Mapeo Objeto-Relacional Parte 3ª

Ahora solo nos falta profundizar en la ejecución de consultas HQL desde Java.

1. Consultas. Introducción

Recordemos que HQL es el lenguaje de consulta de Hibernate, muy parecido a SQL pero en el que las tablas se convierten en clases (colecciones de objetos de estas clases), las filas de las tablas son objetos de esas clases y las relaciones entre las tablas asociaciones.

De forma muy parecida a cómo lo hacemos con conectores, podemos ejecutar consultas desde Java utilizando la interfaz org.hibernate.Query.

Un objeto Query se crea siempre utilizando el método createQuery de la sesión.

```
Query consulta = misesion.createQuery("cadenaHQL");
```

Las consultas pueden ser de varios tipos:

- Consultas de recuperación
- Consultas "escalares"
- Consultas de actualización

El tipo determinará el método a utilizar para ejecutarla.

2. Consultas de recuperación

Una consulta es de recuperación cuando recupera objetos o propiedades.

Para ejecutar la consulta y recuperar el resultado tenemos los métodos list() o iterate().

- Iterator iter = consulta.iterate();
- List <tipo fila> nombrelista = consulta.list();

2.1. El mótodo iterate()

El método **iterate()** devuelve un iterador Java para iterar los resultados de la consulta. En cuanto a programación sería parecido al executeReader de ADO.NET.

En este caso se ejecuta la consulta obteniendo solo los ids de los objetos y en cada llamada del método iterador.next() se ejecuta la consulta propia para obtener el objeto correspondiente. Esto supone un mayor número de accesos a la base de datos y por tanto, mayor tiempo de procesamiento total y mayor carga del servidor. La ventaja es que no requiere que todos los objetos estén cargados en memoria simultáneamente.

Para reducir el número de accesos a la base de datos se puede utilizar el método setFetchSize() que determina cuántas filas recuperar en un acceso a la base de datos:

```
consulta. setFetchSize (10) ;
```

Ejemplo de utilización de iterate():

```
import hibernate1.*;
import java.util.Iterator;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Query;
public class QueryDep {
    public static void main (String[] args) {
        SessionFactory sesionf = SessionFactoryUtil.getSessionFactory();
        Session sesion = sesionf.openSession();
        Query q = sesion.createQuery("from Departamentos");
        q.setFetchSize(5);
        Iterator iter = q.iterate();
        Departamentos depi ;
        while (iter.hasNext()){
            depi = (Departamentos) iter.next();
            System.out.println(depi.getDeptNo() + " * " + depi.getDnombre());
        }
        sesion.close();
    }
}
```

2.2. El mótodo list()

El método **list()** devuelve en una colección todos los resultados de la consulta, cada elemento de la colección corresponde a una fila del resultado.

Este método realiza una única comunicación con la base de datos, se traen todas las filas correspondientes, por lo que se requiere que haya memoria suficiente para almacenar todos los objetos correspondientes a esas filas y si la cantidad de filas a recuperar es considerable el método tardará en ejecutarse.

Siempre tenemos que tener presente el tipo de "filas" que devuelve la consulta.

La forma de trabajar es la siguiente:

1. Definir la cadena HQL --> String mihql = "cadena HQL";
   Crear la Query --> Query q = misesion.createQuery(mihql);

   o bien directamente: --> Query q = misesion.createQuery("cadena HQL");

2. Ejecutar la consulta y recoger el resultado:
   List <tipo fila> lista = q.list();

3. Trabajar con la lista como con una colección cualquiera. Lo habitual es crear un bucle "for each" para recorrer la lista:

```
for (tipo-fila fila: lista) {
        tratar la fila
}
```

En cada iteración del bucle recogemos en *fila* una "fila" del resultado.

Hay que pensar en cómo son las filas del resultado para actuar en consecuencia.

La estructura de las filas del resultado depende de la HQL que se ejecuta, veamos ejemplos de las diferentes situaciones.

2.2.1. Consulta de tipo "from Clase"

Ejemplo:

```
import hibernate1.*;
import java.util.Iterator;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Query;
public class QueryDep {
    public static void main (String[] args) {
        SessionFactory sesionf = SessionFactoryUtil.getSessionFactory();
        Session sesion = sesionf.openSession();
        Query q = sesion.createQuery("from Departamentos");
        List <Departamentos> lista = q.list();
        for (Departamentos dep: lista){
            System.out.println(dep.getDeptNo() + " * " + dep.getDnombre());
        }
        sesion.close();
    }
}
```

En este caso las filas devueltas son objetos Departamentos, luego asignamos el resultado de la consulta a una List <Departamentos>, y en el bucle "for each" el tipo-fila es la clase Departamentos y llamamos a la variable que recoje el elemento de la lista dep

for (<Departamentos> dep: lista)

2.2.2. Consultas sobre clases no asociadas

Si queremos recuperar los datos de una consulta con select o en la que intervienen varias tablas y no tenemos asociada a ninguna clase lo que devuelve esa consulta, podemos utllizar la **clase Object.** Object se refiere a un objeto pero con una definición amplia que acepta 'cualquier cosa'.

