06: Hibernate First & second level cache interview questions and answers

Posted on December 22, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Q. What is a first-level cache in Hibernate?
A. First-level cache always is associated with a “Session” object. Hibernate uses this cache by default. You can’t turn it off. Hibernate caches the SQL statements like insert, update, delete, etc in the first-level cache. By default, Hibernate will flush changes automatically for you

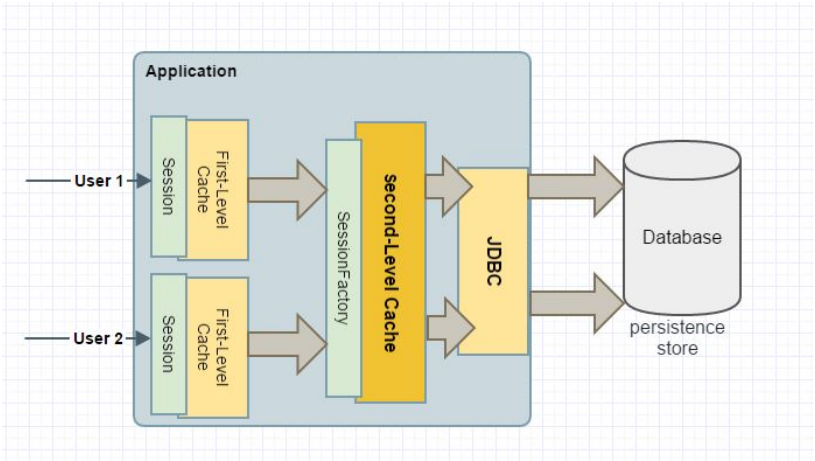
1) before some query executions.
2) when a transaction is committed.

When you **explicitly** call **flush()** you force hibernate to execute the SQL commands on Database. But do understand

that changes are not “committed” yet. So, after doing flush and before doing commit, if you access DB directly, you will not see the changes.

Q. When do you need to explicitly call flush?
A. 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 reason is when you do bulk batch inserts to conserve memory by regularly flushing the cache.



First & Second level cache in Hibernate

Q. What is a second-level cache in Hibernate?
A. Hibernate uses two different caches for objects: first-level cache and second-level cache. First-level cache is associated with the Session object, while second-level cache is associated with the *SessionFactory* object. By default, Hibernate uses first-level cache on a per-transaction basis. Hibernate uses this cache mainly to reduce the number of SQL queries it needs to generate within a given transaction. For example, if an object is modified several times within the same transaction, Hibernate will generate only one SQL UPDATE statement at the end of the transaction, containing all the modifications. The second-level cache needs to be explicitly configured. Hibernate provides a flexible concept to exchange cache providers for the second-level cache. By

- Scala Interview Q&As
- Spring, Hibernate, & M
  - Spring (18)
  - Hibernate (13)
    - 01: ♥♦ 15+ Hiber
    - 01b: ♦ 15+ Hiber
    - 02: Understandir
    - 03: Identifying ar
    - 04: Identifying ar
    - 05: Debugging H
    - 06: Hibernate Fir
    - 07: Hibernate mi
    - 08: Hibernate au
    - 09: Hibernate en
    - 10: Spring, Java
    - 11: Hibernate de
    - 12: Hibernate cu
  - AngularJS (2)
  - Git & SVN (6)
  - JMeter (2)
  - JSF (2)
  - Maven (3)
- Testing & Profiling/Sa
- Other Interview Q&A f
- Free Java Interviev

16 Technical Key Areas

open all | close all
- Best Practice (6)
- Coding (26)
- Concurrency (6)
- Design Concepts (7)
- Design Patterns (11)
- Exception Handling (3
- Java Debugging (21)
- Judging Experience Ir
- Low Latency (7)
- Memory Managemen

default Ehcache is used as caching provider. However more sophisticated caching implementation can be used like the distributed JBoss Cache or Oracle Coherence.

The Hibernate configuration looks like:

```
1 <property name="hibernate.cache.use_second_level_
2 <property name="hibernate.cache.provider_class">o
3
```

The ehcache.xml can be configured to cache objects of type com.myapp.Order as shown below

```
1  <cache name="com.myapp.Order"
2     maxElementsInMemory="300"
3     eternal="true"
4     overflowToDisk="false"
5     timeToIdleSeconds="300"
6     timeToLiveSeconds="300"
7     diskPersistent="false"
8     diskExpiryThreadIntervalSeconds="120"
9     memoryStoreEvictionPolicy="LRU"
10 />
11
```

second-level cache reduces the database traffic by caching loaded objects at the SessionFactory level between transactions. These objects are available to the whole application, not just to the user running the query. The 'second-level' cache exists as long as the session factory is alive. The second-level cache holds on to the 'data' for all properties and associations (and collections if requested) for individual entities that are marked to be cached. It is imperative to implement proper cache expiring strategies as caches are never aware of changes made to the persistent store by another application. he following are the list of possible cache strategies.

- **Read-only**: This is useful for data that is read frequently, but never updated. This is the most simplest and best-performing cache strategy.

⊞ Performance (13)
⊞ QoS (8)
⊞ Scalability (4)
⊞ SDLC (6)
⊞ Security (13)
⊞ Transaction Managen

80+ step by step Java Tutorials

open all | close all
⊞ Setting up Tutorial (6)
⊞ Tutorial - Diagnosis (2
⊞ Akka Tutorial (9)
⊞ Core Java Tutorials (2
⊞ Hadoop & Spark Tuto
⊞ JEE Tutorials (19)
⊞ Scala Tutorials (1)
⊞ Spring & HIbernate Tu
⊞ Tools Tutorials (19)
⊞ Other Tutorials (45)

100+ Java pre-interview coding tests

open all | close all
⊞ Can you write code? (
⊞ ♦ Complete the given
⊞ Converting from A to I
⊞ Designing your classe
⊞ Java Data Structures
⊞ Passing the unit tests
⊞ What is wrong with th
⊞ Writing Code Home A
⊞ Written Test Core Jav
⊞ Written Test JEE (1)

- **Read/write**: Read/write caches may be appropriate if your data needs to be updated. This carry more overhead than read-only caches. In non-JTA environments, each transaction should be completed when session.close() or session.disconnect() is called.

