Hibernate III

Conceptos y características

Concepto

- El **mapeo objeto-relacional (ORM** Object-Relational Mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional.

Herramientas ORM

- Ventajas del uso de un ORM:
  - Ayuda a reducir tiempo de desarrollo
  - Abstracción de la base de datos
  - Reutilización
  - Independencia de la BD → Migrar con facilidad
  - Lenguaje propio para realizar consultas
- Inconvenientes:
  - Aplicaciones más lentas
  - Configuración del entorno más compleja
- Herramientas: Doctrine, Propel, ADOdb Active Record, Hibernate, Oracle Toplink, iPersist, etc.

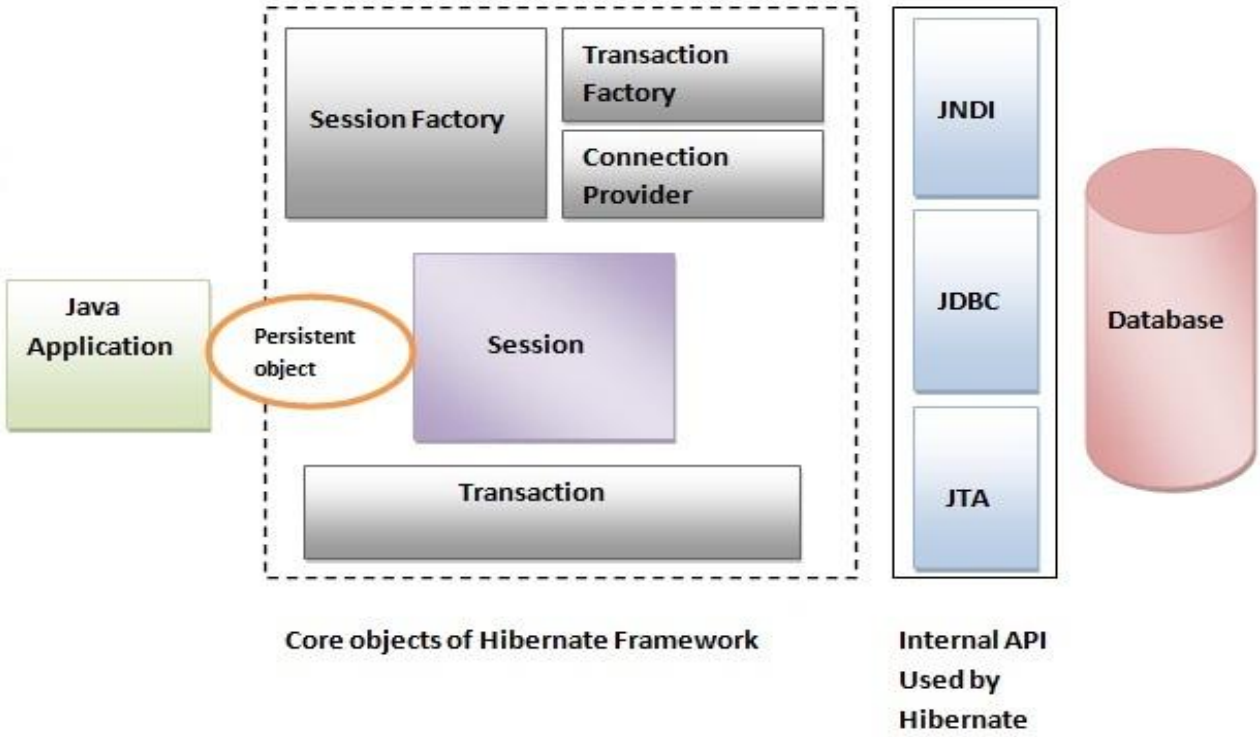
Hibernate

- Hibernate es una herramienta de mapeo objeto-relacional para la plataforma Java (disponible para .NET con el nombre de Nhibernate) que facilita el mapeo de atributos mediante ficheros declarativos (XML)
- Se está convirtiendo en el estándar de facto para almacenamiento persistente cuando queremos independizar la capa de negocio del almacenamiento de la información.
- Con Hibernate no emplearemos habitualmente SQL para acceder a datos, sino que el propio motor de Hibernate, mediante el uso de factorías (patrón de diseño **Factory**) construirá esas consultas para nosotros.
- Hibernate pone a disposición del diseñador un lenguaje llamado **HQL (Hibernate Query Lenguage)** que permite acceder a los datos mediante POO.

