SPRING
COURSE MATERIAL
BY
NAGOOR BABU

{SPRING ORM}

SPRING CORE MODULE INDEX

1. Spring ORM……………………………………………………………………………PAGE 04
2. Application on Spring-Hibernate integration…………………………PAGE 12
3. JPA[Java Persistence API]……………………………………………………PAGE 18
4. Integration of JPA with Spring application in ORM Module……PAGE 32

Spring ORM

Spring ORM:

In enterprise applications both the data models are having their own approaches to represent data in effective manner, these differences are able to provide Paradiagm Mismatches, these mismatches are able to reduce data persistency in enterprise applications.

In general, Object Oriented Data Model and Relational data model are having the following mismatches.

1. Granualarity Mismatch
2. Sub types mismatch
3. Associations Mismatch
4. Identity Mismatch

To improve Data persistency in Enterprise applications we have to resolve the above specified mismtahces between Data models, for this, we have to use "ORM" implementations.

To implement ORM in Enterprise applications we have to use the following ORM implementations.

1. EJBs-Entity Beans
2. JPA
3. Hibernate
4. IBatis
5. JDO

If we want to use Hibernate in enterprise applications then we have to use the following sateps.

1) Persistence Class or Object.
2) Prepare Mapping File.
3) Prepare Hibernate Configuration File
4) Prepare Hibernate Client Application

To prepare Hibernate Client Application we have to use the following steps.

1. Create Configuration class object
2. Create SessionFactory object
3. Create Session Object
4. Create Transaction object if it is required.
5. Persistence operations
6. Close Session Factory and Session objects.

Example:

```
Configuration cfg = new Configuration();
cfg.configure("hibernate.cfg.xml");
SessionFactory session_Factory = cfg.buildSessionFactory();
```

```
Session session = session_Factory.openSession();
Transaction tx = session.beginTransaction();
Employee emp = new Employee();
emp.setEno(111);
emp.setEname("AAA");
emp.setEsal(5000);
emp.setEaddr("Hyd");
session.save(emp);
tx.commit();
System.out.println("Employee Record inserted Succesfully");
session.close();
session_Factory.close();
```

To remove the above boilerplate code, SPRING Framework has provided ORM Module. Spring has provided the complete ORM module in the form of "org.springframework.orm" package.

To abstract the above boilerplate code Spring-ORM module has provided a predefined class in the fomr of "org.springframework.orm.hibernate4.HibernateTemplate" w.r.t Hibernate4 version

Note: If we use Hibernate3.x version then we have to use "org.springframework.orm.hibernate3.HibernateTemplate" class.

org.springframework.orm.hibernate4.HibernateTemplate class has provided the following methods inorder to perform persistence operations.

1. public void persist(Object entity)
2. public Serializable save(Object entity)
3. public void saveOrUpdate(Object entity)
4. public void update(Object entity)
5. public void delete(Object entity)
6. public Object get(Class entityClass, Serializable id)
7. public Object load(Class entityClass, Serializable id)
8. public List loadAll(Class entityClass)

If we want to Integrate Hibernate with Spring then we have to use the following steps.

1) Create Java Project with both Spring[including ORM] and Hibernate Libraries.
2) Create Bean/POJO class.
3) Prepare Hibernate Mapping File
4) Create DAO interface with persistence methods.
5) Create DAO implementation class with HibernateTemplate as property.
6) Prepare Spring Configuration File
7) Prepare Client Application

1. Create Java Project with both Spring[including ORM] and Hibernate Libraries.

Prepare JAVA project in Eclipse IDE and add the following JAR files to Buildpath in the form of the following Libraries.

Spring4_Lib:

- spring-aop-4.0.4.RELEASE.jar
- spring-beans-4.0.4.RELEASE.jar
- spring-context-4.0.4.RELEASE.jar
- spring-context-support-4.0.4.RELEASE.jar
- spring-core-4.0.4.RELEASE.jar
- spring-expression-4.0.4.RELEASE.jar
- spring-jdbc-4.0.4.RELEASE.jar
- commons-io-2.6.jar
- commons-logging-1.2.jar
- spring-tx-4.0.4.RELEASE.jar
- spring-aspects-4.0.4.RELEASE.jar
- spring-orm-4.0.4.RELEASE.jar

Hibernate4_Lib:

- ojdbc6.jar
- antlr-2.7.7.jar
- dom4j-1.6.1.jar
- hibernate-commons-annotations-4.0.5.Final.jar
- hibernate-core-4.3.11.Final.jar
- hibernate-jpa-2.1-api-1.0.0.Final.jar
- jandex-1.1.0.Final.jar
- javassist-3.18.1-GA.jar
- jboss-logging-3.1.3.GA.jar
- jboss-logging-annotations-1.2.0.Beta1.jar
- jboss-transaction-api_1.2_spec-1.0.0.Final.jar
- hibernate-entitymanager-4.3.11.Final.jar

2. Create Bean/POJO class.

```
public class Student{
private String sid;
private String sname;
private String saddr;
setXXX() and getXXX()
}
```

3. Prepare Hibernate Mapping File:

Student.hbm.xml

```
<!DOCTYPE ---- >
<hibernate-mapping>
   <class name="com.durgasoft.pojo.Student" table="student">
      <id name="sid" column="SID"/>
      <property name="sname" column="SNAME"/>
      <property name="saddr" column="SADDR"/>
   </class>
</hibernate-mapping>
```

4. Create DAO interface with persistence methods.

The main intention of DAO interface is to declare all Services.

