Hibernate Caching

Lesson Objectives

1. Understand the type of cache levels in Hibernate.
2. Able to implement First Level Cache
3. Know how to configure using Cache Level 2.
4. Able to implement Query Cache.

Agenda

- Introduction to Caching in Hibernate
- First Level Cache
- Second Level Cache
- Query Cache

Section 01

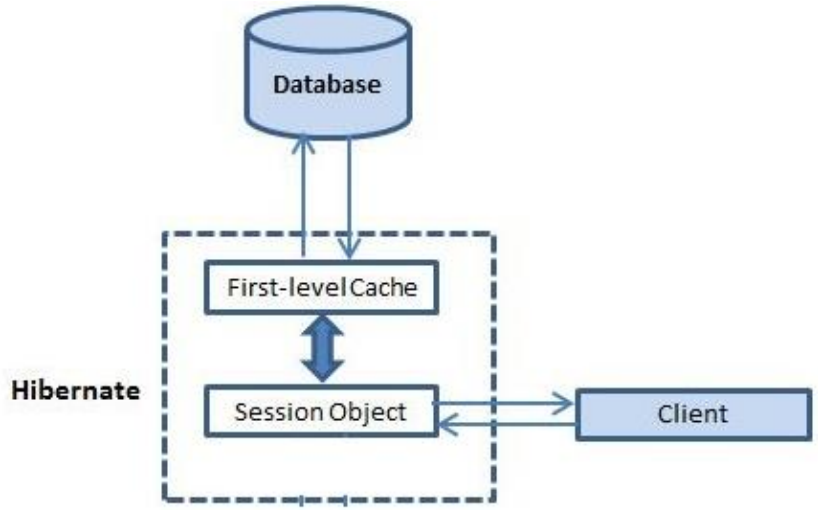
CACHING IN HIBERNATE

Introduction

- Hibernate Cache can be very useful in gaining fast application performance if used correctly.
- Reduce the number of database queries, hence reducing the throughput time of the application.
- Hibernate comes with different types of Cache:
  - **First Level Cache**
  - **Second Level Cache**
  - **Query Cache**

First Level Cache

- Hibernate first level cache is associated with the **Session** object.
- Hibernate first level cache is enabled by default and there is no way to disable it.
- Any object cached in a session will not be visible to other sessions and when the session is closed, all the cached objects will also be lost.



First Level Cache

- Hibernate provides methods through which we can delete selected objects from the cache or clear the cache completely.
  - **evict()** remove a single object from the hibernate first level cache.
  - **clear()** clear the cache: delete all the objects from the cache.
  - **contains()** check if an object is present in the hibernate cache or not

First Level Cache

❖ **Example 1**: load the entity many times in the same session

```
session = HibernateUtils.getSessionFactory().openSession();

Departments department = session.load(Departments.class, new Integer(1));
System.out.println(department.getDeptName());

// Lấy đối tượng department thêm lần nữa
department = (Departments) session.load(Departments.class, new Integer(1));
System.out.println(department.getDeptName());

// Lấy thêm nhiều lần nữa
for (int i = 0; i < 5; i++) {
    department = (Departments) session.load(Departments.class,
                                            new Integer(1));
    System.out.println(department.getDeptName());
}
```

First Level Cache

- **Example 1**: load the entity many times in the same session

Console log:

```
Hibernate: select department0_.dept_id as dept_id1_6_0_,
department0_.dept_name as dept_nam2_6_0_ from dbo.Departments
department0_ where department0_.dept_id=?
IT Tools
IT Tools
IT Tools
IT Tools
IT Tools
IT Tools
IT Tools
IT Tools
```

First Level Cache

❖ **Example 2**: load the entity many times in the different session

```
sessionA = HibernateUtils.getSessionFactory().openSession();
sessionB = HibernateUtils.getSessionFactory().openSession();

Departments department = sessionA.load(Departments.class,
                                       new Integer(1));
System.out.println(department.getDeptName());

// Lấy đối tượng department thêm lần nữa trong sesssionA
department = sessionA.load(Departments.class, new Integer(1));
System.out.println(department.getDeptName());

// Lấy đối tượng department thêm lần nữa trong sessionB
department = (Departments) sessionB.load(Departments.class,
                                         new Integer(1));
System.out.println(department.getDeptName());
```

First Level Cache

❖ **Example 2**: load the entity many times in the same session

Console log:

```
Hibernate:    select    department0_.dept_id    as    dept_id1_6_0_,
department0_.dept_name   as   dept_nam2_6_0_   from   dbo.Departments
department0_ where department0_.dept_id=?
IT Tools
IT Tools
Hibernate:    select    department0_.dept_id    as    dept_id1_6_0_,
department0_.dept_name   as   dept_nam2_6_0_   from   dbo.Departments
department0_ where department0_.dept_id=?
IT Tools
```

