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:

```
<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

```
<property name="cache.provider_class">org.hibernate.cache.EhCacheProvider</property>
<property name="hibernate.cache.use_second_level_cache">true</property>
```

2) Add cache usage setting in hbm file

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

3) Create ehcache.xml file

```
<?xml version="1.0"?>
<ehcache>

<defaultCache
maxElementsInMemory="100"
eternal="true"/>

</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

Here, we are assuming, there is emp1012 table in the oracle database containing some records.

File: Employee.java

```
package com.javatpoint;

public class Employee {
```

```
private int id;
private String name;
private float salary;

public Employee() {}
public Employee(String name, float salary) {
    super();
    this.name = name;
    this.salary = salary;
}
//setters and getters
}
```

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.javatpoint.Employee" table="emp1012">
        <cache usage="read-only" />
        <id name="id">
        <generator class="native"></generator>
        </id>
        <property name="name"></property>
        <property name="salary"></property>
        </class>

        </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

```
<?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">

<!-- Generated by MyEclipse Hibernate Tools.                    -->
<hibernate-configuration>

    <session-factory>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

        <property name="cache.provider_class">org.hibernate.cache.EhCacheProvider</property>
         <property name="hibernate.cache.use_second_level_cache">true</property>

    <mapping resource="employee.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

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

File: ehcache.xml

```
<?xml version="1.0"?>
<ehcache>
<defaultCache
maxElementsInMemory="100"
eternal="false"
timeToIdleSeconds="120"
timeToLiveSeconds="200" />
```

```
<cache name="com.javatpoint.Employee"
maxElementsInMemory="100"
eternal="false"
timeToIdleSeconds="5"
timeToLiveSeconds="200" />
</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

```
package com.javatpoint;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class FetchTest {
public static void main(String[] args) {
Configuration cfg=new Configuration().configure("hibernate.cfg.xml");
SessionFactory factory=cfg.buildSessionFactory();

    Session session1=factory.openSession();
    Employee emp1=(Employee)session1.load(Employee.class,121);
    System.out.println(emp1.getId()+" "+emp1.getName()+" "+emp1.getSalary());
    session1.close();

    Session session2=factory.openSession();
```

```
    Employee emp2=(Employee)session2.load(Employee.class,121);
    System.out.println(emp2.getId()+" "+emp2.getName()+" "+emp2.getSalary());
    session2.close();

}
}
```

Output:

```
Console
<terminated> FetchTest [Java Application] C:\Users\Sonoo Jaiswal\AppData\Local\Genuitec\Common\bina
log4j:WARN No appenders could be found for logger (org.hibernate.cfg.En
log4j:WARN Please initialize the log4j system properly.
Hibernate: select employee0_.id as id0_0_, employee0_.name as name0_0_,
121 Rahul 50000.0
121 Rahul 50000.0
```

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.

download this hibernate example (developed using MyEclipse IDE)

← prev next →