**Spring ORM**

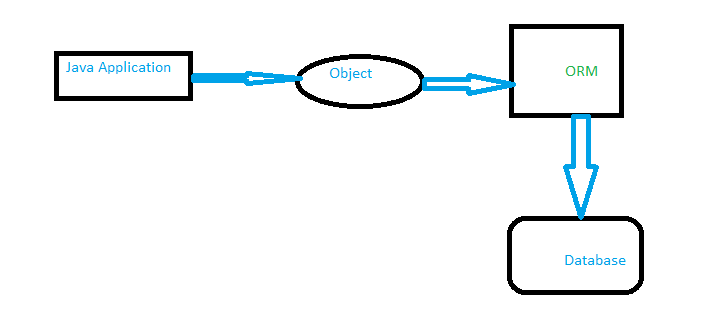
**ORM**

**ORM – Object Relational Mapping**

Object Relational Mapping is a technique for converting data between incompatible type systems using OOPS language

ORM **encapsulates** database interaction inside an object.

**ORM**



ORM allows java objects as representation of a relational database.

**Encaspulation**:

A way to achieve abstraction to objects data

Hides object properties from outer world.

Provides method to get/set object data.

It is also called as “Data hiding”

Spring ORM:

Spring ORM is an module that covers many persistence technologies namely JPA,JDO,Hibernate,ibatis technologies. For each technology , the configuration consists in injecting data source bean into Session Factory or EntityManagerFactory.

EntityManagerFactory provides instances of EntityManager for connecting to same database. All the instances are configured to use the same setting.

List of required and optional dependencies

Required dependencies

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- JPA Vendor -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-entitymanager</artifactId>

<version>${hibernate.version}</version>

</dependency>

<!-- IN MEMORY Database and JDBC Driver -->

<dependency>

<groupId>hsqldb</groupId>

<artifactId>hsqldb</artifactId>

<version>1.8.0.7<

**Advantage of ORM frameworks with Spring:**

1. **Less Coding is required**
2. **Easy to test**
3. **Better Exception handling**

Spring ORM covers many technologies like

Hibernate

iBatis

JPA

Before starting the concept of Spring ORM , we can go through the bean concept.

What is Bean?

Beans are the objects that construct the application. A bean is an object that is instantiated, assembled and otherwise managed by a Spring IOC container. The definition of bean contains and called configuration metadata. This is to create beans lifecycle details finally beans dependencies.

**Spring ORM**

Spring ORM is an module that covers many persistence technologies namely JPA,JDO,Hibernate and iBatis

Every technology the configuration basically consists in injecting a database bean into some kind of SessionFactory or EntityManagerFactory.

**SessionFactory**

1. SessionFactory is to create and manage Sessions. Session is to provide a CRUD interface for mapped classes, and also access to the more versatile Criteria API.
2. SessionFactory is thread safe where as Session is not thread safe

SessionFactory is an instance which will create Session objects.

Example:

 A manufacturing plant is a place where the cars will be produced.

Entity Manager:

**EntityManager** is used to interact with persistence context and **EntityManagerFactory** interacts with **entity manager factory**. Using **EntityManager** methods, we can interact with database. We can save, update and delete the data in database.

Example:

public class BaseDao{

private static final String PERSISTENCE\_UNIT\_NAME = "Employee";

private static EntityManagerFactory factory =

Persistence.createEntityManagerFactory(PERSISTENCE\_UNIT\_NAME);

public void create(MyEntiy person){

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();

// do what ever you need

em.getTransaction().commit();

em.close();

}

}

EntityManager is just a wrapper around a JDBC connection. It's very light weight and can be created and destroyed without performance penalty.

Employee emp = new Employee();

emp.setName("Joe M");

getEntityManager().persist(emp);

Dependencies to Support Spring ORM :

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>5.1.4.RELEASE</version>

</dependency>

EntityManagerFactory provides instances of EntityManager for connecting to same database. All the instances are configured to use the same setting.

