Second Level Cache in Hibernate

**SessionFactory** holds the second level cache data. It is global for all the session objects and not enabled by default. Caching is all about application performance optimization and it sits between your application and the database to avoid the number of database hits as many as possible to give a better performance for performance critical applications.

# **First-level cache:**

The first-level cache is the Session cache and is a mandatory cache through which all requests must pass.  **It works in session scope**.

# **Second-level cache:**

Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache. The second-level cache can be configured on a per-class and per-collection basis and mainly responsible for caching objects across sessions. **It in session factory scope**.

# Query-level cache:

Hibernate also implements a cache for query resultsets that integrates closely with the second-level cache.

This is an optional feature and requires two additional physical cache regions that hold the cached query results and the timestamps when a table was last updated. This is only useful for queries that are run frequently with the same parameters.

To use the query cache, you must first activate it using the hibernate.cache.use\_query\_cache="true" property in the configuration file. By setting this property to true, you make Hibernate create the necessary caches in memory to hold the query and identifier sets.

Next, to use the query cache, you use the **setCacheable(Boolean)** method of the Query class. For example:

Session session = SessionFactory.openSession();

Query query = session.createQuery("FROM EMPLOYEE");

**query.setCacheable(true);**

List users = query.list();

SessionFactory.closeSession();

Hibernate also supports very fine-grained cache support through the concept of a cache region. A cache region is part of the cache that's given a name.

Session session = SessionFactory.openSession();

Query query = session.createQuery("FROM EMPLOYEE");

**query.setCacheable(true);**

**query.setCacheRegion("employee");**

List users = query.list();

SessionFactory.closeSession();

**How second level cache works**

Lets write all the facts point by point:

1. Whenever hibernate session try to load an entity, the very first place it look for cached copy of entity in first level cache (associated with particular hibernate session).
2. If cached copy of entity is present in first level cache, it is returned as result of load method.
3. If there is no cached entity in first level cache, then second level cache is looked up for cached entity.
4. If second level cache has cached entity, it is returned as result of load method. But, before returning the entity, it is stored in first level cache also so that next invocation to load method for entity will return the entity from first level cache itself, and there will not be need to go to second level cache again.
5. If entity is not found in first level cache and second level cache also, then database query is executed and entity is stored in both cache levels, before returning as response of load() method.
6. Second level cache validate itself for modified entities, if modification has been done through hibernate session APIs.
7. If some user or process make changes directly in database, the there is no way that second level cache update itself until “timeToLiveSeconds” duration has passed for that cache region. In this case, it is good idea to invalidate whole cache and let hibernate build its cache once again. You can use below code snippet to invalidate whole hibernate second level cache.

# **Caching Strategies**

1. **Read Only**: This caching strategy should be used for persistent objects that will always read but never updated. It’s good for reading and caching application configuration and other static data that are never updated. This is the simplest strategy with best performance because there is no overload to check if the object is updated in database or not.
2. **Read Write**: It’s good for persistent objects that can be updated by the hibernate application. However if the data is updated either through backend or other applications, then there is no way hibernate will know about it and data might be stale. So while using this strategy, make sure you are using Hibernate API for updating the data.
3. **Nonrestricted Read Write**: If the application only occasionally needs to update data and strict transaction isolation is not required, a nonstrict-read-write cache might be appropriate.
4. **Transactional**: The transactional cache strategy provides support for fully transactional cache providers such as JBoss TreeCache. Such a cache can only be used in a JTA environment and you must specify hibernate.transaction.manager\_lookup\_class.

Since EHCache supports all the above caching strategies, it’s the best choice when you are looking for second level caching in hibernate.

**CONCURRENCY STRATEGIES:**

A concurrency strategy is a mediator which responsible for storing items of data in the cache and retrieving them from the cache. If you are going to enable a second-level cache, you will have to decide, for each persistent class and collection, which cache concurrency strategy to use.

