JPA - difference in the use of the mappedBy property to define the owning entity

What exactly is the difference in the following two declarations

B is the owning side

```
@Entity
class A {
   @Id int id;

   @OneToOne
   B b;
}

@Entity
class B {
   @Id int id;

   @OneToOne(mappedBy="b")
   A a;
}
```

A is the owning side

```
@Entity
class A {
   @Id int id;

   @OneToOne(mappedBy="a")
   B b;
}

@Entity
class B {
   @Id int id;
```

```
    @OneToOne
    A a;
}
```

Thinking of this in "normal SQL" i think it is the same as having two tables each having the other table's foreign key. What i don't understand though is what is the effect of specifying which entity is the owning side i.e using the 'mappedBy' property. What does this actually achieve as i don't believe there is an equivalent in normal SQL.

java hibernate jpa persistence jpql

asked Jun 10 '12 at 12:06

ziggy
7,152 52 160 256

2 Did you check that in both cases the A and B tables will have FK to each other? – Piotr Nowicki Jun 10 '12 at 12:31

4 Answers

The JPA 2.0 specification, section 2.9, writes:

> Relationships may be bidirectional or unidirectional. A bidirectional relationship has both an owning side and an inverse (non-owning) side. A unidirectional relationship has only an owning side. The owning side of a relationship determines the updates to the relationship in the database, as described in section 3.2.4.
>
> The following rules apply to bidirectional relationships:
>
> - The inverse side of a bidirectional relationship must refer to its owning side by use of the `mappedBy` element of the `OneToOne`, `OneToMany`, or `ManyToMany` annotation. The `mappedBy` element designates the property or field in the entity that is the owner of the relationship.
> - The many side of one-to-many / many-to-one bidirectional relationships must be the owning side, hence the `mappedBy` element cannot be specified on the `ManyToOne` annotation.
> - For one-to-one bidirectional relationships, the owning side corresponds to the side that contains the corresponding foreign key.

> - For many-to-many bidirectional relationships either side may be the owning side.

The relevant parts of section 3.2.4 are:

> The state of persistent entities is synchronized to the database at transaction commit. This synchronization involving writing to the database any updates to persistent entities and their relationships as specified above.

and

> Bidirectional relationships between managed entities will be persisted based on references held by the owning side of the relationship. It is the developer's responsibility to keep the in-memory references held on the owning side and those held on the inverse side consistent with each other when they change. In the case of unidirectional one-to-one and one-to-many relationships, it is the developer's responsibility to insure that the semantics of the relationships are adhered to.
>
> > *It is particularly important to ensure that changes to the inverse side of a relationship result in appropriate updates on the owning side, so as to ensure the changes are not lost when they are synchronized to the database.*

answered Jun 10 '12 at 12:59

meriton

49.8k 13 72 138

As other have pointed out, you are wrong about which side is the owning side in your examples. With owning side we mean owning the relationship from an OO perspecitve, in practise that quite often ends up being the opposite of how it is or will be generated in the db if one uses a rdbm as persistence provider.

In normal circumstances the OO model makes it quite clear which sides is the owning side. For example an Order has OrderLines. If we delete an Order all Orderlines should be deleted. If we delete an OrderLine the Order possibly still has a right to existence. Hence, the Order is the owning side

Order possibly still has a right to existence. Hence, the Order is the owning side.

For a more concrete and excellent example, on the effects of which side is the owning side, I refer to @JB Nizet answer.

According to section 2.9 of the JPA 2.0 spec:

> For one-to-one bidirectional relationships, the owning side corresponds to the side that contains the corresponding foreign key.

But in the same section we also have:

> In addition, this specification also requires support for the following alternative mapping strategies: [..] The mapping of unidirectional and bidirectional one-to-one relationships, bidirectional many-to-one/one-to-many relationships, and unidirectional many-to-one relationships by means of join table mappings.

A bit futher down in the same section it continues with:

> Additional mapping annotations (e.g., column and table mapping annotations) may be speci- fied to override or further refine the default mappings and mapping strategies described in Section 2.10. Some implementations make use of that to allow the FK of a birectional OneToOne to be in the target table.

To read some about some strategies to solve that scenario, see: An almost good explaination

I haven't checked but I do hope and believe that 2.1 will remove the first quote. Since the actual database structure should put as little limit as possible on how we can model data as entities.

edited Jun 10 '12 at 15:02

answered Jun 10 '12 at 13:05

esej

2,701 1 11 21

No, JPA specifies that the owning side of a OneToOne bidirectional associatiuon is the one which contains the foreign key. Same for bidirectional OneToMany. – JB Nizet Jun 10 '12 at 13:36

"and it may be used in a OneToOne mapping in which the primary key of the referencing entity is used as a foreign key to the referenced entity." docs.oracle.com/javaee/6/api/javax/persistence/… . The situation *just* doesn't appear so often. – esej Jun 10 '12 at 14:22

And that doesn't change anything. In this case, the primary key column is the join column (hence the name PrimaryKeyJoinColumn), and the entity containing this annotation, and thus the join column, is the owning side. Read @merlyn's answer, containing this excerpt from the JPA spec: "For one-to-one bidirectional relationships, the owning

@merlon's answer, containing this excerpt from the JPA spec: "For one-to-one bidirectional relationships, the owning side corresponds to the side that contains the corresponding foreign key." – JB Nizet Jun 10 '12 at 14:25

1 Yes. I stand corrected. I can do it in EclipseLink, and I totally missed it is not according to 2.0 final spec. The wording in the spec kinda makes me feel the implementation(s) are breaking the spec. I'll update my answer to get rid of the lie. Thank you. – esej Jun 10 '12 at 14:43

In the first example the `A` table is going to have 2 columns `id` and `b_id`, the `B` table is going to have one column, `id`. This makes `A` the owning side.

In the second example `B` is the owning side. `B` has two columns, `id` and `a_id`. `A` is going to have one column, `id`.

And that is the difference :-)

answered Jun 10 '12 at 12:25

siebz0r
7,236 6 40 86

The owning side is the side that JPA considers to know is the association exists or not. Suppose you go with your first example. The owning side is the side where there is no mappedBy attribute. The owning side is thus A, and not B.

This means that if you have an A and a B in database, and you do

```
A a = em.find(A.class, aId);
B b = em.find(B.class, bId);
a.setB(b);
```

JPA will save the association (i.e. it will store the ID of B in the join column of table A).

But if you do

```
A a = em.find(A.class, aId);
B b = em.find(B.class, bId);
b.setA(a);
```

nothing will be changed in database, because you modified the inverse side and forgot to modify the owning side.

answered Jun 10 '12 at 12:41

JB Nizet
505k 49 799 947