Relaciones en JPA

Fuente: http://es.debugmodeon.com/articulo/relaciones-en-jpa

Vamos a usar tres anotaciones diferentes: *@OneToOne*, *@OneToMany* y *@ManyToOne*.

Como ejemplo, pensemos en las entidades Organizacion, Proyecto y Estudiante y en las siguientes restricciones:

- Una organización tiene varios proyectos.
- Cada proyecto está asociado a una organización.
- Un proyecto tiene un estudiante.
- Cada estudiante es asignado a un proyecto.

Antes de comenzar, para más información sobre entidades puedes leer este artículo.

La entidad para nuestra organización la declaramos de la siguiente forma:

```
package entity;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;

@Entity
public class Organizacion implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "org_id")
    private int idOrganizacion;

    @Basic
    @Column(name="org_nombre", nullable = false, length = 100)
    private String nombre;

    @OneToMany(fetch=FetchType.LAZY, mappedBy = "organizacion")
    private List<Proyecto> proyectos;

    public Organizacion() {
    }
```

```
    public Organizacion(String nombre) {
        this.nombre = nombre;
    }

    public int getIdOrganizacion() {
        return idOrganizacion;
    }

    public void setIdOrganizacion(int idOrganizacion) {
        this.idOrganizacion = idOrganizacion;
    }

    public String getNombre() {
        return nombre;
    }

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

    public List<Proyecto> getProyectos() {

        return proyectos;
    }

    public void setProyectos(List<Proyecto> proyectos) {
        this.proyectos = proyectos;
    }

    public void addProyecto( Proyecto proyecto ){
        if( proyectos == null ) {
            proyectos = new ArrayList<Proyecto>();
        }
        proyectos.add(proyecto);
    }
}
```

La anotación *@OneToMany* indica que una organización puede contener varios proyectos. La propiedad **cascade** define con qué tipo de operaciones se realizarán operaciones en "cascada", es decir se propagarán a las entidades relacionadas, en nuestro caso a los proyectos. Esta propiedad puede tener los siguientes valores:

- ***CascadeType.PERSIST*** - Cuando persistamos la entidad todas las entidades que contenga esta variable serán persistidas también.
- ***CascadeType.REMOVE*** - Cuando borremos la entidad todas las entidades que contenga esta variable se borrarán del mismo modo.
- ***CascadeType.REFRESH*** - Cuando actualicemos la entidad todas las entidades que contenga esta variable se actualizarán.
- ***CascadeType.MERGE*** - Cuando hagamos un "merge" de la entidad todas las entidades que contenga esta variable realizarán la misma operación.

- ***CascadeType.ALL*** - Todas las operaciones citadas anteriormente.

Las siguientes anotaciones son equivalentes:

```
@OneToMany(
    cascade = {
        CascadeType.PERSIST,
        CascadeType.REMOVE,
        CascadeType.REFRESH,
        CascadeType.MERGE
    }
)
private ArrayList<Proyecto> proyectos;
```

y

```
@OneToMany(cascade = CascadeType.ALL)
private ArrayList<Proyecto> proyectos;
```

La propiedad **mappedBy = "organizacion"** indica el nombre de la entidad *Organizacion* en el objeto *Proyecto*.

La entidad para nuestro proyecto la declaramos de la siguiente forma:

```
package entity;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToOne;

@Entity
public class Proyecto implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "pro_id")
    private int idProyecto;

    @Basic
    @Column(name = "pro_nombre", length = 50, nullable = false)
    private String nombre;

    @ManyToOne(fetch=FetchType.EAGER, cascade=CascadeType.ALL)
    @JoinColumn(name = "org_id", referencedColumnName="org_id", nullable = false)
    private Organizacion organizacion;
```

```
    @OneToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinColumn(name = "est_id", referencedColumnName="est_id", nullable=false)
    private Estudiante estudiante;

    public Proyecto() {
    }

    public Proyecto(String nombre) {
        this.nombre = nombre;
    }

    public Estudiante getEstudiante() {
        return estudiante;
    }

    public void setEstudiante(Estudiante estudiante) {
        this.estudiante = estudiante;
    }

    public int getIdProyecto() {
        return idProyecto;
    }

    public void setIdProyecto(int idProyecto) {
        this.idProyecto = idProyecto;
    }

    public String getNombre() {
        return nombre;
    }

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

    public Organizacion getOrganizacion() {
        return organizacion;
    }

    public void setOrganizacion(Organizacion organizacion) {
        this.organizacion = organizacion;
    }

}
```

Ahora usamos la anotación *@ManyToOne* para indicar la relación. La anotación *@JoinColumn(name="organization_id", nullable = false)* define el nombre de la columna en la tabla de la base de datos y que este no puede ser null.