1. **Transactional:** Use this strategy for read-mostly data where it is critical to prevent stale data in concurrent transactions,in the rare case of an update.
2. **Read-write:** Again use this strategy for read-mostly data where it is critical to prevent stale data in concurrent transactions,in the rare case of an update.
3. **Nonstrict-read-write:** This strategy makes no guarantee of consistency between the cache and the database. Use this strategy if data hardly ever changes and a small likelihood of stale data is not of critical concern.
4. **Read-only:** A concurrency strategy suitable for data which never changes. Use it for reference data only.

Example on Second Level Cache in Hibernate

# **Maven(pom.xml)**

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>hib2ndlevelcache-1</groupId>

<artifactId>hib2ndlevelcache-1</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>hib2ndlevelcache-1</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<!-- Hibernate Core API -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>4.3.5.Final</version>

</dependency>

<dependency>

<groupId>cglib</groupId>

<artifactId>cglib</artifactId>

<version>2.2.2</version>

</dependency>

<!-- MySQL Driver -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.0.5</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-ehcache</artifactId>

<version>4.3.5.Final</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

<version>1.7.5</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.30</version>

</dependency>

</dependencies>

</project>

# **Hibernate Configuration(hibernate.cfg.xml)**

<?xml version=*'1.0'* encoding=*'UTF-8'*?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=*"show\_sql"*>true</property>

<property name=*"hbm2ddl.auto"*>update</property>

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/test</property>

<property name=*"connection.username"*>deba</property>

<property name=*"connection.password"*>deba</property>

<property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*'show\_sql'*>true</property>

**<property name=*"hibernate.cache.use\_second\_level\_cache"*>true</property>**

**<property name=*"hibernate.cache.region.factory\_class"*>**

**org.hibernate.cache.ehcache.EhCacheRegionFactory**

**</property>**

<mapping resource=*"employee.hbm.xml"* />

</session-factory>

</hibernate-configuration>

# **Hibernate Mapping file (employee.hbm.xml)**

<?xml version=*'1.0'* encoding=*'UTF-8'*?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"com.ddlab.rnd.hibernate.Employee"* table=*"emp"*>

<cache usage=*"read-only"* />

<id name=*"id"*>

<generator class=*"native"*></generator>

</id>

<property name=*"name"*></property>

<property name=*"salary"*></property>

</class>

</hibernate-mapping>

# **Ehcache Configuration(ehcache.xml)**

<?xml version=*"1.0"*?>

<ehcache>

<defaultCache maxElementsInMemory=*"100"* eternal=*"false"*

timeToIdleSeconds=*"120"* timeToLiveSeconds=*"200"* />

<cache name=*"com.ddlab.rnd.hibernate.Employee"*

maxElementsInMemory=*"100"* eternal=*"false"* timeToIdleSeconds=*"5"*

timeToLiveSeconds=*"200"* />

</ehcache>

# **SQL File**

**create** **table** emp (

id **integer** **NOT** **NULL** AUTO\_INCREMENT,

name **VARCHAR**(50),

salary **integer**,

**PRIMARY** **KEY** (id)

);

**select** \* **from** emp;

# **Java Files**

## **Employee.java**

**package** com.ddlab.rnd.hibernate;

**public** **class** Employee {

**private** **int** id;

**private** String name;

**private** **int** salary;

**public** Employee() {

**super**();

}

**public** Employee(String name, **int** salary) {

**super**();

**this**.name = name;

**this**.salary = salary;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getSalary() {

**return** salary;

}

**public** **void** setSalary(**int** salary) {

**this**.salary = salary;

}

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";

}

}

## **StoreTest.java**

**package** com.ddlab.rnd.hibernate;

**import** org.hibernate.\*;

**import** org.hibernate.boot.registry.StandardServiceRegistryBuilder;

**import** org.hibernate.cfg.\*;

**import** org.hibernate.service.ServiceRegistry;

**public** **class** StoreTest {

**public** **static** SessionFactory getSessionFactory() {

Configuration cfg = **new** Configuration().configure("hibernate.cfg.xml");