- **Nonstrict read/write**: This is most appropriate for data that is read often but only occasionally modified.This strategy does not guarantee that two transactions won't simultaneously modify the same data.

- **Transactional**: This is a fully transactional cache that may be used only in a JTA environment.

It can be enabled via the Hibernate mapping files as shown below:

```
<class name="com.myapp.Order">

    <cache usage="read-write"/>
    ....
</class>

```

Note: The usage options are: transactional|read-write | nonstrict-read-write|read-only. The cache can also be enabled at different granular level (e.g. parent, children, etc). The active orders will be cached for 300 seconds.

Q. How does the hibernate second-level cache work?
A. Hibernate always tries to first retrieve objects from the session and if this fails it tries to retrieve them from the second-level cache. If this fails again, the objects are directly loaded from the database. Hibernate's static initialize() method, which populates a proxy object, will attempt to hit the second-level cache before going to the database. The Hibernate class provides static methods for manipulation of proxies.

```
public final class Hibernate extends Object {
    ....
 public static void initialize(Object proxy)  thr
    ....
}

```

As a consequence of using the Hibernate second-level cache, you have to be aware of the fact that each call of a data access method can either result in a cache hit or miss. So, configure your log4j.xml to log your hits and misses.

```
<logger name="org.hibernate.cache">
   <level value="DEBUG" />
</logger>

```

Alternatively, you can use Spring AOP to log the cache access on your DAO methods.

The second level cache is a powerful mechanism for improving performance and scalability of your database driven application. Read-only caches are easy to handle, while read-write caches are more subtle in their behavior. Especially, the interaction with the Hibernate session can lead to unwanted behavior.

Q. What is a query cache in Hibernate?
A. The query cache is responsible for caching the results and to be more precise the keys of the objects returned by queries. Let us have a look how Hibernate uses the query cache to retrieve objects. In order to make use of the query cache we have to modify the person loading example as follows.

```
Query query = session.createQuery("from Order as
query.setInt(0, "Active");
query.setCacheable(true); // the query is cacheab
List l = query.list();

```

You also have to change the hibernate configuration to

enable the query cache. This is done by adding the following line to the Hibernate configuration.

```
1 <property name="hibernate.cache.use_query_cache">
```

Q. Where can you apply a 2nd level cache in Hibernate?
A. Use it when you read certain objects very often.

1) The 2nd level cache is key-value pair based, and works only if you get your **entities by id**. The 2nd level cache works on children objects only if **fetch="select"** is used. When we say fetch="select", then it will always fire separate queries to retrieve the association objects even if it is lazy ="false". So, if you do "from Employee where name = :name" then the 2nd level cache will not be hit.

2) The 2nd level cache is invalidated/updated on a per entity basis when an entity is updated/deleted **via hibernate**. If the entities are updated outside hibernate, then they are not invalidated and you run the risk of working with stale entities. In this scenario, you need to perform an impact analysis of objects being stale, and reviw your cache expiration strategy.

3) When using queries with join statements, then use the "**query cache**".

Q. What are the pitfalls of second level and query caches?
A. Memory is a finite resource, and over use or incorrect usage like caching the Order object and all its referenced objects can cause OutOfMemoryError. Here are some tips to overcome the pitfalls relating to caching.

1. Set entity's keys as query parameters, rather than setting the entire entity object. Critreria representations should also use identifiers as parameters. Write HQL queries to use identifiers in any substitutable parameters such as WHERE clause, IN clause etc.

In the example below, the entire customer and everything he/she references would be held in cache until either the

query cache exceeds its configured limits and it is evicted, or the table is modified and the results become dirty.

```
1 final Customer customer = ... ;
2 final String hql = "FROM Order as order WHERE ord
3 final Query q = session.createQuery(hql);
4 q.setParameter(0, customer);
5 q.setCacheable(true);
6
```

Instead of setting the whole customer object as shown above, just set the id.

```
1 final Order customer = ... ;
2 final String hql = "from Order as order where ord
3 final Query q = session.createQuery(hql);
4 q.setParameter(0, customer.getId());
5 q.setCacheable(true);
6
```

2. Hibernate's query cache implementation is pluggable by decorating Hibernate's query cache implementation. This involves overriding the put( ) method to check if a canonical equivalent of a query results object already exist in the Object[][], and assign the same QueryKey if it exists.

3. If you are in a single JVM using in memory cache only, use hibernate.cache.use_structured_entries=false in your hibernate configuration.

Here are some general performance tips:

1. Session.*load* will always try to use the cache. Session.*find* does not use the cache for the primary object, but cause the cache to be populated. Session.*iterate* always uses the cache for the primary object and any associated objects.

2. While developing, enable the show SQL and monitor the generated SQL.

```
1 <property name="show_sql">true</property>
```

Also enable the "**org.hibernate.cache**" logger in your log4j.xml to monitor cache hits and misses.

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

767 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

389 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

240 views

001B: ♦ Java architecture & design concepts interview questions & answers

202 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003,

and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ 08: Hibernate automatic dirty checking of persistent objects and handling detached objects

09: Hibernate entities with auditable, soft delete & optimistic locking fields ›

Posted in Hibernate, Hibernate Job Interview Essentials, member-paid

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer

 ↑