Javax.persistence Annotations for Hibernate - 2017

**javax.persistence   
OneToOne**

|  |  |
| --- | --- |
| **Optional Element Summary** | |
| [CascadeType](http://docs.oracle.com/javaee/6/api/javax/persistence/CascadeType.html" \o "enum in javax.persistence)[] | [**cascade**](http://docs.oracle.com/javaee/6/api/javax/persistence/OneToOne.html#cascade())           (Optional) The operations that must be cascaded to the target of the association. |
| [FetchType](http://docs.oracle.com/javaee/6/api/javax/persistence/FetchType.html" \o "enum in javax.persistence) | [**fetch**](http://docs.oracle.com/javaee/6/api/javax/persistence/OneToOne.html#fetch())           (Optional) Whether the association should be lazily loaded or must be eagerly fetched. |
| java.lang.String | [**mappedBy**](http://docs.oracle.com/javaee/6/api/javax/persistence/OneToOne.html#mappedBy())           (Optional) The field that owns the relationship. |
| boolean | [**optional**](http://docs.oracle.com/javaee/6/api/javax/persistence/OneToOne.html#optional())            (Optional) Whether the association is optional. |
| boolean | [**orphanRemoval**](http://docs.oracle.com/javaee/6/api/javax/persistence/OneToOne.html#orphanRemoval())           (Optional) Whether to apply the remove operation to entities that have been removed from the relationship and to cascade the remove operation to those entities. |
| java.lang.Class | [**targetEntity**](http://docs.oracle.com/javaee/6/api/javax/persistence/OneToOne.html#targetEntity())           (Optional) The entity class that is the target of the association. |

Defines a single-valued association to another entity that has one-to-one multiplicity. It is not normally necessary to specify the associated target entity explicitly since it can usually be inferred from the type of the object being referenced. If the relationship is bidirectional, the non-owning side must use the mappedBy element of the OneToOne annotation to specify the relationship field or property of the owning side.

The OneToOne annotation may be used within an embeddable class to specify a relationship from the embeddable class to an entity class. If the relationship is bidirectional and the entity containing the embeddable class is on the owning side of the relationship, the non-owning side must use the mappedBy element of the OneToOne annotation to specify the relationship field or property of the embeddable class. The dot (".") notation syntax must be used in the mappedBy element to indicate the relationship attribute within the embedded attribute. The value of each identifier used with the dot notation is the name of the respective embedded field or property.

Example 1: One-to-one association that maps a foreign key column

// On Customer class:

**@OneToOne(optional=false)**

**@JoinColumn(**

**name="CUSTREC\_ID", unique=true, nullable=false, updatable=false)**

public CustomerRecord getCustomerRecord() { return customerRecord; }

// On CustomerRecord class:

**@OneToOne(optional=false, mappedBy="customerRecord")**

public Customer getCustomer() { return customer; }

Example 2: One-to-one association that assumes both the source and target share the same primary key values.

// On Employee class:

@Entity

public class Employee {

@Id Integer id;

**@OneToOne @MapsId**

**EmployeeInfo info;**

...

}

// On EmployeeInfo class:

@Entity

public class EmployeeInfo {

@Id Integer id;

...

}

Example 3: One-to-one association from an embeddable class to another entity.

@Entity

public class Employee {

@Id int id;

@Embedded LocationDetails location;

...

}

@Embeddable

public class LocationDetails {

int officeNumber;

@OneToOne ParkingSpot parkingSpot;

...

}

@Entity

public class ParkingSpot {

@Id int id;

String garage;

@OneToOne(mappedBy="location.parkingSpot") Employee assignedTo;

...