EX:

```
public interface StudentDao {
   public String insert(Student std);
   public String update(Student std);
   public String delete(Student std);
   public Employee getStudent(int eno);
}
```

5. Create DAO implementation class with HibernateTemplate as property.

The main intention of DAO implementation class is to implement all DAO methods. In DAO implementation class every DAO method must be declared with @Transactional annotation inorder to activate Spring Transaction Service.
Note: If we use @Transactional annotation then it is not required to create Transaction object explicitly and iit is not required to perform commit and rollback operations explicitly.

In DAO implementation class we must declare HibernateTemplate property and its respective setXXX() method inorder to inject HibernateTemplate object.
Example:

```
public class StudentDaoImpl implements StudentDao {
   String status = "";
   private HibernateTemplate hibernateTemplate;
   public void setHibernateTemplate(HibernateTemplate hibernateTemplate) {
      this.hibernateTemplate = hibernateTemplate;
```

```
    }

    @Transactional
    public String insert(Student std) {
        try {
            hibernateTemplate.save(std);
            status = "Insertion Success";
        }catch(Exception ex) {
            ex.printStackTrace();
            status = "Insertion Failure";
        }
        return status;
    }

    @Transactional
    public String update(Student std) {
        try {
            hibernateTemplate.update(std);
            status = "Updations Success";
        }catch(Exception ex) {
            ex.printStackTrace();
            status = "Updations Failure";
        }
        return status;
    }

    @Transactional
    public String delete(Student std) {
        try {
            hibernateTemplate.delete(std);
            status = "Deletion Success";
        }catch(Exception ex) {
            ex.printStackTrace();
            status = "Deletion Failure";
        }
        return status;

    }

    @Transactional
    public Employee getStudent(int sid) {
        Student std = null;
        try {
            std = (Student)hibernateTemplate.get(Student.class, sid);
        }catch(Exception ex) {
            ex.printStackTrace();
        }
        return std;
    }
}
```

6. Prepare Spring Configuration File:

In Spring Configuration File we must configure the following beans

1. DriverManagerDataSource
2. LocalSessionFactoryBean
3. HibernateTransactionManager
4. HibernateTemplate
5. StudentDaoImpl

- ✓ Where org.springframework.jdbc.datasource.DriverManagerDataSource configuration is able to provide Spring inbuilt Connectionpooling Mechanism and it will include the properties liike "driverClassName, url, username, password".

- ✓ Where org.springframework.orm.hibernate4.LocalSessionFactoryBean configuration is able to create SessionFactory object by including the properties like
    1. **dataSource :** represents DataSource bean which was configured in Spring Configuration file.
    2. **mappingResources:** It will take list of values contains mapping files in the form of "<list>" tag.
    3. **hibernateProperties**: It will take hibernate properties in the form of "<props>" tag, it must includes mainly "hibernate.dialect" property.

- ✓ Where org.springframework.orm.hibernate4.HibernateTransactionManager configuration will activate Transaction Manager inorder to provide Transaction Support and it will include "sessionFactory" property.

- ✓ Where org.springframework.orm.hibernate4.HibernateTemplate configuration will provide HibernateTemplate object inorder to perform persistence operations and it will include "sessionFactory" and "checkWriteOperations" with false value.

- ✓ Where com.durgasoft.dao.StudentDaoImpl configuration will provide DAO object inorder to access Dao methods and it will include "hibernateTemplate" property.

Note: To use @Transactional annotation in DAO methods , we must use " <tx:annotation-driven/>" tag in spring configuration file.

Example:

applicationContext.xml

```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: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.xsd
      http://www.springframework.org/schema/tx
      http://www.springframework.org/schema/tx/spring-tx.xsd
      http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop.xsd">

<bean name="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
   <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
   <property name="username" value="system"/>
   <property name="password" value="durga"/>
</bean>
<bean name="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
   <property name="dataSource" ref="dataSource"/>
   <property name="mappingResources">
      <list>
      <value>Student.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>
      </props>
   </property>
</bean>
<tx:annotation-driven/>
<bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">
      <property name="sessionFactory" ref="sessionFactory"/>
```

```
</bean>

<bean name="hibernateTemplate" class="org.springframework.orm.hibernate4.HibernateTemplate">
   <property name="sessionFactory" ref="sessionFactory"/>
   <property name="checkWriteOperations" value="false"></property>
</bean>
<bean name="stdDao" class="com.durgasoft.dao.StudentDaoImpl">
   <property name="hibernateTemplate" ref="hibernateTemplate"/>
 </bean>
</beans>
```

7. Prepare Client Application

The main intention of Client Application is to get Dao object and to access Dao object.

