JPA 2.1 Second Level Cache

JPA has two levels of cache, the first use of them without even knowing it, from the beginning of the startup of the application, since it requires no configuration additional. JPA is contextual and that context provides what we call "first-level cache".

Each object that is loaded by the EntityManager end up in this EntityManager cache, since then, every time you get the same object by key, it will be returned immediately, without the need for a new access to the database for this.

This manner the economy of queries is very large, it is sufficient that the same instance of EntityManager is used as the first-level cache is done only insider her. So we have a problem: each instance of EntityManager lives only during a user request, and close shortly thereafter to release server resources.

The first level cache is not shared, so when a user loads a list of users (List<User> users) the other user has to load again the same list.

This means that a new request for every cache is mounted again, then comes the 2nd level cache used for objetcts that should last more than a scope of request.

There are 5 approaches to work with the cache

1. *ALL (All entities and entity-related state and data are cached.)*

2. *DISABLE_SELECTIVE(Caching is enabled for all entities except those for which Cacheable(false) is specified.)*

3. *ENABLE_SELECTIVE(Caching is enabled for all entities for Cacheable(true) is specified.)*

4. *NONE(Caching is disabled for the persistence unit.)*

5. *UNSPECIFIED(Caching behavior is undefined: provider-specific defaults may apply. When we do not specify anything, this is the value taken by shared-cache-mode, in which case, each JPA implementation is free to define which mode is enable and can be done by code).*

   These 5 ways my favorite is

   <shared-cache-mode> ENABLE_SELECTIVE </ shared-cache-mode>

   this configuration says, enables all are with @Cacheble, the second level cache is shared between all instances of EntityManager. In practice this means that if a user uploaded a list of Users it is available, use it to himself again in future requests or for other users, so we have to understand very well knowing what to put in the cache.

   Implementing Cache

   To implement the cache have the Infinispan and EHcache both have specific settings that you can specify for each entity the maximum number of objetcts that will stay in memory and what will be stored on disk.

   <property name="hibernate.cache.region.factory_class" value="org.jboss.as.jpa.hibernate4.infinispan.InfinispanRegionFactory" />

Contained in the documentation and configure the setting of the second level cache provider, you will have to enable it by default it is set to false.

<property name="hibernate.cache.use_second_level_cache" value="true" />

Enable Query Cache

<property name="hibernate.cache.use_query_cache" value="true" />

Enable Cluster cache

This configuration minimizes writes, at the cost of more frequent readings, being:
This setting is most useful for clustered caches and, in Hibernate 3, is enabled by default for clusterd cache implementations.

<property name="hibernate.cache.use_minimal_puts" value="true" />

Enable Statistics

<property name="hibernate.cache.infinispan.statistics" value="false" />

Infinispan has the ability to expose JMX statistics when the statistics are off and infinispan are disabled using this method, the statistics for the infinispan cache mamanger and all manage caches (entity, colection, etc..) are enable.

<property name="hibernate.cache.infinispan.statistics" value="false" />

Evict cached

Evict cached data is also essential in order to save memory when the cache entries are no longer needed. You can configure the cache expiration policy, which determines when the data is updated in the cache(hour, day, an so on) according to the requirements for that entity. Configure data clearing the cache can be done either programmatically or by the configuration file.

```
<property name="hibernate.cache.infinispan.entity.eviction.strategy"
value="LRU" />
<property
name="hibernate.cache.infinispan.entity.eviction.wake_up_interval"
value="2000" />
<property name="hibernate.cache.infinispan.entity.eviction.max_entries"
value="5000" />
<property name="hibernate.cache.infinispan.entity.expiration.lifespan"
value="60000" />
<property name="hibernate.cache.infinispan.entity.expiration.max_idle"
value="30000" />
```

Change an object that is in the cache

Now we have a second-level cache working, what happens when you carry a list of Users that cache and a user edits the registry?

How the change will be made by the JPA itself, the moment that changed object is sent to the bank, it is also updated in the second level cache.

```
/**
* Insert/update entity data into cache when read
* from database and when committed into database.
```

** Forces refresh of cache for items read from database.*
**/*
REFRESH
TypedQuery<User> query = em.createQuery(cq);
query.setHint("javax.persistence.cache.storeMode",
CacheStoreMode.REFRESH);

If the old version of the object was loaded into the first level of any other EntityManager cache, this object would be outdated but working as a request scope of the short lifetime of the EntityManager first level cache is a plus.

As time which lasts a request is usually very small, and this woulld also be the life of the EntityManager, so there was some error would need competition in the same "second" two users from editing the same instance of an entity class - which usually change little, because if it were not so, would not be in the second level cache.

The very important point is the probability of competition versus the benefit of the cache, as each application has a different dynamic, it must be assessed case by case, the application I use these concepts, we will certainly have more queriesthan users querying users changing, with this in mind we can choose to add
to thecache that have not changed their states frequently, usually lists change little compared to the amount of views that happen.

Imagine how many times the same query would be made, always with the same result. Therefore we have the query cache, this feature, however, not yet part of the JPA, then we need to use specific features

of Hibernate, such information contained in specificação see:

Note:

Because support for the second-level cache is not required by theJava Persistence API specification, setting the second-level cache inpersistence.xml mode will have no effect When You Use quepersistence provider does not implement the second-level cache.

The JPA already understand that at certain times we will need specific features of the implementation we are using, allowing us to develop applications withspecific features of an implementation.

Configuring the query cache

We need to put a query.setHint ("javax.persistence.cache.storeMode")throughout the query cache you want to do.

We need to be careful is that every query that is cached, need to have the entities involved with the @ Cacheable, if not hibernate will do so:

He will get the query cache and for each record found in the query he will make aLOAD, being: if the query has 90 records, it does 90 selects to fetch entities, sinceonly the cached query this, and entities not .