IES CHAN DO MONTE
C.S. de Desarrollo de Aplicaciones Multiplataforma Modulo Acceso a datos

1.ANOTACIÓNS

Ata o de agora estivemos facendo o mapeo as clases Java mediante o ficheiro dos arquivos .hbm.xml .

Dende fai algún anos en Java creouse o concepto chamado "El infierno XML". Este inferno consistiu en facíase un uso intensivo de XML, sendo o XML un formato de ficheiro demasiado longo de escribir, moi repetitivo, verboso, ect. Isto levou a crear unha solución para evitar os ficheiros XML de persistencia en Hibernate: O uso de Anotacións JAVA no propio código. Estas anotacións permiten especificar dunha forma máis compacta e sinxela a información de mapeo das clases Java

Actualmente teríase que utilizar as estádar de JPA que se econtran no paquete javax.persistence. Sen embargo hai características específicas de Hibernate que non posee JPA o que fai que aínda sexa necesario usar alguna enotación do paquete `org.hibernate.annotations` pero neste caso Hibernate 4 non as marcou como `java.lang.Deprecated`.

Vexamos un exemplo da clase Profesor pero mapeada con anotacións:

```
@Entity
@Table(name="Profesor")
public class Profesor implements Serializable  {

    @Id
    @Column(name="Id")
    private int id;

    @Column(name="nombre")
    private String nombre;

    @Column(name="ape1")
    private String ape1;

    @Column(name="ape2")
    private String ape2;


    public Profesor(){
    }

    public Profesor(int id, String nombre, String ape1, String ape2) {
        this.id = id;
        this.nombre = nombre;
        this.ape1 = ape1;
        this.ape2 = ape2;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```
    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApe1() {
        return ape1;
    }

    public void setApe1(String ape1) {
        this.ape1 = ape1;
    }

    public String getApe2() {
        return ape2;
    }

    public void setApe2(String ape2) {
        this.ape2 = ape2;
    }
}
```

As anotacións que se usaron son as seguintes:

- @Entity: Aplícase á clase e indica que esta clase Java é unha entidade a persistir. É unha anotación estándar de JPA. No noso exemplo estamos inidcando que a clase Profesor é unha entidade que se pode persistir
- @Table(name=”Profesor”): Aplícase á clase e indica o nome da tabla da base de datos onde se persistirá a clase. É opcional se o nome da clase coincide co da tabla. É unha anotación estádar de JPA. No noso exemplo estamos indicando que a clase Profesor persitirá na tabla Profesor da base de datos.
- @Id: Aplícase a unha propiedade Java e inidca que este atributo é a clave primaria. É unha anotación estádar de JPA. No noso exemplo estamos indicando que a propiedade Java id é a clave primaria.
- @Column(name=”id”): Aplícase a unha propiedade Java e indica o nome da columna da base de datos na que se persistirá a propiedade. É opcional se o nome da propiedade Java coincide co da columna da base de datos. É unha anotación estándar de JPA. No noso exemplo estamos indicando que a propiedade JAVA id persistirá na columna chamada Id.
- @Column(name=”nombre”): Aplícase a unha propiedade Java e indica o nome da columna da base de datos na que persistirá a propiedade. É opcional se o nome da propiedade Java coincide co da columna de base de datos. É unha anotación estándar de JPA. No noso exemplo estamos indicando que a propiedade JAVA nombre persistirá an columna chamada nombre.
- @column(name=”ape1”):É igual ao caso anterior pero para a propiedade ape1.
- @column(name=”ape2”): É igual ao caso anterior pero para a propiedade ape2.

Unha diferencia importante entre usar o ficheiro de mapeo hbm.xml e as anotacións é que no ficheiro é obrigatorio indicar todas as propiedades que qureremos que persistan en base de datos, mentras que usando as anotacións iso non é necesario. Usando anotacións persisten todas as propiedades que teñan os métodos get/set.

2.RELACIONS CON ANOTACIONS

2.1Un a un (unidireccional)

A relación un a un en Hibernate consiste simplemente en que un obxecto teña unha referencia a outro obxecto de forma que ao persistirse o primeiro obxecto tamén se persista o segundo. Nesta lección, a relación vai ser unidireccional, isto é, que a relación un a un será nun único sentido.

Clases Java Antes de entrar en como se implementa en Hibernate, vexamos as clases Java e as táboas que definen a relación un a un. Para o noso exemplo, imos usar as clases:

- Profesor
- Dirección

Estas dúas clases terán unha relación un a un.

```
import java.io.Serializable;
import javax.persistence.*;



@Entity
@Table(name="Profesor")
public class Profesor implements Serializable  {

   @Id
   @Column(name="Id")
   private int id;

   @Column(name="nombre")
   private String nombre;

   @Column(name="ape1")
   private String ape1;

   @Column(name="ape2")
   private String ape2;

   @OneToOne(cascade=CascadeType.ALL)
   @PrimaryKeyJoinColumn
   private Direccion direccion;
}
```

Á propiedade dirección (liña 24) engadíronse dúas anotacións para indicar a relación un a un e que esta

relación se implemente mediante a clave primaria.

- @OneToOne(cascade=CascadeType.ALL): Esta anotación indica a relación un a un entre as 2 táboas. Tamén indicamos o valor de cascade igual que no ficheiro de hibernate.

- @PrimaryKeyJoinColumn: Indicamos que a relación entre as dúas táboas se realiza mediante a clave primaria.

En caso de no incluir la anotación `@PrimaryKeyJoinColumn` se producirá un error indicando que falta la columna `direccion_Id` en la tabla `Profesor`.

Nótese que si utilizamos anotaciones en necesario usar `@PrimaryKeyJoinColumn` mientras que usando el fichero `.hbm.xml` no es necesario indicarlo.