Example:

```
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
StudentDao stdDao = (StudentDao)context.getBean("stdDao");
Student std = new Student();
std.setSid("S-111");
std.setSname("AAA");
std.setSaddr("Hyd");
String status = stdDao.insert(std);
System.out.println(status);
      or
Student std = (Student)stdDao.getStudent(Student.class,"S-111");
System.out.println("Student Details");
System.out.println("--------------------");
System.out.println("Student Id       :"+std.getSid());
System.out.println("Student Name     :"+std.getSname());
System.out.println("Student Address  :"+std.getSaddr());
      or
Student std = new Student();
std.setSid("S-111");
std.setSname("BBB");
std.setSaddr("Hyd");
String status = stdDao.update(std);
System.out.println(status);
     or
String status = stdDao.delete("S-111");
System.out.println(status);
```

Application on Spring-Hibernate integration

Application on Spring-Hibernate integration:

Employee.java

```java
package com.durgasoft.pojo;

public class Employee {
    private int eno;
    private String ename;
    private float esal;
    private String eaddr;

    public int getEno() {
        return eno;
    }
    public void setEno(int eno) {
        this.eno = eno;
    }
    public String getEname() {
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public float getEsal() {
        return esal;
    }
    public void setEsal(float esal) {
        this.esal = esal;
    }
    public String getEaddr() {
        return eaddr;
    }
    public void setEaddr(String eaddr) {
        this.eaddr = eaddr;
    }

    public String toString() {
        return "["+eno+","+ename+","+esal+","+eaddr+"]";
    }

}
```

EmployeeDao.java

```java
package com.durgasoft.dao;

import com.durgasoft.pojo.Employee;

public interface EmployeeDao {
    public String insert(Employee e);
    public String update(Employee e);
    public String delete(Employee e);
    public Employee getEmployee(int eno);
}
```

EmployeeDaoImpl.java

```java
package com.durgasoft.dao;
import org.hibernate.FlushMode;
import org.hibernate.Transaction;
import org.springframework.orm.hibernate4.HibernateTemplate;
import org.springframework.transaction.annotation.Transactional;

import com.durgasoft.pojo.Employee;

public class EmployeeDaoImpl implements EmployeeDao {
    String status = "";
    private HibernateTemplate hibernateTemplate;
    public void setHibernateTemplate(HibernateTemplate hibernateTemplate) {
        this.hibernateTemplate = hibernateTemplate;

    }

    @Override
    @Transactional
    public String insert(Employee e) {
        try {
            hibernateTemplate.save(e);
            status = "Insertion Success";
        }catch(Exception ex) {
            ex.printStackTrace();
            status = "Insertion Failure";
```

```java
        }
        return status;
    }

    @Override
    @Transactional
    public String update(Employee e) {
        try {
            hibernateTemplate.update(e);
            status = "Updations Success";
        }catch(Exception ex) {
            ex.printStackTrace();
            status = "Updations Failure";
        }
        return status;
    }

    @Override
    @Transactional
    public String delete(Employee e) {
        try {
            hibernateTemplate.delete(e);
            status = "Deletion Success";
        }catch(Exception ex) {
            ex.printStackTrace();
            status = "Deletion Failure";
        }
        return status;

    }

    @Override
    @Transactional
    public Employee getEmployee(int eno) {
        Employee emp = null;
        try {
            emp = (Employee)hibernateTemplate.get(Employee.class, eno);
        }catch(Exception ex) {
            ex.printStackTrace();
        }
        return emp;
    }
}
```

Employee.hbm.xml

```xml
<?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">
<hibernate-mapping>
   <class name="com.durgasoft.pojo.Employee" table="emp1">
      <id name="eno" column="ENO"/>
      <property name="ename" column="ENAME"/>
      <property name="esal" column="ESAL"/>
      <property name="eaddr" column="EADDR"/>
   </class>
</hibernate-mapping>
```

applicationContext.xml

```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: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.xsd
      http://www.springframework.org/schema/tx
      http://www.springframework.org/schema/tx/spring-tx.xsd
      http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop.xsd">


 <bean name="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
   <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
   <property name="username" value="system"/>
   <property name="password" value="durga"/>
 </bean>
 <bean name="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
   <property name="dataSource" ref="dataSource"/>
   <property name="mappingResources">
      <list>
```

```
        <value>Employee.hbm.xml</value>
        </list>
        </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</prop>
            <!--  <prop key="hibernate.current_session_context_class">thread</prop> -->
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>
</bean>
<tx:annotation-driven/>
<bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"/>
</bean>

<bean name="hibernateTemplate" class="org.springframework.orm.hibernate4.HibernateTemplate">
    <property name="sessionFactory" ref="sessionFactory"/>
    <property name="checkWriteOperations" value="false"></property>
</bean>
<bean name="empDao" class="com.durgasoft.dao.EmployeeDaoImpl">
    <property name="hibernateTemplate" ref="hibernateTemplate"/>
 </bean>
</beans>
```

Test.java

```
package com.durgasoft.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.orm.hibernate4.HibernateTemplate;

import com.durgasoft.dao.EmployeeDao;
import com.durgasoft.pojo.Employee;

public class Test {

    public static void main(String[] args)throws Exception {
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
```

```
        EmployeeDao empDao = (EmployeeDao)context.getBean("empDao");
        Employee emp = new Employee();
        emp.setEno(111);
        emp.setEname("AAA");
        emp.setEsal(5000);
        emp.setEaddr("Hyd");
        String status = empDao.insert(emp);
        System.out.println(status);
        System.out.println(empDao.getEmployee(111));


    }

}
```

JPA[Java Persistence API]

JPA[Java Persistence API]

Introduction:

- JPA is a an API , it can be used to perform database operations in enterprise applications with ORM implementation tools.

- JPA was provided by J2EE along with EJB3.0 version as a persistence mechanism.

- JPA is a specification provided by SUN Microsystems and it is implemented by some third party vendors like JBOSS, Eclipse Foundations, Apache......

- JPA is following ORM rules regulations to achieve data persistency in enterprise applications and it is implemented by the following tools.