}

## javax.persistence  OneToMany

|  |  |
| --- | --- |
| **Optional Element Summary** | |
| [CascadeType](http://docs.oracle.com/javaee/6/api/javax/persistence/CascadeType.html" \o "enum in javax.persistence)[] | [**cascade**](http://docs.oracle.com/javaee/6/api/javax/persistence/OneToMany.html#cascade())          (Optional) The operations that must be cascaded to the target of the association. |
| [FetchType](http://docs.oracle.com/javaee/6/api/javax/persistence/FetchType.html" \o "enum in javax.persistence) | [**fetch**](http://docs.oracle.com/javaee/6/api/javax/persistence/OneToMany.html#fetch())          (Optional) Whether the association should be lazily loaded or must be eagerly fetched. |
| java.lang.String | [**mappedBy**](http://docs.oracle.com/javaee/6/api/javax/persistence/OneToMany.html#mappedBy())            The field that owns the relationship. |
| boolean | [**orphanRemoval**](http://docs.oracle.com/javaee/6/api/javax/persistence/OneToMany.html#orphanRemoval())         (Optional) Whether to apply the remove operation to entities that have been removed from the relationship and to cascade the remove operation to those entities. |
| java.lang.Class | [**targetEntity**](http://docs.oracle.com/javaee/6/api/javax/persistence/OneToMany.html#targetEntity())         (Optional) The entity class that is the target of the association. |

 Defines a many-valued association with one-to-many multiplicity.

If the collection is defined using generics to specify the element type, the associated target entity type need not be specified; otherwise the target entity class must be specified. If the relationship is bidirectional, the mappedBy element must be used to specify the relationship field or property of the entity that is the owner of the relationship.

The OneToMany annotation may be used within an embeddable class contained within an entity class to specify a relationship to a collection of entities. If the relationship is bidirectional, the mappedBy element must be used to specify the relationship field or property of the entity that is the owner of the relationship. When the collection is a java.util.Map, the cascade element and the orphanRemoval element apply to the map value.

Example 1: One-to-Many association using generics

// In Customer class:

**@OneToMany(cascade=ALL, mappedBy="customer")**

public Set<Order> getOrders() { return orders; }

In Order class:

**@ManyToOne**

**@JoinColumn(name="CUST\_ID", nullable=false)**

public Customer getCustomer() { return customer; }

Example 2: One-to-Many association without using generics

// In Customer class:

**@OneToMany(targetEntity=com.example.Order.class, cascade=ALL,**

**mappedBy="customer")**

public Set getOrders() { return orders; }

// In Order class:

@ManyToOne

@JoinColumn(name="CUST\_ID", nullable=false)

public Customer getCustomer() { return customer; }

Example 3: Unidirectional One-to-Many association using a foreign key mapping

// In Customer class:

**@OneToMany(orphanRemoval=true)**

**@JoinColumn(name="CUST\_ID")** // join column is in table for Order

public Set<Order> getOrders() {return orders;}

## javax.persistence  ManyToOne

|  |  |
| --- | --- |
| **Optional Element Summary** | |
| [CascadeType](http://docs.oracle.com/javaee/6/api/javax/persistence/CascadeType.html" \o "enum in javax.persistence)[] | [**cascade**](http://docs.oracle.com/javaee/6/api/javax/persistence/ManyToOne.html#cascade()) :  (Optional) The operations that must be cascaded to the target of the association. |
| [FetchType](http://docs.oracle.com/javaee/6/api/javax/persistence/FetchType.html" \o "enum in javax.persistence) | [**fetch**](http://docs.oracle.com/javaee/6/api/javax/persistence/ManyToOne.html#fetch()) : (Optional) Whether the association should be lazily loaded or must be eagerly fetched. |
| boolean | [**optional**](http://docs.oracle.com/javaee/6/api/javax/persistence/ManyToOne.html#optional()) : (Optional) Whether the association is optional. |
| java.lang.Class | [**targetEntity**](http://docs.oracle.com/javaee/6/api/javax/persistence/ManyToOne.html#targetEntity()) : (Optional) The entity class that is the target of the association. |

 Defines a single-valued association to another entity class that has many-to-one multiplicity. It is not normally necessary to specify the target entity explicitly since it can usually be inferred from the type of the object being referenced. If the relationship is bidirectional, the non-owning OneToMany entity side must used the mappedBy element to specify the relationship field or property of the entity that is the owner of the relationship.

The ManyToOne annotation may be used within an embeddable class to specify a relationship from the embeddable class to an entity class. If the relationship is bidirectional, the non-owning OneToMany entity side must use the mappedBy element of the OneToMany annotation to specify the relationship field or property of the embeddable field or property on the owning side of the relationship. The dot (".") notation syntax must be used in the mappedBy element to indicate the relationship attribute within the embedded attribute. The value of each identifier used with the dot notation is the name of the respective embedded field or property.

