2. **Hibernate Architecture**

Hibernate architecture ထဲမွာ persistent object, session factory, transaction factory, connection factory, session, transaction etc လိုမ်ိဳး object ေပါင္းမ်ားစြာ ပါ၀င္ပါတယ္။

Hibernate ကို layer ၄ ခု ခြဲထားပါတယ္။

1. Java application layer
2. Hibernate framework layer
3. Backhand api layer
4. Database layer



ဒီပံုကေတာ့ hibernate ကို mapping file ရယ္ configuration file ေတြနဲ႔ ေတြ႕ၿမင္ရတဲ့ high level architecture ပံုပဲ ၿဖစ္ပါတယ္။



hibernate framework က session factory, session, transaction etc တို႔လို object မ်ားစြာကို java ရဲ႕ ရွိၿပီးသား api မ်ားၿဖစ္ၾကတဲ့ DBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface)တို႔နဲ႔ပါင္းစပ္အသံုးၿပဳတယ္။

Hibernate Architecture ရဲ႕ elements မ်ား

SessionFactory

session တစ္ခုရဲ႕ factory တစ္ခုၿဖစ္သလို ConnectionProvider ရဲ႕ client လည္းၿဖစ္တယ္။ သူက data ရဲ႕ second level cache (optional) ကို hold လုပ္ထားတယ္။ org.hibernate.SessionFactory interface က session ရဲ႕ object ကို ရရွိဖို႔ရန္ factory method ကို ထုတ္ေပးပါတယ္။

Session

session object က application နဲ႔ database ထဲမွာ သိမ္းထားတဲ့ data နဲ႔ၾကားထဲမွာ interface တစ္ခုကို ထုတ္ေပးပါတယ္။ သူက short-lived object တစ္ခုၿဖစ္ၿပီးေတာ့ JDBC object ကို wrap လုပ္ေပးပါတယ္။ session က Transaction, Query နဲ႔ Criteria တို႔ရဲ႕ factory လည္း ဟုတ္တယ္။ session က data ရဲ႕ first-level cache (mandatory) ကို hold လုပ္ထားတယ္။ org.hibernate.Session interface က object ေတြ insert, update နဲ႔ delete လုပ္ဖို႔ရန္ method မ်ားကို ထုတ္ေပးပါတယ္။ ၿပီးေတာ့ Transaction, Query နဲ႔ Criteria အတြက္လည္း factory method မ်ားကို ထုတ္ေပးပါတယ္။

Transaction

transaction object က work ရဲ႕ atomic unit ကို သတ္မွတ္ေပးတယ္။ org.hibernate.Transaction interface က transaction management အတြက္ methods မ်ားကို ထုတ္ေပးပါတယ္။

ConnectionProvider

JDBC connections ရဲ႕ factory တစ္ခုၿဖစ္တယ္။ connectionProvider က DriverManager သို႔မဟုတ္ DataSource မွ application ကို abstract လုပ္တယ္။ သူကေတာ့ optional ၿဖစ္တယ္။

TransactionFactory

Transaction ရဲ႕ factory တစ္ခုၿဖစ္တယ္။ သူကလည္း optional ၿဖစ္တယ္။

**Hibernate with Eclipse**

1. HB Using XML (main method နဲ႔ run ရမယ္)

2. HB Using Annotation (main method နဲ႔ run ရမယ္)

**1. HB Using XML**

Hibernate Example using XML in Eclipse

ေအာက္ပါအဆင့္မ်ားကို လုပ္ေဆာင္ရမွာ ၿဖစ္ပါတယ္။

1. persistence class ကို create လုပ္မယ္။
2. persistence class အတြက္ mapping file ကို create လုပ္မယ္။
3. configuration file ကို create လုပ္မယ္။
4. table ေဆာက္မယ္။
5. persistent object ကို retrieves သို႔မဟုတ္ stores လုပ္မယ့္ class တစ္ခုကို create လုပ္မယ္။
6. jar file ကို load လုပ္မယ္။
7. main method ပါတဲ့ class ကို run မယ္။