1. Hibernate ---------> JBOSS
2. EclipseLink -------> Eclipse Foundation
3. Open JPA --------> Apache Software Foundations

Note: If we want to use JPA in enterprise applications then we must use either of the JPA implementations.

If we want to prepare JPA applications with "Hibernate" JPA provider then we have to use the following steps.

1. Create Java project in Eclipse with JPA library which includes all Hibernate JARs.
2. Create Entity class under src folder.
3. Create mapping File or Use JPA annotations in POJO class.
4. Create JPA configuration File[persistence.xml]
5. Create Test Application.

1. Create Java project in Eclipse with JPA library which includes all Hibernate JARs.

This step is same as Java project creation and it will include JPA provider Library that is Hibernate jars.

2. Create Entity class under src folder.

Student.java

```java
package com.durgasoft.entity;
public class Student{
```

```
  private String sid;
  private String sname;
  private String saddr;
  setXXX() and getXXX()
}
```

3. Create mapping File or Use JPA annotations in POJO class.

It is same as Hibernate mapping file, it will provide mapping between Object Oriented Data model elements like class, Id property, normal properties with the relational data model elements like Table name, Primary Key Columns, normal columns,....

EX:

Student.xml

```
<hibernate-mapping>
  <class name="com.durgasoft.entity.Student" table="student">
    <id name="sid" column="SID"/>
    <property name="sname" column="SNAME"/>
    <property name="saddr" column="SADDR"/>
  </class>
</hibernate-mapping>
```

4. Create JPA configuration File[persistence.xml]

JPA configuration File is same as Hibernate COnfiguration File, it include all JPA configuration details which are required to interact with database .

IN general, we will provide the following configuration details in JPA configuration file.

1. Jdbc Parameters like Driver class name, Driver URL, Database user name, Database password.
2. Dialect configurations
3. Mapping File or Annotated classes configuration
4. Cache Mechanisms configurations
5. Transactions configurations

The default name of the JPA configuration file is "persistence.xml".

Ex persistence.xml:

```
<persistence>
```

```
<persistence-unit name="std">
   <!-- <class>com.durgasoft.entity.Student</class>-->
   <mapping-file>Student.xml</mapping-file>
   <properties>
      <property name="javax.persistence.jdbc.driver" value="oracle.jdbc.OracleDriver"/>
      <property name="javax.persistence.jdbc.url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
      <property name="javax.persistence.jdbc.user" value="system"/>
      <property name="javax.persistence.jdbc.password" value="durga"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.Oracle10gDialect"/>

      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
   </properties>
</persistence-unit>
</persistence>
```

- ✓ Where <persistence> tag is root tag in JPA configuration File.
- ✓ Where <mapping-file> tag is able to take mapping file configuration
- ✓ where <propertis> tag will include JPA properties.
- ✓ Where <property> tag will take a single JPA property like driver class name, driver url,....

5. Create Test Application:

The main intention of Test /Client application is to perform persistence operations .

To prepare Test application in JPA we have to use the following steps.

1. Create EntityManagerFactory Object.
2. Create EntityManager Object.
3. Create EntityTransaction Object as per the requirement
4. Perform Persistence operation
5. Perform Commit or rollback operations if we use EntityTransaction.

1. Create EntityManagerFactory Object:

javax.persistence.EntityManagerFactory is a Factory class, it able to manage no of EntityManager object.
To get EntitymanagerFactory class object we have to use the following method from javax.persistence.Persistence class.

public static EntityManagerFactory createEntityManagerFactory(String persistence_Unit_Name);

EX: EntityManagerFactory factory = Persistence.createEntitymanagerFactory("std");

2. Create EntityManager Object:

javax.persistence.EntityManager is an interface, it able to provide predefined Library to perform persistence operations.
To get EntityManager object we have to use the following method from EntiotyManagerFactory.

```
public EntityManager createEntityManager()
```

EX: EntityManager entManager = factory.createEntitymanager();

3. Create EntityTransaction Object as per the requirement

javax.persistence.EntityTransaction is a class, it able to provide Tranmsaction support in JPA applications inorder to perform persistence operations.
To get EntityTramsaction object we have to use the following method from EntityManager.

```
public EntityTransaction getTransaction()
```

EX: EntityTransaction entTransaction = entManager.getTransaction();

Note: EntityTransaction contains the following methods inorder to complete Transaction.

```
public void commit()
public void rollback()
```

Note: EntityTranmsaction is required for only non select operations only, not for select operations.

4. Perform Persistence operation:

To perform Persistence operations we have to use the following methods from EntityManager object.

1. public Object find(Class entity_Class, Serializable pk_Value)
2. public void persist(Object obj)
3. public void remove(Object obj)

Note: To perform Updations , first we have to get Entity object from Database table by using find() method then we have to use set New data to Entity Object then perform commit operation.

