Spring Boot Redis Cache

spring boot with redis cache example code from scratch, give complete code in step by step

here use redis as caching

Here's a complete **Spring Boot + Redis Cache** example from scratch using **Redis as caching**, not as a database.

✅ What You'll Learn

- Enable caching using Redis in Spring Boot
- Use `@Cacheable`, `@CachePut`, `@CacheEvict`
- Setup Redis configuration
- Simple REST controller for demo

✅ Project Setup

1️⃣ `pom.xml` (for Maven)

xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.example</groupId>
 <artifactId>redis-cache-demo</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>jar</packaging>

 <properties>
  <java.version>17</java.version>
  <spring.boot.version>3.1.0</spring.boot.version>
 </properties>

 <dependencies>
  <!-- Spring Boot Starter Web -->
```

```
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Redis for caching -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-redis</artifactId>
    </dependency>

    <!-- Spring Boot Cache -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-cache</artifactId>
    </dependency>

    <!-- Jackson for JSON -->
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
    </dependency>

    <!-- Lombok (optional, for boilerplate reduction) -->
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

2️⃣ `application.yml`

yaml

```yaml
spring:
  data:
    redis:
      host: localhost
      port: 6379
  cache:
    type: redis

logging:
  level:
    org.springframework.data.redis: DEBUG
```

3️⃣ `RedisCacheDemoApplication.java`

java

```java
package com.example.rediscachedemo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cache.annotation.EnableCaching;

@SpringBootApplication
@EnableCaching  // 🔥 Enables Spring Caching
public class RedisCacheDemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(RedisCacheDemoApplication.class, args);
    }
}
```

4️⃣ Model Class: `User.java`

java

```java
package com.example.rediscachedemo.model;

import java.io.Serializable;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class User implements Serializable {
    private int id;
    private String name;
}
```

5️⃣ Service Layer: `UserService.java`

java

```java
package com.example.rediscachedemo.service;

import com.example.rediscachedemo.model.User;
import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.CachePut;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;

import java.util.HashMap;
import java.util.Map;

@Service
public class UserService {

    private final Map<Integer, User> userDb = new HashMap<>();

    // Simulated DB fetch
    @Cacheable(value = "users", key = "#id")
    public User getUserById(int id) {
        System.out.println("Fetching from DB...");
        return userDb.get(id);
```

```
    }

    @CachePut(value = "users", key = "#user.id")
    public User updateUser(User user) {
        userDb.put(user.getId(), user);
        return user;
    }

    @CacheEvict(value = "users", key = "#id")
    public void deleteUser(int id) {
        userDb.remove(id);
    }

    public User createUser(User user) {
        userDb.put(user.getId(), user);
        return user;
    }
}
```

6️⃣ Controller Layer: `UserController.java`

java

```
package com.example.rediscachedemo.controller;

import com.example.rediscachedemo.model.User;
import com.example.rediscachedemo.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private UserService userService;

    @PostMapping
    public User createUser(@RequestBody User user) {
        return userService.createUser(user);
    }
```

```
    @GetMapping("/{id}")
    public User getUser(@PathVariable int id) {
        return userService.getUserById(id);
    }

    @PutMapping
    public User updateUser(@RequestBody User user) {
        return userService.updateUser(user);
    }

    @DeleteMapping("/{id}")
    public String deleteUser(@PathVariable int id) {
        userService.deleteUser(id);
        return "Deleted user with ID: " + id;
    }
}
```

7️⃣ Start Redis Server

If Redis isn't installed:

- **Windows**: Use Memurai or Redis for Windows
- **Linux/Mac**:

```bash
sudo apt install redis
redis-server
```

🔁 Sample API Calls

1. Create User

```http
POST /users
Content-Type: application/json

{
```

```
  "id": 1,
  "name": "Viswanath"
}
```

2. Fetch User (cached after first call)

```http
GET /users/1
```

3. Update User

```http
PUT /users
Content-Type: application/json

{
  "id": 1,
  "name": "VK"
}
```

4. Delete User (also evicts cache)

```http
DELETE /users/1
```

✅ Output Demo

```sql
POST /users    --> Saved in DB
GET /users/1   --> Fetching from DB... (First time)
GET /users/1   --> From cache (No log)
PUT /users     --> Updates cache
DELETE /users/1 --> Evicts cache
```

🧠 Notes

- `@Cacheable`: Checks cache first, stores if absent.
- `@CachePut`: Always updates both DB and cache.
- `@CacheEvict`: Removes from cache.

Let me know if you want the same in Gradle or with a Redis config class using `LettuceConnectionFactory`.