring Framework:

Now, you don't need to follow so many steps. You can simply write this:

```
Employee e1=new Employee(111,"arun",40000);
hibernateTemplate.save(e1);
```

Methods of HibernateTemplate class

Let's see a list of commonly used methods of HibernateTemplate class.

No.	Method	Description
1)	void persist(Object entity)	persists the given object.
2)	Serializable save(Object entity)	persists the given object and returns id.
3)	void saveOrUpdate(Object entity)	persists or updates the given object. If id is found, it updates the record otherwise saves the record.
4)	void update(Object entity)	updates the given object.
5)	void delete(Object entity)	deletes the given object on the basis of id.
6)	Object get(Class entityClass, Serializable id)	returns the persistent object on the basis of given id.
7)	Object load(Class entityClass, Serializable id)	returns the persistent object on the basis of given id.
8)	List loadAll(Class entityClass)	returns the all the persistent objects.

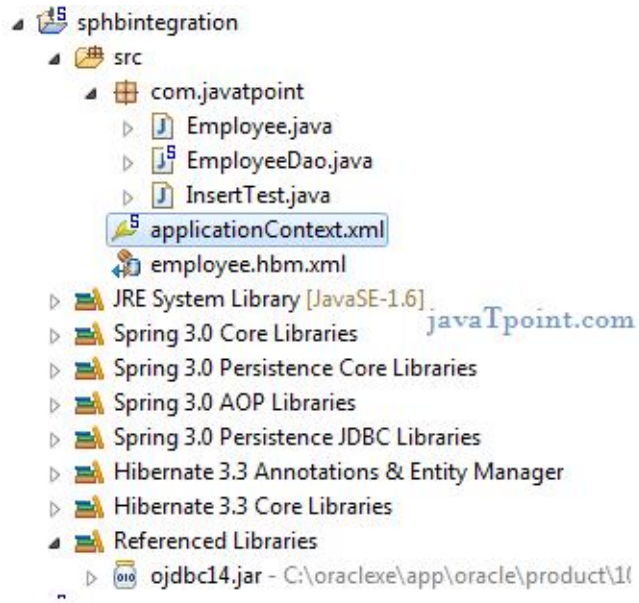
Steps

Let's see what are the simple steps for hibernate and spring integration:

1. **create table in the database** It is optional.
2. **create applicationContext.xml file** It contains information of DataSource, SessionFactory etc.
3. **create Employee.java file** It is the persistent class
4. **create employee.hbm.xml file** It is the mapping file.
5. **create EmployeeDao.java file** It is the dao class that uses HibernateTemplate.
6. **create InsertTest.java file** It calls methods of EmployeeDao class.

Example of Hibernate and spring integration

In this example, we are going to integrate the hibernate application with spring. Let's see the **directory structure** of spring and hibernate example.



1) create the table in the database

In this example, we are using the Oracle as the database, but you may use any database. Let's create the table in the oracle database

```
CREATE TABLE  "EMP558"
   (    "ID" NUMBER(10,0) NOT NULL ENABLE,
    "NAME" VARCHAR2(255 CHAR),
    "SALARY" FLOAT(126),
     PRIMARY KEY ("ID") ENABLE
   )
/
```

2) Employee.java

It is a simple POJO class. Here it works as the persistent class for hibernate.

```
package com.javatpoint;


public class Employee {
private int id;
private String name;
private float salary;


//getters and setters
```

```
}
```

3) employee.hbm.xml

This mapping file contains all the information of the persistent class.

```
<?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.javatpoint.Employee" table="emp558">
        <id name="id">
        <generator class="assigned"></generator>
        </id>

        <property name="name"></property>
        <property name="salary"></property>
</class>

</hibernate-mapping>
```

