Hibernate Second Level Cache – 2021-2022

**The first-level cache**

The first level cache type is the session cache. The session cache caches object within the current session but this is not enough for long level i.e. session factory scope. Hibernate uses first-level cache by default and you have nothing to do to use first-level cache. **Once the session is closed, first-level cache is terminated** as well. This is actually desirable, as it allows for concurrent sessions to work with entity instances in isolation from each other.

**The Second-level cache**

On the other hand, second-level cache is *SessionFactory*-scoped, meaning it is shared by all sessions created with the same session factory.

Hibernate second-level caching is designed to be unaware of the actual cache provider used. Hibernate only needs to be provided with an implementation of the *org.hibernate.cache.spi.RegionFactory* interface which encapsulates all details specific to actual cache providers. Basically, it acts as a bridge between Hibernate and cache providers.

**Enabling Second-Level Caching**

**hibernate.cache.use\_second\_level\_cache=true hibernate.cache.region.factory\_class=org.hibernate.cache.ehcache.EhCacheRegionFactory**

**Cache Concurrency Strategy**

***READ\_ONLY***: Used only for entities that never change (exception is thrown if an attempt to update such an entity is made). It is very simple and performant. Very suitable for some static reference data that don't change

***NONSTRICT\_READ\_WRITE***: Cache is updated after a transaction that changed the affected data has been committed. Thus, strong consistency is not guaranteed and there is a small time window in which stale data may be obtained from cache. This kind of strategy is suitable for use cases that can tolerate eventual consistency.

***READ\_WRITE***: This strategy guarantees strong consistency which it achieves by using ‘soft' locks: When a cached entity is updated, a soft lock is stored in the cache for that entity as well, which is released after the transaction is committed. All concurrent transactions that access soft-locked entries will fetch the corresponding data directly from database

***TRANSACTIONAL***: Cache changes are done in distributed XA transactions. A change in a cached entity is either committed or rolled back in both database and cache in the same XA transaction.

**Collection Cache**

**Collections are not cached by default, and we need to explicitly mark them as cacheable**. For example.

@Entity

@Cacheable

**@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ\_WRITE)**

public class Foo {

...

**@Cacheable**

**@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ\_WRITE)**

**@OneToMany**

**private Collection<Bar> bars;**

// getters and setters

}

**Query Cache**

Results of HQL queries can also be cached. This is useful if you frequently execute a query on entities that rarely change.

To enable query cache, set the value of *hibernate.cache.use\_query\_cache* property to *true*:

**hibernate.cache.use\_query\_cache=true**

Then, for each query you have to explicitly indicate that the query is cacheable (via an *org.hibernate.cacheable* query hint):

entityManager.createQuery("select f from Foo f")

.setHint("org.hibernate.cacheable", **true**).getResultList();

**Complete example is given below for Spring Boot**

**Dependency for pom.xml**

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-ehcache</artifactId>

</dependency>

application.properties

# For Hibernate Second Level Caching

**spring.jpa.properties.hibernate.cache.use\_second\_level\_cache=true**

**spring.jpa.properties.hibernate.cache.region.factory\_class=org.hibernate.cache.ehcache.EhCacheRegionFactory**

**Entity Layer**

@Data

@Entity

@Table(name = "appuser")

@Cacheable

@Cache(usage = CacheConcurrencyStrategy.***READ\_WRITE***)

**public** **class** AppUser {

@Id

@GeneratedValue

@Column(name = "ID")

**private** **long** id;

@Column(name = "NAME")

**private** String name;

**public** AppUser() {}

**public** AppUser(String name) {

**this**.name = name;

}

}

@Repository

**public** **interface** UserRepository **extends** JpaRepository<AppUser, Long> {

}

@Service

**public** **class** UserService {

@Autowired

**private** UserRepository repo;

**public** **void** saveUser(AppUser user) {

repo.save(user);

}

**public** AppUser getUserById(Long id) {

AppUser user = repo.findById(id).get();

**int** cacheSize = CacheManager.***ALL\_CACHE\_MANAGERS***.size();

System.***out***.println("Cache Size: "+cacheSize);

**int** size = CacheManager.***ALL\_CACHE\_MANAGERS***.get(0)

.getCache("com.ddlab.rnd.cache.AppUser").getSize();

System.***out***.println("Cache Size : "+size);

**return** user;

}

}

**public** **class** TestUserCache {

**public** **static** **void** testCache(ApplicationContext applicationContext) {

UserService service = applicationContext.getBean(UserService.**class**);

AppUser user = **new** AppUser("Hati");

service.saveUser(user);

user = service.getUserById(Long.*valueOf*(320));

System.***out***.println("User : "+user);

}

}

@SpringBootApplication

**public** **class** StandaloneJPAHibernateMappingApp {

**public** **static** **void** main(String[] args) {

ApplicationContext applicationContext = SpringApplication.*run*(StandaloneJPAHibernateMappingApp.**class**, args);

**TestUserCache.*testCache*(applicationContext);**

}

}