Hibernate – VII

Hibernate Cache

Table of Contents

Introduction

Caching is a mechanism for storing the loaded objects into a cache memory. This will improve the performance of application.

The advantage of cache mechanism

- ✓ It will reduce the hitting database (Whenever we want to load the same object from the database then instead of hitting the database once again, it loads from the local cache memory).
- ✓ So that the no. of round trips between our application and database will decreased.
- ✓ So we can say it will increases the performance of the application.

Hibernate supports three types of cache

1. The First Level Cache **(Session Level Cache)**.
2. The Second Level Cache **(Session Factory level cache)**.
3. The Query Cache

The First Level Cache (Session Level cache)

- ✓ By default, the first level cache is enabled in each hibernate application.
- ✓ Programmers are not responsible to enable or disable the First level cache.
- ✓ It is by-default associated with the session object and scope of the cache is limited to one session only.
- ✓ When we load an object first time from the database then the loaded object will be stored in the cache memory maintained by that session object.
- ✓ If we load the same object once again, with in the same session, then the object will be loaded from the local cache memory not from the database.
- ✓ If we load the same object once again, with in the same session, then the object will be loaded from the local cache memory not from the database.

Let's see in the below Application
Nothing will be change in **hibernate.cfg.xml** any other file.
Same as it is, we have to observe only the TestApplication.java and console.

```
public class TestApplication {

	public static void main(String[] args) {
		Configuration configuration = new Configuration().configure();
		SessionFactory sessionFactory = configuration.buildSessionFactory();

		Session session = sessionFactory.openSession();
		Transaction txn = session.beginTransaction();

		Order order1 = (Order) session.get(Order.class, new Long(3));
		System.out.println ("Product Name is "+order1.getProductName ());

		Order order2 = (Order) session.get(Order.class, new Long(3));
		System.out.println ("Product Name is "+order2.getProductName ());

		txn.commit();
		session.close();
	}
}
```

- As when you see in console hibernate fire select query only once.
- In the above **TestApplication.java** we are trying to get same object using same session object in that case in **Console** we can see it will fire select query only once.

```
        Session session = sessionFactory.openSession();

        Order order1 = (Order) session.get(Order.class, new Long(3));
        System.out.println ("Product Name is "+order1.getProductName ());

        Order order2 = (Order) session.get(Order.class, new Long(3));
        System.out.println ("Product Name is "+order2.getProductName ());

        Session session = sessionFactory.openSession();

        Order order3 = (Order) session.get(Order.class, new Long(3));
        System.out.println ("Product Name is "+order3.getProductName ());


        session.close();
}
```

- In the above case it will fire 2 time select query , For both the Session object.
- You can check the console.

The Second Level Cache (SessionFactory Level cache) introduced in hibernate 3.0

✓ It is not provided by the hibernate, programmer is responsible to enable or disable it, and it provided by 3rd parties vendor. There are different different vendor which provide like..

Cache	Provider class	Query Cache Supported
EHCache	org.hibernate.cache.EhcacheProvider	Yes
OSCache	org.hibernate.cache.OScacheProvider	Yes
SwarmCache	org.hibernate.cache.SwarmCacheProvider	
JBoss Cache 1.x	org.hibernate.cache.TreeCacheProvider	Yes
JBoss Cache 2	org.hibernate.cache.jbc2.JBossCacheRegionFactory	Yes

✓ In the second level cache, first hibernate verify whether that object is available in the local cache memory or not of that particular session.

✓ If not available then hibernate verify whether the object is available in factory cache [second level cache], or not if not available then hibernate will hit the database and loads the object from there, and then first stores in the local cache of the session then in the second level cache.

- ✓ When another session need to load the same object from the database, then hibernate copies that object from second level cache into the local cache of this new session

- To enable second level cache we require some changes see in the below application using **ehcache**.
- Here we required **ehcache.xml** file we have to change inside the **hibernate.cfg.xml** and add **dependency** in **pom.xml file**

```
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactid>hibernate-ehcache</artifactId>
        <version>5.0.1.Final</version>
</dependency>
```

- We have to add these properties in **hibernate.cfg.xml file**

```
<!-- Enable Second Level Cache -->
<property name="hibernate.cache.use_second_level_cache">true</property>
<property name="hibernate.cache.use_query_cache">true</property>
<property name="hibernate.cache.provider_class">
        org.hibernate.cache.EhCacheProvider
</property>
<property name="hibernate.cache.region.factory_class">
        org.hibernate.cache.ehcache.EhCacheRegionFactory
</property>
```

- Add **ehcache.xml file**

```
<ehcache>
        <!-- Default for all objects -->
        <defaultCache
          eternal="true"
          timeToLiveSeconds="200"
          timeToIdleSeconds="100"
          maxElementsInMemory="200" />

        <!-- For specified object -->
        <cache name="com.Biditvats.domain.Product"
           eternal="true"
           timeToLiveSeconds="200"
           timeToIdleSeconds="100"
           maxElementsInMemory="200" />

</ehcache>
```

- Remember **ehcache.xml**
  - ✓ Default cache will reflect all the pojo classes in our application.
  - ✓ if **eternal=”true”** then we should not write timeToIdealSeconds, timeToLiveSeconds, hibernate will take care about those values.
  - ✓ So if you want to give values manually better eternal=”false” always, so that we can assign values into **timeToIdealSeconds, timeToLiveSeconds** manually.
  - ✓ **timeToIdleSeconds=”seconds”** means, if the object in the global cache is ideal, means not using by any other class or object then it will be waited for some time that we specified and delated from the cache if time is exceeds more than timeToIdealSeconds value.
  - ✓ **timeToLiveSeconds=”seconds”** means, the other Session or class using this object or not, maybe it is using by other sessions or may not, whatever the situation might be, once it

completed the time specified timeToLiveSeconds, then it will be removed from the cache by hibernate.