Hibernate Tutorial 12 Caching Objects

By Gary Mak
hibernatetutorials@metaarchit.com
September 2006

1. Level of caching

The general concept of caching persistent objects is that when an object is first read from external storage, a copy of it will be stored in the cache. For the subsequent readings of the same object, it can be retrieved from the cache directly. Since caches are typically stored in memory or local disk, it will be faster to read an object from cache than external storage. If using properly, caching can greatly improve the performance of our application.

As a high performance O/R mapping framework, Hibernate supports the caching of persistent objects at different levels. Suppose we get an object with same identifier for two times within a session, will Hibernate query the database for two times?

```
Session session = factory.openSession();
try {
    Book book1 = (Book) session.get(Book.class, id);
    Book book2 = (Book) session.get(Book.class, id);
} finally {
    session.close();
}
```

If we inspect the SQL statements executed by Hibernate, we will find that only one database query is made. That means Hibernate is caching our objects in the same session. This kind of caching is called “first level caching”, whose caching scope is a session.

But how about getting an object with same identifier for two times in two different sessions?

```
Session session1 = factory.openSession();
try {
    Book book1 = (Book) session1.get(Book.class, id);
} finally {
    session1.close();
}
Session session2 = factory.openSession();
try {
    Book book2 = (Book) session2.get(Book.class, id);
} finally {
    session2.close();
}
```

We will find that two database queries are made. That means Hibernate is not caching the persistent objects across different sessions by default. We need to turn on this “second level caching” whose caching scope is a session factory.

2. The second level caching

To turn on the second level caching, the first step is to choose a cache provider in the Hibernate configuration file “hibernate.cfg.xml”. Hibernate supports several kinds of cache implementation, such as EHCache, OSCache, SwarmCache and JBossCache. In a non-distributed environment, we may simply choose EHCache, which is also the default cache provider for Hibernate.

```
<hibernate-configuration>
    <session-factory>
        ...
        <property name="cache.provider_class">
            org.hibernate.cache.EhCacheProvider
        </property>
        ...
    </session-factory>
</hibernate-configuration>
```

We can configure EHCache through a configuration file “ehcache.xml” located in the source root folder. We can specify different settings for different “cache regions”, which are used for storing different kinds of objects. The parameter “eternal” indicates that the elements will be expired in a time period if set to false. The parameters “timeToIdleSeconds” and “timeToLiveSeconds” indicate how long the elements will be expired.

```
<ehcache>
    <diskStore path="java.io.tmpdir" />
    <defaultCache
        maxElementsInMemory="10000"
        eternal="false"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        overflowToDisk="true"
        />
    <cache name="com.metaarchit.bookshop.Book"
        maxElementsInMemory="10000"
        eternal="false"
        timeToIdleSeconds="300"
        timeToLiveSeconds="600"
        overflowToDisk="true"
        />
</ehcache>
```

To monitor the caching activities of Hibernate at runtime, we can add the following line to the log4j configuration file “log4j.properties”.

```
log4j.logger.org.hibernate.cache=debug
```

Now, we need to enable caching for a particular persistent class. The cached objects will be stored at a region having the same name as the persistent class, e.g. “com.metaarchit.bookshop.Book”. There are several cache usages we can choose for a persistent class. If the persistent objects are read only and never modified, the most efficient usage is “read-only”.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        <cache usage="read-only" />
        ...
    </class>
</hibernate-mapping>
```

After the caching of Book class being enabled, we can see that only one SQL query is made even the book object is loaded for two times in two different sessions.

```
Session session1 = factory.openSession();
try {
    Book book1 = (Book) session1.get(Book.class, id);
} finally {
    session1.close();
}
Session session2 = factory.openSession();
try {
    Book book2 = (Book) session2.get(Book.class, id);
} finally {
    session2.close();
}
```

However, if we modify the book object in one session and flush the changes, an exception will be thrown for updating a read-only object.

```
Session session1 = factory.openSession();
try {
    Book book1 = (Book) session1.get(Book.class, id);
    book1.setName("New Book");
    session1.save(book1);
    session1.flush();
} finally {
    session1.close();
}
```

So for an updatable object, we should choose another cache usage “read-write”. Hibernate will invalidate the object from cache before it is updated.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        <cache usage="read-write" />
        ...
    </class>
</hibernate-mapping>
```

In some cases, e.g. the database updated by other applications, you may want to invalidate the cached objects manually. You can do it through the methods provided by the session factory. You can invalidate either one instance or all instances of a persistent class.

```
factory.evict(Book.class);

factory.evict(Book.class, id);

factory.evictEntity("com.metaarchit.bookshop.Book");

factory.evictEntity("com.metaarchit.bookshop.Book", id);
```

3. Caching associations

For the associated publisher object of a book, will it be also cached when its parent book object is cached?

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        <cache usage="read-write" />
        ...
        <many-to-one name="publisher" class="Publisher" column="PUBLISHER_ID" />
    </class>
</hibernate-mapping>
```

If you initialize the association in two different sessions, you will find that it is loaded for two times. It is because Hibernate caches only the identifier of publisher in the region of Book.

