H2K

H2K Infosys, LLC

Dream, Strive & Achieve Victory

Live - Green - Everywhere

www.H2KINFOSYS.com

USA: 770-777-1269

Training@H2KInfosys.com

UK : (020) 3371 7615

HIBERNATE

Part II

So far we have seen very basic O/R mapping using hibernate but there are three most important mapping topics which we have to learn in detail. These are the mapping of collections, the mapping of associations between entity classes and Component Mappings

Collections Mappings:

If an entity or class has collection of values for a particular variable, then we can map those values using any one of the collection interfaces available in java.

Collection type	Mapping	initialized with
java.util.Set	<set> element	java.util.HashSet
Java.util.SortedSet	<set> element	java.util.TreeSet
Java.util.List	<list> element	java.util.ArrayList
Java.util.Collection	<bag> or <ibag> element	java.util.ArrayList
Java.util.Map	<map> element	java.util.HashMap
Java.util.SortedMap	<map> element	java.util.TreeMap

Set:

```
<set name="certificates" cascade="all">
         <key column="employee_id"/>
         <one-to-many class="Certificate"/>
</set>
```

List:

```
<list name="certificates" cascade="all">
         <key column="employee_id"/>
         <list-index column="idx"/>
         <one-to-many class="Certificate"/>
</list>
```

Bag:

```
<bag name="certificates" cascade="all">
         <key column="employee_id"/>
         <one-to-many class="Certificate"/>
</bag>
```

Map:

```
<map name="certificates" cascade="all">
         <key column="employee_id"/>
         <index column="certificate_type" type="string"/>
         <one-to-many class="Certificate"/>
</map>
```

Association Mappings

The mapping of associations between entity classes and the relationships between tables is the soul of ORM.

Mapping type	Description
Many-to-One	Mapping many-to-one relationship using Hibernate
One-to-One	Mapping one-to-one relationship using Hibernate
One-to-Many	Mapping one-to-many relationship using Hibernate
Many-to-Many	Mapping many-to-many relationship using Hibernate

One to One:

```
      <one-to-one name="address" column="address"
       class="Address" not-null="true"/>
```

One to Many:

```
<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <set name="certificates" cascade="all">
         <key column="employee_id"/>
         <one-to-many class="Certificate"/>
      </set>
      <property name="firstName" column="first_name" type="string"/>
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>

   <class name="Certificate" table="CERTIFICATE">
      <meta attribute="class-description">
         This class contains the certificate records.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="name" column="certificate_name" type="string"/>
   </class>

</hibernate-mapping>
```

Many to Many:

```
<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <set name="certificates" cascade="save-update" table="EMP_CERT">
         <key column="employee_id"/>
         <many-to-many column="certificate_id" class="Certificate"/>
      </set>
      <property name="firstName" column="first_name" type="string"/>
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>

   <class name="Certificate" table="CERTIFICATE">
      <meta attribute="class-description">
         This class contains the certificate records.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="name" column="certificate_name" type="string"/>
   </class>
</hibernate-mapping>
```

Component Mappings:

It is very much possible that an Entity class can have a reference to another class as a member variable. If the referred class does not have its own life cycle and completely depends on the life cycle of the owning entity class, then the referred class hence therefore is called as the Component class.

Mapping type	Description
Component Mappings	Mapping for a class having a reference to another class as a member variable.

```
<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <component name="address" class="Address">
         <property name="street" column="street_name" type="string"/>
         <property name="city" column="city_name" type="string"/>
         <property name="state" column="state_name" type="string"/>
         <property name="zipcode" column="zipcode" type="string"/>
      </component>
      <property name="firstName" column="first_name" type="string"/>
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
   <class name="Certificate" table="CERTIFICATE">
      <meta attribute="class-description">
         This class contains the certificate records.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="name" column="certificate_name" type="string"/>
   </class>

</hibernate-mapping>
```

Hibernate Annotations

Hibernate Annotations is the powerful way to provide the metadata for the Object and Relational Table mapping. All the metadata is clubbed into the POJO java file along with the code this helps the user to understand the table structure and POJO simultaneously during the development. (You will need **hibernate-annotations.jar**, **hibernate-comons-annotations.jar** and **ejb3-persistence.jar**)

```
@Entity
@Table(name = "EMPLOYEE")
public class Employee {
   @Id @GeneratedValue
   @Column(name = "id")
   private int id;

   @Column(name = "first_name")
   private String firstName;

   @Column(name = "last_name")
   private String lastName;

   @Column(name = "salary")
   private int salary;

   public Employee() {}
         . . . // getters and setters here
}
```

Hibernate Query Language

Hibernate Query Language (HQL) is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties. HQL queries are translated by Hibernate into conventional SQL queries which in turns perform action on database.

FROM Clause

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
```

```
List results = query.list();
```

AS Clause: String hql = "**FROM Employee AS E**";

SELECT Clause: String hql = **"SELECT E.firstName FROM Employee E";**

WHERE clause: String hql = **"FROM Employee E WHERE E.id = 10";**

ORDER BY Clause:

```
String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";
```

```
String hql = "FROM Employee E WHERE E.id > 10 " +
             "ORDER BY E.firstName DESC, E.salary DESC ";
```

GROUP BY Clause

```
String hql = "SELECT SUM(E.salary), E.firtName FROM Employee E " +
             "GROUP BY E.firstName";
```

Using Named Parameters

Hibernate supports named parameters in its HQL queries.

