HIBERNATE (Parte mapeo)

Por Óscar Fernández Pastoriza

Componente con bag // set

Colecciones no indexadas.

Como se instancian o declaran

> - `bag` → Se declara como Collection<> y se instancia como ArrayList<>. Admite valores repetidos.
> - `set` → Se declara como Set<> y se instancia como HashSet<>. No admite valores repetidos.

Estructura XML

Parten de una etiqueta principal con dos atributos necesarios, `name=""` y `table=""`.

> - El atributo `name=""` hace referencia al atributo del POJO. **P.ej:** *telefonos*
> - El atributo `table=""` hace referencia a la tabla de la base de datos que tendrá los valores de la lista.

Su primera etiqueta hija debe ser `<key>`, que contendrá un atributo `column=""` o una etiqueta hija `<column>`.

> - Sea como sea, es imprescindible pone referenciar la columna que ejerce como **CLAVE FORÁNEA** desde la `table=""` a la tabla original del XML `<hibernate-mapping table="AQUI_TABLA">`
> - En el caso de la etiqueta hija, se pueden especificar más cosas, como si admite nulos o el tipo de datos.

A continuación, hay tres grandes grupos: Elemento, Elemento Compuesto o Relación one2many (que se describe más adelante)

- Elemento
  - Se representan con una etiqueta `<element>` y cuentan con un atributo `column=""` o etiqueta hija importante `<column>` .
  - La columna debe ser la columna que se quiere referenciar presente en la `table=""` definida anteriormente.
- Elemento Compuesto
  - Se representan con una etiqueta `<composite-element>` y un atributo `class=""` que hace referencia a la clase **.java** que se quiere mapear.
  - Funcionan a partir de aquí como un mapeo normal de una entidad, se deben definir las `<property>` .

Componente con list // idbag

Colecciones indexadas.

Como se instancian o declaran

- `list` → Se declara como List<> y se instancia como ArrayList<>
- `idbag` → Se declara como Collection<> y se instancia como ArrayList<>

Estructura XML del list

Además de la estructura de las colecciones no indexadas, requieren de una etiqueta `<index>` (o también se puede usar `<list-index>` en list). Funcionan de manera similar aunque supuestamente `<list-index>` está mejor optimizada por abajo.

Creo que se puede usar para fechas y otros, aunque lo más básico es usarlo con un entero autoincremental.

- La etiqueta debe tener un atributo `column=""` que hace referencia a la columna que ejercerá como index en esa colección.

- Opcionalmente, aunque recomendado, se puede usar el atributo `base=""` para establecer un índice mínimo desde el que comenzará el incremento.

Estructura XML del idbag

Además de la estructura de las colecciones no indexadas, requieren al principio de una etiqueta `<collection-id>` que indexará la colección y que requiere de lo siguiente:

- Un atributo `type=""` que establece el tipo del index.
- Un atributo `column=""` que establece la columna de la tabla especificada que se usará como index.
- Una etiqueta hija `generator` con un único atributo `class=""` .

ANEXO: Qué tipos de generators

- `identity` ⇒ Se genera automáticamente por la bbdd.
- `assigned` ⇒ Es tu responsabilidad asignarla manualmente antes de persistirlo.
- `increment` ⇒ Se genera automáticamente por Hibernate.
- `foreign` ⇒ Usa la calve primaria de otra tabla como su propia clave.

Componente con map

Funcionan como unas colecciones no indexadas, pero requieren de una etiqueta `<map-key>` que se usará como clave del mapa.

Por ejemplo, en el caso de un mapa de las horas extras:

```
 key column="NSS_Empregado"
map-key column="Fecha" type="java.sql.Date"
element column="Horas" type="java.lang.Double"
```

Fumadas varias (sortedmap y demás)

Opcionalmente, se puede usar el atributo `sort=""` en la etiqueta `<map>` para ordenar el mapa, en ese caso se debe declarar el mapa como un SortedMap<> e instanciarlo como un TreeMap<>.

- El atributo `sort=""` debe apuntar hacia una clase propia que implemente **Comparator<?>** u otros valores por defecto, como `natural` para el orden natural.

