Hibernate

Conceptes i característiques

Consorci d'Educació de Barcelona
Generalitat de Catalunya
Ajuntament de Barcelona

Concepte

El mapeig objecte-relacional (ORM Object-Relational Mapping) és una tècnica de programació per convertir dades entre: el sistema de tipus utilitzat en un llenguatge de programació orientat a objectes i el que s'utilitzat en una base de dades relacional.

Eines ORM

- Avantatges de l'ús d'un ORM:
  - Ajuda a reduir temps de desenvolupament.
  - Abstracció de la base de dades.
  - Reutilització.
  - Independència de la BD → Migracions amb més facilitat.
  - Llenguatge propi per realitzar consultes.
- Inconvenients:
  - Aplicacions més lentes.
  - Configuració de l'entorn més complexa.
- Eines: Doctrine, Propel, ADOdb Active Record, Hibernate, Oracle Toplink, iPersist, etc.

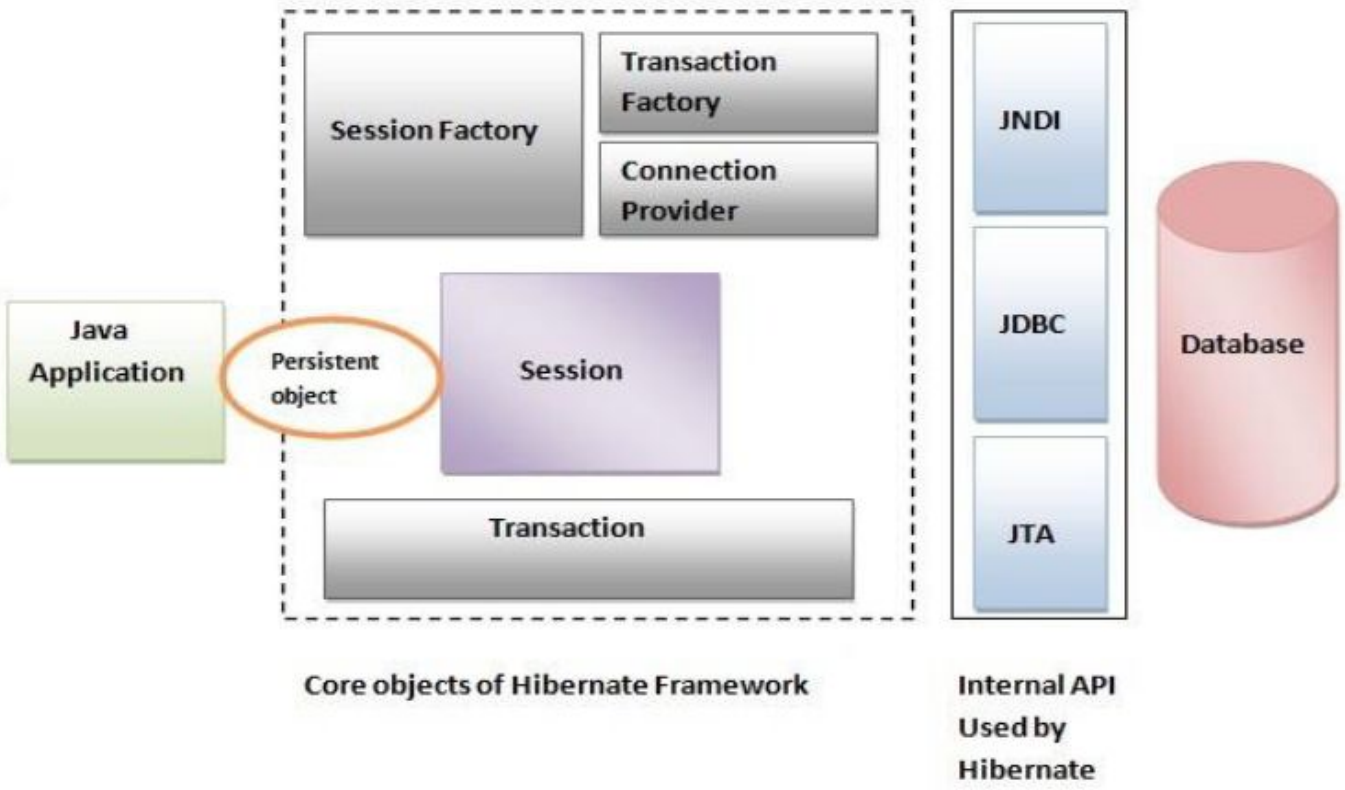
Hibernate

- Hibernate és una eina de mapeig objecte-relacional per a Java (disponible per a .NET amb el nom de Nhibernate) que facilita el mapeig d'atributs mitjançant fitxers declaratius (XML).
- És un dels estàndards per l'emmagatzematge persistent quan volem independitzar la capa de negoci de l'emmagatzematge de la informació.
- Amb Hibernate no emprarem habitualment SQL per a accedir a dades, sinó que el propi motor de Hibernate, mitjançant l'ús de factories (patró de disseny Factory) construirà aquestes consultes per a nosaltres.
- Hibernate posa a la disposició del dissenyador un llenguatge anomenat HQL (Hibernate Query Lenguage) que permet accedir a les dades mitjançant POO.