ServiceRegistry serviceRegistry = **new** StandardServiceRegistryBuilder().applySettings(cfg.getProperties())

.build();

SessionFactory factory = cfg.buildSessionFactory(serviceRegistry);

**return** factory;

}

**public** **static** **void** main(String[] args) {

**try** {

SessionFactory factory = *getSessionFactory*();

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

session.save(**new** Employee("Deb", 50000));

session.save(**new** Employee("Mishra", 70000));

tx.commit();

session.close();

factory.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

## **FetchTest.java**

**package** com.ddlab.rnd.hibernate;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.boot.registry.StandardServiceRegistryBuilder;

**import** org.hibernate.cfg.Configuration;

**import** org.hibernate.service.ServiceRegistry;

**public** **class** FetchTest {

**public static SessionFactory getSessionFactory() {**

**Configuration cfg = new Configuration().configure("hibernate.cfg.xml");**

**ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder().applySettings(cfg.getProperties())**

**.build();**

**SessionFactory factory = cfg.buildSessionFactory(serviceRegistry);**

**return factory;**

**}**

**public** **static** **void** main(String[] args) {

SessionFactory factory = *getSessionFactory*();

Session session1 = factory.openSession();

**int** rowNo = 2;

Employee emp1 = (Employee) session1.load(Employee.**class**, rowNo);

System.***out***.println(emp1.getId() + " " + emp1.getName() + " " + emp1.getSalary());

session1.close();

Session session2 = factory.openSession();

Employee emp2 = (Employee) session2.load(Employee.**class**, rowNo);

System.***out***.println(emp2.getId() + " " + emp2.getName() + " " + emp2.getSalary());

session2.close();

factory.close();

}

}

## **EvictionTest.java**

**package** com.ddlab.rnd.hibernate;

**import** java.util.Map;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.boot.registry.StandardServiceRegistryBuilder;

**import** org.hibernate.cfg.Configuration;

**import** org.hibernate.metadata.ClassMetadata;

**import** org.hibernate.service.ServiceRegistry;

**public** **class** EvictionTest {

**public** **static** SessionFactory getSessionFactory() {

Configuration cfg = **new** Configuration().configure("hibernate.cfg.xml");

ServiceRegistry serviceRegistry = **new** StandardServiceRegistryBuilder().applySettings(cfg.getProperties())

.build();

SessionFactory factory = cfg.buildSessionFactory(serviceRegistry);

**return** factory;

}

/\*\*

\* Evicts all second level cache hibernate entites. This is generally only

\* needed when an external application modifies the databaase.

\*/

**public static void evict2ndLevelCache( SessionFactory sessionFactory ) {**

**try {**

**Map<String, ClassMetadata> classesMetadata = sessionFactory.getAllClassMetadata();**

**for (String entityName : classesMetadata.keySet()) {**

**System.*out*.println("Evicting Entity from 2nd level cache: " + entityName);**

**sessionFactory.~~evictEntity~~(entityName);**

**}**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**}**

**public** **static** **void** main(String[] args) {

SessionFactory factory = *getSessionFactory*();

Session session1 = factory.openSession();

**int** rowNo = 2;

Employee emp1 = (Employee) session1.load(Employee.**class**, rowNo);

System.***out***.println(emp1.getId() + " " + emp1.getName() + " " + emp1.getSalary());

session1.close();

//If you evict the entity, the query will be fired once agains

*evict2ndLevelCache*(factory);

//System.out.println(HibernateUtil.getSessionFactory().getStatistics().getEntityFetchCount()); //Prints 1

// System.out.println(HibernateUtil.getSessionFactory().getStatistics().getSecondLevelCacheHitCount()); //Prints 0

Session session2 = factory.openSession();

Employee emp2 = (Employee) session2.load(Employee.**class**, rowNo);

System.***out***.println(emp2.getId() + " " + emp2.getName() + " " + emp2.getSalary());

session2.close();

factory.close();

}

}