EX:

```
EntityTransaction entTranction = entManager.getTransaction();
entTransaction.begin();
```

```
Student std = (Student)entManager.find(Student.class, "S-111");
std.setSname("BBB'");
std.setSaddr("Sec");
entTransaction.commit();
System.out.println("Student Updated Successfully");
```

EX:

Test.java

```
public class Test {

    public static void main(String[] args)throws Exception {
        EntityManagerFactory factory = Persistence.createEntityManagerFactory("std");
        EntityManager entManager = factory.createEntityManager();
        Student std = new Student();
        std.setSid("S-111");
                std.setSname("AAA");
                std.setSaddr("Hyd");
        EntityTransaction tx = entManager.getTransaction();
        tx.begin();
        entManager.persist(std);
        tx.commit();
        System.out.println("Student Inserted Succssfully");
    }
}
```

Simple JPA Example with XML Mapping File:

Employee.java

```
package com.durgasoft.pojo;
public class Employee {
    private int eno;
    private String ename;
    private float esal;
    private String eaddr;
    public int getEno() {
        return eno;
    }
    public void setEno(int eno) {
        this.eno = eno;
    }
    public String getEname() {
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public float getEsal() {
        return esal;
    }
    public void setEsal(float esal) {
        this.esal = esal;
    }
    public String getEaddr() {
        return eaddr;
    }
    public void setEaddr(String eaddr) {
        this.eaddr = eaddr;
    }
}
```

Employee.xml

```
<?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">
<hibernate-mapping>
<class name="com.durgasoft.pojo.Employee" table="emp1">
    <id name="eno" column="ENO"/>
    <property name="ename" column="ENAME"/>
    <property name="esal" column="ESAL"/>
    <property name="eaddr" column="EADDR"/>
</class>
</hibernate-mapping>
```

src/META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
   version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">
<persistence-unit name="emp">
   <!--  <class>com.durgasoft.pojo.Employee</class>-->
   <mapping-file>Employee.xml</mapping-file>
   <properties>
      <property name="javax.persistence.jdbc.driver" value="oracle.jdbc.OracleDriver"/>
      <property name="javax.persistence.jdbc.url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
      <property name="javax.persistence.jdbc.user" value="system"/>
      <property name="javax.persistence.jdbc.password" value="durga"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.Oracle10gDialect"/>

      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
   </properties>
</persistence-unit>
</persistence>
```

Test.java

```
package com.durgasoft.test;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.durgasoft.pojo.Employee;

public class Test {

   public static void main(String[] args)throws Exception {
      EntityManagerFactory factory = Persistence.createEntityManagerFactory("emp");
      EntityManager entManager = factory.createEntityManager();
      Employee emp = new Employee();
      emp.setEno(111);
      emp.setEname("AAA");
      emp.setEsal(5000);
      emp.setEaddr("Hyd");
      EntityTransaction tx = entManager.getTransaction();
      tx.begin();
```

```
        entManager.persist(emp);
        tx.commit();
        System.out.println("Employee Inserted Succssfully");
    }
}
```

Simple JPA Example with Annotations:

If we want to use Annotations in JPA application then we have to use the following steps.

1.Use javax.persistence provided annotations in Entity class:

1. @Entity
2. @Table
3. @Id
4. @Column

2. Configure Annotated class in persistence.xml file:

```
<persistence>
     <persistence-unit name="std">
    <class>com.durgasoft.entity.Student</class>
          ------
     </persistence-unit>
  </persistence>
```

Example:

Employee.java

```
package com.durgasoft.pojo;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="emp2")
public class Employee {
    @Id
    @Column(name="ENO")
    private int eno;
```

```java
    @Column(name="ENAME")
    private String ename;
    @Column(name="ESAL")
    private float esal;
    @Column(name="EADDR")
    private String eaddr;
    public int getEno() {
        return eno;
    }
    public void setEno(int eno) {
        this.eno = eno;
    }
    public String getEname() {
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public float getEsal() {
        return esal;
    }
    public void setEsal(float esal) {
        this.esal = esal;
    }
    public String getEaddr() {
        return eaddr;
    }
    public void setEaddr(String eaddr) {
        this.eaddr = eaddr;
    }


}
```

src/META-INF/persistence.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
    version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">
<persistence-unit name="emp">
    <class>com.durgasoft.pojo.Employee</class>
    <!--  <mapping-file>Employee.xml</mapping-file> -->
    <properties>
        <property name="javax.persistence.jdbc.driver" value="oracle.jdbc.OracleDriver"/>
        <property name="javax.persistence.jdbc.url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
        <property name="javax.persistence.jdbc.user" value="system"/>
        <property name="javax.persistence.jdbc.password" value="durga"/>
        <property name="hibernate.dialect" value="org.hibernate.dialect.Oracle10gDialect"/>

        <property name="hibernate.show_sql" value="true"/>
        <property name="hibernate.format_sql" value="true"/>
    </properties>
</persistence-unit>
</persistence>
```

Test.java

```java
package com.durgasoft.test;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.durgasoft.pojo.Employee;

public class Test {

    public static void main(String[] args)throws Exception {
        EntityManagerFactory factory = Persistence.createEntityManagerFactory("emp");
        EntityManager entManager = factory.createEntityManager();
        Employee emp = new Employee();
        emp.setEno(111);
        emp.setEname("AAA");
        emp.setEsal(5000);
        emp.setEaddr("Hyd");
        EntityTransaction tx = entManager.getTransaction();
        tx.begin();
        entManager.persist(emp);
        tx.commit();
        System.out.println("Employee Inserted Succssfully");
    }
}
```

JPA With ECLIPSE Link Implementattion:

If we want to use JPA with EclipseLink implementation then we have to use the following steps.

1. Create JPA project.
2. Create Entity Class with Annotations
3. Create JPA configuration File
4. Create Test Application

1. Create JPA project.

1. Right Click on "Project Explorer".
2. Select on "New".
3. Select "Others"
4. Search and Select JPA Project
5. Click on "Next" button.
6. Provide package name:app6
7. Click on "next" button.
8. Click on "Next" button.
9. Click on "Download Libraries" icon.
10. Select EclipseLink2.5.2 library.
11. Click on "Next" button.
12. Select "Check box" of Accepct Licence of this Aggrement.
13. Click on "Finish" Button.
14. Click on "Finish" button.
15. Click on "Open Perspective".

