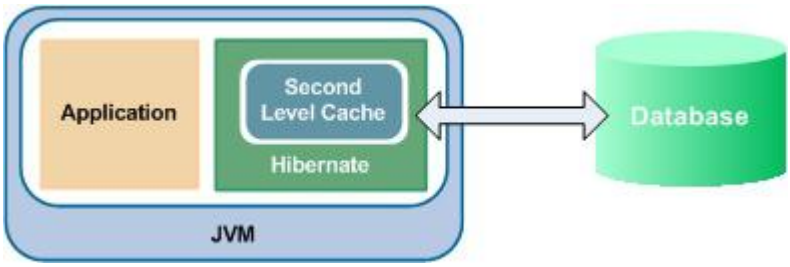
1. Hibernate EhCache Configuration Tutorial

Caching is facility provided by ORM frameworks which help users to get fast running web application, while help framework itself to reduce number of queries made to database in a single transaction. Hibernate also provide this caching functionality, in two layers.

- **Fist level cache**: This is enabled by default and works in session scope.
- **Second level cache**: This is apart from first level cache which is available to be used globally in session factory scope.

In this tutorial, I am giving an example using ehcache configuration as second level cache in hibernate.



Hibernate with EhCache

Sections in this post:

How second level cache works
About EhCache
Configuring EhCache
Configuring entity objects
Query caching
Example application
Sourcecode download

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.

About EhCache

Terracotta Ehcache is a popular open source Java cache that can be used as a Hibernate second level cache. It can be used as a standalone second level cache, or can be configured for clustering to provide a replicated coherent second level cache.

Hibernate ships with the ehcache library. If you want any particular version of ehcache, visit the Terracotta **Ehcache download site**:

http://www.terracotta.org/products/enterprise-ehcache

The **maven dependency** is for Ehcache 2.0 and any upgrades is:

```
<dependency>
   <groupId>net.sf.ehcache</groupId>
   <artifactId>ehcache</artifactId>
   <version>[2.0.0]</version>
   <type>pom</type>
</dependency>
```

Configuring EhCache

To configure ehcache, you need to do two steps:

1. configure Hibernate for second level caching
2. specify the second level cache provider

Hibernate 4.x and above

```
<property key="hibernate.cache.use_second_level_cache">true</property>
<property
name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</pr
operty>
```

Hibernate 3.3 and above

```
<property key="hibernate.cache.use_second_level_cache">true</property>
<property
name="hibernate.cache.region.factory_class">net.sf.ehcache.hibernate.EhCacheRegionFactory</prope
rty>
```

Hibernate 3.2 and below

```
<property key="hibernate.cache.use_second_level_cache">true</property>
<property
name="hibernate.cache.region.provider_class">net.sf.ehcache.hibernate.EhCacheProvider</property>
```

Configuring entity objects

This may done in two ways.

1) If you are using **hbm.xml files** then use below configuration:

```
<class name="com.application.entity.DepartmentEntity" table="...">
   <cache usage="read-write"/>
</class>
```

2) Otherwise, if you are using annotations, use these **annotations**:

```
@Entity
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY,
region="department")
public class DepartmentEntity implements Serializable
{
   //code
}
```

For both options, **caching strategy** can be of following types:

- **none** : No caching will happen.
- **read-only** : If your application needs to read, but not modify, instances of a persistent class, a read-only cache can be used.
- **read-write** : If the application needs to update data, a read-write cache might be appropriate.
- **nonstrict-read-write** : If the application only occasionally needs to update data (i.e. if it is extremely unlikely that two transactions would try to update the same item simultaneously), and strict transaction isolation is not required, a nonstrict-read-write cache might be appropriate.
- **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.

Query caching

You can also enable query caching. To do so configure it in your hbm.xml:

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

and where queries are defined in your code, add the method call **setCacheable(true)** to the queries that should be cached:

```
sessionFactory.getCurrentSession().createQuery("...").setCacheable(true).list();
```

By default, Ehcache will create separate cache regions for each entity that you configure for caching. You can change the defaults for these regions by adding the configuration to your ehcache.xml. To provide this configuration file, use this property in hibernate configuration:

```
<property name="net.sf.ehcache.configurationResourceName">/ehcache.xml</property>
```

And use below configuration to override the default configuration:

```
<cache
   name="com.somecompany.someproject.domain.Country"
   maxElementsInMemory="10000"
   eternal="false"
   timeToIdleSeconds="300"
   timeToLiveSeconds="600"
   overflowToDisk="true"
/>
```

Please note that in ehcache.xml, **if eternal=”true” then we should not write timeToIdealSeconds, timeToLiveSeconds**, hibernate will take care about those values
So if you want to give values manually better use eternal=”false” always, so that we can assign values into timeToIdealSeconds, timeToLiveSeconds manually.

timeToIdealSeconds=”seconds” means, if the object in the global cache is ideal, means not using by any other class or object then it will be waited for some time we specified and deleted from the global cache if time is exceeds more than timeToIdealSeconds value.

timeToLiveSeconds=”seconds” means, the other Session or class using this object or not, i mean may be it is using by other sessions or may not, what ever the situation might be, once it competed the time specified timeToLiveSeconds, then it will be removed from the global cache by hibernate.