Relaciones one2one

Las relaciones one2one básicamente son las que relacionan una entidad con otra.

Se declaran directamente como un objeto y no se instancian, es decir, deben ser referencias nulas.

Es una etiqueta `<one-to-one>` con tres atributos clave.

- El atributo `name=""` debe apuntar directamente a la referencia del atributo del POJO.
- El atributo `class=""` debe apuntar directamente a la clase del atributo del POJO.

El lado Jefe (fuerte)

- El atributo `cascade=""` establece las operación que se deben hacer el lado débil (esclavo) en caso de sufrir una modificación, lo más bruto es poner `"all"`.

El lado esclavo (débil)

- El atributo `constrained="true"` establece que es el lado débil y que está restringido (constrained) a lo que decida el lado fuerte.

❗ Aparentemente no es necesario, pero si muy recomendable que exista una referencia bidireccional. Es decir, si un Empleado tiene un ContactoDeEmergencia, el Empleado debería tener un atributo `contactoDeEmergencia` y el ContactoDeEmergencia debería tener un atributo `empleado`.

Relaciones many2one y one2many

Estas relaciones relacionan colecciones con un objeto o un objeto en una colección.

No es necesario que siempre que exista una many2one, haya un one2many inversa.

many2one

- Esta etiqueta `<many-to-one>` necesita de un atributo en el POJO, que debe ser referenciado a través del atributo `name=""` y con el atributo `class=""` .
- Esta etiqueta debe tener una etiqueta hija `<column>` .

one2many

- Esta etiqueta (`<one-to-many>`) debe encontrarse dentro de una colección, ya sea indexada o no.
- La colección contenedora debe tener el atributo `inverse="true"` sólo si hay una relación many2one inversa.
- La propia etiqueta debe tener el atributo `class=""` que debe apuntar hacia una clase.

Relaciones many2many puras

Es una relación muchos a muchos.

Deben encontrarse dentro de una colección.

- Deben tener un atributo `class=""` que referencie a la clase con la que se relaciona.
- Deben tener una etiqueta hija `<column>` o un atributo `column=""` que establece cual es la clave foránea que se usará para relacionar esa clase con la tabla definida en la colección.

Relaciones many2many con atributos

Es una relación muchos a muchos que tiene atributos en la relación.

Empezando, necesitaremos dos POJOS más, uno que hará realmente de relación y otro que establece la clave o el id.

- El POJO con id debe tener como únicos campos las claves de las dos entidades a relacionar. P.ej: codProxecto y codDepto.
- El POJO que ejercerá de relación deberá tener los siguientes campos:
  - PojoID id → Establecerá su propia ID.
  - EntidadA entidadA → P.ej: Proxecto
  - EntidadB entidadB → P.ej: Departamento
  - Valores extra → P.ej: Double horasDedicadas

Ahora pasemos realmente con el mapeo y en qué consiste:

Únicamente necesitaremos un fichero hbm para mapear todo:

- Tendrá una clave que tendrá la siguiente estructura:

```
<composite-id name="id" class="POJOS.EmpregadoProxectoId">
      <key-property name="nssempregado" type="string">
            <column name="NSSEmpregado" length="15"/>
      </key-property>
      <key-property name="numProxecto" type="int">
            <column name="NumProxecto"/>
      </key-property>
/composite-id>
```

- Además, se establecerá como `<property>` cada propiedad o valor extra.
- Por último, tendrá una relación `<many-to-one>` a cada uno de las entidades que participan en la relación.

❗ Además, en cada entidad que participa, se deberá establecer un atributo del POJO (P.ej: EmpregadoProxecto empregadosProxectos) y una relación `<one-to-many>` dentro de la correspondiente colección.

Relaciones reflexivas

- Básicamente son una `<many-to-one>` o `<one-to-many>` sobre la misma entidad.

Componentes

Los componentes albergan ciertas columnas de la misma tabla que están relacionadas en un POJO a parte, en vez de mapear todo directamente como properties.

P.ej:

- Si queremos guardar la dirección de un empleado, meteremos las columas relativas a la dirección en un POJO Dirección a parte, en vez de tener en el empleado los campos de la dirección. Entendamos que la dirección es única para un empleado.