Moving the Curve

Technology, code, and thoughts by Stevi Deter

SaveOrUpdate versus Merge in Hibernate

We all have those problems that we encounter just infrequently enough that when we see them again, we know we've solved this, but can't remember how.

The NonUniqueObjectException thrown when using Session.saveOrUpdate() in Hibernate is one of mine. I'll be adding new functionality to a complex application. All my unit tests work fine. Then in testing the UI, trying to save an object, I start getting an exception with the message "a different object with the same identifier value was already associated with the session." Here's some example code from Java Persistence with Hibernate.

```
Session session = sessionFactory1.openSession();
Transaction tx = session.beginTransaction();
Item item = (Item) session.get(Item.class, new Long(1234));
tx.commit();
session.close(); // end of first session, item is detached

item.getId(); // The database identity is "1234"
item.setDescription("my new description");
Session session2 = sessionFactory.openSession();
Transaction tx2 = session2.beginTransaction();
Item item2 = (Item) session2.get(Item.class, new Long(1234));
session2.update(item); // Throws NonUniqueObjectException
tx2.commit();
session2.close();
```

To understand the cause of this exception, it's important to understand detached objects and what happens when you call saveOrUpdate() (or just update()) on a detached object.

When we close an individual Hibernate Session, the persistent objects we are working with are detached. This means the data is still in the application's memory, but Hibernate is no longer responsible for tracking changes to the objects.

If we then modify our detached object and want to update it, we have to reattach the object. During that reattachment process, Hibernate will check to see if there are any other copies of the same object. If it finds any, it has to tell us it doesn't know what the "real" copy is any more. Perhaps other changes were made to

those other copies that we expect to be saved, but Hibernate doesn't know about them, because it wasn't managing them at the time.

Rather than save possibly bad data, Hibernate tells us about the problem via the NonUniqueObjectException.

So what are we to do? In Hibernate 3, we have merge() (in Hibernate 2, use saveOrUpdateCopy()). This method will force Hibernate to copy any changes from other detached instances onto the instance you want to save, and thus merges all the changes in memory before the save.

```
Session session = sessionFactory1.openSession();
Transaction tx = session.beginTransaction();
Item item = (Item) session.get(Item.class, new Long(1234));
tx.commit();
session.close(); // end of first session, item is detached

item.getId(); // The database identity is "1234"
item.setDescription("my new description");
Session session2 = sessionFactory.openSession();
Transaction tx2 = session2.beginTransaction();
Item item2 = (Item) session2.get(Item.class, new Long(1234));
Item item3 = session2.merge(item); // Success!
tx2.commit();
session2.close();
```

It's important to note that merge returns a reference to the newly updated version of the instance. It isn't reattaching item to the Session. If you test for instance equality (item == item3), you'll find it returns false in this case. You will probably want to work with item3 from this point forward.

It's also important to note that the Java Persistence API (JPA) doesn't have a concept of detached and reattached objects, and uses EntityManager.persist() and EntityManager.merge().

I've found in general that when using Hibernate, saveOrUpdate() is usually sufficient for my needs. I usually only need to use merge when I have objects that can have references to objects of the same type. Most recently, the cause of the exception was in the code validating that the reference wasn't recursive. I was loading the same object into my session as part of the validation, causing the error.

Where have you encountered this problem? Did merge work for you or did you need another solution? Do you prefer to always use merge, or prefer to use it only as needed for specific cases?

No related posts.

This entry was posted in hibernate, Java and tagged hibernate, merge, NonUniqueObjectException, ORM, saveOrUpdate on December 7, 2008 [http://www.stevideter.com/2008/12/07/saveorupdate-versus-merge-in-hibernate/] .

65 thoughts on “SaveOrUpdate versus Merge in Hibernate”

Sambath P
November 18, 2011 at 10:05 pm

Thanks for the clear explanation of when to use merge.

Manjul
February 17, 2012 at 1:04 am

Nice explanation..

ysf
February 17, 2012 at 6:06 am

very good explantion saveOrUpdate and merge command’s difference.

bharti
March 26, 2012 at 4:17 am

your post cleared my fundamentals on merge.