**Working with REDIS**

My Doubt :- Suppose for a class I want to have 6 fields. 4 i want to store in DataBase and 2 in Cache memory. Now for creating an entity to work with RDBMS we use @Entity annotation and to work with Redis we use @RedisHash now how we will work on this requirement because i cant have the two annotations declared for a singlee class. Should i declare 4 fields in a single class with @Entity and 2 fields in other class with @RedisHash? Then how i would be retrieving the data in a DTO object??

**ChatGPT said:**

ChatGPT

To address your requirement of storing 4 fields in the relational database (RDBMS) and 2 fields in the cache (Redis), while maintaining a clean and manageable design, here's an approach that you can follow:

**1. Split the Data into Two Classes**

Yes, you should split the data into two classes:

* One annotated with @Entity for storing data in the database.
* Another annotated with @RedisHash for storing data in Redis.

**2. Create a Common DTO for Data Retrieval**

To combine the data from both sources (RDBMS and Redis) into a single object for use in your application, you can create a **DTO (Data Transfer Object)** that aggregates data from both the database and cache.

**Implementation Steps**

**Class 1: Entity for RDBMS**

This class will hold the fields that you want to persist in the relational database.

java

CopyEdit

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Column;

@Entity

public class MyDatabaseEntity {

@Id

private Long id;

@Column

private String field1;

@Column

private String field2;

@Column

private String field3;

@Column

private String field4;

// Getters and Setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getField1() { return field1; }

public void setField1(String field1) { this.field1 = field1; }

public String getField2() { return field2; }

public void setField2(String field2) { this.field2 = field2; }

public String getField3() { return field3; }

public void setField3(String field3) { this.field3 = field3; }

public String getField4() { return field4; }

public void setField4(String field4) { this.field4 = field4; }

}

**Class 2: Entity for Redis**

This class will hold the fields that you want to cache in Redis.

java

CopyEdit

import org.springframework.data.annotation.Id;

import org.springframework.data.redis.core.RedisHash;

@RedisHash("MyRedisEntity")

public class MyRedisEntity {

@Id

private Long id;

private String field5;

private String field6;

// Getters and Setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getField5() { return field5; }

public void setField5(String field5) { this.field5 = field5; }

public String getField6() { return field6; }

public void setField6(String field6) { this.field6 = field6; }

}

**Class 3: DTO for Aggregating Data**

The DTO will be used to combine the fields from both MyDatabaseEntity and MyRedisEntity.

java

CopyEdit

public class MyDTO {

private Long id;

private String field1;

private String field2;

private String field3;

private String field4;

private String field5;

private String field6;

// Getters and Setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getField1() { return field1; }

public void setField1(String field1) { this.field1 = field1; }

public String getField2() { return field2; }

public void setField2(String field2) { this.field2 = field2; }

public String getField3() { return field3; }

public void setField3(String field3) { this.field3 = field3; }

public String getField4() { return field4; }

public void setField4(String field4) { this.field4 = field4; }

public String getField5() { return field5; }

public void setField5(String field5) { this.field5 = field5; }

public String getField6() { return field6; }

public void setField6(String field6) { this.field6 = field6; }

}

**Service Layer: Combining the Data**

You can create a service method to fetch data from both RDBMS and Redis and populate the MyDTO object.

java

CopyEdit

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class MyService {

@Autowired

private MyDatabaseRepository dbRepository;

@Autowired

private MyRedisRepository redisRepository;

public MyDTO getDataById(Long id) {

// Fetch data from RDBMS

MyDatabaseEntity dbEntity = dbRepository.findById(id).orElseThrow(() -> new RuntimeException("DB Entity not found"));

// Fetch data from Redis

MyRedisEntity redisEntity = redisRepository.findById(id).orElse(new MyRedisEntity());

// Combine data into DTO

MyDTO dto = new MyDTO();

dto.setId(dbEntity.getId());

dto.setField1(dbEntity.getField1());

dto.setField2(dbEntity.getField2());

dto.setField3(dbEntity.getField3());

dto.setField4(dbEntity.getField4());

dto.setField5(redisEntity.getField5());

dto.setField6(redisEntity.getField6());

return dto;

}

}

**Repository Interfaces**

The repository interfaces will handle the interaction with the respective storage systems.

For RDBMS:

java

CopyEdit

import org.springframework.data.jpa.repository.JpaRepository;

public interface MyDatabaseRepository extends JpaRepository<MyDatabaseEntity, Long> {

}

For Redis:

java

CopyEdit

import org.springframework.data.repository.CrudRepository;

public interface MyRedisRepository extends CrudRepository<MyRedisEntity, Long> {

}

**Advantages of This Approach**

1. **Separation of Concerns**: The data specific to the database and cache are cleanly separated into their respective classes.
2. **Flexibility**: You can independently modify the database or cache schema without affecting the other.
3. **DTO Simplifies Aggregation**: The service layer combines the data into a single object, making it easy to use in the application.

Let me know if you have further questions!