1. Create the Persistent class

simple Persistent class တစ္ခုကို create လုပ္ေတာ့မယ္ဆိုရင္ ေအာက္ပါ rules မ်ားကို လိုက္နာေဆာင္ရြက္သင့္ပါတယ္။

* **A no-arg constructor**: default constructor တစ္ခုကေတာ့ ပါကိုပါရမယ္ ဒါမွသာ hibernate က Persistent class ရဲ႕ instance ကို newInstance() method နဲ႔ ေဆာက္လို႔ရမွာ ၿဖစ္ပါတယ္။
* **identifier property တစ္ခုေပးပါ** : အဲ့တာက id လို attribute မ်ိဳးကို assign လုပ္လို႔ ပိုမိုေကာင္းမြန္ေစပါတယ္။ id ဆိုတဲ့ field ကေတာ့ db ထဲမွာ primary key ပံုစံနဲ႔ ရွိေနမွာ ၿဖစ္ပါတယ္။
* **getter and setter methods မ်ားထည့္သြင္းပါ** : Hibernate က default အားၿဖင့္ getter setter name မ်ားကို အသိအမွတ္ၿပဳထားေပးၿပီးသား။
* **non-final class ကို ဦးစားေပးပါ** : hibernate က proxies ရဲ႕ concept ကို အသံုးၿပဳတယ္ ဆိုလိုသည္မွာ အဲ့တာက persistence class အေပၚမွာ depends ၿဖစ္ေနပါတယ္။ application programmer ေတြကေတာ့ lazy association fetching အတြက္ proxies မ်ားကို သံုးဖို႔ မၿဖစ္ႏိုင္ပါဘူး။

Employee.java

**public** **class** Employee {

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

**private** String firstName, lastName;

**getter//setter**

2. Persistent class အတြက္ mapping file ကို create လုပ္ၿခင္း

mapping file ရဲ႕ နာမည္ကို အစဥ္အလာအတိုင္းပဲ ေပးသင့္ပါတယ္။ **class\_name.hbm.xml** အဲ့လို format အတိုင္းပဲ ၿဖစ္သင့္တယ္။

* **hibernate-mapping** : mapping file ရဲ႕ root element တစ္ခုၿဖစ္တယ္။ သူ႕ထဲမွာ mapping file အားလံုးပါ၀င္ပါတယ္။
* **class** : hibernate-mapping element ရဲ႕ sub element ၿဖစ္တယ္။ သူက Persistent class ကို သတ္မွတ္ေပးပါတယ္။
* **generator** : id ရဲ႕ sub element တစ္ခုၿဖစ္တယ္။ သူ႕ကို primary key generate လုပ္ဖို႔ရန္ အသံုးၿပဳတယ္။ သူထဲမွာ assigned, increment, hilo, sequence, native etc လိုမ်ိဳး generator class မ်ားစြာ ပါ၀င္ႏိုင္ပါတယ္။
* **property** : class ရဲ႕ sub element တစ္ခုၿဖစ္ပါတယ္။ သူက persistence class ရဲ႕ property name ကို သတ္မွတ္ေပးပါတယ္။

သူ႕ကို eclipse မွာ ယူမယ္ဆိုရင္ xml file type ကို မယူပါ။ file ဆိုတဲ့ type ကို ယူၿပီးမွာ name ေပးတဲ့အခါမွ format ၿဖစ္တဲ့ xml ကို ထည့္ရိုက္ေပးလိုက္တာ ၿဖစ္ပါတယ္။

