Object States in Hibernate –

1. Transient,
2. Persistent and
3. Detached

Object states in Hibernate plays a vital role in the execution of code in an application. Hibernate has provided three different states for an object of a **pojo (Plain Old Java Object** ) class. These three states are also called as life cycle states of an object.

**1. Transient Object State:**

An object which is not associated with hibernate session and does not represent a row in the database is considered as transient. It will be garbage collected if no other object refers to it.

An object that is created for the first time using the new() operator is in transient state. When the object is in transient sate then it will not contain any identifier (primary key value). You have to use save, persist or saveOrUpdate methods to persist the transient object.

Employee emp = **new** Employee();

emp.setName("Ravi Raj");

// here emp object is in a transient state

**2. Persistent Object State:**

An object that is associated with the hibernate session is called as Persistent object. When the object is in persistent state, then it represent one row of the database and consists of an identifier value. You can make a transient instance persistent by associating it with a Session.

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

//emp object is now in a persistent state

**3. Detached Object State:**

Object which is just removed from hibernate session is called as detached object. The state of the detached object is called as detached state.

When the object is in detached sate then it contain identity but you can’t do persistence operation with that identity.

Any changes made to the detached objects are not saved to the database. The detached object can be reattached to the new session and save to the database using update, saveOrUpdate and merge methods.

Example

**package** com.visix.mnrao.bean;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.AnnotationConfiguration;

**import** com.jwt.hibernate.Student;

**public** **class** ObjectStatesDemo {

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

// Transient object state

Student student = **new** Student();

student.setId(101);

student.setName("MNRAO");

student.setRoll("10");

student.setDegree("B.Tech");

student.setPhone("9999");

// Transient object state

Session session = **new** AnnotationConfiguration().configure()

.buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

// Persistent object state

session.save(student);

t.commit();

// Persistent object state

session.close();

// Detached object state

}

}

**Hibernate Example With Annotation :**

This example is the same as the first example except that it uses annotations. In our first example we created *.hbm.xml* file for database and pojo class mapping, here there is no need to create hbm files, instead we will use annotations to do the object relational mapping.

**Steps to create the hibernate application with Annotation**

1. Add the jar file
2. Create the Persistent class
3. Add mapping of Persistent class in configuration file
4. Create the class that retrieves or stores the persistent object

Step 1: Add the jar file for annotation

In addition to the already existing jar files you need to add the following jar files to the classpath

1.hibernate-commons-annotations.jar

2.ejb3-persistence.jar

3.hibernate-annotations.jar –

Following jar files are enough for getting annotation support in hibernate.

1.antlr-2.7.6.jar

2.commons-collections-3.1.jar

3.dom4j-1.6.1.jar

4.hibernate-commons-annotations-3.2.0.Final.jar

5.hibernate-core-3.6.7.Final.jar

6.hibernate-jpa-2.0-api-1.0.1.Final.jar

7.javassist.jar

8.jms-1.1.jar 9.jsr250-api-1.0.jar

10.jta-1.1.jar

11.log4j-1.2.16.jar

12.mysql connector.jar

13.slf4j-api-1.6.1.jar

14.slf4j-log4j12-1.6.1.jar

**Step 2 : Create the Persistent class**

**package** com.visix.mnrao.bean;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "STUDENT")

**public** **class** Student {

**private** **long** stusid;

**private** String name;

**private** String degree;

**private** String roll;

**private** String phone;

@Id

**public** **long** getStusid() {

**return** stusid;

}

**public** **void** setStusid(**long** stusid) {

**this**.stusid = stusid;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** String getDegree() {

**return** degree;

}

**public** **void** setDegree(String degree) {

**this**.degree = degree;

}

**public** String getRoll() {

**return** roll;

}

**public** **void** setRoll(String roll) {

**this**.roll = roll;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

}

In the above class, we are creating the same persistent class which we have created in the previous example. But here, we are using annotation.  
  
**@Entity** annotation marks this class as an entity.  
  
**@Table** annotation specifies the table name where data of this entity is to be persisted. If you are not using @Table annotation in Entity class, hibernate will use the class name as the table name by default.  
  
**@Id** annotation marks the identifier for this entity.  
  
**@Column** annotation specifies the details of the column for this property or field. If @Column annotation is not specified, property name will be used as the column name bydefault.

**Step 3 : Mapping of Persistent class in configuration file**

The next change you need to do here is, instead of adding the .hbm.xml file to the hibernate.cfg.xml file, we add the fully qualified name of the annotated class to the mapping element.

