Implement Hibernate 2nd level Cache with Redis, Spring Boot, and Spring Data JPA

Pavan Jadda

Feb 17 · 3 min read

This blog post explains the process of implementing Redis Cache as Hibernate 2nd level cache using the Spring Boot, Spring Data JPA. **Source code** and config files are uploaded to Github.

pavankjadda/Redis-Hibernate2ndLevelCache-SpringBoot

Implement Hibernate 2nd level Cache with Redis, Spring Boot, and Spring Data JPA ...

github.com

Setup Redis

1. There are a couple of ways to set up Redis

(i) Download **the tar file** and follow the instructions from **official Redis docs** for detailed step by step instructions

(ii) Or use a **docker-compose file** uploaded in my source code.

2. For this demo, I use the docker-compose file, as it is easy to set up and bring up new Redis instance. Make sure the docker is running on your local machine before executing the following command.

3. So clone the repository and move into that directory

```
$ git clone https://github.com/pavankjadda/SpringBoot-RedisCache
```

```
$ cd SpringBoot-RedisCache
```

```
$ docker-compose -f src/main/resources/docker/docker-
redis.yml up
```

4. The Redis instance should be up and running at **localhost:6379**

Spring Boot Application

1. The demo Spring Boot application has an Employee domain object with EmployeeController and EmployeeRepository defined using Spring Data JPA. It uses the H2 in-memory database to persist data

```
@Data
@Entity
@Cache(region = “employeeCache”, usage =
CacheConcurrencyStrategy.READ_WRITE)
public class Employee
{
   @Id
   private Long id;
   private String firstName;
   private String lastName;
   private String email;
   private String phone;
}
```

2. The JPA/Hibernate **@Cache** annotation creates **employeeCache** map in Redis cache and takes care of **CachePut** and **CacheEvict** operations. By default **findAll(), create, update, delete** method calls hit the database since they involve in data changes and **findById()** method hits Redis cache.

3. The data in data.sql file (present in src/main/resources directory) will be loaded in the H2 in-memory database by the Spring Boot

```
INSERT INTO EMPLOYEE( ID , FIRST_NAME , LAST_NAME , EMAIL ,
PHONE) VALUES(1,'John','Doe','jdoe@example.com','123-484-
4994');
```

```
INSERT INTO EMPLOYEE( ID , FIRST_NAME , LAST_NAME , EMAIL , PHONE) VALUES(2,'Jack','Ryan','jryan@example.com','123-484-4994');
```

```
INSERT INTO EMPLOYEE( ID , FIRST_NAME , LAST_NAME , EMAIL , PHONE) VALUES(3,'Tome','Cruise','tcruise@example.com','123-484-4994');
```

4. We use the open-source library **Redisson** to connect to Redis instance and use Redis as Hibernate 2nd level cache. Please see their Github repo for more information.

5. The library **redisson-hibernate-53** provides **org.redisson.hibernate.RedissonRegionFactory** class, which implements hibernate 2nd level cache

```
<dependency>
  <groupId>org.redisson</groupId>
  <artifactId>redisson-hibernate-53</artifactId>
  <version>3.12.1</version>
</dependency>
```

6. Define the config values in application.yml as shown below and create **redisson-dev.yaml** file, that holds Redis server details

```
1
2
3  ### Server information
4  server:
5    port: 8080
6
7  ### Spring Properties
```

application.yml

```
singleServerConfig:
  address: "redis://localhost:6379"
```

redisson-dev.yaml hosted with ❤ by GitHub view raw

redission.yaml

7. Run the application and go to **http://localhost:8080/api/v1/employee/find/all** to see all the employees. You should be able to see the following JSON

```
[
  {
    "id": 1,
    "firstName": "John",
    "lastName": "Doe",
    "email": "jdoe@example.com",
    "phone": "123-484-4994"
  },
  {
    "id": 2,
    "firstName": "Jack",
    "lastName": "Ryan",
    "email": "jryan@example.com",
    "phone": "123-484-4994"
  },
  {
    "id": 3,
    "firstName": "Tome",
    "lastName": "Cruise",
    "email": "tcruise@example.com",
    "phone": "123-484-4994"
  }
]
```

8. Check the hibernate logs in the IDE, it should show the following statement. All the employees are retrieved from the database

```
Hibernate: select employee0_.id as id1_0_, employee0_.email
as email2_0_, employee0_.first_name as first_na3_0_,
employee0_.last_name as last_nam4_0_, employee0_.phone as
phone5_0_ from employee employee0_
```

8. Now go to **http://localhost:8080/api/v1/employee/find/1** to see one of the employee information and you will see the following JSON

```
{
  "id": 1,
  "firstName": "John",
  "lastName": "Doe",
  "email": "jdoe@example.com",
  "phone": "123-484-4994"
}
```

and the hibernate logs should be empty, as this employee record came from the cache.

. . .

That's it, you have successfully implemented Redis as Hibernate 2nd level cache with Spring Boot and Spring Data JPA. Checkout source code on Github.

. . .

Also, check out **my other article** on implementing Hibernate 2nd level cache with Hazelcast, Spring Boot and Spring Data JPA.