**Caching in Hibernate**

Hibernate caching improves the performance of the application by pooling the object in the cache.

There are mainly two types of caching: first level cache and second level cache.

First Level Cache

Session object holds the first level cache data. It is enabled by default. The first level cache data will not be available to entire application. An application can use many session object.

Second Level Cache

SessionFactory object holds the second level cache data. The data stored in the second level cache will be available to entire application. But we need to enable it explicitely.

Second Level Cache implementations are provided by different vendors such as:

**Hibernate Second Level Cache**

**Hibernate second level cache** uses *a common cache for all the session object of a session factory*. It is useful if you have multiple session objects from a session factory.

**SessionFactory** holds the second level cache data. It is global for all the session objects and not enabled by default.

Different vendors have provided the implementation of Second Level Cache.

1. EH Cache
2. OS Cache
3. Swarm Cache
4. JBoss Cache

Each implementation provides different cache usage functionality. There are four ways to use second level cache.

1. **read-only:** caching will work for read only operation.
2. **nonstrict-read-write:** caching will work for read and write but one at a time.
3. **read-write:** caching will work for read and write, can be used simultaneously.
4. **transactional:** caching will work for transaction.

The cache-usage property can be applied to class or collection level in hbm.xml file. The example to define cache usage is given below:

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

Let's see the second level cache implementation and cache usage.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Implementation** | **read-only** | **nonstrict-read-write** | **read-write** | **transactional** |
| EH Cache | Yes | Yes | Yes | No |
| OS Cache | Yes | Yes | Yes | No |
| Swarm Cache | Yes | Yes | No | No |
| JBoss Cache | No | No | No | Yes |

## 3 extra steps for second level cache example using EH cache

1) Add 2 configuration setting in hibernate.cfg.xml file

1. **<property** name="cache.provider\_class"**>**org.hibernate.cache.EhCacheProvider**</property>**
2. **<property** name="hibernate.cache.use\_second\_level\_cache"**>**true**</property>**

Add cache usage setting in hbm file

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

Create ehcache.xml file

1. **<?xml** version="1.0"**?>**
2. **<ehcache>**
4. **<defaultCache**
5. maxElementsInMemory="100"
6. eternal="true"**/>**
8. **</ehcache>**

### Hibernate Second Level Cache Example

To understand the second level cache through example, we need to create following pages:

1. Employee.java
2. employee.hbm.xml
3. hibernate.cfg.xml
4. ehcache.xml
5. FetchTest.java

*File: Employee.java*

1. **package** com.javatpoint;
3. **public** **class** Employee {
4. **private** **int** id;
5. **private** String name;
6. **private** **float** salary;
8. **public** Employee() {}
9. **public** Employee(String name, **float** salary) {
10. **super**();
11. **this**.name = name;
12. **this**.salary = salary;
13. }
14. //setters and getters
15. }

*File: employee.hbm.xml*

1. **<?xml** version='1.0' encoding='UTF-8'**?>**
2. <!DOCTYPE hibernate-mapping PUBLIC
3. "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4. "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"**>**
6. **<hibernate-mapping>**
7. **<class** name="com.javatpoint.Employee" table="emp1012"**>**
8. **<cache** usage="read-only" **/>**
9. **<id** name="id"**>**
10. **<generator** class="native"**></generator>**
11. **</id>**
12. **<property** name="name"**></property>**
13. **<property** name="salary"**></property>**
14. **</class>**
16. **</hibernate-mapping>**



Here, we are using **read-only** cache usage for the class. The cache usage can also be used in collection.

*File: hibernate.cfg.xml*

1. **<?xml** version='1.0' encoding='UTF-8'**?>**
2. <!DOCTYPE hibernate-configuration PUBLIC
3. "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4. "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd"**>**
6. <!-- Generated by MyEclipse Hibernate Tools.                   -->
7. **<hibernate-configuration>**
9. **<session-factory>**
10. **<property** name="show\_sql"**>**true**</property>**
11. **<property** name="hbm2ddl.auto"**>**update**</property>**
12. **<property** name="dialect"**>**org.hibernate.dialect.Oracle9Dialect**</property>**
13. **<property** name="connection.url"**>**jdbc:oracle:thin:@localhost:1521:xe**</property>**
14. **<property** name="connection.username"**>**system**</property>**
15. **<property** name="connection.password"**>**oracle**</property>**
16. **<property** name="connection.driver\_class"**>**oracle.jdbc.driver.OracleDriver**</property>**
18. **<property** name="cache.provider\_class"**>**org.hibernate.cache.EhCacheProvider**</property>**
19. **<property** name="hibernate.cache.use\_second\_level\_cache"**>**true**</property>**
21. **<mapping** resource="employee.hbm.xml"**/>**
22. **</session-factory>**
24. **</hibernate-configuration>**

To implement second level cache, we need to define **cache.provider\_class** property in the configuration file.

*File: ehcache.xml*

1. <?xml version="1.0"?>
2. <ehcache>
3. <defaultCache
4. maxElementsInMemory="100"
5. eternal="false"
6. timeToIdleSeconds="120"
7. timeToLiveSeconds="200" />
9. <cache name="com.javatpoint.Employee"
10. maxElementsInMemory="100"
11. eternal="false"
12. timeToIdleSeconds="5"
13. timeToLiveSeconds="200" />
14. </ehcache>

You need to create ehcache.xml file to define the cache property.

**defaultCache** will be used for all the persistent classes. We can also define persistent class explicitely by using the cache element.

**eternal** If we specify eternal="true", we don't need to define timeToIdleSeconds and timeToLiveSeconds attributes because it will be handled by hibernate internally. Specifying eternal="false" gives control to the programmer, but we need to define timeToIdleSeconds and timeToLiveSeconds attributes.

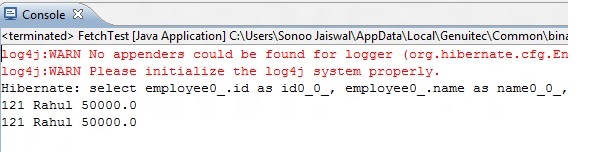
**timeToIdleSeconds** It defines that how many seconds object can be idle in the second level cache.

**timeToLiveSeconds** It defines that how many seconds object can be stored in the second level cache whether it is idle or not.

*File: FetchTest.java*

1. **package** com.javatpoint;
3. **import** org.hibernate.Session;
4. **import** org.hibernate.SessionFactory;
5. **import** org.hibernate.cfg.Configuration;
7. **public** **class** FetchTest {
8. **public** **static** **void** main(String[] args) {
9. Configuration cfg=**new** Configuration().configure("hibernate.cfg.xml");
10. SessionFactory factory=cfg.buildSessionFactory();
12. Session session1=factory.openSession();
13. Employee emp1=(Employee)session1.load(Employee.**class**,121);
14. System.out.println(emp1.getId()+" "+emp1.getName()+" "+emp1.getSalary());
15. session1.close();
17. Session session2=factory.openSession();
18. Employee emp2=(Employee)session2.load(Employee.**class**,121);
19. System.out.println(emp2.getId()+" "+emp2.getName()+" "+emp2.getSalary());
20. session2.close();
22. }
23. }

#### Output:



As we can see here, hibernate does not fire query twice. If you don't use second level cache, hibernate will fire query twice because both query uses different session objects.