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.

**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 only needs to be provided with an implementation of the *org.hibernate.cache.spi.RegionFactory* interface

**How to Enable Second-Level Caching**

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

**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

}

**For Hibernate Second Level Caching in Spring**

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

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

**Difference between Hibernate Session and SessionFactory**

* **SessionFactory** is a factory class for Session objects. It is available for the whole application while a Session is only available for particular transaction.
* **Session** is short-lived while SessionFactory objects are long-lived. SessionFactory provides a second level cache and Session provides a first level cache.
* **SessionFactory** is an interface. SessionFactory can be created by providing Configuration object and it is threadsafe
* **A Session** is used to get a physical connection with a database and it is not threadsafe.

**Cache Implementation in SpringBoot**

**Application.properties**

spring.datasource.url=jdbc:postgresql://localhost:5432/postgres

spring.datasource.username=postgres

spring.datasource.password=postgres

# The SQL dialect makes Hibernate generate better SQL for the chosen database

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect

# Hibernate ddl auto (create, create-drop, validate, update)

spring.jpa.hibernate.ddl-auto = update

spring.jpa.properties.hibernate.generate\_statistics=true

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

spring.jpa.properties.hibernate.temp.use\_jdbc\_metadata\_defaults=false

**Application Configuration**

@EnableCaching

@Configuration

**public** **class** AppConfig {

}

**Entity Layer**

@Entity(name = "Project")

@Table(name = "Project")

@Getter

@Setter

@ToString

**public** **class** Project {

@Id

@GeneratedValue

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

@Column(name = "name")

**private** String name;

@Column(name = "description")

**private** String description;

// @OneToMany(cascade = CascadeType.ALL)

@OneToMany(fetch = FetchType.***EAGER***, cascade = CascadeType.***ALL***)

**private** List<Employee> empList;

}

Repository Layer

@Repository

**public** **interface** ProjectRepository **extends** CrudRepository<Project, Long> {

@Cacheable(value = "project")

Optional<Project> findById(Long id);

@CacheEvict(cacheNames = "project", beforeInvocation = **false**, key = "#project.id")

Project save(Project project);

}

Service Layer

**public** **interface** ProjectService {

Project save(Project project);

Project getById(Long id);

}

Service Implementation

@Service

**public** **class** ProjectServiceImpl **implements** ProjectService {

@Autowired

**private** ProjectRepository projRepo;

@Override

**public** Project save(Project project) {

**return** projRepo.save(project);

}

@Override

**public** Project getById(Long id) {

Optional<Project> projOpt = projRepo.findById(id);

**return** projOpt.get();

}

}

AutoRun for Testing

@Component

**public** **class** AutoRun {

@Autowired

**private** ProjectService projService;

**private** **void** saveProject1() {

Address adrs1 = **new** Address();

adrs1.setCity("Chennai");

Address adrs2 = **new** Address();

adrs2.setCity("Saint Petersburg");

Employee emp1 = **new** Employee();

emp1.setName("Ravi Chopda");

emp1.setAdrs(adrs1);

Employee emp2 = **new** Employee();

emp2.setName("Satiago Millet");

emp2.setAdrs(adrs2);

List<Employee> empList = List.*of*(emp1, emp2);

Project p1 = **new** Project();

p1.setName("Horamavu");

p1.setEmpList(empList);

projService.save(p1);

}

**private** **void** saveProject2() {

Address adrs1 = **new** Address();

adrs1.setCity("Bangalore");

Address adrs2 = **new** Address();

adrs2.setCity("Moscow");

Employee emp1 = **new** Employee();

emp1.setName("John Abraham");

emp1.setAdrs(adrs1);

Employee emp2 = **new** Employee();

emp2.setName("Shyam Sunder");

emp2.setAdrs(adrs2);

List<Employee> empList = List.*of*(emp1, emp2);

Project p1 = **new** Project();

p1.setName("Dumalica");

p1.setEmpList(empList);

projService.save(p1);

}

**private** **void** getProjectDetails(Long id) {

Project project = projService.getById(id);

System.***out***.println("Project: " + project);

}

@EventListener(ApplicationReadyEvent.**class**)

**public** **void** run() {

System.***out***.println("--------- Application Running -----------");

saveProject1();

saveProject2();

sleep(10);

getProjectDetails(Long.*valueOf*(2));

getProjectDetails(Long.*valueOf*(1));

System.***out***.println("Waiting for 20 seconds ...");

sleep(20);

getProjectDetails(Long.*valueOf*(1));

getProjectDetails(Long.*valueOf*(2));

}

**private** **void** sleep(**long** time) {

**try** {

TimeUnit.***SECONDS***.sleep(time);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}