Hibernate cache

Довольно часто в java приложениях с целью снижения нагрузки на БД используют кеш. Не много людей реально понимают как работает кеш под капотом, добавить просто аннотацию не всегда достаточно, нужно понимать как работает система. Поэтому этой статье я попытаюсь раскрыть тему про то, как работает кеш популярного ORM фреймворка. Итак, для начала немного теории.  
  
Прежде всего Hibernate cache — это 3 уровня кеширования:

* Кеш первого уровня (First-level cache);
* Кеш второго уровня (Second-level cache);
* Кеш запросов (Query cache);

Кеш первого уровня

Кеш первого уровня всегда привязан к объекту сессии. Hibernate всегда по умолчанию использует этот кеш и его нельзя отключить. Давайте сразу рассмотрим следующий код:

SharedDoc persistedDoc = (SharedDoc) session.load(SharedDoc.class, docId);

System.out.println(persistedDoc.getName());

user1.setDoc(persistedDoc);

persistedDoc = (SharedDoc) session.load(SharedDoc.class, docId);

System.out.println(persistedDoc.getName());

user2.setDoc(persistedDoc);

Возможно, Вы ожидаете, что будет выполнено 2 запроса в БД? Это не так. В этом примере будет выполнен 1 запрос в базу, несмотря на то, что делается 2 вызова load(), так как эти вызовы происходят в контексте одной сессии. Во время второй попытки загрузить план с тем же идентификатором будет использован кеш сессии.  
Один важный момент — при использовании метода load() Hibernate не выгружает из БД данные до тех пор пока они не потребуются. Иными словами — в момент, когда осуществляется первый вызов load, мы получаем прокси объект или сами данные в случае, если данные уже были в кеше сессии. Поэтому в коде присутствует getName() чтобы 100% вытянуть данные из БД. Тут также открывается прекрасная возможность для потенциальной оптимизации. В случае прокси объекта мы можем связать два объекта не делая запрос в базу, в отличии от метода get(). При использовании методов save(), update(), saveOrUpdate(), load(), get(), list(), iterate(), scroll() всегда будет задействован кеш первого уровня. Собственно, тут нечего больше добавить.

Кеш второго уровня

Если кеш первого уровня привязан к объекту сессии, то кеш второго уровня привязан к объекту-фабрике сессий (Session Factory object). Что как бы подразумевает, что видимость этого кеша гораздо шире кеша первого уровня. Пример:

Session session = factory.openSession();

SharedDoc doc = (SharedDoc) session.load(SharedDoc.class, 1L);

System.out.println(doc.getName());

session.close();

session = factory.openSession();

doc = (SharedDoc) session.load(SharedDoc.class, 1L);

System.out.println(doc.getName());

session.close();

В данном примере будет выполнено 2 запроса в базу, это связано с тем, что по умолчанию кеш второго уровня отключен. Для включения необходимо добавить следующие строки в Вашем конфигурационном файле JPA (persistence.xml):

<property name="hibernate.cache.provider\_class" value="net.sf.ehcache.hibernate.SingletonEhCacheProvider"/>

*//или в более старых версиях*

*//<property name="hibernate.cache.provider\_class" value="org.hibernate.cache.EhCacheProvider"/>*

<property name="hibernate.cache.use\_second\_level\_cache" value="true"/>

Обратите внимание на первую строку. На самом деле, хибернейт сам не реализует кеширование как таковое. А лишь предоставляет структуру для его реализации, поэтому подключить можно любую реализацию, которая соответствует спецификации нашего ORM фреймворка. Из популярных реализаций можна выделить [следующие](http://docs.jboss.org/hibernate/core/3.3/reference/en/html/performance.html#performance-cache):

* EHCache
* OSCache
* SwarmCache
* JBoss TreeCache

Помимо всего этого, вероятней всего, Вам также понадобится отдельно настроить и саму реализацию кеша. В случае с EHCache это нужно сделать в файле [ehcache.xml](http://ehcache.org/ehcache.xml). Ну и в завершение еще нужно указать самому хибернейту, что именно кешировать. К счастью, это очень легко можно сделать с помощью аннотаций, например так:

@Entity

@Table(name = "shared\_doc")

@Cache(usage = CacheConcurrencyStrategy.READ\_WRITE)

public class SharedDoc{

private Set<User> users;