Example 1:

**@ManyToOne(optional=false)**

**@JoinColumn(name="CUST\_ID", nullable=false, updatable=false)**

public Customer getCustomer() { return customer; }

Example 2:

@Entity

public class Employee {

@Id int id;

@Embedded JobInfo jobInfo;

...

}

@Embeddable

public class JobInfo {

String jobDescription;

**@ManyToOne ProgramManager pm; // Bidirectional**

}

@Entity

public class ProgramManager {

@Id int id;

**@OneToMany(mappedBy="jobInfo.pm")**

Collection<Employee> manages;

}

## javax.persistence  ManyToMany

|  |  |
| --- | --- |
| **Optional Element Summary** | |
| [CascadeType](http://docs.oracle.com/javaee/6/api/javax/persistence/CascadeType.html" \o "enum in javax.persistence)[] | [**cascade**](http://docs.oracle.com/javaee/6/api/javax/persistence/ManyToMany.html#cascade()) : (Optional) The operations that must be cascaded to the target of the association. |
| [FetchType](http://docs.oracle.com/javaee/6/api/javax/persistence/FetchType.html" \o "enum in javax.persistence) | [**fetch**](http://docs.oracle.com/javaee/6/api/javax/persistence/ManyToMany.html#fetch()) : (Optional) Whether the association should be lazily loaded or must be eagerly fetched. |
| java.lang.String | [**mappedBy**](http://docs.oracle.com/javaee/6/api/javax/persistence/ManyToMany.html#mappedBy()) : The field that owns the relationship. |
| java.lang.Class | [**targetEntity**](http://docs.oracle.com/javaee/6/api/javax/persistence/ManyToMany.html#targetEntity()) : (Optional) The entity class that is the target of the association. |

 Defines a many-valued association with many-to-many multiplicity.

Every many-to-many association has two sides, the owning side and the non-owning, or inverse, side. The join table is specified on the owning side. If the association is bidirectional, either side may be designated as the owning side. If the relationship is bidirectional, the non-owning side must use the mappedBy element of the ManyToMany annotation to specify the relationship field or property of the owning side.

The join table for the relationship, if not defaulted, is specified on the owning side.

The ManyToMany annotation may be used within an embeddable class contained within an entity class to specify a relationship to a collection of entities. If the relationship is bidirectional and the entity containing the embeddable class is the owner of the relationship, the non-owning side must use the mappedBy element of the ManyToMany annotation to specify the relationship field or property of the embeddable class. The dot (".") notation syntax must be used in the mappedBy element to indicate the relationship attribute within the embedded attribute. The value of each identifier used with the dot notation is the name of the respective embedded field or property.

Example 1:

// In Customer class:

**@ManyToMany**

**@JoinTable(name="CUST\_PHONES")**

public Set<PhoneNumber> getPhones() { return phones; }

// In PhoneNumber class:

**@ManyToMany(mappedBy="phones")**

public Set<Customer> getCustomers() { return customers; }

Example 2:

// In Customer class:

**@ManyToMany(targetEntity=com.example.PhoneNumber.class)**

public Set getPhones() { return phones; }

// In PhoneNumber class:

**@ManyToMany(targetEntity=com.example.Customer.class, mappedBy="phones")**

public Set getCustomers() { return customers; }

Example 3:

// In Customer class:

**@ManyToMany**

**@JoinTable(name="CUST\_PHONE",**

**joinColumns=**

**@JoinColumn(name="CUST\_ID", referencedColumnName="ID"),**

**inverseJoinColumns=**

**@JoinColumn(name="PHONE\_ID", referencedColumnName="ID")**

**)**

public Set<PhoneNumber> getPhones() { return phones; }

// In PhoneNumberClass:

@ManyToMany(mappedBy="phones")

public Set<Customer> getCustomers() { return customers; }

## javax.persistence  Enum CascadeType

|  |
| --- |
| [**ALL**](http://docs.oracle.com/javaee/6/api/javax/persistence/CascadeType.html#ALL) : Cascade all operations |
| [**DETACH**](http://docs.oracle.com/javaee/6/api/javax/persistence/CascadeType.html#DETACH) : Cascade detach operation |
| [**MERGE**](http://docs.oracle.com/javaee/6/api/javax/persistence/CascadeType.html#MERGE) :  Cascade merge operation |
| [**PERSIST**](http://docs.oracle.com/javaee/6/api/javax/persistence/CascadeType.html#PERSIST) : Cascade persist operation |
| [**REFRESH**](http://docs.oracle.com/javaee/6/api/javax/persistence/CascadeType.html#REFRESH) : Cascade refresh operation |
| [**REMOVE**](http://docs.oracle.com/javaee/6/api/javax/persistence/CascadeType.html#REMOVE) : Cascade remove operation |

Defines the set of cascadable operations that are propagated to the associated entity. The value cascade=ALL is equivalent to cascade={PERSIST, MERGE, REMOVE, REFRESH, DETACH}.

## javax.persistence  Enum FetchType

Defines strategies for fetching data from the database. The EAGER strategy is a requirement on the persistence provider runtime that data must be eagerly fetched. The LAZY strategy is a hint to the persistence provider runtime that data should be fetched lazily when it is first accessed. The implementation is permitted to eagerly fetch data for which the LAZY strategy hint has been specified.

Example:

@Basic(fetch=LAZY)

protected String getName() { return name; }

|  |
| --- |
| [**EAGER**](http://docs.oracle.com/javaee/6/api/javax/persistence/FetchType.html#EAGER) : Defines that data must be eagerly fetched. |
| [**LAZY**](http://docs.oracle.com/javaee/6/api/javax/persistence/FetchType.html#LAZY) :  Defines that data can be lazily fetched. |

## javax.persistence  Annotation Type GeneratedValue

|  |  |
| --- | --- |
| **Optional Element Summary** | |
| java.lang.String | [**generator**](http://docs.oracle.com/javaee/6/api/javax/persistence/GeneratedValue.html#generator()) : (Optional) The name of the primary key generator to use as specified in the SequenceGenerator or TableGenerator annotation. |
| [GenerationType](http://docs.oracle.com/javaee/6/api/javax/persistence/GenerationType.html" \o "enum in javax.persistence) | [**strategy**](http://docs.oracle.com/javaee/6/api/javax/persistence/GeneratedValue.html#strategy()) : (Optional) The primary key generation strategy that the persistence provider must use to generate the annotated entity primary key. |

Provides for the specification of generation strategies for the values of primary keys.

The GeneratedValue annotation may be applied to a primary key property or field of an entity or mapped superclass in conjunction with the Id annotation. The use of the GeneratedValue annotation is only required to be supported for simple primary keys. Use of the GeneratedValue annotation is not supported for derived primary keys.

Example 1:

@Id

**@GeneratedValue(strategy=SEQUENCE, generator="CUST\_SEQ")**

@Column(name="CUST\_ID")

public Long getId() { return id; }

Example 2:

@Id

**@GeneratedValue(strategy=TABLE, generator="CUST\_GEN")**

@Column(name="CUST\_ID")

Long id;

## javax.persistence  Enum GenerationType

|  |
| --- |
| [**AUTO**](http://docs.oracle.com/javaee/6/api/javax/persistence/GenerationType.html#AUTO) : Indicates that the persistence provider should pick an appropriate strategy for the particular database. |
| [**IDENTITY**](http://docs.oracle.com/javaee/6/api/javax/persistence/GenerationType.html#IDENTITY) : Indicates that the persistence provider must assign primary keys for the entity using a database identity column. |
| [**SEQUENCE**](http://docs.oracle.com/javaee/6/api/javax/persistence/GenerationType.html#SEQUENCE) : Indicates that the persistence provider must assign primary keys for the entity using a database sequence. |
| [**TABLE**](http://docs.oracle.com/javaee/6/api/javax/persistence/GenerationType.html#TABLE) : Indicates that the persistence provider must assign primary keys for the entity using an underlying database table to ensure uniqueness. |

Defines the types of primary key generation strategies.