No código de Direccion non é necesario indicar nada sobre a relación, tal e como explicamos no caso do ficheiro de Hibernate

---

```
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="Direccion")
public class Direccion implements Serializable  {

   @Id
   @Column(name="Id")
   private int id;

   @Column(name="calle")
   private String calle;

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

   @Column(name="poblacion")
   private String poblacion;

   @Column(name="provincia")
```

```
    private String provincia;
}
```

Código Java

Agora que xa temos as clases Java preparadas para poder persistirlas, vexamos o código necesario para persistilas.

```
Direccion direccion=new Direccion(1, "Plaza del ayuntamiento", 8, "Xativa", "Valencia");
Profesor profesor=new Profesor(1, "Juan", "Perez", "García");
profesor.setDireccion(direccion);

Session session=sessionFactory.openSession();
session.beginTransaction();

session.save(profesor);

session.getTransaction().commit();
session.close();
```

Como podes ver, non hai nada novo no código Java para persistir unha relación un a un, simplemente creamos as 2 clases (Líñas 1 e 2) e establecemos a relación entre ambas asignándolle ao obxecto Profesor a referencia ao obxecto Direccion (Líña 3). Por último, simplemente persistimos a clase Profesor tal e como se explicou anteriormente.

Ao executar o exemplo de Hibernate, vemos como se crearon as filas nas Táboas Profesor e Direccion mentres que desde Java só se persistiu a clase Profesor.

Se revisamos o rexistro que se xera ao persistir os 2 obxectos, podemos ver que primeiro se realiza unha orde SELECT contra a táboa Direccion para comprobar se xa existe a dirección na base de datos. Isto faíxao Hibernate xa que se xa existe non é necesario inserir a fila da dirección, pero se a fila xa existe pero os datos son distintos, Hibernate lanzará un UPDATE para modificalos.

2.2Un a un (bidireccional)

Esta lección é moi semellante á anterior, pero neste caso a relación entre as clases Profesor e

Direccion será bidireccional.

Clases Java

Antes de abordar cómo se implementa en Hibernate, vexamos as clases Java e as táboas que definen a relación un a un.

Para o noso exemplo, usaremos as clases:

- Profesor

- Direccion

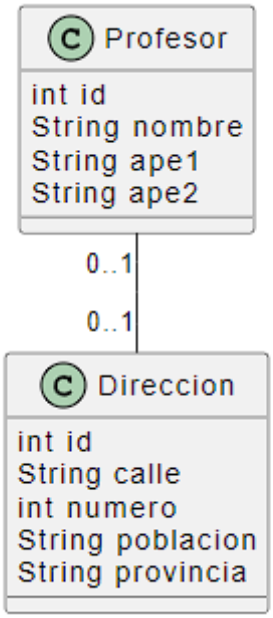
Estas dúas clases terán unha relación un a un.

```
public class Profesor implements Serializable  {
    private int id;
    private String nombre;
    private String ape1;
    private String ape2;
    private Direccion direccion;

    public Profesor(){
    }

    public Profesor(int id, String nombre, String ape1, String ape2) {
        this.id = id;
        this.nombre = nombre;
        this.ape1 = ape1;
        this.ape2 = ape2;
    }
}

public class Direccion implements Serializable  {
    private int id;
    private String calle;
    private int numero;
    private String poblacion;
    private String provincia;
    private Profesor profesor;

    public Direccion(){
    }

    public Direccion(int id, String calle, int numero, String poblacion, String provincia) {
        this.id = id;
        this.calle = calle;
        this.numero = numero;
        this.poblacion = poblacion;
        this.provincia = provincia;
    }
}
```

No listado 1, podemos ver como a clase Profesor ten unha propiedade chamada "direccion" da clase Direccion (liña 6) e, ademais, a clase Direccion tamén posúe unha referencia a Profesor, xa que

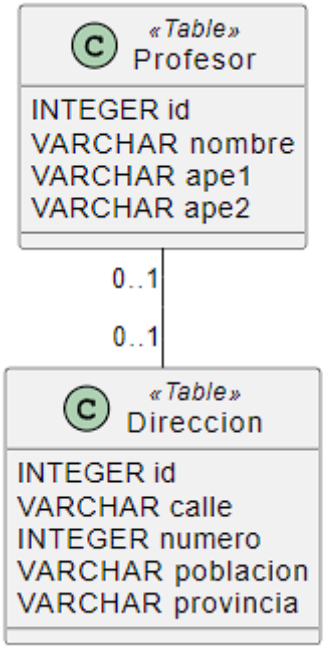
definimos que a relación é bidireccional tanto desde Profesor cara a Direccion como desde Direccion cara a Profesor.

No seguinte diagrama UML, pódese observar a relación desde Profesor cara a Direccion e viceversa.



Tablas

A tabla de base de datos quedarían da siguinte forma:



Podemos apreciar que no deseño das táboas da base de datos xa non existe unha columna en Profesor coa clave primaria de Direccion ou viceversa, xa que se a houbese, sería unha relación de moitos a un. Entón, ¿como se establece a relación entre as dúas filas? Simplemente porque tanto Profesor como Direccion deben ter a mesma clave primaria e desta forma establécese a relación.

ANOTACIONS

A continuación vemos a clase Profesor mediante anotacións.

```
import java.io.Serializable;
import javax.persistence.*;


@Entity
@Table(name="Profesor")
public class Profesor implements Serializable  {

    @Id
    @Column(name="Id")
    private int id;

    @Column(name="nombre")
    private String nombre;

    @Column(name="ape1")
    private String ape1;

    @Column(name="ape2")
    private String ape2;

    @OneToOne(cascade=CascadeType.ALL)
    @PrimaryKeyJoinColumn
    private Direccion direccion;
}
```

Á propiedade dirección (liña 24) engadíronse dúas anotacións para indicar a relación un a un e que esta relación se implementa mediante a clave primaria.

- `**@OneToOne(cascade=CascadeType.ALL)**`: Esta anotación indica a relación un a un entre as 2 táboas. Tamén indicamos o valor de cascade igual que no ficheiro de Hibernate.

- `**@PrimaryKeyJoinColumn**`: Indicamos que a relación entre as dúas táboas se realiza mediante a clave primaria.

En caso de no incluir la anotación `@PrimaryKeyJoinColumn` se producirá un error indicando que falta la columna `direccion_Id` en la tabla `Profesor`.