ၿပီးေတာ့ သူ႕ကို local package ၿဖစ္တဲ့ hb ရဲ႕ အၿပင္ဘက္မွာ ေဆာက္ေပးပါ။ ရွင္းရွင္းေၿပာေၿပာရရင္ java file ေတြက local package အတြင္းထဲမွာ ၿဖစ္ၿပီးေတာ့ xml format မ်ားကေတာ့ local package ရဲ႕ အၿပင္ဘက္မွာ ၿဖစ္ရမွာ ၿဖစ္ပါတယ္။

employee.hbm.xml

<hibernate-mapping>

<class name=*"hibernate\_with\_eclipse.hb\_with\_xml.hb.Employee"* table=*"emp1000"*>

<id name=*"id"*>

<generator class=*"assigned"*></generator>

</id>

<property name=*"firstName"*></property>

<property name=*"lastName"*></property>

</class>

</hibernate-mapping>

3. Configuration file ကို create လုပ္ၿခင္း

mapping file နဲ႔ database နဲ႔ပတ္သက္တဲ့ information မ်ားပါ၀င္ပါတယ္။ ဥပမာ connection\_url, driver\_class, username, password etc. တို႔ၿဖစ္ၾကပါတယ္။database ထဲမွာ table ကို automatically create လုပ္ဖို႔ရန္ hbm2ddl.auto property ကို အသံုးၿပဳပါတယ္။ Dialect class ရဲ႕ detail အေၾကာင္းကိုေတာ့ next topic မွာ ေၿပာပါမယ္။ သူ႕နာမည္ကေတာ့ hibernate.cfg.xml ၿဖစ္ရပါမယ္။သူ႕ကိုေတာ့ soucrce folder တစ္ခုေဆာက္ၿပီး ထည့္လိုက္ပါတယ္။ xml နဲ႔ေရးထားတဲ့အတြက္ေၾကာင့္ mapping resource ၿဖစ္ေနတာပါ။

hibernate.cfg.xml

<hibernate-configuration>

<session-factory>

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

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

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

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

<property name=*"connection.password"*>root</property>

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

<mapping resource=*"hibernate\_with\_eclipse/hb\_with\_xml/employee.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

5. db မွာ သက္ဆိုင္ရာ table name အတိုင္း ေဆာက္မယ္။

CREATE TABLE `hibernate`.`emp1000` (

`id` INT NOT NULL,

`firstName` VARCHAR(45) NULL,

`secondName` VARCHAR(45) NULL,

PRIMARY KEY (`id`));

6. class တစ္ခုေဆာက္ၿပီး အဲ့ class ထဲကို objects မ်ား retrieves သို႔မဟုတ္ stores လုပ္ၿခင္း

StoreData.java (main method ပါ၀င္ပါတယ္)

**public** **class** StoreData {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Employee e1 = **new** Employee();

e1.setId(1);

e1.setFirstName("Gaurav");

e1.setLastName("Chawla");

session.save(e1);

t.commit();

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

factory.close();

session.close();}}

7. hibernate အတြက္ လိုအပ္ေသာ jar file မ်ားကို download ဆြဲၿခင္း နဲ႔ eclipse ထဲထည့္သြင္းၿခင္း

လိုအပ္ေသာ jar file မ်ားကို download ဆြဲၿပီး resource folder ေအာက္မွာ jar ဆိုတဲ့ folder တစ္ခုေဆာက္ၿပီး ထည့္ေပးလိုက္တယ္။ ၿပီးရင္ build path မွာ add external jar ကို click ႏွိပ္လိုက္ရင္ folder dialog တက္လာတယ္ အဲ့တာဆိုရင္ ခုနက ကိုယ္ resource ေအာက္မွာထည့္ထားတဲ့ folder path လမ္းေၾကာင္းေအာက္က jar file ေတြကုို ေရြးေပးလိုက္ရံုပဲ။