Arquitectura de Hibernate

- Las interfaces de Hibernate son:
  - La interfaz **SessionFactory** (org.hibernate.SessionFactory): permite obtener instancias de **Session.** Esta interfaz debe compartirse entre muchos hilos de ejecución. Normalmente hay una única **SessionFactory** para toda la aplicación. Si la aplicación accede a varias bases de datos se necesitará una **SessionFactory** por cada base de datos. Por otro lado, la clase Session nos ofrece métodos como **save(Object), createQuery(String), beginTransaction(), close(),** etc.
  - La interfaz **Configuration** (org.hibernate.cfg.Configuration)**:** se utiliza para configurar Hibernate. La aplicación utiliza una instancia de **Configuration** para especificar la ubicación de los documentos que indican el mapeado de los objetos y a continuación crea la **SessionFactory**
  - La interfaz **Query** (org.hibernate.Query): permite ejecutar consultas y controla cómo se realizan. Las consultas se escriben en **HQL**
  - La interfaz **Transaction** (org.hibernate.Transaction): permite realizar modificaciones o consultas en la base de datos según el paradigma ACID

Una forma de imaginarse esta arquitectura es:



Estructura de los ficheros de mapeo

- Hibernate utiliza unos ficheros de mapeo para relacionar las tablas de la base de datos con los objetos Java. Estos ficheros están en formato XML y tienen la extensión **.hbm.xml**.
- En el proyecto anterior se han creado los ficheros **Empleados.hbm.xml** y **Departamentos.hbm.xml** asociados a las tablas emple y depart respectivamente.
- Veamos la estructura de estos ficheros

```
<?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 06-mar-2017 1:59:28 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>
```

- **hibernate-mapping**: todos los ficheros de mapeo comienzan y acaban con esta etiqueta.
- **class:** esta etiqueta engloba la clase con sus atributos indicando el mapeo a la tabla de la base de datos.
    - **name:** nombre de la clase
    - **table:** nombre de la tabla que representa el objeto
    - **catalog:** nombre de la base de datos
- Dentro de class distinguimos la etiqueta **id** en la cual se indica en **name** el campo que representa al atributo clave y en **column** su nombre sobre la tabla, en **type** el tipo de datos. También tenemos la propiedad **generator** que indica la naturaleza del campo clave. En este caso es “assigned” porque es el usuario el que se encarga de asignar la clave. Podría ser “increment” si es un campo generado por la base de datos. Este atributo se correspondería con el campo dept_no de la tabla depart.

- Resto de atributos se indican en las etiquetas **property** asociando el nombre del campo de la clase con el nombre de la columna de la tabla y el tipo de datos.
- La etiqueta **set** se utiliza para mapear colecciones. Dentro de set se definen varios atributos:
    - **name**: indica el nombre del atributo generado
    - **table:** el nombre de la tabla de donde se tomará la colección
    - El elemento **key** define el nombre de la columna identificadora en la asociación
    - El elemento **one-to-many** define la relación (un departamentos puede tener muchos empleados)
    - **class:** indica de qué tipo son los elementos de la colección.
- Los tipos que declaramos en los ficheros de mapeo no son tipos de datos Java ni SQL. Se llaman **tipos de mapeo Hibernate.**

Clases persistentes

- Las clases referenciadas en el elemento **class** de los ficheros de mapeo hacen referencia a las clases generadas en nuestro proyecto como Emple.java y Depart.java. A estas clases se les llama **clases persistentes.**
- Las clases persistentes son las clases que implementan las entidades del problema, deben implementar la interfaz **Serializable.** Equivalen a una tabla de la base de datos, y un registro o fila es un objeto persistente de esa clase
- Estas clases representan un objeto emple y un objeto depart, por lo tanto, podemos crear objetos empleados y departamentos a partir de ellas. Tienen unos atributos privados y unos métodos públicos (*getters* y *setters*) para acceder a los mismos.
- A estas reglas se les suele llamar modelo de programación **POJO** (Plain Old Java Objects)

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

```
    public String getLoc() {
            return this.loc;
    }

    public void setLoc(String loc) {
            this.loc = loc;
    }

    public Set<Emple> getEmples() {
            return this.emples;
    }

    public void setEmples(Set<Emple> emples) {
            this.emples = emples;
    }

}
```

Problemas Hibernate_III

- Realiza el mapeo de la base de datos world y observa los ficheros de mapeo y las clases generadas