Notar que si utilizamos anotaciones es necesario usar `@PrimaryKeyJoinColumn` mientras que usando el fichero `.hbm.xml` no es necesario indicarlo.

O código de Direccion é similar ao anterior.

```java
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;


@Entity
@Table(name="Direccion")
public class Direccion implements Serializable  {

    @Id
    @Column(name="Id")
    private int id;

    @Column(name="calle")
    private String calle;

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

    @Column(name="poblacion")
    private String poblacion;

    @Column(name="provincia")
    private String provincia;

    @OneToOne(cascade=CascadeType.ALL)
    @PrimaryKeyJoinColumn
    private Profesor profesor;
}
```

Ao igual que no caso anterior, vemos cómo debemos incluír as etiquetas @OneToOne e @PrimaryKeyJoinColumn (Liñas 22 e 23) para establecer a relación entre Direccion e Profesor.

Código Java

Agora que xa temos preparadas as clases Java para que poidan persistir, vexamos o código necesario para persistilas.

```java
Direccion direccion1=new Direccion(3, "Calle de la sarten", 23, "Manises", "Valencia");
Profesor profesor1=new Profesor(3, "Sergio", "Mateo", "Ramis");
profesor1.setDireccion(direccion1);
direccion1.setProfesor(profesor1);

Direccion direccion2=new Direccion(4, "Calle Luis lamarca", 45, "Torrente", "Valencia");
Profesor profesor2=new Profesor(4, "Paco", "Moreno", "Díaz");
profesor2.setDireccion(direccion2);
direccion2.setProfesor(profesor2);


Session session=sessionFactory.openSession();
session.beginTransaction();

session.save(profesor1);
session.save(direccion2);

session.getTransaction().commit();
session.close();
```

O exemplo inclúe dous casos:

- Crear un obxecto direccion1 e outro profesor1 (liñas 1 e 2), crear as relacións (liñas 3 e 4) e, finalmente, na liña 16 gardar o obxecto profesor1.

- Crear un obxecto direccion2 e outro profesor2 (liñas 6 e 7), crear as relacións (liñas 8 e 9) e, finalmente, na liña 17 gardar o obxecto direccion2.

En ambos casos el resultado aparente es el mismo , se guarda tanto el objeto `Direccion` como el objeto `Profesor` al ser la relación bidireccional aunque realmente los 2 casos **no** son iguales; veamos ahora el porqué.
En el primer caso, si persistimos un objeto `Profesor` se inserta directamente dicho objeto en la base de datos por lo que **no** puede existir ya la fila pero sí que se permite que la `Direccion` ya exista, actualizándose en dicho caso. Pero en el segundo caso si persistimos el objeto `Direccion` lo que ocurre es lo contrario, no podrá existir la fila de la dirección pero sí podrá existir la fila del `Profesor`.

2.3Un a moitos (desordenada)

A relación un a moitos consiste simplemente en que un obxecto pai teña unha lista sen ordear de outros obxectos fillo de forma que ao persistir o obxecto principal tamén se persista a lista de obxectos fillo. Esta relación tamén adoita chamarse mestre-detalle ou pai-fillo.

Clases Java Antes de entrar en cómo se implementa en Hibernate, vexamos as clases Java e as táboas que definen a relación un a moitos. Para o noso exemplo, imos usar as clases:

- Profesor

- CorreoElectronico

```java
public class Profesor implements Serializable  {
    private int id;
    private String nombre;
    private String ape1;
    private String ape2;
    private Set<CorreoElectronico> correosElectronicos;


    public Profesor(){
    }

    public Profesor(int id, String nombre, String ape1, String ape2) {
        this.id = id;
        this.nombre = nombre;
        this.ape1 = ape1;
        this.ape2 = ape2;
    }

}

public class CorreoElectronico implements Serializable {
    private int idCorreo;
    private String direccionCorreo;
    private Profesor profesor;

    public CorreoElectronico() {

    }

    public CorreoElectronico(int idCorreo,String direccionCorreo,Profesor profesor) {
        this.idCorreo=idCorreo;
        this.direccionCorreo=direccionCorreo;
        this.profesor=profesor;
    }
}
```

No listado 1, pódese observar como a clase Profesor ten unha propiedade chamada "correosElectronicos" da clase CorreoElectronico (líña 6) e ademais, a clase CorreoElectronico tamén posúe unha referencia a Profesor, xa que definimos que a relación é bidireccional desde Profesor a CorreoElectronico e viceversa.

O mecanismo que usamos en Java para almacenar unha serie de obxectos fillo é a interface Set. Non imos usar a interface List ou un array xa que estas formas implican unha ordeación dos obxectos fillo, mentres que usando un Set non hai ningún tipo de orde. Na seguinte lección explícase como usar unha lista ordenada de obxectos fillo.

Táboas

As táboas da base de datos quedarían da seguinte forma:

class Profesor <>
Profesor : INTEGER id
Profesor : VARCHAR nombre
Profesor : VARCHAR ape1
Profesor : VARCHAR ape2

class CorreoElectronico <>

CorreoElectronico: INTEGER idCorreo
CorreoElectronico: VARCHAR direccionCorreo
CorreoElectronico: INTEGER idProfesor

Profesor "1" -- "0..n" CorreoElectronico

Profesor.hbm.xml

O ficheiro Profesor.hbm.xml quedará da seguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
<hibernate-mapping>
  <class name="ejemplo05.Profesor" >
    <id column="Id" name="id" type="integer"/>
    <property name="nombre" />
    <property name="ape1" />
    <property name="ape2" />