**2. HB using Annotation**

hibernate application ကို xml နဲ႔ create လုပ္လို႔ရလို annotation နဲ႔လည္း create လုပ္လို႔ရပါတယ္။ ဥပမာ @Entity, @Id, @Table etc တို႔ ၿဖစ္ၾကပါတယ္။ Hibernate Annotations က JPA 2 specification ကို အေၿခခံထားတာၿဖစ္ၿပီး features ေတြအားလံုးကိုလည္း

support လုပ္ေပးထားပါတယ္။ JAP annotations အားလံုးကို javax.persistence package ထဲမွာ သတ္မွတ္ထားတာ ၿဖစ္ပါတယ္။ Hibernate EntityManager က JPA specification က သတ္မွတ္ထားတဲ့ interfaces နဲ႔ life cycle ကို implements လုပ္ထားပါတယ္။ annotation နဲ႔ hibernate ကို develop လုပ္ၿခင္းက ဘာေကာင္းလဲဆိုေတာ့ mapping (hbm) file ကို create လုပ္စရာမလိုေတာ့ပါ။ hibernate annotations က meta data ကို ေပးႏိုင္ရန္ လုပ္ေဆာင္ပါတယ္။

1. Create persistence class

@Entity annotation - ဒီ class က entity ၿဖစ္တဲ့ဆိုတာကို သိသာေစဖို႔အတြက္ ၿဖစ္တယ္။

@Table annotation – entity ထဲက data ေတြကို ဘယ္ table ထဲကို persist လုပ္ရမလဲဆိုတာက ေၿပာၿခင္းၿဖစ္ပါတယ္။ @Table annotation ဆိုတဲ့ annotation ကို အသံုးမၿပဳခဲ့ဘူးဆိုရင္ေတာ့ class name နဲ႔တူတဲ့ table name ကို default အားၿဖင့္ သြားၿပီး persist လုပ္ေပးလိမ့္မယ္။

Student.java

@Entity

@Table(name = "emp500")

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

@Id

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

**private** String firstName, lastName;

**getter//setter**

2. Create the Configuration file

annotation နဲ႔ေရးထားတဲ့အတြက္ေၾကာင့္ mapping class ၿဖစ္ေနတာပါ။

hibernate.cfg.xml

<hibernate-configuration>

<session-factory>

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

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

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

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

<property name=*"connection.password"*>root</property>

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

<mapping class=*"hibernate\_with\_eclipse.hb\_with\_annotation.Student"*/>

<mapping resource=*"hibernate\_with\_eclipse/hb\_with\_xml/employee.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

3. db မွာ သက္ဆိုင္ရာ table name အတိုင္း ေဆာက္မယ္။

CREATE TABLE `hibernate`.`emp500` (

`id` INT NOT NULL,

`firstName` VARCHAR(45) NULL,

`lastName` VARCHAR(45) NULL,

PRIMARY KEY (`id`));

4. Create the class that retrieves or stores the persistent object.

StoredDataAnnotation.java

**public** **class** StoredDataAnnotation {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Student e1 = **new** Student();

e1.setId(101);

e1.setFirstName("Gaurav");

e1.setLastName("Chawla");

session.save(e1);

t.commit();

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

factory.close();

session.close();}}

Hibernate Example

1. HB Web Application

2. HB Generator Class

3. HB Dialects

1. **HB Web Application**

Web Application with Hibernate (using XML)

presentation logic အတြက္ jsp ကို သံုးမယ္။ representing data အတြက္ Bean class ကို သံုးမယ္။ database codes အတြက္ DAO class ကုိ သံုးမယ္။ အခု example မွာဆိုရင္ JSP file ကိုသံုးၿပီး user ဆီက data ကို ရယူပါမယ္။

index.jsp

user ဆီက data ယူမယ္ ၿပီးရင္ register.jsp ဆီကို post method နဲ႔ သြားမယ္။

<form action=*"pages/hibernate\_example/hb\_web\_application/register.jsp"* method=*"post"*>