Example application

In our example application, I have one DepartmentEntity for which I want to enable second level cache using ehcache. Lets record the changes step by step:

1) 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="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
      <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernatedemo</property>
      <property name="hibernate.connection.password">password</property>
      <property name="hibernate.connection.username">root</property>
      <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
      <property name="show_sql">true</property>
      <property name="hbm2ddl.auto">create</property>
      <property
name="hibernate.cache.provider_class">org.hibernate.cache.EhCacheProvider</property>
      <mapping class="hibernate.test.dto.DepartmentEntity"></mapping>
   </session-factory>
</hibernate-configuration>
```

2) DepartmentEntity.java

```
package hibernate.test.dto;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;

import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;

@Entity (name = "dept")
@Table(name = "DEPARTMENT", uniqueConstraints = {
      @UniqueConstraint(columnNames = "ID"),
      @UniqueConstraint(columnNames = "NAME") })

@Cache(usage=CacheConcurrencyStrategy.READ_ONLY, region="department")
```

```
public class DepartmentEntity implements Serializable {

   private static final long serialVersionUID = 1L;

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "ID", unique = true, nullable = false)
   private Integer id;

   @Column(name = "NAME", unique = true, nullable = false, length = 100)
   private String name;

   public Integer getId() {
      return id;
   }

   public void setId(Integer id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

3) HibernateUtil.java

```
package hibernate.test;

import java.io.File;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

public class HibernateUtil
{
   private static final SessionFactory sessionFactory = buildSessionFactory();

   private static SessionFactory buildSessionFactory()
   {
      try
      {
         // Create the SessionFactory from hibernate.cfg.xml
         return new AnnotationConfiguration().configure(new
File("hibernate.cgf.xml")).buildSessionFactory();
      }
      catch (Throwable ex) {
         // Make sure you log the exception, as it might be swallowed
         System.err.println("Initial SessionFactory creation failed." + ex);
         throw new ExceptionInInitializerError(ex);
      }
   }

   public static SessionFactory getSessionFactory() {
      return sessionFactory;
   }
```

```
    public static void shutdown() {
        // Close caches and connection pools
        getSessionFactory().close();
    }
}
```

4) TestHibernateEhcache.java

```
public class TestHibernateEhcache
{
    public static void main(String[] args)
    {
        storeData();

        try
        {
            //Open the hibernate session
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.beginTransaction();

            //fetch the department entity from database first time
            DepartmentEntity department = (DepartmentEntity) session.load(DepartmentEntity.class, new
Integer(1));
            System.out.println(department.getName());

            //fetch the department entity again; Fetched from first level cache
            department = (DepartmentEntity) session.load(DepartmentEntity.class, new Integer(1));
            System.out.println(department.getName());

            //Let's close the session
            session.getTransaction().commit();
            session.close();

            //Try to get department in new session
            Session anotherSession = HibernateUtil.getSessionFactory().openSession();
            anotherSession.beginTransaction();

            //Here entity is already in second level cache so no database query will be hit
            department = (DepartmentEntity) anotherSession.load(DepartmentEntity.class, new Integer(1));
            System.out.println(department.getName());

            anotherSession.getTransaction().commit();
            anotherSession.close();
        }
        finally
        {
            System.out.println(HibernateUtil.getSessionFactory().getStatistics().getEntityFetchCount());
//Prints 1
            System.out.println(HibernateUtil.getSessionFactory().getStatistics().getSecondLevelCacheHitC
ount()); //Prints 1

            HibernateUtil.shutdown();
        }
    }

    private static void storeData()
    {
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
```

```
        DepartmentEntity department = new DepartmentEntity();
        department.setName("Human Resource");

        session.save(department);
        session.getTransaction().commit();
    }
}
```

Output:

```
Hibernate: insert into DEPARTMENT (NAME) values (?)
Hibernate: select department0_.ID as ID0_0_, department0_.NAME as NAME0_0_ from
DEPARTMENT department0_ where department0_.ID=?
Human Resource
Human Resource
Human Resource
1
1
```

In above output, first time the department is fetched from database. but next two times it is fetched from cache. Last fetch is from second level cache.

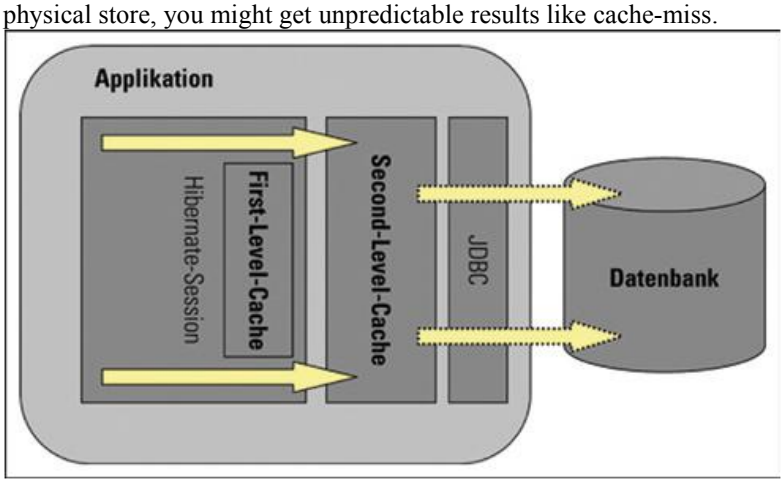
Source Link : http://howtodoinjava.com/hibernate/hibernate-ehcache-configuration-tutorial/
Source code : https://docs.google.com/file/d/0B7yo2HclmjI4eWpOYmZReWRvcEk/edit?usp=sharing