    <set name="correosElectronicos"  cascade="all" inverse="true"  >
        <key>
            <column name="idProfesor"  />
        </key>
        <one-to-many class="ejemplo05.CorreoElectronico" />
    </set>
  </class>
</hibernate-mapping>
```

O ficheiro básicamente contén o que se explicou nas leccións anteriores, agás polo tag `**<set>**` (liñas 10 a 15).

Tag `<set>`

O tag `**<set>**` utilízase para definir unha relación un a moitos desordenada entre as dúas clases de Java.

Atributos

- `**name**`: É o nome da propiedade de Java do tipo Set na cal se almacenan todos os obxectos fillo. No noso exemplo, o valor é "correosElectronicos", xa que é a propiedade que contén o Set.

- `**cascade**`: Como xa explicamos en leccións anteriores, este atributo indica que se realizan as mesmas operacións co obxecto pai que cos obxectos fillo, é dicir, se un se borra, tamén os outros, etc. O seu valor habitual é "all".

- `**inverse**`: Este atributo utilízase para minimizar as SQLs que Hibernate lanza contra a base de datos. Neste caso concreto, debe establecerse como true para evitar unha sentenza SQL de UPDATE por cada fillo.

Tags anidados

- `**key**`: Este tag contén outro anidado chamado `column` co atributo `name`, que indica o nome dunha columna da base de datos. Esta columna debe ser da táboa fillo e ser o nome da clave allea á táboa pai. No noso exemplo, é "idProfesor", xa que é o nome da clave allea na táboa CorreoElectronico.

- `**one-to-many**`: Este tag contén o atributo `class` co FQCN (nome cualificado completo) da clase de Java fillo. No noso exemplo, é o nome da clase CorreoElectronico cuxo FQCN é "exemplo05.CorreoElectronico".

CorreoElectronico.hbm.xml

O ficheiro CorreoElectronico.hbm.xml quedaría da seguinte forma:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
<hibernate-mapping>
  <class name="ejemplo05.CorreoElectronico" >
    <id column="IdCorreo" name="idCorreo" type="integer"/>
    <property name="direccionCorreo" />


    <many-to-one name="profesor">
            <column name="idProfesor"  />
    </many-to-one>


  </class>
</hibernate-mapping>
```

O ficheiro básicamente contén o que se explicou nas leccións anteriores, agás polo tag `**<many-to-one>**` (liñas 9 a 11).

Tag `<many-to-one>`

O tag `**<many-to-one>**` utilízase para definir unha relación moitos a un entre as dúas clases de Java.

Atributos

- `**name**`: É o nome da propiedade de Java que enlaza co obxecto pai. No noso exemplo, o valor é "profesor", xa que é a propiedade que contén a referencia á clase Profesor.

Tags anidados

- `**column**`: Este tag contén o atributo `name`, que indica o nome dunha columna da base de datos. Esta columna debe ser da táboa fillo e ser o nome da clave allea á táboa pai. No noso exemplo, é "idProfesor", xa que é o nome da clave allea na táboa CorreoElectronico.

Anotacións

Para usar anotacións, deberemos modificar o código fonte das clases de Java e non usar os ficheiros .hbm.xml.

O código fonte da clase Profesor quedaría da seguinte forma:

```java
@Entity
@Table(name="Profesor")
public class Profesor implements Serializable  {

    @Id
    @Column(name="Id")
    private int id;

    @Column(name="nombre")
    private String nombre;

    @Column(name="ape1")
    private String ape1;

    @Column(name="ape2")
    private String ape2;

    @OneToMany(mappedBy="profesor",cascade= CascadeType.ALL)
    private Set<CorreoElectronico> correosElectronicos;


    public Profesor(){
    }

    public Profesor(int id, String nombre, String ape1, String ape2) {
        this.id = id;
        this.nombre = nombre;
        this.ape1 = ape1;
        this.ape2 = ape2;
    }
}
```

A propiedade `**correosElectronicos**` engadiuse unha anotación para indicar a relación un a moitos.

- `**OneToMany**`: Como o seu nome indica, informa a Hibernate de que esta propiedade conterá a lista de fillos.

- `**mappedBy**`: Este atributo conterá o nome da propiedade de Java da clase fillo que enlaza coa clase pai. No noso exemplo, é o nome da propiedade `**profesor**` que se atopa na clase CorreoElectronico.

- `**cascade**`: Este atributo ten o mesmo significado que o do ficheiro de mapeo de Hibernate.

O código da clase CorreoElectronico é o seguinte:

```java
@Entity
@Table(name="CorreoElectronico")
public class CorreoElectronico implements Serializable {

    @Id
    @Column(name="IdCorreo")
    private int idCorreo;

    @Column(name="DireccionCorreo")
    private String direccionCorreo;

    @ManyToOne
    @JoinColumn(name="IdProfesor")
    private Profesor profesor;

    public CorreoElectronico() {

    }

    public CorreoElectronico(int idCorreo,String direccionCorreo,Profesor profesor) {
        this.idCorreo=idCorreo;
        this.direccionCorreo=direccionCorreo;
        this.profesor=profesor;
    }
}
```

A propiedade `**profesor**` engadiuse dúas anotacións para indicar a relación:

- `**ManyToOne**`: Ao ser o outro lado da relación, indicamos que desde este lado é unha relación moitos a un.

- `**JoinColumn**`: Indicaremos o nome da columna que, na táboa filla, contén a clave allea á táboa pai. No noso exemplo, é a columna da base de datos `**IdProfesor**` que se atopa na táboa CorreoElectronico, a cal enlaza coa táboa Profesor.

Código Java

Agora que xa temos preparadas as clases Java para que poidan persistirse, vexamos o código necesario para persistilas.

```java
Profesor profesor=new Profesor(7, "Sara", "Barrrera", "Salas");
Set<CorreoElectronico> correosElectronicos=new HashSet<>();
correosElectronicos.add(new CorreoElectronico(3, "sara@yahoo.com",profesor));
correosElectronicos.add(new CorreoElectronico(2, "sara@hotmail.com",profesor));
correosElectronicos.add(new CorreoElectronico(1, "sara@gmail.com",profesor));

profesor.setCorreosElectronicos(correosElectronicos);

Session session=sessionFactory.openSession();
session.beginTransaction();

session.save(profesor);


session.getTransaction().commit();
session.close();
```

A explicación do código é a seguinte:

Na liña 1 créase o obxecto Profesor. Na segunda liña créase o obxecto HashSet que implementa a interfaz Set, a cal conterá a lista de fillos. Desde as liñas 3 a 5 créanse os obxectos CorreoElectronico e añádense ao Set. Na liña 7 establécese a relación entre a lista de fillos (CorreoElectronico) e o pai (Profesor). Na liña 12 gardase o obxecto Profesor e automaticamente tamén se gardan os seus fillos.