Name:<input type=*"text"* name=*"name"*/><br><br/>

Password:<input type=*"password"* name=*"password"*/><br><br/>

Email ID:<input type=*"text"* name=*"email"*/><br><br/>

<input type=*"submit"* value=*"register"*/>

</form>

register.jsp

register.jsp file ကို user class ကေနလာတဲ့ object ထဲက information ေတြကို သိမ္းထားဖို႔နဲ႔ request parameters အကုန္လံုးကို ရယူဖို႔ရအတြက္ ေရးသားထားၿခင္းၿဖစ္ပါတယ္။ သူက user entity class object ကို userDao ရဲ႕ register method ေခၚၿပီး pass လုပ္ထားပါတယ္။ register.jsp က process ကို အဓိကခ်ိတ္ဆက္ေပးတဲ့ file တစ္ခုၿဖစ္ပါတယ္။

<%@page import=*"hibernate\_example.hb\_web\_application.UserDao"*%>

<jsp:useBean id=*"obj"* class=*"hibernate\_example.hb\_web\_application.User"*>

</jsp:useBean>

<jsp:setProperty property=*"\*"* name=*"obj"* />

<%

**int** i = UserDao.register(obj);

**if** (i > 0)

out.print("You are successfully registered");

%>

User.java

**public** **class** User {

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

**private** String name, password, email;

**public** User() {}

//getter and setter

user.hbm.xml

database ထဲက table နဲ႔ user entity နဲ႔ map လုပ္ေပးတဲ့ file ၿဖစ္ပါတယ္။

<hibernate-mapping>

<class name=*"hibernate\_example.hb\_web\_application.User"* table=*"u400"*>

<id name=*"id"*>

<generator class=*"increment"*></generator>

</id>

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

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

<property name=*"email"*></property>

</class>

</hibernate-mapping>

UserDao.java

parameter ပါလာတဲ့ user object ကို သိမ္းထားေပးၿပီး register page ကို return ၿပန္ေပးလိုက္တယ္။

**public** **class** UserDao {

**public** **static** **int** register(User u) {

**int** i = 0;

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

i = (Integer) session.save(u);

t.commit();

session.close();

**return** i;}}

hibernate.cfg.xml

database information နဲ႔ mapping file information ကို သိမ္းဆည္းေပးပါတယ္။

<mapping resource=*"hibernate\_example/hb\_web\_application/user.hbm.xml"*/>

web.xml

<welcome-file>pages/hibernate\_example/hb\_web\_application/index.jsp</welcome-file>

2. HB Generator Class

<generator> class က id ရဲ႕ sub element တစ္ခုၿဖစ္တယ္။ persistent class ရဲ႕ object ေတြအတြက္ unique identifie ကို ထုတ္ေပးဖို႔ရန္ အသံုးၿပဳပါတယ္။ Hibernate Framework ထဲမွာ generator class မ်ားစြာ သတ္မွတ္ေပးထားပါတယ္။

generator class အားလံုးက org.hibernate.id.IdentifierGenerator interface ကို implements လုပ္ထားပါတယ္။ application programme ေတြက IdentifierGenerator interface ကို implementing လုပ္ၿခင္းအားၿဖင့္ one's own generator classes ေတြကို create လုပ္တယ္။ Hibernate framework ေတြက built-in generator classes ေၿမာက္မ်ားစြာကို ထုတ္ထားေပးၿပီးသား ၿဖစ္ပါတယ္။

1. assigned
2. increment
3. sequence
4. hilo
5. native
6. identity
7. seqhilo
8. uuid
9. guid
10. select
11. foreign
12. sequence-identity

1) assigned

<generator> element မရွိခဲ့ဘူးဆိုရင္ ဒီေကာင္က default အေနနဲ႔၀င္ေနမွာ ၿဖစ္ပါတယ္။

....

 <hibernate-mapping>

  <**class** ...>

    <id ...>

     <generator **class**="assigned"></generator>

    </id>

    .....

  </**class**>

 </hibernate-mapping>

2) increment