Hibernate Ehcache Usage

Ehcache is an **open source,** standards-based cache for boosting performance, **offloading** your database, and simplifying scalability. It's the most widely-used Java-based cache because it's **robust, proven,** and **full-featured.** Ehcache scales from in-process, with one or more nodes, all the way to mixed in-process/out-of-process configurations with terabyte-sized caches.

Add **ehcache.xml** file into classpath. In maven, create a file named ehcache.xml in **resources** folder. Then copy the following content:

```
<?xml version="1.0" encoding="utf-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="ehcache.xsd"
         updateCheck="true" monitoring="autodetect"
         dynamicConfig="true">
    <diskStore path="java.io.tmpdir"/>
    <defaultCache
            maxElementsInMemory="1"
            eternal="false"
            timeToIdleSeconds="120"
            timeToLiveSeconds="120"
            overflowToDisk="true"
            />

    <cache name="Article"
           maxElementsInMemory="500"
           eternal="true"
           timeToIdleSeconds="0"
           timeToLiveSeconds="100"
           overflowToDisk="false"
            />
</ehcache>
```

In this file, you can individually change configurations of entities. For example, we changed Article entity configuration by explicitly defining settings. By the way, do not use packet name of the entity, such as example.test.Article. Instead just use the name of the Entity: Article.

Lets explain these configuration statements:

maxElementsInMemory="2" informs the Hibernate to place 2 records (in Hibernate style, 2 objects) in **RAM.** If the **object** size grater then specified max size, in this **case** max size is others will be in the hard disk
eternal="false" indicates the records should not be stored in hard disk permanently.
timeToIdleSeconds="120" the records may live in RAM and afterwards they will be moved to hard disk
timeToLiveSeconds="300" The maximum storage period in the hard disk will be 300 seconds and afterwards the records are permanently deleted from the hard disk (not from database table).
overflowToDisk="true" When the records retrieved are more than **maxElementsInMemory** value, the excess records may be stored in hard disk specified in the **ehcache.xml** file.

Possible **directories** where the records can be **stored:**

user.home: User's home directory
user.dir: User's current working directory
java.io.tmpdir: Default temp file path

The user.home, user.dir and java.io.tmpdir are the folders where the second-level cache records can be stored on the hard disk. The programmer should choose one.

Add the following statements info hibernate.cfg.xml file:

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

Then add the following **dependency** and **repository** into pom.xml:

```
<repositories>
        <repository>
            <id>terracotta-releases</id>
            <url>http://www.terracotta.org/download/reflector/releases</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
<dependencies>
  <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-ehcache</artifactId>
      <version>4.3.5.Final</version>
```

```
    </dependency>
  </dependencies>
```

Lastly, which entity will be cached should be defined by **using** annotations or entity config file:

You have to use the following annotation:

```
@org.hibernate.annotations.Cache(usage =CacheConcurrencyStrategy.READ_ONLY)
```

An example:

```
@Entity
@Table(name = "article"
        , catalog = "softwarevol"
        , uniqueConstraints = @UniqueConstraint(columnNames = "titleSearch")
)
@org.hibernate.annotations.Cache(usage =CacheConcurrencyStrategy.READ_ONLY)
public class Article implements java.io.Serializable {
...
}
```

Test codes:

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
Article article=(Article)session.load(Article.class,(short)1);
System.out.println(article.getTitle());

session.getTransaction().commit();
Session session1 = HibernateUtil.getSessionFactory().getCurrentSession();
session1.beginTransaction();
article=(Article)session1.load(Article.class,(short)1);
System.out.println(article.getTitle());
session1.getTransaction().commit();
```

When running this code, second request will be fetched from cache not database.

Remove Operations

```
SessionFactory factory=HibernateUtil.getSessionFactory();
Cache cache=factory.getCache();
//Removes Article instance which has id 1
cache.evictEntity(Article.class,(short)1);

//Removes all Article type instances
cache.evictEntityRegion(Article.class);


/* For the second-level cache, there are methods defined on
SessionFactory for evicting the cached state of an instance,
entire class, collection instance or entire collection role. */
cache.evictEntity(Cat.class, catId); //evict a particular Cat
cache.evictEntityRegion(Cat.class);  //evict all Cats
cache.evictCollection("Cat.kittens", catId); //evict a particular collection of kittens
cache.evictCollectionRegion("Cat.kittens"); //evict all kitten collections
```