**hibernate.cfg.xml**

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=*"hibernate.dialect"*>org.hibernate.dialect.Oracle9Dialect</property>

<property name=*"hibernate.connection.driver\_class"*>oracle.jdbc.driver.OracleDriver</property>

<property name=*"hibernate.connection.url"*>jdbc:oracle:thin:@localhost:1521:ORCL</property>

<property name=*"hibernate.connection.username"*>scott</property>

<property name=*"hibernate.connection.password"*>tiger</property>

<property name=*"show\_sql"*>true</property>

<property name=*"format\_sql"*>true</property>

<property name=*"hbm2ddl.auto"*>create </property>

<mapping class=*"com.visix.mnrao.bean.Student"* />

</session-factory>

</hibernate-configuration>

**Step 4 : Create Test class that retrieves or stores the persistent object**

**package** com.visix.mnrao.test;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.AnnotationConfiguration;

**import** com.visix.mnrao.bean.Student;

**public** **class** Test {

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

Session session = **new** AnnotationConfiguration().configure(“**hibernate.cfg.xml**”)

.buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

Student student1 = **new** Student();

student1.setStusid(1);

student1.setName("MNRAO");

student1.setRoll("101");

student1.setDegree("B.Tech");

student1.setPhone("99999");

Student student2 = **new** Student();

student2.setStusid(2);

student2.setName("Ravi");

student2.setRoll("102");

student2.setDegree("B.Tech");

student2.setPhone("934499");

session.persist(student1);

session.persist(student2);

t.commit();

session.close();

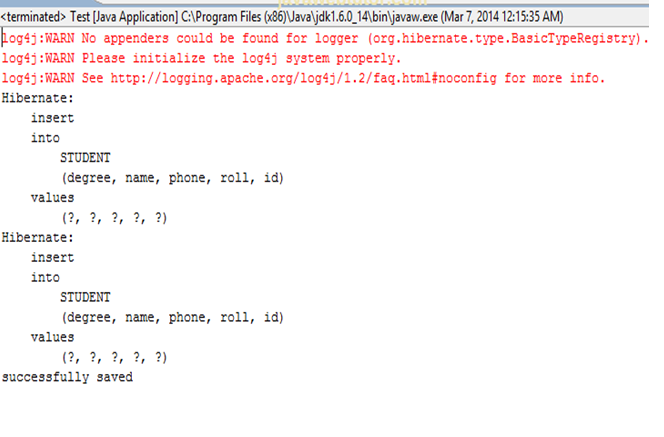
System.***out***.println("successfully saved");

}

}

**Output :**

If data is inserted into DB you can see generated sql in console as given below



**Difference between hibernate save(), saveOrUpdate() and persist() methods :**

Session interface in Hibernate provides a couple of methods to move an object from new or transient state to persistent state e.g. save(), saveOrUpdate() and persist() are used to store an object into the database, but there are some significant differences between them.

save() 🡪 generates a new identifier and INSERT record into database

save() method will fail if the primary key is already persistent i.e. object already exists in the database. This is why, you should only call save() with an absolutely new object which doesn't have any database identifier. Calling save() method on the detached object will throw exception.

saveOrUpdate 🡪 can either INSERT or UPDATE based upon existence of record.

The Session.save() method does an INSERT to store the object into the database and return the identifier generated by the database.

On the other hand, saveOrUpdate() can do INSERT or UPDATE depending upon whether object exists in database or not.

saveOrUpdate is more flexible in terms of use but it involves an extra processing to find out whether record already exists in table or not. saveOrUpdate does a select first to determine if it needs to do an insert or an update. It will insert data if primary key not exist otherwise it will update data.

Another key difference between save() and saveOrUpdate() method is that,

save() method is used to bring a transient object to persistent state

**(but)**

saveOurUpdate() can bring both transient (new) and detached (existing) object into persistent state. It is often used to re-attach a detached object into Session.

**Conclusion:**

save() method saves records into database by INSERT SQL query, Generates a new identifier and return the Serializable identifier back.

On the other hand saveOrUpdate() method either INSERT or UPDATE based upon existence of object in database. If persistence object already exists in database then UPDATE SQL will execute and if there is no corresponding object in database than INSERT will run.

**Example:**

Below is example of both save and saveOrUpdate() method. You can easily identify the difference in generated SQL statement in the console.