```
Session session1 = factory.openSession();
try {
    Book book1 = (Book) session1.get(Book.class, id);
    Hibernate.initialize(book1.getPublisher());
} finally {
    session1.close();
}
```

```
Session session2 = factory.openSession();
try {
    Book book2 = (Book) session2.get(Book.class, id);
    Hibernate.initialize(book2.getPublisher());
} finally {
    session2.close();
}
```

To cache the publisher objects in their own region, we need to enable the caching for the Publisher class also. We can do it the same way as for Book class. The cached publisher objects will be stored at a region with the name “com.metaarchit.bookshop.Publisher”.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Publisher" table="PUBLISHER">
        <cache usage="read-write" />
        ...
    </class>
</hibernate-mapping>
```

4. Caching collections

For the associated chapters of a book to be cached, we enable caching for the Chapter class. The cached chapter objects will be stored at a region with the name “com.metaarchit.bookshop.Chapter”.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        <cache usage="read-write" />
        ...
        <set name="chapters" table="BOOK_CHAPTER">
            <key column="BOOK_ID" />
            <one-to-many class="Chapter" />
        </set>
    </class>
</hibernate-mapping>

<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Chapter" table="CHAPTER">
        <cache usage="read-write" />
        ...
        <many-to-one name="book" class="Book" column="BOOK_ID" />
    </class>
</hibernate-mapping>
```

Then we can try initializing the collection in two different sessions. We expect that no query should

be made for the second session.

```
Session session1 = factory.openSession();
try {
    Book book1 = (Book) session1.get(Book.class, id);
    Hibernate.initialize(book1.getChapters());
} finally {
    session1.close();
}
Session session2 = factory.openSession();
try {
    Book book2 = (Book) session2.get(Book.class, id);
    Hibernate.initialize(book2.getChapters());
} finally {
    session2.close();
}
```

By inspecting the SQL statements, we can find that there is still one query made. The reason is that, unlike a many-to-one association, a collection will not be cached by default. We need to turn on it manually by specifying cache usage to the collection. For the chapter collection of a book, it will be stored at a region with the name “com.metaarchit.bookshop.Book.chapters”.

How can a collection be cached by Hibernate? If the collection is storing simple values, the values themselves will be cached. If the collection is storing persistent objects, the identifiers of the objects will be cached at the collection region and the persistent objects themselves will be cached at their own region.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        <cache usage="read-write" />
        ...
        <set name="chapters" table="BOOK_CHAPTER">
            <cache usage="read-write" />
            <key column="BOOK_ID" />
            <one-to-many class="Chapter" />
        </set>
    </class>
</hibernate-mapping>
```

To invalidate a particular collection or all the collections in a region, we can use the following methods provided by the session factory.

```
factory.evictCollection("com.metaarchit.bookshop.Book.chapters");

factory.evictCollection("com.metaarchit.bookshop.Book.chapters", id);
```

For a bi-directional one-to-many/many-to-one association, you should call the evictCollection()

method on the collection end after the single end is being updated.

```
Session session1 = factory.openSession();
try {
    Book book1 = (Book) session1.get(Book.class, id);
    Chapter chapter = (Chapter) book1.getChapters().iterator().next();
    chapter.setBook(null);
    session1.saveOrUpdate(chapter);
    session1.flush();
    factory.evictCollection("com.metaarchit.bookshop.Book.chapters", id);
} finally {
    session1.close();
}
Session session2 = factory.openSession();
try {
    Book book2 = (Book) session2.get(Book.class, id);
    Hibernate.initialize(book2.getChapters());
} finally {
    session2.close();
}
```

5. Caching queries

In addition to caching objects loaded by a session, a query with HQL can also be cached. Suppose that we are running the same query in two different sessions.

```
Session session1 = factory.openSession();
try {
    Query query = session1.createQuery("from Book where name like ?");
    query.setString(0, "%Hibernate%");
    List books = query.list();
} finally {
    session1.close();
}
Session session2 = factory.openSession();
try {
    Query query = session2.createQuery("from Book where name like ?");
    query.setString(0, "%Hibernate%");
    List books = query.list();
} finally {
    session2.close();
}
```

By default, the HQL queries are not cached. We must first enable the query cache in the Hibernate configuration file.

```
<hibernate-configuration>
    <session-factory>
        ...
        <property name="cache.use_query_cache">true</property>
        ...
    </session-factory>
</hibernate-configuration>
```

Then we need to set the query to be cacheable before execution. The query result is cached at a region with the name “org.hibernate.cache.StandardQueryCache” by default.

How can a query be cached by Hibernate? If the query is returning simple values, the values themselves will be cached. If the query is returning persistent objects, the identifiers of the objects will be cached at the query region and the persistent objects themselves will be cached at their own region.

```
Session session1 = factory.openSession();
try {
    Query query = session1.createQuery("from Book where name like ?");
    query.setString(0, "%Hibernate%");
    query.setCacheable(true);
    List books = query.list();
} finally {
    session1.close();
}
Session session2 = factory.openSession();
try {
    Query query = session2.createQuery("from Book where name like ?");
    query.setString(0, "%Hibernate%");
    query.setCacheable(true);
    List books = query.list();
} finally {
    session2.close();
}
```

We can also specify the cache region for a query. This enables separating query caches at different regions and reducing the number of caches in one particular region.

```
...
query.setCacheable(true);
query.setCacheRegion("com.metaarchit.bookshop.BookQuery");
```