```
String hql = "FROM Employee E WHERE E.id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("employee_id", 15334);
List results = query.list();
```

UPDATE Clause

```
String hql = "UPDATE Employee set salary = :salary " +
             "WHERE id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("salary", 1000);
query.setParameter("employee_id", 10);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

DELETE Clause

```
String hql = "DELETE FROM Employee WHERE id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("employee_id", 10);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

INSERT Clause

```
String hql = "INSERT INTO Employee(firstName, lastName, salary)" +
          "SELECT firstName, lastName, salary FROM old_employee";
Query query = session.createQuery(hql);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

Aggregate Methods

S.N.	Functions	Description
1	avg(property name)	The average of a property's value
2	count(property name or *)	The number of times a property occurs in the results
3	max(property name)	The maximum value of the property values
4	min(property name)	The minimum value of the property values
5	sum(property name)	The sum total of the property values

Pagination using Query

Method	Description
Query setFirstResult(int startPosition)	This method takes an integer that represents the first row in your result set, starting with row 0.
Query setMaxResults(int maxResult)	This method tells Hibernate to retrieve a fixed number **maxResults** of objects.

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
query.setFirstResult(1);
query.setMaxResults(10);
List results = query.list();
```

Hibernate Criteria Queries

Criteria API allow you to build up a criteria query object programmatically where you can apply filtration rules and logical conditions.

The Hibernate **Session** interface provides **createCriteria**() method which can be used to create a Criteria object that returns instances of the persistence object's class when your application executes a criteria query.

Simple criteria example which does not add any Restriction:

```
Criteria cr = session.createCriteria(Employee.class);
List results = cr.list();
```

Criteria with Restriction:

```
Criteria cr = session.createCriteria(Employee.class);
Criterion salary = Restrictions.eq("salary", 2000);
```

cr.add(salary);
List results = cr.list();

add() method takes **Criterion** object as input which is returned by method on Restriction. And you can add as many restrictions you want (applicable to your business logic). Following are the few more examples covering different scenarios and can be used as per requirement:

Restriction	Meaning
Restrictions.eq("salary", 2000)	having salary equal to 2000
Restrictions.gt("salary", 2000)	having salary greater than 2000
Restrictions.lt("salary", 2000)	having salary Less than 2000
Restrictions.like("firstName", "Be%")	having fistName starting with Be
Restrictions.ilike("firstName", "Be%")	Case sensitive restriction
Restrictions.between("salary", 1000, 2000)	Salary range between 1000 to 2000
Restrictions.isNull("city")	City is Null
Restrictions.isNotNull("City")	City is Not Null
Restrictions.isEmpty("CellNo")	CellNo is Empty
Restrictions.isNotEmpty("CellNo")	CellNo is Not Empty
Restrictions.or(crton1, crton2)	Logical OR in two criterion
Restrictions.and(crton1, crton2)	Logical AND in two criterion

Using LogicalExpression restrictions

```
Criteria cr = session.createCriteria(Employee.class);

Criterion salary = Restrictions.gt("salary", 2000);
Criterion name = Restrictions.ilike("firstNname","Be%");

// to get records matching with OR conditions
LogicalExpression orExp = Restrictions.or(salary, name);
cr.add( orExp );


// To get records matching with AND condistions
LogicalExpression andExp = Restrictions.and(salary, name);
cr.add( andExp );

List results = cr.list()
```

Pagination using Criteria:

```
Criteria cr = session.createCriteria(Employee.class);
cr.setFirstResult(1);
cr.setMaxResults(10);
List results = cr.list();
```

Method	Description

public Criteria setFirstResult(int firstResult)	This method takes an integer that represents the first row in your result set, starting with row 0.
public Criteria setMaxResults(int maxResults)	This method tells Hibernate to retrieve a fixed number **maxResults** of objects.

Sorting the Results:

The Criteria API provides the org.hibernate.criterion.**Order** class to sort your result set in either ascending or descending order, according to one of your object's properties.

```
Criteria cr = session.createCriteria(Employee.class);
// To get records having salary more than 2000
cr.add(Restrictions.gt("salary", 2000));

// To sort records in descening order
cr.addOrder(Order.desc("salary"));

// To sort records in ascending order
cr.addOrder(Order.asc("salary"));

List results = cr.list();
```

Projections & Aggregations:

The Criteria API provides the org.hibernate.criterion.**Projections** class which can be used to get average, maximum or minimum of the property values. The Projections class is similar to the **Restrictions** class in that it provides several static factory methods for obtaining **Projection** instances.

```
Criteria cr = session.createCriteria(Employee.class);

// To get sum of a property.
cr.setProjection(Projections.sum("salary"));
```

setProjection	Meaning
Projections.rowCount()	To get total row count.
Projections.avg("salary")	To get average of a property.
Projections.countDistinct("City")	To get distinct count of a property.
Projections.max("salary")	To get maximum of a property.
Projections.min("salary")	To get minimum of a property.
Projections.sum("salary")	To get sum of a property.