2.2.2.1. Consultas de tipo from a, b

Cuando la consulta es del tipo from a, b (siendo a y b el nombre de dos clases asociadas a sendas tablas), cada 'fila' del resultado está formada por un objeto de clase a y un objeto de clase b, por lo que los resultados se reciben en un array de objetos, donde el primer elemento del array se corresponde con la primera clase que ponemos a la derecha de FROM, el siguiente elemento con la siguiente clase y así sucesivamente (si la from incluyera más clases). El siguiente ejemplo realiza una consulta para obtener los datos de los empleados y de sus departamentos. El resultado de la consulta se recibe en un List de array de objetos, cada elemento del List es un array de objetos (Object[]), es como si dijéramos "un array de cosas" donde el primer elemento del array pertenece a la clase Empleados y el segundo a la clase Departamentos:

```
import hibernate1.*;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Query;
public class QueryClasesNoAsociadas {
   public static void main (String[] args) {
      SessionFactory sesionf = SessionFactoryUtil.getSessionFactory();
      Session sesion = sesionf.openSession();
      System.out.println("INICIO");
      Query q = sesion.createQuery("from Departamentos d, Empleados e "
            + "where d.deptNo=e.departamentos.deptNo order by e.apellido");
      List <Object[]> lista = q.list();
      Departamentos dep;
```

```
        Empleados emp;
        for (Object[] elto: lista){
            dep = (Departamentos) elto[0];
            emp = (Empleados) elto[1];
            System.out.println(dep.getDeptNo() + " * " + dep.getDnombre()+
                    " * " + emp.getApellido());
        }
        System.out.println("FIN");
        sesion.close();
    }
}
```

En este caso la query recupera departamentos y empleados, cada "fila" devuelta contiene un departamento y un empleado por lo que ahora no podemos asignar el resultado de q.list() a una lista de objetos de una determinada clase (no tenemos ninguna clase definida así) por lo que lo asignamos a una lista de arrays de objetos. Cada elemento de esta lista será un array de objetos, en este caso concreto será un array de dos elementos, el primero un objeto de tipo Departamentos y el segundo un objeto de tipo Empleados.

Para recorrer la lista hemos utilizado un bucle for "each" y ahora en cada iteración del bucle recuperamos el array con los dos objetos y los tratamos. Como conocemos la clase de cada elemento del array, podemos aplicar al elemento el cast correspondiente para asignar a un objeto de la clase correspondiente: emp = (Empleados) elto[1]; donde emp es de tipo Departamentos.

2.2.2.2. Consultas de tipo select ....

Recuerda que el método list() carga todos los valores y si trabajamos con tablas de muchas columnas pero solo vamos a utilizar pocas, para ahorrar recursos podríamos seleccionar solo las columnas necesarias:

```
q = sesion.createQuery("select d.deptNo, d.loc, e.empNo, e.apellido "
            + "from Departamentos d, Empleados e "
            + "where d.deptNo=e.departamentos.deptNo order by e.apellido");
```

Veamos lo que pasa si añadimos a la query que hemos probado la cláusula select anterior.

Añadela y ejecuta. ¿Qué ocurre?

Error-→ "...java.lang.Byte cannot be cast to hibernate1.Departamentos"

¿Por qué? En la asignación a dep el primer objeto recuperado no es un departamento sino un nº.

El siguiente código sí funciona:

```
q = sesion.createQuery("select d.deptNo, d.loc, e.empNo, e.apellido "
        + "from Departamentos d, Empleados e "
        + "where d.deptNo=e.departamentos.deptNo order by e.apellido");
List <Object[]> lista2 = q.list();
for (Object[] elto2: lista2){
    System.out.println(elto2[0] + " * " + elto2[1] + " * " + elto2[2] +
                       " * " + elto2[3]);
}
System.out.println("FIN");
sesion.close();
```

Lo que devuelve la query es una lista en la que cada elemento de esa lista (cada fila) es un array de 4 elementos (cada una de las propiedades devueltas). Como esos elementos son de distintos tipos los asignamos a un array [] de Object (el tipo comodín).

Y utilizamos los elementos del array directamente, según sean números o strings.

3. Consultas "escalares"

Aunque una consulta escalar es una consulta de recuperación, la hemos incluido en un punto aparte porque se ejecuta de diferente forma.

Entendemos como consulta "escalar" una consulta HQL que devuelve un solo resultado (que puede ser un único objeto o un único valor. En este caso podemos simplificar el tratamiento del resultado utilizando UniqueResult():

```
System.out.println("*******Datos del departamento 10");
Departamentos dep10 = (Departamentos) sesion.createQuery("from Departamentos D where D.deptNo=10").uniqueResult();
System.out.println(dep10.getDeptNo() + " * " + dep10.getDnombre());
```

En este caso la query devuelve un solo departamento (un solo objeto de un tipo determinado). asignamos pues el resultado de la query directamente a un objeto de la clase correspondiente.