Transient, Persistent and Detached Objects in Hibernate



**Transient State:**  
A New instance of  a persistent class which is not associated with a ***Session***, has no representation in the ***database***and no identifier value is considered ***transient*** by Hibernate:

1. UserDetail user = **new** UserDetail();
2. user.setUserName("Dinesh Rajput");
3. // user is in a transient state

**Persistent State:**  
A persistent instance has a representation in the ***database***, an identifier value and is associated with a ***Session***. You can make a transient instance persistent by associating it with a ***Session***:

1. Long id = (Long) session.save(user);
2. // user is now in a persistent state

**Detached State:**  
Now, if we close the ***Hibernate Session***, the ***persistent*** instance will become a ***detached*** instance: it isn't attached to a ***Session*** anymore (but can still be modified and reattached to a new Session later though).

1. session.close();
2. //user in detached state

**Difference between Transient and Detached States:**  
Transient objects do not have association with the databases and session objects. They are simple objects and not persisted to the database. Once the last reference is lost, that means the object itself is lost. And of course , garbage collected. The commits and rollbacks will have no effects on these objects. They can become into persistent objects through the save method calls of Session object.  
  
The detached object have corresponding entries in the database. These are persistent and not connected to the Session object.

A new instance of a a persistent class which is not associated with a Session, has no representation in the database and no identifier value is considered ***transient*** by Hibernate:

Person person = new Person();

person.setName("Foobar");

// person is in a transient state

A ***persistent*** instance has a representation in the database, an identifier value and is associated with a Session. You can make a transient instance ***persistent*** by associating it with a Session:

Long id = (Long) session.save(person);

// person is now in a persistent state

Now, if we close the Hibernate Session, the persistent instance will become a ***detached*** instance: it isn't attached to a Session anymore (but can still be modified and reattached to a new Sessionlater though).

# **Hibernate Converting Object From Detached to Persistent state**

import org.hibernate.\*;

import org.hibernate.cfg.\*;

public class ClientLogicProgram {

    public static void main(String... args) {

        Configuration cfg = new Configuration();

        cfg.configure("hibernate.cfg.xml");

        SessionFactory factory = cfg.buildSessionFactory();

        Session session1 = factory.openSession();

         Product p=null;          //Transient state..

         Object o=session1.get(Product.class, new Integer(1001));

         p=(Product)o;           //now p is in Persistent state..

        session1.close();

        p.setPrice(36000);            // p is in Detached state

        Session session2=factory.openSession();

         Transaction tx=session2.beginTransaction();

            session2.update(p);      // now p reached to Persistent state

         tx.commit();

        session2.close();

        factory.close();

}

}

How to Save data in Oracle data using sequence using Hibernate

# Oracle Query

drop sequence userInfoSeq;

create sequence userInfoSeq START WITH 5 INCREMENT BY 5;

drop table userInfo;

create table userInfo(

id number(3) PRIMARY KEY,

name varchar2(30)

);

INSERT INTO userInfo VALUES (userInfoSeq.NEXTVAL, 'John Abraham');

INSERT INTO userInfo VALUES (userInfoSeq.NEXTVAL, 'Vidya Balan');

select \* from userInfo;

# Maven(pom.xml)

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>saveDataInOracle1</groupId>

<artifactId>saveDataInOracle1</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>saveDataInOracle1</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<spring.version>4.1.2.RELEASE</spring.version>

<spring.security.version>3.2.3.RELEASE</spring.security.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

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

</dependency>

<!-- C3P0 library -->

<dependency>

<groupId>com.mchange</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.5</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>4.3.7.Final</version>

</dependency>

<dependency>

<groupId>ojdbc6</groupId>

<artifactId>ojdbc6</artifactId>

<version>6</version>

<scope>system</scope>

<systemPath>${basedir}/lib/ojdbc6.jar</systemPath>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.3</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