2.4 Un a moitos (ordenada)

A relación un a moitos consiste simplemente en que un obxecto pai teña unha lista ordenada de outros obxectos fillo de forma que ao persistirse o obxecto principal tamén se persista a lista de obxectos fillo. Esta relación tamén adoita chamarse mestre-detalle ou pai-fillo.

Clases Java

Antes de entrar en cómo se implementa en Hibernate, vexamos as clases Java e as táboas que definen a relación un a moitos. Para o noso exemplo, imos usar as clases:

- Profesor
- CorreoElectronico

Estas dúas clases van ter unha relación un a moitos.

```
public class Profesor implements Serializable  {
    private int id;
    private String nombre;
    private String ape1;
    private String ape2;
    private List<CorreoElectronico> correosElectronicos;


    public Profesor(){
    }

    public Profesor(int id, String nombre, String ape1, String ape2) {
        this.id = id;
        this.nombre = nombre;
        this.ape1 = ape1;
        this.ape2 = ape2;
    }

}

public class CorreoElectronico implements Serializable {
    private int idCorreo;
    private String direccionCorreo;
    private Profesor profesor;

    public CorreoElectronico() {

    }

    public CorreoElectronico(int idCorreo,String direccionCorreo,Profesor profesor) {
        this.idCorreo=idCorreo;
        this.direccionCorreo=direccionCorreo;
        this.profesor=profesor;
    }
}
```

No listado 1, podemos ver como a clase Profesor ten unha propiedade chamada `**correosElectronicos**` da clase `**CorreoElectronico**` (liña 6) e ademais, a clase `**CorreoElectronico**` tamén posúe unha referencia a `Profesor`, xa que definimos que a relación é bidireccional desde `Profesor` a `**Direccion**` e viceversa.

O mecanismo que usamos en Java para almacenar unha serie de obxectos fillo é a interface `**List**`. Usamos esta interface xa que nos permite que os obxectos fillo estean ordenados.

No seguinte diagrama UML, pódese ver a relación desde `Profesor` a `**CorreoElectronico**` e viceversa.class **Profesor**

Profesor : int id
Profesor : String nombre
Profesor : String ape1
Profesor : String ape2

class CorreoElectronico
CorreoElectronico: int idCorreo
CorreoElectronico: String direccionCorreo

Profesor "1" -- "0..n" CorreoElectronico: correosElectronicos {Ordenada}

Tablas

La tablas de base de datos quedarían de la siguiente forma:

class Profesor <>

Profesor : INTEGER id

Profesor : VARCHAR nombre

Profesor : VARCHAR ape1

Profesor : VARCHAR ape2

class CorreoElectronico <>

CorreoElectronico: INTEGER idCorreo

CorreoElectronico: VARCHAR direccionCorreo

CorreoElectronico: INTEGER idProfesor

CorreoElectronico: INTEGER idx

Profesor "1" -- "0..n" CorreoElectronico

Podemos ver cómo a táboa `**CorreoElectronico**` contén como chave allea a chave primaria da táboa `Profesor`, e desta forma establécese a relación un a moitos.

Se ha añadido la columna `idx` la cual almacenará el orden de los objetos `hijos` dentro de la lista. Sin esta columna sería imposible posteriormente saber el orden en el que se encontraban los elementos.

Ficheiro de mapeo ".hbm.xml"

Ao persistir dúas clases, serán necesarios dous ficheiros de persistencia:

- `Profesor.hbm.xml`
- `CorreoElectronico.hbm.xml`

Profesor.hbm.xml

O ficheiro Profesor.hbm.xml quedará da seguiente forma:

1| Fichero Profesor.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://www.hibernate.
<hibernate-mapping>
  <class name="ejemplo05.Profesor" >
    <id column="Id" name="id" type="integer"/>
    <property name="nombre" />
    <property name="ape1" />
    <property name="ape2" />

    <list name="correosElectronicos"  cascade="all" inverse="false"  >
        <key>
            <column name="idProfesor"  />
        </key>
        <list-index>
            <column name="Idx"  />
        </list-index>
        <one-to-many class="ejemplo07.CorreoElectronico" />
    </list>
  </class>
</hibernate-mapping>
```

Tag list

O tag `<**list**>` utilízase para definir unha relación un a moitos entre as dúas clases Java nas que hai unha orde.

Atributos

- `**name**`: É o nome da propiedade Java do tipo List na cal se almacenan todos os obxectos fillos. No noso exemplo, o valor é `**correosElectronicos**` xa que é a propiedade que contén a List.

- `**cascade**`: Como xa explicamos en leccións anteriores, este atributo indica que se realizan as mesmas operacións co obxecto pai ca cos obxectos fillos, isto é, se un se borra, tamén os outros, etc. O seu valor habitual é `**all**`. Máis información en Cascade.

- `**inverse**`: Neste caso, o atributo `**inverse**` debe ter o valor `**false**`, xa que desta forma gardarase o valor da orde de cada obxecto. Se se establecese en `**true**`, non se gardaría o valor.

Recuerda poner el valor del atributo `inverse` a `false`