With these steps JPA project will be created in projects Explorer part..

2. Create Entity Class with Annotations

Create package "com.durgasoft.entity" under src folder and create Entity class.

Employee.java

```
package com.durgasoft.entity;
import java.io.Serializable;
import java.lang.String;
import javax.persistence.*;
@Entity
@Table(name="emp1")
public class Employee implements Serializable {

```

```java

    @Id
    @Column(name="ENO")
    private int eno;
    @Column(name="ENAME")
    private String ename;
    @Column(name="ESAL")
    private float esal;
    @Column(name="EADDR")
    private String eaddr;
    private static final long serialVersionUID = 1L;

    public Employee() {
        super();
    }
    public int getEno() {
        return this.eno;
    }

    public void setEno(int eno) {
        this.eno = eno;
    }
    public String getEname() {
        return this.ename;
    }

    public void setEname(String ename) {
        this.ename = ename;
    }
    public float getEsal() {
        return this.esal;
    }

    public void setEsal(float esal) {
        this.esal = esal;
    }
    public String getEaddr() {
        return this.eaddr;
    }

    public void setEaddr(String eaddr) {
        this.eaddr = eaddr;
    }
```

```

}
```

3. Create JPA configuration File

Open persistence.xml file which is existed under"src\META-INF" folder and provide the following details.

persistence.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
<persistence-unit name="emp">
<class>com.durgasoft.entity.Employee</class>
<properties>
<property name="javax.persistence.jdbc.driver" value="oracle.jdbc.OracleDriver"/>
<property name="javax.persistence.jdbc.url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="javax.persistence.jdbc.user" value="system"/>
<property name="javax.persistence.jdbc.password" value="durga"/>
</properties>
</persistence-unit>
</persistence>
```

4. Create Test Application

Create a package "com.durgasoft.test" and prepare Test class:

Test.java

```java
package com.durgasoft.test;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.durgasoft.entity.Employee;

public class Test {

```

```
    public static void main(String[] args)throws Exception {
    EntityManagerFactory factory = Persistence.createEntityManagerFactory("emp");
        EntityManager entityManager = factory.createEntityManager();
        /*
        EntityTransaction entityTransaction = entityManager.getTransaction();
        entityTransaction.begin();
        Employee emp = new Employee();
        emp.setEno(111);
        emp.setEname("AAA");
        emp.setEsal(5000);
        emp.setEaddr("Hyd");
        entityManager.persist(emp);
        entityTransaction.commit();
        System.out.println("Employee Inserted Successfully");
        */
        /*
        Employee emp = entityManager.find(Employee.class, 111);
        System.out.println("Employee Details");
        System.out.println("------------------------");
        System.out.println("Employee Number   :"+emp.getEno());
        System.out.println("Employee Name     :"+emp.getEname());
        System.out.println("Employee Salary   :"+emp.getEsal());
        System.out.println("Employee Address  :"+emp.getEaddr());
        */
        /*
        EntityTransaction entityTransaction = entityManager.getTransaction();
        entityTransaction.begin();
        Employee emp = entityManager.find(Employee.class, 111);
        emp.setEname("BBB");
        emp.setEsal(7000);
        emp.setEaddr("Sec");
        entityTransaction.commit();
        System.out.println("Employee updated Successfully");
        */
        EntityTransaction entityTransaction = entityManager.getTransaction();
        entityTransaction.begin();
        Employee emp = entityManager.find(Employee.class, 111);
        entityManager.remove(emp);
        entityTransaction.commit();
        System.out.println("Employee Deleted Successfully");
        entityManager.close();
    }
}
```

Integration of JPA with Spring application in ORM Module

Integration of JPA with Spring application in ORM Module:

1. Create Java Project with both Spring Library and Hibernate Library.
2. Create Dao interface
3. Create Dao implementation classs.
4. Create POJO / Entity class.
5. Create Hibernate Mapping File.
6. Create Spring Configuration File.
7. Create Test Application.

1. Create Java Project with both Spring Library and Hibernate Library.

Spring Library:

- commons-logging-1.2.jar
- spring-aop-4.3.9.RELEASE.jar
- spring-beans-4.3.9.RELEASE.jar
- spring-context-4.3.9.RELEASE.jar
- spring-context-support-4.3.9.RELEASE.jar
- spring-core-4.3.9.RELEASE.jar
- spring-expression-4.3.9.RELEASE.jar
- spring-jdbc-4.3.9.RELEASE.jar
- spring-orm-4.3.9.RELEASE.jar
- spring-tx-4.3.9.RELEASE.jar

Hibernate Library:

- hibernate3.jar
- antlr-2.7.6.jar
- commons-collections-3.1.jar
- dom4j-1.6.1.jar
- javassist-3.12.0.GA.jar
- jta-1.1.jar
- slf4j-api-1.6.1.jar
- hibernate-jpa-2.0-api-1.0.1.Final.jar
- ojdbc6.jar

2. Create Dao interface

EmployeeDao.java

```java
package com.durgasoft.dao;

import com.durgasoft.entity.Employee;

public interface EmployeeDao {
    public String insertEmployee(Employee emp);
    public Employee findEmployee(int eno);
    public String updateEmployee(Employee emp);
    public String removeEmployee(int eno);
}
```

3.Create Dao implementation class