# Spring-Hibernate Configuration (app-context.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"* xmlns:jee=*"http://www.springframework.org/schema/jee"*

xmlns:tx=*"http://www.springframework.org/schema/tx"* xmlns:jdbc=*"http://www.springframework.org/schema/jdbc"*

xmlns:osgi=*"http://www.springframework.org/schema/osgi"* xmlns:security=*"http://www.springframework.org/schema/security"*

xsi:schemaLocation=*"http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd*

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

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

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

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

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

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

*http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-3.0.3.xsd"*>

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

<bean id=*"dataSourceInternal"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*

destroy-method=*"close"*>

**<property name=*"driverClass"* value=*"oracle.jdbc.driver.OracleDriver"* />**

**<property name=*"jdbcUrl"* value=*"jdbc:oracle:thin:@localhost:1521:xe"* />**

**<property name=*"user"* value=*"test"* />**

**<property name=*"password"* value=*"test1"* />**

<!-- these are C3P0 properties -->

<property name=*"acquireIncrement"* value=*"5"* />

<property name=*"initialPoolSize"* value=*"5"* />

<property name=*"minPoolSize"* value=*"5"* />

<property name=*"maxPoolSize"* value=*"20"* />

</bean>

<!-- This is the lazy DataSource proxy that interacts with the target DataSource once a real statement is sent to the database. Users use this DataSource to set up their Hibernate session factory, which in turn forces the Hibernate second-level cache and also everything that interacts with that Hibernate session factory to use it. -->

<bean id=*"dataSource"* class=*"org.springframework.jdbc.datasource.LazyConnectionDataSourceProxy"*>

<property name=*"targetDataSource"*><ref bean=*"dataSourceInternal"* /></property>

</bean>

<!-- <bean id="hibSessionFactory" class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean"> -->

<bean id=*"hibSessionFactory"* class=*"org.springframework.orm.hibernate4.LocalSessionFactoryBean"*>

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

<!--<property name="hibernateProperties"> <value> hibernate.show\_sql=true

</value> </property> -->

<property name=*"hibernateProperties"*>

<value>

hibernate.id.new\_generator\_mappings=true,

<!-- hibernate.show\_sql=true -->

<!-- hibernate.hbm2ddl.auto=update -->

<!-- hibernate.current\_session\_context\_class=thread -->

</value>

</property>

<property name=*"annotatedClasses"*>

<list>

<value>com.ddlab.rnd.hibernate.UserInfo</value>

</list>

</property>

</bean>

</beans>

# Java Files

## UserInfo.java

**package** com.ddlab.rnd.hibernate;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.SequenceGenerator;

**import** javax.persistence.Table;

@Entity

@Table(name = "userInfo")

**public** **class** UserInfo {

**@Id**

**@GeneratedValue(strategy = GenerationType.*AUTO*, generator = "my\_seq\_gen")**

**@SequenceGenerator(name = "my\_seq\_gen", sequenceName = "userInfoSeq")**

**// sequenceName is the actual oracle sequence**

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

@Column(name = "name")

**private** String name;

**public** UserInfo(String name) {

**this**.name = name;

}

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

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

## TestSaveInOracle.java

**package** com.ddlab.rnd.hibernate;

**import** org.hibernate.HibernateException;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** TestSaveInOracle {

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

ApplicationContext context = **new** ClassPathXmlApplicationContext("app-context.xml");

SessionFactory sessionFactory = (SessionFactory) context.getBean("hibSessionFactory");

Session session = sessionFactory.openSession();

Transaction transaction = **null**;

**try** {

transaction = session.beginTransaction();

UserInfo info = **new** UserInfo("Deb");

session.save(info);

transaction.commit();

} **catch** (HibernateException e) {

**if** (transaction != **null**)

transaction.rollback();

e.printStackTrace();

} **finally** {

session.close();

}

System.***out***.println("successfully saved into database");

}

}

## save, persist, update, merge, saveOrUpdate

Both save() and persist() are used to **insert** a **new** entity in the database. You're calling them on entities that already exist in the database. So they do nothing.

The main difference between them is that save() is Hibernate-proprietary, whereas persist() is a standard JPA method. Additionally, save() is guaranteed to assign and return an ID for the entity, whereas persist() is not.

update() is used to attach a detached entity to the session.

saveOrUpdate() is used to either save or update an entity depending on the state (new or detached) of the entity.

Merge Does Following

Merge has intelligence. It has lot of pre-checks before it go actual merge(if required)

1. if Object is transient, It simply fires INSERT query makes object persistent(attached to session)
2. if Object is detached, fires select query to check whether data modified or not if modified, fires UPDATE query otherwise just ignore merge task.

where as session.update

1. throws exception if object is transient.
2. if Object is detached, it simply fires UPDATE query irrespective of data changes to object.

session.merge is expensive than update