```
inverse="false"
```

Tags anidados

- `**key**`: Este tag contén outro anidado chamado `**column**` co atributo `**name**` que indica o nome dunha columna da base de datos. Esta columna debe ser da táboa fillo e ser o nome da clave allea á táboa pai. No noso exemplo, é `**idProfesor**`, xa que é o nome da clave allea na táboa CorreoElectronico.

- `**list-index**`**:** Este tag contén outro anidado chamado `**column**` co atributo `**name**` que indica o nome dunha columna da base de datos. Esta columna debe ser da táboa fillo e ser a columna onde se garda a orde que ocupa dentro da lista. No noso exemplo, é `**idx**`.

- `**one-to-many**`: Este tag contén o atributo `**class**` co FQCN da clase Java fillo. No noso exemplo, é o nome da clase CorreoElectronico cuxo FQCN é `**exemplo05.CorreoElectronico**`.

O ficheiro CorreoElectronico.hbm.xml quedará da seguinte forma:

1| Fichero CorreoElectronico.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://www.hibernate.c
<hibernate-mapping>
  <class name="ejemplo05.CorreoElectronico" >
    <id column="IdCorreo" name="idCorreo" type="integer"/>
    <property name="direccionCorreo" />


    <many-to-one name="profesor">
            <column name="idProfesor"  />
    </many-to-one>


  </class>
</hibernate-mapping>
```

El ficheiro básicamente contén o que se explicou nas leccións anteriores, excepto polo tag `<**many-to-one**>` (liñas 9 a 11).

Tag many-to-one

O tag `<**many-to-one**>` úsase para definir unha relación moitos a un entre as dúas clases Java.

Atributos

- `**name**`: É o nome da propiedade Java que enlaza co obxecto pai. No noso exemplo, o valor é `**profesor**`, xa que é a propiedade que contén a referencia á clase Profesor.

Tags anidados

- `**column**`: Este tag contén o atributo `**name**` que indica o nome dunha columna da base de datos. Esta columna debe ser da táboa fillo e ser o nome da clave allea á táboa pai. No noso exemplo, é `**idProfesor**`, xa que é o nome da clave allea na táboa CorreoElectronico.

Anotacións

Para usar anotacións, deberemos modificar o código fonte das clases Java e non usar os ficheiros .hbm.xml.

O código fonte da clase Profesor queda do seguinte xeito:

```java
@Entity
@Table(name="Profesor")
public class Profesor implements Serializable  {

    @Id
    @Column(name="Id")
    private int id;

    @Column(name="nombre")
    private String nombre;

    @Column(name="ape1")
    private String ape1;

    @Column(name="ape2")
    private String ape2;

    @OneToMany(cascade= CascadeType.ALL)
    @JoinColumn(name="IdProfesor")
    @IndexColumn(name="idx")
    private List<CorreoElectronico> correosElectronicos;


    public Profesor(){
    }

    public Profesor(int id, String nombre, String ape1, String ape2) {
        this.id = id;
        this.nombre = nombre;
        this.ape1 = ape1;
        this.ape2 = ape2;
    }
}
```

À propiedade `**correosElectronicos**` engadiuse unha anotación para indicar a relación un a moitos.

- `**OneToMany**`: Como o nome indica, indica a Hibernate que esta propiedade conterá a lista de fillos.

- `cascade`: Este atributo ten o mesmo significado que o do ficheiro de mapeo de Hibernate. Máis información en Cascade.

- `**JoinColumn**`: Indicaremos o nome da columna que, na táboa fillo, contén a clave allea á táboa pai. No noso exemplo, é a columna da base de datos `**IdProfesor**` que se atopa na táboa CorreoElectronico e que enlaza coa táboa Profesor.

- `**IndexColumn**`: Indicaremos o nome da columna que, na táboa fillo, contén a orde dentro da lista de fillos. No noso exemplo, é a columna da base de datos `**idx**` que se atopa na táboa CorreoElectronico.

O código da clase CorreoElectronico é o seguinte:

```java
@Entity
@Table(name="CorreoElectronico")
public class CorreoElectronico implements Serializable {

    @Id
    @Column(name="IdCorreo")
    private int idCorreo;

    @Column(name="DireccionCorreo")
    private String direccionCorreo;

    @ManyToOne
    @JoinColumn(name="IdProfesor")
    private Profesor profesor;

    public CorreoElectronico() {

    }

    public CorreoElectronico(int idCorreo,String direccionCorreo,Profesor profesor) {
        this.idCorreo=idCorreo;
        this.direccionCorreo=direccionCorreo;
        this.profesor=profesor;
    }
}
```

Á propiedade `**profesor**` engadíronse dúas anotacións para indicar a relación:

- `**ManyToOne**`: Ao ser o outro lado da relación, indicamos que desde este lado é unha relación moitos a un.

- `**JoinColumn**`: Indicaremos o nome da columna que, na táboa fillo, contén a clave allea á táboa pai. No noso exemplo, é a columna da base de datos `**IdProfesor**` que se atopa na táboa CorreoElectronico e que enlaza coa táboa Profesor.

O código Java é o seguinte:

1 | Persistiendo la clase Profesor

```java
Profesor profesor=new Profesor(9, "Rosa", "Díaz", "Del Toro");
List<CorreoElectronico> correosElectronicos=new ArrayList<>();
correosElectronicos.add(new CorreoElectronico(3, "rosa@yahoo.com",profesor));
correosElectronicos.add(new CorreoElectronico(2, "rosa@hotmail.com",profesor));
correosElectronicos.add(new CorreoElectronico(1, "rosa@gmail.com",profesor));

profesor.setCorreosElectronicos(correosElectronicos);

Session session=sessionFactory.openSession();
session.beginTransaction();

session.save(profesor);

session.getTransaction().commit();
session.close();
```

2.5 Moitos a moitos

A relación moitos a moitos consiste en que un obxecto A teña unha lista de outros obxectos B e tamén que o obxecto B, á súa vez, teña a lista de obxectos A. Deste xeito, ao persistirse calquera obxecto, tamén se persistirá a lista de obxectos que posúe.

Clases Java

Antes de entrar en como se implementa en Hibernate, vexamos as clases Java e as táboas que definen a relación un a moitos.