UniqueResult() se puede emplear también en las consultas de grupo (de resumen) cuando se devuelve un solo valor.
Ejemplo:

```
public class QueryFuncionesGrupo {
    public static void main (String[] args) {
        SessionFactory sesionf = SessionFactoryUtil.getSessionFactory();
        Session sesion = sesionf.openSession();
        System.out.println("INICIO");
        Query q = sesion.createQuery("select avg(salario) from Empleados");
        Double media = (Double) q.uniqueResult();
        System.out.println("Salario medio: " + media);
        sesion.close();
```

4. Consultas de actualización

El lenguaje HQL también incorpora sentencias de actualización parecidas a las de SQL: INSERT INTO, DELETE y UPDATE.

Cuando la query está basada en este tipo de hql el método para ejecutarla es **executeUpdate()**.

La sintaxis para la operación **INSERT INTO** es la siguiente:

insert into nombreClase (lista propiedades) sentencia-select

Donde:
- Solo se soporta la forma INSERT INTO ... SELECT ..., no la forma INSERT INTO ... VALUES
Es decir, solo podemos insertar datos procedentes de otra consulta.

- La lista de propiedades es análoga a la lista de columnas en la declaración INSERT de SQL, pero en este caso obligatoria.

- La sentencia-select puede ser cualquier consulta select de HQL válida, hay que tener en cuenta que los tipos devueltos por la consulta coincidan con los esperados por el insert.

- Para el caso de la propiedad *id* hay dos opciones: se puede especificar en la lista de propiedades (en tal caso su valor se toma de la expresión de selección correspondiente) o se puede omitir en la lista de propiedades (en este caso se utiliza un valor generado). Esta última opción solamente está disponible cuando se utilizan generadores de id que operan en la base de datos (por ejemplo cuando la clave primaria es de autoincremento).

Ejemplo:

String hql = "insert into departamentos (deptNo, nombre, loc) select deptNo, nombre, loc from Nuevos";
int deptosInsertados = misesion.createQuery(hql).executeUpdate ( ) ;
System.out.println("Nº de filas insertadas:" + deptosInsertados);

La sintaxis para la operación **UPDATE** es la siguiente:

update nombreClase set {propiedad = nuevovalor} [,...n] + where condicion

Ejemplo:
```
String hql = "update departamentos dep set dep.loc = "" where dep.loc = 'FOL'";
int deptosModificados = misesion.createQuery(hql).executeUpdate ( ) ;
System.out.println("Nº de filas actualizadas:" + deptosModificados);
```

La sintaxis para la operación **DELETE** es la siguiente:

delete from nombreClase where condicion

Ejemplo:

```
String hql = "delete from departamentos dep where dep.loc = '"";
int deptosEliminados = misesion.createQuery(hql).executeUpdate ( ) ;
System.out.println("Nº de filas eliminadas:" + deptosEliminados);
```

5. Parámetros en las consultas

Con Hibernate también podemos definir consultas con parámetros. En este caso se utiliza la misma interfaz Query.

El parámetro puede ser de posición, en este caso se utiliza ?, o un parámetro por nombre, en este caso se utiliza el prefijo : delante del nombre del parámetro.

Hibernate numera los parámetros desde cero, el primero que aparece estará en la posición 0, el siguiente en la 1, y así sucesivamente. Las ventajas de los parámetros con nombre son las siguientes:

- son insensibles al orden en que aparecen en la cadena de consulta,
- pueden aparecer múltiples veces en la misma petición,
- son auto-documentados.

Antes de ejecutar la consulta debemos asignar valores a los parámetros utilizando los métodos set*tipo* de la consulta. Estos métodos suelen tener la misma estructura:

set*tipo*(int posicion| String nombre, valor)

donde:

- posicion es la posición del parámetro cuando utilizamos parámetros de posición ?
- nombre es el nombre del parámetro cuando utilizamos parámetros con nombre
- valor es el valor que le asignamos

Según el tipo podemos tener:

- setCharacter()
- setDate()
- setDouble()
- setInteger()
- setString()

Por ejemplo:

```
Query q = misesion.createQuery("from Empleados emp  where emp.departamentos.deptNo=:ndep
and emp.oficina = :ofi");
q.setInteger ( "ndep" , 10) ;
q.setString("ofi", "DIRECTOR") ;
```

Aquí hemos asignado los valores utilizando parámetros por nombre.

Si utilizamos parámetros de posición:

```
Query q = misesion.createQuery("from Empleados emp  where emp.departamentos.deptNo? and emp.oficina = ?");
q.setString( 1, "DIRECTOR");
q.setInteger ( 0 , 10) ;
```

No se pueden mezclar en una misma query parámetros de posición y por nombre.

También tenemos un método interesante:

setParameterList(String nombre, Collection listaValores)

Permite asignar una colección de valores como parámetros:

```
Query q =misesion.createQuery("from Empleados emp where emp.nombre in :empNombres ");
q.setParameterList("empNombres", new String[]{"Ana","Juan"});
```

Aquí hemos señalado unos pocos métodos, puedes consultar todos los métodos en:

https://docs.jboss.org/hibernate/orm/3.2/api/org/hibernate/Query.html