

Hibernate Caching

Design by: DieuNT1



1. Introduction to Caching in Hibernate

2. First Level Cache

3. Second Level Cache

4. Query Cache

5. Question and Answer

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.

Section 01

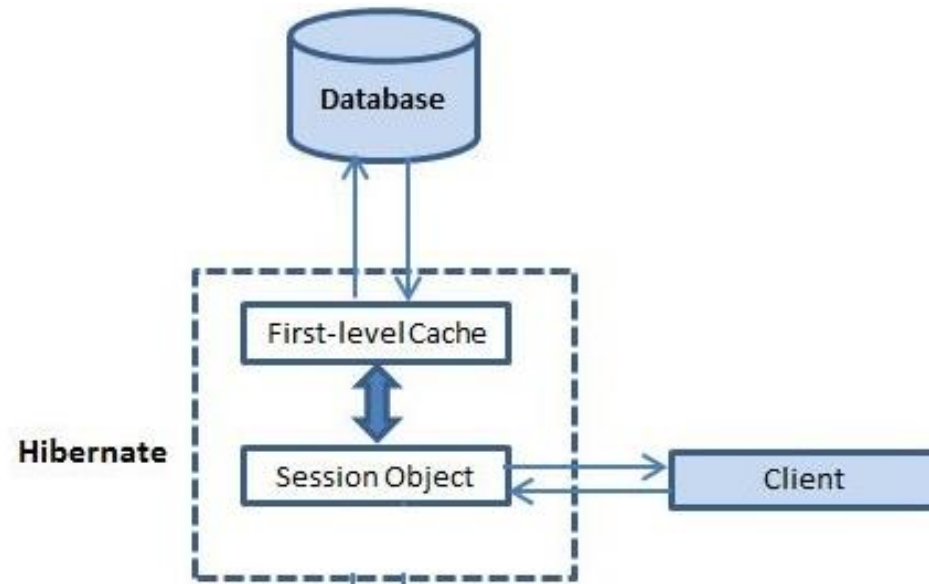
Caching in Hibernate

Caching in Hibernate refers to the technique of storing frequently accessed data in memory **to improve the performance** of an application that uses Hibernate.

- 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.



- 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

```
Session session = HibernateUtil.getSessionFactory().openSession();
...
AntEntity entity = (AntEntity) session.load(AntEntity.class, new Integer(1));
entity = (AntEntity) session.load(AntEntity.class, new Integer(1));

session.evict(entity); //session.clear();

entity = (AntEntity) session.load(AntEntity.class, new Integer(1));
...
```

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

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 sessionA
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());
```

- **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

- **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());
```

- **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.

Second Level Cache

Is **SessionFactory-scoped**, meaning it is shared by all sessions created with the same session factory



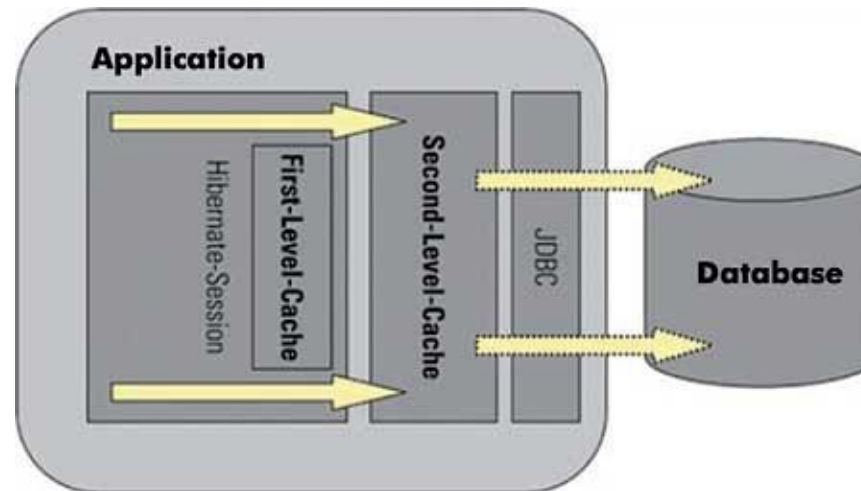
When an entity instance **is looked up by its id**, and if second-level caching is enabled for that entity, the following happens:

1. If an instance is already present in the first-level cache, it is returned from there
2. 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
3. Otherwise, the necessary data are loaded from the database and an instance is assembled and returned.

Second Level Cache

Providers in Hibernate

- **Ehcache:** Ehcache is a popular open-source caching library that provides an efficient in-memory caching solution for Hibernate.
- It supports various features such as expiration **policies**, **distributed** caching, and **persistent** caching.
- With the *Hibernate 6* using *Ehcache 3*.



Enable L2 Cache Configuration

- With **EHCache 3**: hibernate's built-in integration for JCache so we need to include the hibernate-jcache module.

```
<!-- https://mvnrepository.com/artifact/org.hibernate.orm/hibernate-jcache -->
<dependency>
  <groupId>org.hibernate.orm</groupId>
  <artifactId>hibernate-jcache</artifactId>
  <version>6.2.6.Final</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-ehcache -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-ehcache</artifactId>
  <version>5.6.15.Final</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.ehcache/ehcache -->
<dependency>
  <groupId>org.ehcache</groupId>
  <artifactId>ehcache</artifactId>
  <version>3.8.1</version>
</dependency>
```

Enable L2 Cache Configuration

- Since Java 11, JAXB has been removed from the JDK distribution so we need to import it explicitly.
- It is required to parse the **ehcache.xml** file when we bootstrap the application and the configuration is parsed.

```
<!-- https://mvnrepository.com/artifact/com.sun.xml.bind/jaxb-core -->
<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-core</artifactId>
  <version>2.3.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.xml.bind/jaxb-api -->
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.sun.xml.bind/jaxb-impl -->
<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-impl</artifactId>
  <version>2.2.4-1</version>
</dependency>
```



Enabling Second-Level Caching

- To enable the second-level cache support, we need to enable it in `hibernate.cf.xml` file:

```
<!-- Enable second level cache 3 -->  
<property name="hibernate.cache.region.factory_class">jcache</property>  
<property name="hibernate.javax.cache.provider">  
    org.ehcache.jsr107.EhcacheCachingProvider</property>  
<property name="hibernate.cache.use_second_level_cache">true</property>
```

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
}
```

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 sessionA  
    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()
```

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

- 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>
```


Section 03

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!

