Cache Handling

The table defined by the object dispatched through input entity is mapped to execute the access. JPA uses the cache to improve this performance issue. Current standard version defines 1 level Cache only. In JPA 2.0, 2level Cache definition is added.

1 Level Cache

It is the cache defined in the Entity Manager which plays a role of storing objects read within the transaction and used between the start and end of transaction. JPA implements extracts the relevant implements from the 2nd Cache from the 2nd if loading the same object over once in one transaction. In addition, if changing the property of object in one transaction scope, it is automatically reflected in DB at the time of transaction termination. That is, if re-inquiry is performed for the same object in one transaction, reduce the number of DB approach using Cache, improving the application performance.

Sample Source

```
@Test
public void testFindUser() throws Exception {
    newTransaction();

    SetUpInitCacheData.initializeData(em);

    User user = (User) em.find(User.class, "User1");

    Set roles = (Set)user.getRoles();
    roles.iterator();

    // 1. Read from Cache
    user = (User) em.find(User.class, "User1");

    roles = (Set)user.getRoles();
    roles.iterator();

    closeTransaction();
}
```

If created as above, Persistence object persisted through SetUpInitCacheData.initializeData(em) in same transaction is saved in 1LC, when inquiring the same Persistence object like #1 code next, it can be inquired through Cache without re-approaching DB.

2 Level Cache

2 Level Cache is defined in JPA 2.0. This chapter intends to guide the support of JPA in hibernate. 2 Level Cache is the Cache in the application unit and supports Cache function in application viewpoint. It is processed as the method to save the Persistence object loaded through several transactions from the Session Factory level. In persistence.xml file, define hibernate.cache.use_second_level_cache, hibernate.cache.provider_class and define <cache> property of Persistence Class mapping file saved in 2 Level Cache and 2LC can be applied for specific Persistence configuring relevant application. Following is the part of persistence.xml file where property for 2 Level Cache is defined.

Config file

```
<persistence-unit name="HSQLMCUnit" transaction-type="RESOURCE_LOCAL">
    ...
    <properties>
        <property name="hibernate.cache.use_second_level_cache" value="true"/>
        <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.EhCacheProvider"/>
    ...
    </properties>
```

```
</persistence-unit>
```

Following is the part of Entity class where cache property is set as READ_WRITE.

```
// Designate in Department
@Cache(usage=CacheConcurrencyStrategy.READ_WRITE)
public class Department {

  // Designate in Join of Department and User
  @Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
  @OneToMany(targetEntity=User.class, mappedBy="department"
                  ,cascade={CascadeType.PERSIST, CascadeType.MERGE})
  private Set<User> users ;
}

// Designate in User
@Entity
@Cache(usage=CacheConcurrencyStrategy.READ_WRITE)
public class User {
}
```

From above, Cache can be designated in 3 places such as Department, User, Set<User>. Each of them designates cache for Join of User and Department as well as Department and User. Cache property can be defined as the following value in addition to READ_WRITE defined above.

- READ_OLNY: can be used if Persistence object is not changed. Since there is no updating, it can be used safely in distributed environment and provide the fastest performance.
- NONSTRICT_READ_WRITE: use if it is not required to strictly apply transaction isolation.
- TRANSACTIONAL: guarantee complete transaction but provide the slowest performance.

Sample Source

```
@Test
public void testFindDepartment() throws Exception {

   // data input.
   newTransaction();
   SetUpInitCacheData.initializeData(em);
   closeTransaction();

   //To use Hibernate method, create Hibernate Session Factory(obtain from Entity Manager)
   newHibernateSessionFactory();

   // Extract data using 2Level Cache(use method of Hibernate)
   newSession();
   Department department = (Department) session.get(Department.class, "Dept1");

   Set users = department.getUsers();
   users.iterator();
   closeSession();

   // Extract data using 2Level Cache(use method of Hibernate)
   newSession();
   department = (Department) session.get(Department.class, "Dept1");

   users = department.getUsers();
   users.iterator();

   // Hibernate Session close
   closeSession();
}
```

Above example shows that 2 Level Cache value is returned without accessing DB through 2nd Session.get().