```java
package com.durgasoft.dao;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.durgasoft.entity.Employee;

@Repository
public class EmployeeDaoImpl implements EmployeeDao {

    String status = "";

    @PersistenceContext
    private EntityManager entityManager;

    @Transactional
    @Override
    public String insertEmployee(Employee emp) {
        entityManager.persist(emp);
        status = "SUCCESS";
        return status;
```

```
    }

    @Override
    public Employee findEmployee(int eno) {
        Employee emp = (Employee)entityManager.find(Employee.class, eno);
        return emp;
    }

    @Transactional
    @Override
    public String updateEmployee(Employee emp) {
        Employee employee = (Employee)entityManager.find(Employee.class, emp.getEno());
        employee.setEname(emp.getEname());
        employee.setEsal(emp.getEsal());
        employee.setEaddr(emp.getEaddr());
        status = "SUCCESS";
        return status;
    }

    @Transactional
    @Override
    public String removeEmployee(int eno) {
        Employee employee = (Employee)entityManager.find(Employee.class, eno);
        entityManager.remove(employee);
        status = "SUCCESS";
        return status;
    }

}
```

- ✓ Where "@PersistenceContext" will inject EntityManager object into Dao implementation class.
- ✓ Where "@Transactional" annotation will inject Transaction service in DAO methiods, where it is not required to Create EntityTransaction object and not required to perform commit() or rollback() operations.

4. Create POJO / Entity class:

Employee.java

```
package com.durgasoft.entity;

```

```java
public class Employee {
    private int eno;
    private String ename;
    private float esal;
    private String eaddr;

    public int getEno() {
        return eno;
    }
    public void setEno(int eno) {
        this.eno = eno;
    }
    public String getEname() {
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public float getEsal() {
        return esal;
    }
    public void setEsal(float esal) {
        this.esal = esal;
    }
    public String getEaddr() {
        return eaddr;
    }
    public void setEaddr(String eaddr) {
        this.eaddr = eaddr;
    }
}
```

5. Create Hibernate Mapping File.

Employee.xml

```xml
<?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">
<hibernate-mapping>
    <class name="com.durgasoft.entity.Employee" table="emp1">
        <id name="eno"/>
        <property name="ename"/>
        <property name="esal"/>
        <property name="eaddr"/>
    </class>
</hibernate-mapping>
```

6. Create Spring Configuration 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:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:context="http://www.springframework.org/schema/context"

    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd
         http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
<context:annotation-config/>
<tx:annotation-driven/>
<bean id="dataSource"     class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
        <property name="username" value="system"/>
        <property name="password" value="durga"/>
</bean>
<bean id="entityManagerFactoryBean" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="dataSource" ref="dataSource"/>
  <property name="persistenceUnitName" value="emp"/>
  <property name="jpaVendorAdapter">
       <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"/>
  </property>
  <property name="mappingResources">
```

```
    <list>
      <value>Employee.xml</value>
    </list>
  </property>
  <property name="jpaProperties">
    <props>
      <prop key="hibernate.dialect">org.hibernate.dialect.OracleDialect</prop>
      <prop key="hibernate.show_sql">true</prop>
    </props>
  </property>
</bean>
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
   <property name="entityManagerFactory" ref="entityManagerFactoryBean"/>
</bean>
<bean id="empDao" class="com.durgasoft.dao.EmployeeDaoImpl"/>
</beans>
```

- ✓ Where "org.springframework.jdbc.datasource.DriverManagerDataSource" will provide dataSource object, it required the following dependencies.
  1. driverClassName
  2. url
  3. username
  4. password

- ✓ Where "org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean" will provide EntityManager object in Spring Application and it require the following dependencies.
  1. dataSource
  2. persistenceUnitName
  3. jpaVendorAdapter
  4. mappingResources
  5. jpaProperties

- ✓ Where "dataSource" will take a DataSource reference which we configure in Spring configuration File.

- ✓ Where "persistenceUnitName" will take persistence name like "emp".

- ✓ Where "jpaVendorAdpter"property will take the class like "org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" and it will provide all Hibernate implementation of JPA to Spring application.

- ✓ Where "mappingResources" property will take all mapping Files which we are using in spring applications in the form of <list> type.

- ✓ Where "jpaProperties" will take JPA implementation properties like dialect, show_sql,... in the form of "<props>" type.

- ✓ Where "org.springframework.orm.jpa.JpaTransactionManager" will provide Transaction implementation provided by JPA vendor.

- ✓ Where "<context:annotation-config/>" tag will activate the annotations like "@Repository"
- ✓ , "@Service", "@Component".......

- ✓ Where "<tx:annotation-driven/>" tag will activate @Transactional annotation which will inject Transaction Service in Spring application.

