JPA Implementation Patterns: Removing Entities

by Vincent Partington MVB · **Aug. 05, 09 · Database Zone · Tutorial**

Just like retrieving an entity, removing an entity is pretty simple. In fact it's all you need to do is pass the entity to the EntityManager.remove method to remove the entity from the database when the transaction is committed (Of course you'd actually invoke a remove method on your DAO which in turn invokes EntityManager.remote). That's all there is to it. Usually. Because when you're using associations (be they bidirectional or not) things get more interesting.

Removing entities that are part of an association

Consider the example with the Order and OrderLine classes we've discussed previously. Let's say we want to remove and OrderLine from an Order and we go about it in this simple manner:

```
orderLineDao.remove(lineToDelete);
```

There is a problem with this code. When you tell the entity manager to remove the entity, it will *not* automatically be removed from any associations that point to it. Just like JPA does not automatically manage bidirectional associations. In this case that would be the orderLines set in the Order object pointed to by the OrderLine.order property. If I were to word this statement as a failing JUnit test case, it would be this one:

```
OrderLine orderLineToRemove = orderLineDao.findById(someOrderLineId);
Order parentOrder = orderLineToRemove.getOrder();
int sizeBeforeRemoval = parentOrder.getOrderLines().size();
orderLineDao.remove(orderLineToRemove);
assertEquals(sizeBeforeRemoval - 1, parentOrder.getOrderLines().size());
```

Implications

The failure of this test case has two subtle and therefore nasty implications:

- Any code that uses the Order object *after* we have removed the OrderLine but will still see that removed OrderLine. Only after committing the transaction, starting a new transaction, *and* reloading the Order in a new transaction, it will *not* show up in the Order.orderLines set anymore. In simple scenarios we won't run into this problem, but when things get more complex we can be surprised by these "zombie" OrderLines appearing.
- When the PERSIST operation is cascaded from the Order class to the Order.orderLines association and the containing Order object is not removed in the same transaction, we will receive an error such as "deleted entity passed to persist". Different from the "detached entity passed to persist" error we talked about in a previous blog, this error is caused by the fact that the Order object has a reference to an already deleted OrderLine object. That reference is then discovered when the JPA provider flushes the entities in the persistence context to the database, causing it try and persist the already deleted entity. And hence the error appears.

The simple solution

To remedy this problem, we *also* have to remove the OrderLine from the Order.orderLines set. That sounds awfully familiar... In fact, when managing bidirectional associations we also had to make sure both sides of the association were in a consistent state. And that means we can reuse the pattern we used there. By adding an invocation of orderLineToRemove.setOrder(null); to the test it will succeed:

```
OrderLine orderLineToRemove = orderLineDao.findById(someOrderLineId);
Order parentOrder = orderLineToRemove.getOrder();
int sizeBeforeRemoval = parentOrder.getOrderLines().size();
orderLineToRemove.setOrder(null);
orderLineDao.remove(orderLineToRemove);
assertEquals(sizeBeforeRemoval - 1, parentOrder.getOrderLines().size());
```

The pattern

But doing it like this makes our code brittle as it depends on the users of our domain objects to invoke the right methods. The DAO should be responsible for this. A very nice way to solve this problem is to use the @PreRemoveentity lifecycle hook like this:

```
@Entity
public class OrderLine {
[...]

@PreRemove
public void preRemove() {
```

```
setOrder(null);
}
}
```

Now we can just invoke OrderLineDao.remove() to get rid of an unwanted OrderLine object.

Originally this blog suggested introducing a HasPreRemove interface with a preRemove method that would be invoked by the DAO. But Sakuraba commented below that the @PreRemove annotation is just the thing we need here. Thanks again, Sakuraba!

Removing orphans

But what, you ask, would happen if we were to just remove the OrderLine from the Order.orderLines set like so:

```
Order parentOrder = orderLineToRemove.getOrder();
parentOrder.removeOrderLine(orderLineToRemove);
```

?

The OrderLine would indeed be removed from the Order.orderLines set. And not just in this transaction. If we retrieve the Order object again in a new transaction, the removed OrderLine still would not show up. But if we were to look in the database, we would see that the OrderLine is still there. It just has its OrderLine.order field set to null. What we are seeing here is an "orphaned" set element. There are two ways to solve this problem:

- Explicitly remove the OrderLine object as we discussed above.
- If you are using Hibernate as your JPA provider, you can let Hibernate remove these orphans automatically. Add a org.hibernate.annotations.Cascade annotation with a value of a org.hibernate.annotations.CascadeType.DELETE_ORPHAN next to the @OneToMany for which you want Hibernate to do this. See the Hibernate documentation for an example.

While the second solution is vendor-specific it does have the nice feature of not requiring your code to invoke a DAO remove method every time you remove an entity from a set. But to make it obvious you are using a vendor specific extension, you should refer to those annotations using the full package name (as Java Persistence with Hibernate suggests too):

```
@OneToMany(mappedBy = "order", cascade = CascadeType.ALL)
@org.hibernate.annotations.Cascade(org.hibernate.annotations.CascadeType.DELETE_ORPHAN)
private Set<OrderLine> orderLines = new HashSet<OrderLine>();
```

Please note that CascadeType.ALL does not include the DELETE_ORPHAN cascade type. That's why the example here explicitly sets it.

So we can conclude that removing entities is simple unless you are dealing with associations. In that case you need to take extra precautions to make sure the entity is removed from any objects referring to it **and** *from the database at the same time. See you at the next blog! In the meantime, please drop any remarks in the comment section below.*

From http://blog.xebia.com

Like This Article? Read More From DZone

One Big DAO, or One DAO Per Table/Object?

JPA - Basic Projections

A Beginner's Guide to ACID and Database Transactions

Free DZone Refcard

Graph-Powered Search: Neo4j & Elasticsearch

Topics: JAVA , PERSISTENCE

Hibernating Rhinos

Example schemas and queries for hybrid data models based on relational + JSON
MariaDB