**Entity Class**

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "STUDENT")

**public** **class** Student {

**private** **long** id;

**private** String name;

**private** String degree;

**private** String roll;

**private** String phone;

@Id

**public** **long** getId() {

**return** id;

}

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

**this**.id = id;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** String getDegree() {

**return** degree;

}

**public** **void** setDegree(String degree) {

**this**.degree = degree;

}

**public** String getRoll() {

**return** roll;

}

**public** **void** setRoll(String roll) {

**this**.roll = roll;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

}

**Configuration File:**

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*"hibernate.connection.url"*>jdbc:mysql://localhost:3306/hibernate</property>

<property name=*"hibernate.connection.username"*>root</property>

<property name=*"hibernate.connection.password"*>mnrao</property>

<property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"show\_sql"*>true</property>

<property name=*"format\_sql"*>true</property>

<property name=*"hbm2ddl.auto"*>create </property>

<mapping class=*"com.visix.mnrai.bean.Student"* />

</session-factory>

</hibernate-configuration>

**Using save() method**

**package** com.visix.mnrao.student;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.AnnotationConfiguration;

**import** com.visix.mnrao.bean.Student;

**public** **class** SaveTest {

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

Session session = **new** AnnotationConfiguration().configure()

.buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

Student student = **new** Student();

student.setId(1);

student.setName("mnrao");

student.setRoll("101");

student.setDegree("B.E");

student.setPhone("99999");

Long studentId = (Long) session.save(student);

t.commit();

session.close();

System.***out***.println(" Identifier : " + studentId);

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

}

}

Output:

log4j:WARN No appenders could be found for logger (org.hibernate.type.BasicTypeRegistry). log4j:WARN Please initialize the log4j system properly.

log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

Hibernate:

insert

into

STUDENT

(degree, name, phone, roll, id)

values (?, ?, ?, ?, ?)

Identifier : 1

successfully saved.

**Look at the above output carefully, save() method first generates identifier and then inserts data into database. We printed the value of the identifier after saving the object. You can clearly see the insert query in the console output**

**Using saveOrUpdate() method:**

**package** com.visix.mnrao.student;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.AnnotationConfiguration;

**public** **class** SaveOrUpdateTest {

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

Session session = **new** AnnotationConfiguration().configure()

.buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

Student student = **new** Student();

student.setId(101);

student.setName("MNRAO");

student.setRoll("10");

student.setDegree("B.Tech");

student.setPhone("8888");

session.saveOrUpdate(student);

t.commit();

session.close();

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

}

}

Output:

log4j:WARN No appenders could be found for logger (org.hibernate.type.BasicTypeRegistry). log4j:WARN Please initialize the log4j system properly.

log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

Hibernate:

select

student\_.id,

student\_.degree as degree0\_,

student\_.name as name0\_,

student\_.phone as phone0\_,

student\_.roll as roll0\_

from STUDENT student\_

where student\_.id=?

Hibernate:

insert

into

STUDENT

(degree, name, phone, roll, id)

values (?, ?, ?, ?, ?)

successfully saved

The above output shows that the entity with specified identifier does not exist in database by running select query. So it applied insert query to persist data.

Now check the output for such situation where entity for the given identifier is already exist in database. In the above example just change the name and phone number. Modified code is given below.

**package** com.visix.mnrao.student;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.AnnotationConfiguration;

**public** **class** SaveOrUpdateTest {

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

Session session = **new** AnnotationConfiguration().configure()

.buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

Student student = **new** Student();

student.setId(101);

student.setName("MNRAO");

student.setRoll("10");

student.setDegree("B.Tech");

student.setPhone("9999");

session.saveOrUpdate(student);

t.commit();

session.close();

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

}

}

**Output:**

log4j:WARN No appenders could be found for logger (org.hibernate.type.BasicTypeRegistry). log4j:WARN Please initialize the log4j system properly.

log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

Hibernate:

select

student\_.id,

student\_.degree as degree0\_,

student\_.name as name0\_,

student\_.phone as phone0\_,

student\_.roll as roll0\_

from

STUDENT student\_

where

student\_.id=?

Hibernate:

update STUDENT

set

degree=?,

name=?,

phone=?,

roll=?

where id=?

successfully saved

In the above output, first select query runs to check existence of data for given identifier and since data is there, so update query runs to update data.

**Difference between save() and persist() methods**

* Main difference of save() and persist() is return type of the save() method is java.io.Serializable it returns the generated identity value whereas the return type of persist method is void i.e, it will not return any value. persist() is similar to Session.save() i.e. it is used to move a transient object to the persistent state by storing it into the database but it doesn't return the database identifier.

* Another key difference is persist() method can be used only within the boundary of a transaction whereas save method can be used inside or outside the transaction boundaries. persist() method guarantees that it will not execute an INSERT statement if it is called outside of transaction boundaries. But in the case of save() INSERT happens immediately, no matter if you are inside or outside of a transaction. This is not good in a long-running conversation with an extended Session/persistence context.