Para o noso exemplo imos usar as clases:

- Profesor

- Modulo

Estas dúas clases van ter unha relación moitos a moitos.

```
public class Profesor implements Serializable  {
    private int id;
    private String nombre;
    private String ape1;
    private String ape2;
    private Set<Modulo> modulos=new HashSet();


    public Profesor(){
    }

    public Profesor(int id, String nombre, String ape1, String ape2) {
        this.id = id;
        this.nombre = nombre;
        this.ape1 = ape1;
        this.ape2 = ape2;
    }

}

public class Modulo implements Serializable {
    private int idModulo;
    private String nombre;
    private Set<Profesor> profesores=new HashSet();

    public Modulo() {

    }

    public Modulo(int idModulo, String nombre) {
        this.idModulo = idModulo;
        this.nombre = nombre;

    }
}
```

No listado 1, podemos ver como a clase Profesor ten unha propiedade do tipo Set chamada "modulos" da clase Modulo (liña 6), e ademais, a clase Modulo tamén posúe un Set de obxectos Profesor (liña 24).

O mecanismo que usamos en Java para almacenar a lista de obxectos é a interface Set. Non imos usar a interface List ou un array, xa que estas formas implican unha orde dos obxectos, mentres que usando un Set non hai ningún tipo de orde.

No seguinte diagrama UML, pódese ver a relación entre Profesor e Modulo.

class Profesor

Profesor : int id

Profesor : String nombre

Profesor : String ape1

Profesor : String ape2

class Modulo

Modulo: int idModulo

Modulo: String nombre

Profesor "1" --> "0..n" Modulo: modulos

Profesor "0..n" <-- "1" Modulo: profesores

Tablas

As tablas de base de datos quedarían da siguiente forma:

class Profesor <>

Profesor : INTEGER id

Profesor : VARCHAR nombre

Profesor : VARCHAR ape1

Profesor : VARCHAR ape2

class Modulo <>

Modulo: INTEGER idModulo

Modulo: VARCHAR nombre

class ProfesorModulo <>

ProfesorModulo: INTEGER idProfesor

ProfesorModulo: INTEGER idModulo

Profesor "1" -- "0..n" ProfesorModulo

ProfesorModulo "0..n" -- "1" Modulo

Podemos ver como neste caso as tablas Profesor e Modulo se relacionan mediante a nova tabla ProfesorModulo que contén as claves primarias de ambas táboas.

Fichero de mapeo ".hbm.xml"

Ao persistir dúas clases, serán necesarios dous ficheiros de persistencia:

- Profesor.hbm.xml

- Modulo.hbm.xml

Profesor.hbm.xml

Profesor.hbm.xml quedará da seguinte maneira:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
<hibernate-mapping>
  <class name="ejemplo09.Profesor" >
    <id column="Id" name="id" type="integer"/>
    <property name="nombre" />
    <property name="ape1" />
    <property name="ape2" />

    <set name="modulos" table="ProfesorModulo"  cascade="all" inverse="true"  >
        <key>
            <column name="idProfesor"  />
        </key>
        <many-to-many column="IdModulo" class="ejemplo09.Modulo" />
    </set>
  </class>
</hibernate-mapping>
```

O ficheiro básicamente contén o que se explicou nas leccións anteriores, excepto polo tag <**set**> (liñas 10 a 15).

Tag set

O tag <**set**> utilízase para definir unha lista desordenada entre as dúas clases de Java.

Atributos

- **name**: É o nome da propiedade de Java do tipo Set na cal se almacenan todos os obxectos relacionados. No noso exemplo, o valor é modulos, xa que é a propiedade que contén o Set.

- **table**: É o nome da táboa da base de datos que contén a relación moitos a moitos. No noso exemplo, é a táboa ProfesorModulo.

- **cascade**: Como xa explicamos en leccións anteriores, este atributo indica que se realizan as mesmas operacións co obxecto principal que cos obxectos relacionados, isto é, se un se borra, os outros tamén, etc. O seu valor habitual é all. Máis información en Cascade.

- **inverse**: No caso das relacións moitos a moitos, é necesario poñer un lado da relación co valor true e o outro co valor false. En Profesor poñeremos o valor true e na parte de Modulo estableceremolo a false, aínda que non hai problema en estabelecelo ao revés.

Recuerda poner en este lado de la relación el valor de `inverse="true"` y en el otro lado de la relación `inverse="false"`.
Si ambos valores son `true` no se realizará ninguna inserción en la tabla `ProfesorModulo` y si ambos valores son `false` se realizará dos veces la inserción en la tabla `ProfesorModulo` dando un error de clave primaria duplicada.

Es decir que en este caso `inverse` controla cuándo se realiza la inserción en la tabla `ProfesorModulo`: si se realiza al guardar el `Profesor` o al guardar el `Modulo`.

Tags anidados

- **key**: Este tag contén outro anidado chamado column co atributo name, o cal contén o nome dunha columna da base de datos. Esta columna debe ser unha da táboa da relación moitos a moitos e ser o nome da columna que contén a clave allea da táboa que estamos persistindo. No noso exemplo, é idProfesor, xa que é o nome da clave allea que se atopa na táboa ProfesorModulo.

- **many-to-many:** Este tag contén o atributo class co FQCN (Fully Qualified Class Name) da clase Java coa que se establece a relación. No noso exemplo, é o nome da clase Modulo, cuxo FQCN é exemplo09.Modulo.

Modulo.hbm.xml

O ficheiro Modulo.hbm.xml quedará da seguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "ht
<hibernate-mapping>
  <class name="ejemplo09.Modulo" >
    <id column="IdModulo" name="idModulo" type="integer"/>
    <property name="nombre" />

    <set name="profesores" table="ProfesorModulo"  cascade="all" inverse="false"  >
        <key>
            <column name="idModulo"  />
        </key>
        <many-to-many column="IdProfesor" class="ejemplo09.Profesor" />
    </set>
  </class>
</hibernate-mapping>
```

O ficheiro básicamente contén o que se explicou nas leccións anteriores, agás pola etiqueta <many-to-one> (liñas 8 a 13).

Etiqueta set

A etiqueta <**set**> úsase para definir unha lista desordenada entre as dúas clases de Java.

Atributos

- **name**: É o nome da propiedade de Java do tipo Set na cal se almacenan todos os obxectos relacionados. No noso exemplo, o valor é profesores xa que é a propiedade que contén o Set.