First Level Cache

❖ **Example 3**: remove all of the objects from the cache

```
session = HibernateUtils.getSessionFactory().openSession();

// Lấy đối tượng department lần đầu tiên
Departments department = session.load(Departments.class,
                                      new Integer(1));
System.out.println(department.getDeptName());

// Lấy tiếp lần thứ 2
department = session.load(Departments.class, new Integer(1));
System.out.println(department.getDeptName());
// Xóa bỏ khỏi session - hay First Level Cache
session.evict(department);
// session.clear();

// Lấy tiếp đối tượng department lần nữa
department = (Departments) session.load(Departments.class,
                                        new Integer(1));
System.out.println(department.getDeptName());
```

First Level Cache

- **Example 3**: load the entity many times in the same session

Console log:

```
Hibernate:    select    department0_.dept_id    as    dept_id1_6_0_,
department0_.dept_name   as   dept_nam2_6_0_   from   dbo.Departments
department0_ where department0_.dept_id=?
IT Tools
IT Tools
Hibernate:    select    department0_.dept_id    as    dept_id1_6_0_,
department0_.dept_name   as   dept_nam2_6_0_   from   dbo.Departments
department0_ where department0_.dept_id=?
IT Tools
```

Section 02

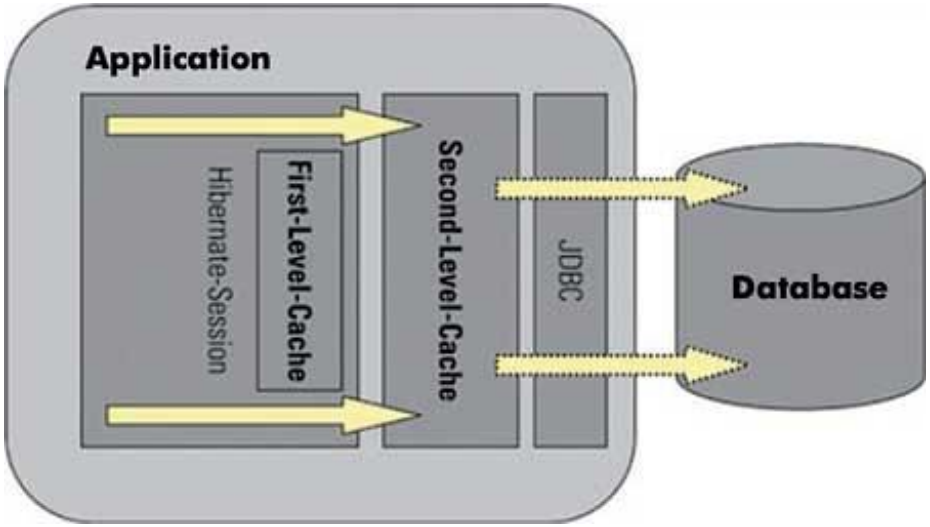
SECOND LEVEL CACHE

Second Level Cache

- **First Level Cache** is a session scoped cache which ensures that each entity instance is loaded only once in the persistent context.
- Once the session is closed, first-level cache is terminated as well. *This is actually desirable, as it allows for concurrent sessions to work with entity instances in isolation from each other*.
- **Second-level cache** is *SessionFactory*-scoped, meaning it is shared by all sessions created with the same session factory.
- When an entity instance is looked up by its id, and if second-level caching is enabled for that entity, the following happens:
  - ✓ If an instance is already present in the first-level cache, it is returned from there
  - ✓ If an instance is not found in the first-level cache, and the corresponding instance state is cached in the second-level cache, then the data is fetched from there and an instance is assembled and returned
  - ✓ Otherwise, the necessary data are loaded from the database and an instance is assembled and returned.

Second Level Cache

- Hibernate **second-level caching** is designed to be unaware[không biết] of the actual cache provider used.
- Hibernate only needs to be provided with an implementation of the org.hibernate.cache.spi.RegionFactory interface which encapsulates all details specific to actual cache providers.
- *It acts as a bridge between Hibernate and cache providers*.
- Hibernate Second Level cache providers include **EHCache** and **Infinispan.** Use **Ehcache** as a cache provider:



Second Level Cache

- We add the **Ehcache** region factory implementation to the classpath with the following Maven dependency:

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-ehcache</artifactId>
    <version>5.4.12.Final</version>
</dependency>

<!-- Ehcache -->
<!-- https://mvnrepository.com/artifact/
                              net.sf.ehcache/ehcache -->
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
    <version>2.10.5</version>
</dependency>
```

Second Level Cache

- To L2 caching is enabled and we give it the name of the region factory class.
- **hibernate.cfg.xml**:

```
<property  name="hibernate.cache.use_second_level_cache“
           value="true" />
<property name="hibernate.cache.region.factory_class"
       value="org.hibernate.cache.ehcache.EhCacheRegionFactory" />
```

Making an Entity Cacheable

Entity

```
@Entity
@Table(name = "Departments", schema = "dbo")
@Cacheable
@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Departments {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "dept_id")
    private int deptId;

    @Column(name = "dept_name")
    private String deptName;

    @OneToMany(mappedBy = "department", fetch = FetchType.LAZY)
    private Set<Employees> employees;

    // getter and setter methods
}
```

Cache Concurrency Strategy

- ***READ_ONLY***: Used only for entities that never change (exception is thrown if an attempt to update such an entity is made). It is very simple and performant. Very suitable for some static reference data that don't change
- ***NONSTRICT_READ_WRITE***: Cache is updated after a transaction that changed the affected data has been committed. Thus, strong consistency is not guaranteed and there is a small time window in which stale data may be obtained from cache. This kind of strategy is suitable for use cases that can tolerate eventual consistency
- ***READ_WRITE***: This strategy guarantees strong consistency which it achieves by using ‘soft' locks: When a cached entity is updated, a soft lock is stored in the cache for that entity as well, which is released after the transaction is committed. All concurrent transactions that access soft-locked entries will fetch the corresponding data directly from database
- ***TRANSACTIONAL***: Cache changes are done in distributed XA transactions. A change in a cached entity is either committed or rolled back in both database and cache in the same XA transaction

Cache Management

- If expiration and eviction policies are not defined, the cache could grow indefinitely and eventually consume all of available memory.
- In most cases, Hibernate leaves cache management duties like these to cache providers, as they are indeed specific to each cache implementation.
- For example, we could define the following Ehcache configuration to limit the maximum number of cached *Departments* instances to 1000:

```
<ehcache>
    <cache name="fa.training.entities.Departments"
           maxElementsInMemory="1000" />
</ehcache>
```

Second Level Cache

- **Example:**

```
try {
            sessionA = HibernateUtils.getSessionFactory().openSession();
            sessionB = HibernateUtils.getSessionFactory().openSession();

            Departments department = sessionA.load(Departments.class, deptId);
            System.out.println(department.getDeptName());

            // Lấy đối tượng department thêm lần nữa trong sesssionA
            department = sessionA.load(Departments.class, deptId);
            System.out.println(department.getDeptName());

            // Lấy đối tượng department thêm lần nữa trong sessionB
            department = (Departments) sessionB.load(Departments.class, deptId);
            System.out.println(department.getDeptName());

        } finally {
            if (sessionA != null) {
                sessionA.close();
            }
            if (sessionB != null) {
                sessionB.close();
            }
}
```

Second Level Cache

- **Console log:**

```
Hibernate: select department0_.dept_id as dept_id1_6_0_,
department0_.dept_name as dept_nam2_6_0_ from dbo.Departments
department0_ where department0_.dept_id=?
IT Tools
IT Tools
IT Tools
```

- **Remove all of the objects from cache level 2:**

  sessionFactory.getCache().evictEntity(User.class, user);

  sessionFactory.getCache().evictAllRegions()

Section 03

QUERY CACHE

Query Cache

- Aside from caching entities and collections, Hibernate offers a query cache too. This is useful for frequently executed queries with fixed parameter values.
  - To use query caching, you will first need to enable it with the following configuration property:

```
<property name="hibernate.cache.use_query_cache" value="true" />
```

  - Caching query using Hibernate native API

```
List<Person> persons = session.createQuery( "select p " + "from Person p " +
"where p.name = :name") .setParameter( "name", "John Doe")
.setCacheable(true) .list();
```

Query Cache

- Hibernate defined the CacheMode enumeration to describe the ways of interactions with the cached

CacheMode	Description
CacheMode.NORMAL	Default. Reads/writes data from/into the cache
CacheMode.REFRESH	Doesn't read from cache, but writes to the cache upon loading from the database
CacheMode.PUT	Doesn't read from cache, but writes to the cache as it reads from the database
CacheMode.GET	Read from the cache, but doesn't write to cache
CacheMode.IGNORE	Doesn't read/write data from/into the cache

```
List<Person> persons = session.createQuery( "select p from Person p" )
.setCacheable( true ) .setCacheMode(CacheMode.REFRESH) .list();
```

Summary

- Introduction to Caching in Hibernate
- First Level Cache
- Second Level Cache
- Query Cache

Thank you