HIbernate commit() and flush()

I googled a lot and read about `org.hibernate.Transaction.commit()` and `org.hibernate.Session.flush()` a lot, know purpose of each method, but still have a question.

Is it good practice to call `org.hibernate.Session.flush()` method by hand? As said in `org.hibernate.Session` docs,

> Must be called at the end of a unit of work, before commiting the transaction and closing the session (depending on flush-mode, Transaction.commit() calls this method).

Could you explain me purpose of calling `org.hibernate.Session.flush()` by hand if `org.hibernate.Transaction.commit()` will call it automatically?

Thanks!

java hibernate orm

asked Jan 29 '13 at 11:29
bsiamionau
4,360 3 22 49

You could inject the `sessionFactory` using the `@Transactional` annotation, If you looked into it. You would not need your code to be transactional everywhere then onwards (it requires in some situations though). – Lion Jan 29 '13 at 11:42

I just came across an interesting situation in my code, I had to merge an entity that already had an id generated from the client app, and then refresh the same entity to get database generated fields such as date created and modified. Without calling session.flush() it would throw an object not found exception because the merge call is ignored sometimes untill the end of the transaction, when I added flush just after the merge call it forces hibernate to do the actual query and then later refreshing the object works because it now exists in the db! – inkalimeva Sep 28 at 13:58

6 Answers

In the Hibernate Manual you can see this example

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();


for ( int i=0; i<100000; i++ ) {
    Customer customer = new Customer(.....);
    session.save(customer);
    if ( i % 20 == 0 ) { //20, same as the JDBC batch size
        //flush a batch of inserts and release memory:
        session.flush();
        session.clear();
    }
}

tx.commit();
session.close();
```

Without the call to the flush method, your first-level cache would throw an OutOfMemoryException

Also you can look at this post about flushing

edited Feb 10 '14 at 9:59
Eugenio Laghi
154 1 2 18

answered Jan 29 '13 at 11:38
Aleksei Bulgak
2,379 3 25 46

could you give reference to this article? – bsiamionau Jan 29 '13 at 11:41

1 docs.jboss.org/hibernate/orm/4.1/devguide/en-US/html/... – Aleksei Bulgak Jan 29 '13 at 11:41

This answer gives a specific case in which it's a good idea to call flush explicitly, but I don't think it addresses the general question of whether one should always do so. – Dave L. Jan 28 at 20:42

One common case for explicitly flushing is when you create a new persistent entity and you want it to have an artificial primary key generated and assigned to it, so that you can use it later on in the same transaction. In that case calling flush would result in your entity being given an id.

Another case is if there are a lot of things in the 1st-level cache and you'd like to clear it out periodically (in order to reduce the amount of memory used by the cache) but you still want to commit the whole thing together. This is the case that Aleksei's answer covers (+1 from me).