La entidad para nuestro estudiante la declaramos de la siguiente forma:

```
package entity;
```

```
import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;

@Entity
public class Estudiante implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "est_id")
    private int estudianteId;

    @Basic
    @Column(name = "est_nombre", length = 50, nullable = false)
    private String nombre;

    @OneToOne(fetch=FetchType.EAGER, mappedBy="estudiante")
    private Proyecto proyecto;

    public Estudiante() {
    }

    public Estudiante(String nombre) {
        this.nombre = nombre;
    }

    public int getEstudianteId() {
        return estudianteId;
    }

    public void setEstudianteId(int estudianteId) {
        this.estudianteId = estudianteId;
    }

    public String getNombre() {
        return nombre;
    }

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

    public Proyecto getProyecto() {
        return proyecto;
    }
```

```
    public void setProyecto(Proyecto proyecto) {
        this.proyecto = proyecto;
    }

}
```

Para correr este ejemplo he usado Apache OpenJPA, con lo que el archivo persistence.xml quedará de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="DemoJPACSPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>entity.Estudiante</class>
    <class>entity.Organizacion</class>
    <class>entity.Proyecto</class>
    <properties>
      <property name="hibernate.connection.username" value="root"/>
      <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
      <property name="hibernate.connection.password" value="mysql"/>
      <property name="hibernate.connection.url"
value="jdbc:mysql://localhost:3306/demojpa"/>
      <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.NoCacheProvider"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
    </properties>
  </persistence-unit>
</persistence>
```

Creamos una clase para insertar datos:

```
package pruebas;

import entity.Estudiante;
import entity.Organizacion;
import entity.Proyecto;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class Prueba01 {

    public static void main(String[] args) {

        EntityManagerFactory factory = Persistence.createEntityManagerFactory("DemoJPA");
        EntityManager em = factory.createEntityManager();
        em.getTransaction().begin();

        Organizacion organizacion = new Organizacion("The Apache Software Foundation");

        Proyecto proyecto = new Proyecto("Streaming LOB support (for OpenJPA)");
```

```
        Estudiante estudiante = new Estudiante("Ignacio Andreu");
        estudiante.setProyecto(proyecto);
        proyecto.setEstudiante(estudiante);
        proyecto.setOrganizacion(organizacion);
        organizacion.addProyecto(proyecto);

        em.persist(estudiante);
        em.persist(organizacion);
        em.persist(proyecto);

        proyecto = new Proyecto("Maven Dependency Visualization");
        estudiante = new Estudiante("Peter Kolbus");
        estudiante.setProyecto(proyecto);
        proyecto.setEstudiante(estudiante);
        proyecto.setOrganizacion(organizacion);
        organizacion.addProyecto(proyecto);

        em.persist(estudiante);
        em.persist(organizacion);
        em.persist(proyecto);

        organizacion = new Organizacion("Mono Proyecto");
        proyecto = new Proyecto("Gendarme Tasks");
        estudiante = new Estudiante("Néstor Salceda");
        estudiante.setProyecto(proyecto);
        proyecto.setEstudiante(estudiante);
        proyecto.setOrganizacion(organizacion);
        organizacion.addProyecto(proyecto);

        em.persist(estudiante);
        em.persist(organizacion);
        em.persist(proyecto);

        em.getTransaction().commit();
        em.close();
        factory.close();

    }

}
```

Y finalmente una clase para leer datos:

```
package pruebas;

import entity.Organizacion;
import entity.Proyecto;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;
```

```
public class Prueba02 {

    public static void main(String[] args) {
        List<String> rpta = new ArrayList<String>();
        EntityManagerFactory factory = Persistence.createEntityManagerFactory("DemoJPA");
        EntityManager em = factory.createEntityManager();
        Query q = em.createQuery("select o from Organizacion o");
        for (Organizacion org : (List<Organizacion>) q.getResultList()) {
            rpta.add("Organizacion: " + org.getNombre());
            if (org.getProyectos() != null && org.getProyectos().size() > 0) {
                for (Proyecto p : org.getProyectos()) {
                    rpta.add(" - " + p.getNombre() + " asignado a " +
                        p.getEstudiante().getNombre());
                }
            } else {
                rpta.add("No contiene proyectos todavia");
            }
        }
        em.close();
        factory.close();
        for (String dato : rpta) {
            System.out.println(dato);
        }
    }
}
```

La salida por pantalla sería algo como:

```
Organizacion: The Apache Software Foundation
 - Streaming LOB support (for OpenJPA) asignado a Ignacio Andreu
 - Maven Dependency Visualization asignado a Peter Kolbus
Organizacion:  Mono Proyecto
 - Gendarme Tasks asignado a Néstor Salceda
```

Comentarios

Comentario 1

Practico artículo pero creo que has dejado de lado uno de los conceptos básicos que más trastornan a los neófitos: ¿Cuál es el "owning side" de una relación? ¿Por qué? , ¿Qué implicaciones tiene ser el owning side o el inverse?

En mi experiencia esto es importante y no nos queda totalmente claro desde la documentación de JPA sin haber manejado anteriormente otros ORM .

¿Crees que sería interesante añadirlo al artículo?

Comentario 2

Según mi forma de verlo no existe diferencia entre usar un ORM o no para pensar en qué parte de la relación contiene la otra, quiero decir, si vemos el ejemplo que trata el artículo una organización tiene

varios proyectos y un proyecto sólo tiene una relación, con lo que siendo conscientes de cómo serían las tablas es evidente cómo tenemos que hacer las entidades.

Algo un poco diferente sería una relación uno a uno donde no queremos que una tabla guarde referencia a otra, es decir uno a uno y unidireccional, en ese caso también sería evidente cómo crear las entidades para conseguirlo.

Un ORM te abstrae del manejo a "bajo nivel" de la base de datos, sobre todo de creación de SQL enrevesado pero es necesario saber cómo funciona lo que hay por debajo, al menos a groso modo, para saber qué hacer por arriba.

En Apache OpenJPA existe una forma de manejar las relacione inversas de forma simple, quizás escriba sobre eso, total es muy sencillito.

Crees que sería interesante escribir sobre el owning side de una relación? Si es así quizás podamos intentarlo, o quizás te podrías animar tú si te ves con fuerzas.