}

Только после всех этих манипуляций кеш второго уровня будет включен и в примере выше будет выполнен только 1 запрос в базу.  
Еще одна важная деталь про кеш второго уровня про которую стоило бы упомянуть — хибернейт не хранит сами объекты Ваших классов. Он хранит информацию в виде массивов строк, чисел и т. д. И идентификатор объекта выступает указателем на эту информацию. Концептуально это нечто вроде Map, в которой id объекта — ключ, а массивы данных — значение. Приблизительно можно представить себе это так:

1 -> { "Pupkin", 1, null , {1,2,5} }

Что есть очень разумно, учитывая [сколько лишней памяти](http://habrahabr.ru/blogs/java/134102/) занимает каждый объект.  
Помимо вышесказанного, следует помнить — зависимости Вашего класса по умолчанию также не кешируются. Например, если рассмотреть класс выше — SharedDoc, то при выборке коллекция users будет доставаться из БД, а не из кеша второго уровня. Если Вы хотите также кешировать и зависимости, то класс должен выглядеть так:

@Entity

@Table(name = "shared\_doc")

@Cache(usage = CacheConcurrencyStrategy.READ\_WRITE)

public class SharedDoc{

@Cache(usage = CacheConcurrencyStrategy.READ\_WRITE)

private Set<User> users;

}

И последняя деталь — чтение из кеша второго уровня происходит только в том случае, если нужный объект не был найден в кеше первого уровня.

Кеш запросов

Перепишем первый пример так:

Query query = session.createQuery("from SharedDoc doc where doc.name = :name");

SharedDoc persistedDoc = (SharedDoc) query.setParameter("name", "first").uniqueResult();

System.out.println(persistedDoc.getName());

user1.setDoc(persistedDoc);

persistedDoc = (SharedDoc) query.setParameter("name", "first").uniqueResult();

System.out.println(persistedDoc.getName());

user2.setDoc(persistedDoc);

Результаты такого рода запросов не сохраняются ни кешом первого, ни второго уровня. Это как раз то место, где можно использовать кеш запросов. Он тоже по умолчанию отключен. Для включения нужно добавить следующую строку в конфигурационный файл:

<property name="hibernate.cache.use\_query\_cache" value="true"/>

а также переписать пример выше добавив после создания объекта Query (то же справедливо и для Criteria):

Query query = session.createQuery("from SharedDoc doc where doc.name = :name");

query.setCacheable(true);

Кеш запросов похож на кеш второго уровня. Но в отличии от него — ключом к данным кеша выступает не идентификатор объекта, а совокупность параметров запроса. А сами данные — это идентификаторы объектов соответствующих критериям запроса. Таким образом, этот кеш рационально использовать с кешем второго уровня.

Стратегии кеширования

Стратегии кеширования определяют поведения кеша в определенных ситуациях. Выделяют четыре группы:

* Read-only
* Read-write
* Nonstrict-read-write
* Transactional

Подробней можно прочитать [тут](http://docs.jboss.org/hibernate/core/3.3/reference/en/html/performance.html#performance-cache-mapping).

Cache region

Регион или область — это логический разделитель памяти вашего кеша. Для каждого региона можна настроить свою политику кеширования (для EhCache в том же ehcache.xml). Если регион не указан, то используется регион по умолчанию, который имеет полное имя вашего класса для которого применяется кеширование. В коде выглядит так:

@Cache(usage = CacheConcurrencyStrategy.READ\_WRITE, region = "STATIC\_DATA")

А для кеша запросов так:

query.setCacheRegion("STATIC\_DATA");

*//или в случае критерии*

criteria.setCacheRegion("STATIC\_DATA");

Что еще нужно знать?

Во время разработки приложения, особенно сначала, очень удобно видеть действительно ли кешируются те или иные запросы, для этого нужно указать фабрике сессий следующие свойства:

<property name="hibernate.show\_sql" value="true"/>

<property name="hibernate.format\_sql" value="true"/>

В дополнение фабрика сессий также может генерировать и сохранять статистику использования всех объектов, регионов, зависимостей в кеше:

<property name="hibernate.generate\_statistics" value="true"/>

<property name="hibernate.cache.use\_structured\_entries" value="true"/>

Для этого есть объекты Statistics для фабрики и SessionStatistics для сессии.  
  
Методы сессии:  
flush() — синхронизирует объекты сессии с БД и в то же время обновляет сам кеш сессии.  
evict() — нужен для удаления объекта из кеша cессии.  
contains() — определяет находится ли объект в кеше сессии или нет.  
clear() — очищает весь кеш.

Заключение

Вот собственно и все. Естественно, что вне статьи осталось еще не мало разных нюансов, которые возникают походу работы с кешем, а также немало проблем. Но это уже тема для другой статьи.