Arquitectura d'Hibernate

- Les interfícies d'Hibernate són:
  - La interfície SessionFactory (org.hibernate.SessionFactory): permet obtenir instàncies de l'objecte Session. Aquesta interfície ha de compartir-se entre molts fils d'execució. Normalment hi ha una única SessionFactory per a tota l'aplicació. Si l'aplicació accedeix a diverses bases de dades es necessitarà una SessionFactory per cada base de dades. D'altra banda, la classe Session ens ofereix mètodes com save(Object), createQuery(String), beginTransaction(), close(), etc.
  - La interfície Configuration (org.hibernate.cfg.Configuration): s'utilitza per configurar Hibernate. L'aplicació utilitza una instància de Configuration per a especificar la ubicació dels documents que indiquen el mapatge dels objectes i a continuació crea la SessionFactory.
  - La interfície Query (org.hibernate.Query): permet executar consultes i controla com es realitzen. Les consultes s'escriuen en HQL.
  - La interfície Transaction (org.hibernate.Transaction): permet realitzar modificacions o consultes en la base de dades segons el paradigma ACID

Arquitectura d'Hibernate gràficament



Estructura dels fitxers de mapeig

- Hibernate utilitza uns fitxers de mapeig per relacionar les taules de la base de dades amb els objectes Java. Aquests fitxers estan en format XML i tenen l'extensió .hbm.xml.
- Al projecte anterior s'han creat els fitxers Empleats.hbm.xml i Departaments.hbm.xml associats a les taules emple i depart respectivament.
- Vegem l'estructura d'aquests fitxers:

Estructura dels fitxers de mapeig

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd" >
<!-- Generated 30-mar-2022 10:59:28 by Hibernate Tools 5.2.1.Final -->
<hibernate-mapping>
  <class name="primero.Depart" table="depart" catalog="ejemplo" optimistic-lock="version">
      <id name="deptNo" type="int">
          <column name="dept_no" />
          <generator class="assigned" />
      </id>
      <property name="dnombre" type="string">
          <column name="dnombre" length="30" />
      </property>
      <property name="loc" type="string">
          <column name="loc" length="30" />
      </property>
      <set name="emples" table="emple" inverse="true" lazy="true" fetch="select">
          <key>
              <column name="dept_no" />
          </key>
          <one-to-many class="primero.Emple" />
      </set>
  </class>
</hibernate-mapping>
```

Estructura dels fitxers de mapeig

- **hibernate-mapping**: tots els fitxers de mapeig comencen i acaben amb aquesta etiqueta.
- **class**: aquesta etiqueta engloba la classe amb els seus atributs indicant el mapatge a la taula de la base de dades.
  - **name**: nom de la classe.
  - **table**: nom de la taula que representa l'objecte.
  - **catalog**: nom de la base de dades.
- Dins de class distingim l'etiqueta **id** en la qual s'indica en **name** el camp que representa a l'atribut clau i en **column** el seu nom sobre la taula, en **type** el tipus de dades. També tenim la propietat **generator** que indica la naturalesa del camp clau. En aquest cas és "assigned" perquè és l'usuari el que s'encarrega d'assignar la clau. Podria ser "increment" si és un camp generat per la base de dades. Aquest atribut es correspondria amb el camp dept_no de la taula depart.

Estructura dels fitxers de mapeig

- La resta d'atributs s'indiquen en les etiquetes **property** associant el nom del camp de la classe amb el nom de la columna de la taula i el tipus de dades.
- L'etiqueta **set** s'utilitza per a mapejar col·leccions. Dins de set es defineixen diversos atributs:
  - **name**: indica el nom de l'atribut generat.
  - **table**: el nom de la taula d'on es prendrà la col·lecció.
  - L'element **key** defineix el nom de la columna identificadora en l'associació.
  - L'element **one-to-many** defineix la relació (un departaments pot tenir molts empleats).
  - **class**: indica de quin tipus són els elements de la col·lecció.
- Els tipus que declarem en els fitxers de mapeig no són tipus de dades Java ni SQL. Es diuen: tipus de mapeig Hibernate.

Classes permanents

- Les classes referenciades en l'element class dels fitxers de mapeig fan referència a les classes generades al nostre projecte com Emple.java i Depart.java. A aquestes classes se'n diu classes permanents.
- Les classes permanents són les classes que implementen les entitats del problema, han d'implementar la interfície Serializable. Equivalen a una taula de la base de dades, i un registre o fila és un objecte permanent d'aquesta classe.
- Aquestes classes representen un objecte emple i un objecte depart, per tant, podem crear objectes emprats i departaments a partir d'elles. Tenen uns atributs privats i uns mètodes públics (getters i setters) per accedir a aquests.
- A aquestes normes se'ls sol dir model de programació POJO (Plain Old Java Objects).

```java
import java.util.HashSet;
import java.util.Set;
public class Depart implements java.io.Serializable {
    private int deptNo;
    private String dnom;
    private String loc;
    private Set<Emple> empleats = new HashSet<Emple>(0);
    public Depart() {
    }
    public Depart(int deptNo) {
        this.deptNo = deptNo;
    }
    public Depart(int deptNo, String dnom, String loc, Set<Emple> empleats) {
        this.deptNo = deptNo;
        this.dnom = dnom;
        this.loc = loc;
        this.empleats = empleats;
    }
    public int getDeptNo() {
        return this.deptNo;
    }
    public void setDeptNo(int deptNo) {
        this.deptNo = deptNo;
    }
```

```
    public String getDnom() {
        return this.dnom;
    }
    public void setDnom(String dnom) {
        this.dnom = dnom;
    }
    public String getLoc() {
        return this.loc;
    }
    public void setLoc(String loc) {
        this.loc = loc;
    }
    public Set<Emple> getempleats() {
        return this.empleats;
    }
    public void setempleats(Set<Emple> empleats) {
        this.empleats = empleats;
    }
}
```

Activitat

- **A1.-**Realitza el mapeig de la base de dades world i observa els fitxers de mapeig i les classes generades