SessionFactory and Session are hibernate-specific. The EntityManager invokes the hibernate session under the hood. And if you need some specific features that are not available in the EntityManager, you can obtain the session by calling

Session session = entityManager.unwrap(Session.class);

What is ORM?

Object-relational mapping is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to create their own ORM tools.

Simpler Way of ORM:

A simple answer is that you wrap your tables or stored procedures in classes in your programming language, so that instead of writing SQL statements to interact with your database, you use methods and properties of objects.

Instead of

String sql = "select CCR\_INSR\_NAME from CMS\_CPRG\_RGST\_CNFG where CCR\_CPRG\_RGID=10" Result res = sql.Execute(); String regId = res[0]["CCR\_INSR\_NAME"];

Use like this

CardProgram p = repository.GetInsertionName(10); String name = p.insertionName;

**Integrated transaction management:**

By the help of Spring framework, we can wrap our mapping code with an explicit template wrapper class or AOP style method interceptor.

Frameworks supported by Spring ORM

1. Hibernate

2. ibatis

3. JPA(Java Persistence API)

4. TopLink

5. JDO

Spring Hibernate Integration:

Steps:

1. Create Java Project with both Spring and Hibernate libraries

2. Create Pojo/bean class

3. Prepare hibernate mapping file Or else annotations

4. Create DAO interface.

5. Create DAO implementation class.

6. Create Spring Configuration file.

7. Create Client application.

Create Java Project with both Spring and Hibernate libraries

1. Spring lib

2. Hibernate lib

Create Pojo/bean class

public Class Employee{

private int eno;

private String ename;

}

Prepare hibernate mapping file Or else annotations:

Employee.hbm.xml

or

@Entity

@Table(name=””)

public Class Employee{

@Column(name=””)

private int eno;

private String ename;

}

Create DAO interface.

Public interface EmployeeDao

{

public String insertEmployee(Employee emp);

}

Create DAO implementation Class:

Create Configuration object

Create SessionFactory Object

Create Transaction Object Creation

This can be achieved using HibernateTemplate itself. This is the abstraction layer of Spring framework.

Public String EmployeeDaoImpl implementation EmployeeDao

{

}

Create Spring Configuration file.

1. DriverManagerDataSource Configuration - to get connection objects

property to declare:

→ DriverClassName

→ url (jdbc:oracle:)

→ userName

→ password

2. LocalSessionFactoryBean – this bean is for loading [prop name : session factory]

→ dataSource

→ MappingResources (Employee.hbm.xml)

→ HibernateProperty (hibernate.dialect)

3. HibernateSessionFactoryObject – this will provide by hiberate TransactionManager prop name

We required session factory

4. Hibernate TransactionManager – this is for transaction object, where transaction object is needed in CRUD operations except select operation.

5. Hibernate Template[prop name : hibernateTemplate]

sessionFactory

chechWriteOperations : true

6. EmployeeDaoImpl Configuration

Create Client Application

Whether it's License/Open Source

Spring ORM is open source

List of required and optional dependencies

Required dependencies

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- JPA Vendor -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-entitymanager</artifactId>

<version>${hibernate.version}</version>

</dependency>

<!-- IN MEMORY Database and JDBC Driver -->

<dependency>

List of required and optional dependencies

Required dependencies

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- JPA Vendor -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-entitymanager</artifactId>

<version>${hibernate.version}</version>

</dependency>

<!-- IN MEMORY Database and JDBC Driver -->

<dependency>

<groupId>hsqldb</groupId>

<artifactId>hsqldb</artifactId>

<version>1.8.0.7<version>

</dependency>

**Persistence**:

Meaning of Persistence is prolonged existence of something.

Persistence deals with storing and retrieving of application data starts with EJB 3.0

Persistence layer of enterprise serves as an intermediatory between the business functions of the app and data stores in an application.