2. How Hibernate Second Level Cache Works?

Caching is facility provided by ORM frameworks which help users to get fast running web application, while help framework itself to reduce number of queries made to database in a single transaction. Hibernate also provide this caching functionality, in two layers.

- **Fist level cache**: This is enabled by default and works in session scope.
- **Second level cache**: This is apart from first level cache which is available to be used globally in session factory scope.

Above statement means, **second level cache is created in session factory scope** and is **available to be used in all sessions** which are created using that particular session factory.
It also means that **once session factory is closed, all cache associated with it die** and cache manager also closed down.
Further, It also means that if you have two instances of session factory (normally no application does that), you will have two cache managers in your application and while accessing cache stored in physical store, you might get unpredictable results like cache-miss.



Hibernate first and second level cache

In this tutorial, I am giving concepts around hibernate second level cache and give example using code snippets.

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.

```
/**
 * Evicts all second level cache hibernate entites. This is generally only
 * needed when an external application modifies the databaase.
 */
public void evict2ndLevelCache() {
   try {
      Map<String, ClassMetadata> classesMetadata = sessionFactory.getAllClassMetadata();
      for (String entityName : classesMetadata.keySet()) {
         logger.info("Evicting Entity from 2nd level cache: " + entityName);
         sessionFactory.evictEntity(entityName);
      }
   } catch (Exception e) {
      logger.logp(Level.SEVERE, "SessionController", "evict2ndLevelCache", "Error evicting 2nd
level hibernate cache entities: ", e);
   }
}
```

To understand more using examples, I wrote an application for testing in which I configured EhCache as 2nd level cache. Lets see various scenarios:

a) Entity is fetched very first time

```
DepartmentEntity department = (DepartmentEntity) session.load(DepartmentEntity.class, new
Integer(1));
System.out.println(department.getName());

System.out.println(HibernateUtil.getSessionFactory().getStatistics().getEntityFetchCount());  //Prints 1
System.out.println(HibernateUtil.getSessionFactory().getStatistics().getSecondLevelCacheHitCount());
   //Prints 0
```

Output: 1 0

Explanation: Entity is not present in either 1st or 2nd level cache so, it is fetched from database.

b) Entity is fetched second time

```
//Entity is fecthed very first time
DepartmentEntity department = (DepartmentEntity) session.load(DepartmentEntity.class, new
Integer(1));
System.out.println(department.getName());
```

```
//fetch the department entity again
department = (DepartmentEntity) session.load(DepartmentEntity.class, new Integer(1));
System.out.println(department.getName());

System.out.println(HibernateUtil.getSessionFactory().getStatistics().getEntityFetchCount());   //Prints 1
System.out.println(HibernateUtil.getSessionFactory().getStatistics().getSecondLevelCacheHitCount());
   //Prints 0
```

Output: 1 0
Explanation: Entity is present in first level cache so, it is fetched from there. No need to go to second level cache.

c) Entity is evicted from first level cache and fetched again

```
//Entity is fecthed very first time
DepartmentEntity department = (DepartmentEntity) session.load(DepartmentEntity.class, new
Integer(1));
System.out.println(department.getName());

//fetch the department entity again
department = (DepartmentEntity) session.load(DepartmentEntity.class, new Integer(1));
System.out.println(department.getName());

//Evict from first level cache
session.evict(department);

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

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

Output: 1 1
Explanation: First time entity is fetched from database. Which cause it store in 1st and 2nd level cache. Second load call fetched from first level cache. Then we evicted entity from 1st level cache. So third load() call goes to second level cache and getSecondLevelCacheHitCount() returns 1.

d) Access second level cache from another session

```
//Entity is fecthed very first time
DepartmentEntity department = (DepartmentEntity) session.load(DepartmentEntity.class, new
Integer(1));
System.out.println(department.getName());

//fetch the department entity again
department = (DepartmentEntity) session.load(DepartmentEntity.class, new Integer(1));
System.out.println(department.getName());

//Evict from first level cache
session.evict(department);

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

department = (DepartmentEntity) anotherSession.load(DepartmentEntity.class, new Integer(1));
System.out.println(department.getName());

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

```
//Prints 2
```

Output: 1 2
Explanation: When another session created from same session factory try to get entity, it is successfully looked up in second level cache and no database call is made.
So now we are clear on how second level cache is used by hibernate.

Source link : http://howtodoinjava.com/hibernate/how-hibernate-second-level-cache-works/
Source code: https://docs.google.com/file/d/0B7yo2HclmjI4cVdNNXFWQmQ1ZTQ/edit?usp=sharing