- **table**: É o nome da táboa da base de datos que contén a relación moitos a moitos. No noso exemplo, é a táboa ProfesorModulo.

- **cascade**: Como xa explicamos en leccións anteriores, este atributo indica que se realizan as mesmas operacións co obxecto principal que cos obxectos relacionados, isto é, se un se borra os outros tamén, etc. O seu valor habitual é all. Máis información en Cascade.

- **inverse**: No caso das relacións moitos a moitos é necesario poñer un lado da relación co valor true e o outro co valor false. En Modulo poñeremos o valor a false e na parte de Profesor establécelo a true, aínda que non hai problema en establecelo ao revés.

Tags anidados

-**key:** Este tag contén outro anidado chamado column co atributo name, o cal contén o nome dunha columna da base de datos. Esta columna debe ser unha da táboa da relación moitos a moitos e ser o nome da columna que contén a clave allea da táboa que estamos a persistir. No noso exemplo, é idModulo xa que é o nome da clave allea que se atopa na táboa ProfesorModulo.

-**many-to-many:** Este tag contén o atributo class co FQCN da clase Java coa que se establece a relación. No noso exemplo, é o nome da clase Modulo cuxo FQCN é exemplo09.Profesor.

Anotacións

Para usar anotacións, deberemos modificar o código fonte das clases Java e non usar os ficheiros .hbm.xml.

O código fonte da clase Profesor quedaría do seguinte xeito:

```
@Entity
@Table(name="Profesor")
public class Profesor implements Serializable  {

   @Id
   @Column(name="Id")
   private int id;

   @Column(name="nombre")
   private String nombre;

   @Column(name="ape1")
   private String ape1;

   @Column(name="ape2")
   private String ape2;

   @ManyToMany(cascade = {CascadeType.ALL})
   @JoinTable(name="ProfesorModulo", joinColumns={@JoinColumn(name="IdProfesor")},
inverseJoinColumns={@JoinColumn(name="IdModulo")})
   private Set<Modulo> modulos=new HashSet();



   public Profesor(){
   }

   public Profesor(int id, String nombre, String ape1, String ape2) {
      this.id = id;
      this.nombre = nombre;
      this.ape1 = ape1;
      this.ape2 = ape2;
   }
```

Á propiedade módulos (liña 20) añadíronse dúas anotacións para indicar a relación moitos a moitos.

- **ManyToMany**: Como o nome indica, dicirlle a Hibernate que a propiedade conterá unha lista de obxectos

que participan nunha relación moitos a moitos.

- **cascade**: Este atributo ten o mesmo significado que o do ficheiro de mapeo de Hibernate. Máis información en Cascade.

- **JoinTable**: Esta anotación contén a información sobre a táboa que realiza a relación moitos a moitos.

- **name**: Nome da táboa que realiza a relación moitos a moitos. No noso exemplo é ProfesorModulo.

- **joinColumns**: Contén cada unha das columnas que forman a clave primaria desta clase que estamos a definir. Cada columna indícase mediante unha anotación @JoinColumn e no atributo name contén o nome da columna.

- **inverseJoinColumns**: Contén cada unha das columnas que forman a clave primaria da clase coa que temos a relación. Cada columna indícase mediante unha anotación @JoinColumn e no atributo name contén o nome da columna.

O código da clase Modulo é o seguinte:

```
@Entity
@Table(name="Modulo")
public class Modulo implements Serializable {

    @Id
    @Column(name="IdModulo")
    private int idModulo;

    @Column(name="nombre")
    private String nombre;

    @ManyToMany(cascade = {CascadeType.ALL},mappedBy="modulos")
    private Set<Profesor> profesores=new HashSet();

    public Modulo() {

    }

    public Modulo(int idModulo, String nombre) {
        this.idModulo = idModulo;
        this.nombre = nombre;

    }
}
```

Á propiedade profesores (liña 13) añadíronse dúas anotacións para indicar a relación moitos a moitos.

- ManyToMany: Indica que a propiedade contén unha lista de obxectos que participan nunha relación moitos a moitos.

- cascade: Este atributo ten o mesmo significado que o do ficheiro de mapeo de Hibernate. Máis información en Cascade.

- mappedBy: Contén o nome da propiedade Java da outra clase dende a cal se relaciona con esta. No noso exemplo, é a propiedade modulos.

Al poner el atributo `mappedBy` ya no es necesario incluir la anotación `@JoinTable` ya que dicha información ya se indica en el otro lado de la relación.

Código Java

Agora que xa temos as clases Java listas para ser persistidas, vexamos o código necesario para facelo.

```
Profesor profesor1=new Profesor(11, "Isabel", "Fuertes", "Gascón");
Profesor profesor2=new Profesor(12, "Jose", "Valenciano", "Gimeno");

Modulo modulo1=new Modulo(1, "Sistemas Operativos en Red");
Modulo modulo2=new Modulo(2, "Entornos de desarrollo");
Modulo modulo3=new Modulo(3, "Sistemas Informáticos");

profesor1.getModulos().add(modulo1);
profesor1.getModulos().add(modulo2);
profesor2.getModulos().add(modulo3);

modulo1.getProfesores().add(profesor1);
modulo2.getProfesores().add(profesor1);
modulo3.getProfesores().add(profesor2);


Session session=sessionFactory.openSession();
session.beginTransaction();

session.save(profesor1);
session.save(profesor2);

session.getTransaction().commit();
session.close();
```

A explicación do código é a seguinte:

- Nas liñas 1 e 2 créanse dous obxectos Profesor.

- Nas liñas 4, 5 e 6 créanse tres obxectos Módulo.

- Das liñas 8 a 10, engádense os módulos aos profesores.

- Das liñas 12 a 14, engádense os profesores aos módulos.

- Nas liñas 20 e 21, gárdanse os dous obxectos Profesor e automáticamente tamén se gardan os módulos.

Observa como o código Java segue sendo sinxelo e practicamente non se complica ao gardalo na base de datos. Estamos a engadir a complexidade no ficheiro de mapeo de Hibernate ou nas anotacións.