How to maintain Hibernate cache consistency running two Java applications?

Asked 16 years, 3 months ago Modified 12 years ago Viewed 13k times

10

Our design has one jvm that is a jboss/webapp (read/write) that is used to maintain the data via hibernate (using jpa) to the db. The model has 10-15 persistent classes with 3-5 levels of depth in the relationships.

We then have a separate jvm that is the server using this data. As it is running continuously we just have one long db session (read only).

There is currently no intra-jvm cache involved - so we manually signal one jvm from the other.

Now when the webapp changes some data, it signals the server to reload the changed data. What we have found is that we need to tell hibernate to purge the data and then reload it. Just doing a fetch/merge with the db does not do the job - mainly in respect of the objects several layers down the hierarchy.

Any thoughts on whether there is anything fundamentally wrong with this design or if anyone is doing this and has had better luck with working with hibernate on the reloads.

Thanks, Chris

java hibernate caching

Share

Improve this question

Follow

edited Nov 14, 2008 at 14:42

erickson

269k ● 59 ● 401 ● 497

asked Sep 7, 2008 at 20:09

Chris Kimpton

5,541 ● 6 ● 48 ● 73

4 Answers

Sorted by: Highest score (default)

14

A Hibernate session loads all data it reads from the DB into what they call the *first-level cache*. Once a row is loaded from the DB, any subsequent fetches for a row with the same PK will return the data from this cache. Furthermore, Hibernate gaurentees reference equality for objects with the same PK in a single Session.

From what I understand, your read-only server application never closes its Hibernate session. So when the DB gets updated by the read-write application, the Session on read-only server is unaware of the change. Effectively, your read-only application is loading an in-memory copy of the database and using that copy, which gets stale in due course.

The simplest and best course of action I can suggest is to close and open Sessions as needed. This sidesteps the whole problem. Hibernate Sessions are intended to be a window for a short-lived interaction with the DB. I agree that there is a performance gain by not reloading the object-graph again and again; but you need to measure it and convince yourself that it is worth the pains.

Another option is to close and reopen the Session periodically. This ensures that the read-only application works with data not older than a given time interval. But there definitely is a window where the read-only application works with stale data (although the design guarantees that it gets the up-to-date data eventually). This might be permissible in many applications - you need to evaluate your situation.

The third option is to use a *second level cache* implementation, and use short-lived Sessions. There are various caching packages that work with Hibernate with relative merits and demerits.

answered Sep 8, 2008 at 19:45

Binil Thomas

13.8k ● 10 ● 58 ● 70

Using JPA, and @PersistenceContext to get the EntityManager - but it seems we need @PersistenceUnit for an EntityManagerFactory and to refresh the data, we get a new EntityManager. Sounds fine - but heavy handed way to get things updated, but if thats what people do...

– Chris Kimpton Sep 10, 2008 at 13:50

5

Chris, I'm a little confused about your circumstances. If I understand correctly, you have a both a web app (read/write) a standalone application (read-only?) using Hibernate to access a shared database. The changes you make with the web app aren't visible to the standalone app. Is that right?

If so, have you considered using a different second-level cache implementation? I'm wondering if you might be able to use a clustered cache that is shared by both the web application and the standalone application. I believe that SwarmCache, which is integrated with Hibernate, will allow this, but I haven't tried it myself.

In general, though, you should know that the contents of a given cache will never be aware of activity by another application (that's why I suggest having both apps share a cache). Good luck!

Share Improve this answer

Follow

answered Sep 7, 2008 at 21:16

erickson

269k ● 59 ● 401 ● 497

2

From my point of view, you should change your underline Hibernate cache to that one, which supports clustered mode. It could be a JBoss Cache or a Swarm Cache. The first one has a better support of data synchronization (replication and invalidation) and also supports JTA.

Then you will able to configure cache synchronization between webapp and server. Also look at isolation level if you will use JBoss Cache. I believe you should use *READ_COMMITTED* mode if you want to get new data on a server from the same session.

Share Improve this answer

Follow

answered Feb 22, 2009 at 7:40

Andrey Vityuk

1,009 ● 1 ● 7 ● 12

1

The most used practice is to have a Container-Managed Entity Manager so that two or more applications in the same container (ie Glassfish, Tomcat, Websphere) can share the same caches. But if you don't use an Application container, because you use Play! for instance, then I would build some webservices in the *primary Application* to read/write consistently in the cache.

I think using stale data is an open door for disaster. Just like Singletons become Multitons, read-only applications are often a *write sometimes*.

Belt and braces :)

Share Improve this answer

Follow

answered Nov 27, 2012 at 7:30

Nicolas Zozol

7,018 ● 4 ● 54 ● 75