4) EmployeeDao.java

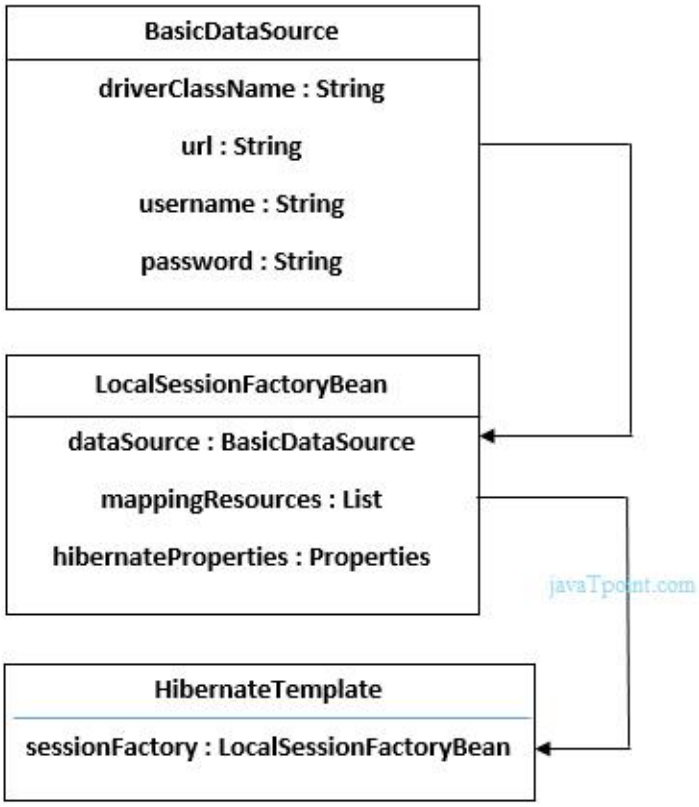
It is a java class that uses the **HibernateTemplate** class method to persist the object of Employee class.

```
package com.javatpoint;
import org.springframework.orm.hibernate3.HibernateTemplate;
import java.util.*;
public class EmployeeDao {
HibernateTemplate template;
public void setTemplate(HibernateTemplate template) {
    this.template = template;
}
//method to save employee
public void saveEmployee(Employee e){
    template.save(e);
}
```

```
//method to update employee
public void updateEmployee(Employee e){
    template.update(e);
}
//method to delete employee
public void deleteEmployee(Employee e){
    template.delete(e);
}
//method to return one employee of given id
public Employee getById(int id){
    Employee e=(Employee)template.get(Employee.class,id);
    return e;
}
//method to return all employees
public List<Employee> getEmployees(){
    List<Employee> list=new ArrayList<Employee>();
    list=template.loadAll(Employee.class);
    return list;
}
}
```

5) applicationContext.xml

In this file, we are providing all the informations of the database in the **BasicDataSource** object. This object is used in the **LocalSessionFactoryBean** class object, containing some other informations such as mappingResources and hibernateProperties. The object of **LocalSessionFactoryBean** class is used in the HibernateTemplate class. Let's see the code of applicationContext.xml file.



File: 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-3.0.xsd">



   <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
      <property name="driverClassName"  value="oracle.jdbc.driver.OracleDriver"></property>
      <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"></property>
      <property name="username" value="system"></property>
      <property name="password" value="oracle"></property>
   </bean>

   <bean id="mysessionFactory"  class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
      <property name="dataSource" ref="dataSource"></property>
```

```
    <property name="mappingResources">
    <list>
    <value>employee.hbm.xml</value>
    </list>
    </property>

    <property name="hibernateProperties">
       <props>
          <prop key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>
          <prop key="hibernate.hbm2ddl.auto">update</prop>
          <prop key="hibernate.show_sql">true</prop>

       </props>
    </property>
</bean>

<bean id="template" class="org.springframework.orm.hibernate3.HibernateTemplate">
<property name="sessionFactory" ref="mysessionFactory"></property>
</bean>

<bean id="d" class="com.javatpoint.EmployeeDao">
<property name="template" ref="template"></property>
</bean>



</beans>
```

6) InsertTest.java

This class uses the EmployeeDao class object and calls its saveEmployee method by passing the object of Employee class.

```
package com.javatpoint;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
```

```
public class InsertTest {
public static void main(String[] args) {

    Resource r=new ClassPathResource("applicationContext.xml");
    BeanFactory factory=new XmlBeanFactory(r);

    EmployeeDao dao=(EmployeeDao)factory.getBean("d");

    Employee e=new Employee();
    e.setId(114);
    e.setName("varun");
    e.setSalary(50000);

    dao.saveEmployee(e);

}
}
```

Now, if you see the table in the oracle database, record is inserted successfully.

download this example (developed using MyEclipse IDE)

Enabling automatic table creation, showing sql queries etc.

You can enable many hibernate properties like automatic table creation by hbm2ddl.auto etc. in applicationContext.xml file. Let's see the code:

```
<property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>
            <prop key="hibernate.hbm2ddl.auto">update</prop>
            <prop key="hibernate.show_sql">true</prop>

        </props>
```

If you write this code, you don't need to create table because table will be created automatically.

← prev

next →