Herencia en JPA

Rodrigo García Carmona (v1.1.1)

Las bases de datos relaciones no entienden el concepto de herencia, por lo que usar herencia en JPA puede resultar complicado. No existe una forma única de implementar la herencia en una base de datos, así que deberemos escoger cómo vamos a representarla en el modelo relacional.

JPA ofrece tres formas de modelar la herencia, todos ellos a través de la anotación Inheritance:

- **Una tabla para todas las clases:** *SINGLE_TABLE*
- **Una tabla para cada clase:** *JOINED*
- **Una tabla para cada subclase:** *TABLE_PER_CLASS*

Herencia de Tabla Única

Ésta es la estrategia que se usará por defecto si no especificamos ninguna otra. Es la solución más sencilla y la que suele dar el mejor rendimiento… si podemos modificar nuestras tablas para añadir una columna discriminadora, claro está. Es especialmente útil para la herencia *superpuesta*.

En esta estrategia se usa una única tabla para almacenar todas las instancias de todas las clases que forman parte de la herencia. Esta tabla tendrá una columna para cada atributo de cada clase, además de una columna discriminadora, que se usa para determinar a qué clase corresponde cada una de las filas. Esta columna contendrá un valor que es único para cada clase.

Una estructura de tabla única tiene el siguiente aspecto:

project

id	proj_type	name	budget
1	L	Marketing	5000.0
2	S	Legal	null

Y se implementa usando las siguientes clases:

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="proj_type")
@ForceDiscriminator
@Table(name="projects")
public abstract class Project {
    @Id
    // Other Id annotations as needed
```

```
    private Integer id;
    ...
}

@Entity
@DiscriminatorValue("L")
public class LargeProject extends Project, implements Serializable {
    private Double budget;
}

@Entity
@DiscriminatorValue("S")
public class SmallProject extends Project, implements Serializable {
}
```

Herencia con una Tabla por Clase

Esta estrategia es especialmente adecuada para aquellos modelos relacionales que sigan la estructura definida en el modelo de objetos, por lo que suele utilizarse cuando este último modelo es el que se diseña en primer lugar. Es especialmente útil para la herencia *incompleta* y *disjunta*.

En esta estrategia se define una tabla para cada clase, tanto padre como hija. Cada tabla de una clase hija contiene únicamente los atributos que la diferencian de la clase padre, y la clase padre debe contener además una columna discriminadora, como en la estrategia anterior.

Además, cada tabla debe almacenar la clave primaria del objeto, aunque sólo la definiremos en la clase raíz. Todas las clases de la jerarquía deben compartir el mismo atributo id.

Un conjunto de tablas para esta estrategia tiene el siguiente aspecto:

projects

id	proj_type	name
1	L	Marketing
2	S	Legal

smallprojects

id
2

largeprojects

id	budget
1	5000.0

Y se implementan utilizando las siguientes clases:

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
@DiscriminatorColumn(name="proj_type")
@Table(name="projects")
public abstract class Project {
    @Id
    // Other Id annotations as needed
    private integer id;
    ...
}

@Entity
@DiscriminatorValue("L")
@Table(name="largeprojects")
public class LargeProject extends Project, implements Serializable {
    private Double budget;
}

@Entity
@DiscriminatorValue("S")
@Table(name="smallprojects")
public class SmallProject extends Project, implements Serializable {
}
```

Herencia con una Tabla por Subclase

Utilizaremos esta estrategia cuando queramos mantener la herencia únicamente en el modelo de objetos, haciendo al modelo relacional ignorante de este hecho. Además, es especialmente útil cuando tengamos herencia *completa* y *disjunta*.

En esta estrategia se usa una clase padre abstracta y varias clases hijas. Cada clase hija es representada en el modelo relacional en forma de una tabla que almacena todos sus atributos y todos los atributos de su superclase.

Aquí puede verse un ejemplo de tablas que siguen esta estrategia:

smallprojects

id	name
2	Legal

largeprojects

id	name	budget
1	Marketing	5000.0

Y aquí las clases que utilizamos para implementarlas:

```Java
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public abstract class Project {
@Id
// Other Id annotations as needed
private Integer id;
@Column(name=”name”)
private String name;
…
}

@Entity
@Table(name=”largeprojects”)
public class LargeProject extends Project, implements Serializable {
private Double budget;
}

@Entity
@Table(name=”smallprojects”)
public class SmallProject extends Project, implements Serializable {
}
```