7. Create Test Application:

```java
package com.durgasoft.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;

import com.durgasoft.dao.EmployeeDao;
import com.durgasoft.entity.Employee;

public class Test {

   public static void main(String[] args) {
       ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
       EmployeeDao empDao = (EmployeeDao) context.getBean("empDao");
       /*
       Employee emp = new Employee();
       emp.setEno(111);
       emp.setEname("AAA");
       emp.setEsal(5000);
       emp.setEaddr("Hyd");
       String status = empDao.insertEmployee(emp);
       System.out.println(status);
       */
       /*
```

```
        Employee emp = empDao.findEmployee(111);
        System.out.println("Employee Details");
        System.out.println("----------------------");
        System.out.println("Employee Number   :"+emp.getEno());
        System.out.println("Employee Name     :"+emp.getEname());
        System.out.println("Employee Salary   :"+emp.getEsal());
        System.out.println("Employee Address  :"+emp.getEaddr());
        */
        /*
        Employee emp = new Employee();
        emp.setEno(111);
        emp.setEname("BBB");
        emp.setEsal(7000);
        emp.setEaddr("Pune");
        String status = empDao.updateEmployee(emp);
        System.out.println(status);
        */
        String status = empDao.removeEmployee(111);
        System.out.println(status);
    }

}
```

Example:

Employee.java

```
package com.durgasoft.pojo;

public class Employee {
    private int eno;
    private String ename;
    private float esal;
    private String eaddr;

    public int getEno() {
        return eno;
    }
    public void setEno(int eno) {
        this.eno = eno;
    }
    public String getEname() {
        return ename;
```

```
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public float getEsal() {
        return esal;
    }
    public void setEsal(float esal) {
        this.esal = esal;
    }
    public String getEaddr() {
        return eaddr;
    }
    public void setEaddr(String eaddr) {
        this.eaddr = eaddr;
    }

    public String toString() {
        return "["+eno+","+ename+","+esal+","+eaddr+"]";
    }

}
```

EmployeeDao.java

```
package com.durgasoft.dao;

import com.durgasoft.entity.Employee;

public interface EmployeeDao {
    public String insertEmployee(Employee emp);
    public Employee findEmployee(int eno);
    public String updateEmployee(Employee emp);
    public String removeEmployee(int eno);
}
```

EmployeeDaoImpl.java

```
package com.durgasoft.dao;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

```

```java
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.durgasoft.entity.Employee;

@Repository
public class EmployeeDaoImpl implements EmployeeDao {

    String status = "";

    @PersistenceContext
    private EntityManager entityManager;

    @Transactional
    @Override
    public String insertEmployee(Employee emp) {
        entityManager.persist(emp);
        status = "SUCCESS";
        return status;
    }

    @Override
    public Employee findEmployee(int eno) {
        Employee emp = (Employee)entityManager.find(Employee.class, eno);
        return emp;
    }

    @Transactional
    @Override
    public String updateEmployee(Employee emp) {
        Employee employee = (Employee)entityManager.find(Employee.class, emp.getEno());
        employee.setEname(emp.getEname());
        employee.setEsal(emp.getEsal());
        employee.setEaddr(emp.getEaddr());
        status = "SUCCESS";
        return status;
    }

    @Transactional
    @Override
    public String removeEmployee(int eno) {
        Employee employee = (Employee)entityManager.find(Employee.class, eno);
        entityManager.remove(employee);
        status = "SUCCESS";
        return status;
    }

}
```

Employee.hbm.xml

```
<?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">
<hibernate-mapping>
    <class name="com.durgasoft.pojo.Employee" table="emp2">
        <id name="eno" column="ENO"/>
        <property name="ename" column="ENAME"/>
        <property name="esal" column="ESAL"/>
        <property name="eaddr" column="EADDR"/>
    </class>
</hibernate-mapping>
```

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:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:context="http://www.springframework.org/schema/context"

    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd
         http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
<context:annotation-config/>
<tx:annotation-driven/>
```

```
<bean id="dataSource"	class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
        <property name="username" value="system"/>
        <property name="password" value="durga"/>
</bean>
<bean id="entityManagerFactoryBean" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="dataSource" ref="dataSource"/>
  <property name="persistenceUnitName" value="emp"/>
  <property name="jpaVendorAdapter">
       <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"/>
  </property>
  <property name="mappingResources">
    <list>
        <value>Employee.xml</value>
    </list>
  </property>
  <property name="jpaProperties">
    <props>
        <prop key="hibernate.dialect">org.hibernate.dialect.OracleDialect</prop>
        <prop key="hibernate.show_sql">true</prop>
    </props>
  </property>
</bean>
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
   <property name="entityManagerFactory" ref="entityManagerFactoryBean"/>
</bean>
<bean id="empDao" class="com.durgasoft.dao.EmployeeDaoImpl"/>
</beans>
```

Test.java

```
package com.durgasoft.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;

import com.durgasoft.dao.EmployeeDao;
import com.durgasoft.entity.Employee;
```

```

public class Test {

   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
      EmployeeDao empDao = (EmployeeDao) context.getBean("empDao");
      /*
      Employee emp = new Employee();
      emp.setEno(111);
      emp.setEname("AAA");
      emp.setEsal(5000);
      emp.setEaddr("Hyd");
      String status = empDao.insertEmployee(emp);
      System.out.println(status);
      */
      /*
      Employee emp = empDao.findEmployee(111);
      System.out.println("Employee Details");
      System.out.println("----------------------");
      System.out.println("Employee Number   :"+emp.getEno());
      System.out.println("Employee Name     :"+emp.getEname());
      System.out.println("Employee Salary   :"+emp.getEsal());
      System.out.println("Employee Address  :"+emp.getEaddr());
      */
      /*
      Employee emp = new Employee();
      emp.setEno(111);
      emp.setEname("BBB");
      emp.setEsal(7000);
      emp.setEaddr("Pune");
      String status = empDao.updateEmployee(emp);
      System.out.println(status);
      */
      String status = empDao.removeEmployee(111);
      System.out.println(status);
   }

}
```