Theory: CRUD Repositories

26 minutes 0 / 5 problems solved

§1. Crud Operations

CRUD stands for basic database functions: **Create, Read, Update, and Delete**. Obviously, almost all the database integrated applications need to implement these operations for each entity. The implementations for different entities are similar to each other. So, creating CRUD methods manually means writing a lot of boilerplate code unless you let the `CrudRepository` interface carry about routine implementations for you.

In this topic, you will see how to use this interface and how to make it provide the specific methods you may need.

§2. Basic project

Here you will need the basic Spring Boot project with H2 and data-JPA dependencies. The Gradle file should look like this:

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    runtimeOnly 'com.h2database:h2'
    testImplementation('org.springframework.boot:spring-boot-starter-test') {
        exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
    }
}
```

§3. H2 setup

Here we will use the in-memory H2 database 'JpaCrudDB' with the following setup in the `application.properties` file:

```
spring.datasource.url=jdbc:h2:file:~/JpaCrudDB
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=create
```

§4. Prepare the Application

With the console Spring Boot Application, you will be able to check the crud methods quickly inside the console. We create a `@Bean` annotated method in our `@SpringBootApplication` class. The method returns `CommandLineRunner`. We won't go deeper into the `CommandLineRunner` bean as it is used just to show the results.

```
@SpringBootApplication
public class JpaRepositoriesApplication {

    public static void main(String[] args) {
        SpringApplication.run(JpaRepositoriesApplication.class, args);
    }

    @Bean
    public CommandLineRunner runApplication() {

        return (args -> {

            // call methods you want to use

        });

    }

}
```

Current topic:

CRUD Repositories

Topic depends on:

- ✕ Optional
- ✕ H2 database
- ✕ Introduction to JPA

Table of contents:

↑ CRUD Repositories

§1. Crud Operations

§2. Basic project

§3. H2 setup

§4. Prepare the Application

§5. Create an entity

§6. Crud Repository

§7. Create Methods

§8. Read Methods

§9. Update Methods

§10. Delete Methods

§11. Other Available Methods

§12. Conclusion

Feedback & Comments

§5. Create an entity

Now we will create the `Task` entity to represent some task information. The `Task` class contains the `id` field annotated with `@Id` and `@GeneratedValue`:

```
@Entity
@Table(name = "task")
public class Task {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String title;
    private String summary;

    private boolean enabled;

    private int priority;



    // constructors

    // getters and setters
```

There is also the `int priority` field. The task with the highest priority would have 0 priority and the lowest priority isn't defined.

§6. Crud Repository

Finally, it's time to create the `TaskRepository` interface by extending `CrudRepository`. You also have to define the entity type and its id field type:

```
public interface TaskRepository extends CrudRepository<Task, Long> { }
```

Remember to add the `TaskRepository` object as a parameter to a `CommandLineRunner` returning method in the `@SpringBootApplication` class:

```
@Bean
public CommandLineRunner runApplication(TaskRepository taskRepository) {
    return (args -> {
        // call methods you want to use
    });
}
```

Now you can add methods' definitions here following some methods name patterns, and the `CrudRepository` interface will take care of all the implementation. Let's dive into these name patterns.

§7. Create Methods

You don't even have to define saving methods in your interface to use them, so you can just do the following to save some tasks to a database:

```
  private void performCreateOperations(TaskRepository taskRepository) {

   Task articleTask = new Task("Finish article", "Finish the article and send for review", true, 0);
        taskRepository.save(articleTask);


   Task speechTask = new Task("Meeting speech", "Prepare the speech for the meeting", false, 2);
        taskRepository.save(speechTask);


   Task drTask = new Task("Call dr Robbins", "Cancel the visit", true, 1);
        taskRepository.save(drTask);




   Task bookingTask = new Task("Book a hotel", "Book a hotel for vacation", true, 4);

        Task savedTask = taskRepository.save(bookingTask);

  }
```

We used the `save(Task task)` method to save a single entity. It returns the saved object. If you compare the `savedTask` with the `bookingTask`, you will see its filled `Id` field in the `savedTask`, because it was generated while saving to the db.

§8. Read Methods

The read methods have a lot of different options.

One of the most popular methods for getting an entity by its `@Id` annotated field is predefined and you don't have to define in `TaskRepository`.

```
Optional<Task> taskOptional = taskRepository.findById(1L);
if (taskOptional.isPresent()) {
    System.out.println(taskOptional.get());
}
```

It returns an `Optional<Task>` value, which may contain the `Task` if the one was found.

There is one more predefined `findAll()` method. As you can guess, it returns all the tasks from the database.

Now let's explore the special options of reading. With our `Task` entity we can find/read tasks from the database by exact field value, e.g. find by title or by enabled value:

```
public interface TaskRepository extends CrudRepository<Task, Long> {

    Task findByTitle(String title);

    List<Task> findAllByEnabled(boolean isEnabled);
}
```

Note the difference in definitions:

`findByTitle` returns a single task if it is presented in a db, but if it finds more than one task with similar titles it will end up with a `NonUniqueResultException`;

`findAllByEnabled` returns all found tasks in a list

You can also make the method return an `Optional<Task>` value:

```
Optional<Task> findByTitle(String title);
```

Let's test the read methods:

```
System.out.println("Task list:");
for (Task task : taskRepository.findAll()) {
    System.out.println(task);
}

System.out.println("List of enabled tasks:");
for (Task task : taskRepository.findAllByEnabled(true)) {
    System.out.println(task);
}
```

There are some options to get lists of tasks ordered by different fields' values. For example, the method returning a list of all enabled tasks ordered by priority. You can define it in a `TaskRepository`:

```
List<Task> findAllByEnabledOrderByPriorityAsc(boolean isEnabled);
```

Finally, you can search for tasks containing any keyword in a summary field with the `findBySummaryContaining(String text)` method, like it is done here:

```
for (Task task : taskRepository.findBySummaryContaining("hotel")) {
    System.out.println(task);
}
```

§9. Update Methods

There are no special methods to update a task. You need to make a few steps to do it:

- use any of read methods to fetch a task you'd like to change,
- make your changes to the fetched task
- use the save method.

Look at the update operation example:

```
private void performUpdateOperations(TaskRepository taskRepository) {
    System.out.println("Updating tasks's title");
    Task task = taskRepository.findByTitle("Meeting speech");

    if (task != null) {
        System.out.println(task);
        task.setEnabled(true);
        taskRepository.save(task);
        System.out.println(taskRepository.findByTitle("Meeting speech"));

    }

}
```

§10. Delete Methods

You can remove all of the tasks from the database with the `deleteAll()` method. You can also delete a task by id or by title value:

```
void deleteByTitle(String title);
```

Here all the read options are also available: delete by field containing a special value, delete all with a priority higher than 3, etc.

§11. Other Available Methods

CRUD operations are not all the `CrudRepository` interface provides. For example, you can use `count()` method to get the number of all tasks or `countByEnabled(boolean enabled)` to get the number of enabled/disabled tasks:

```
long countByEnabled(boolean enabled);
```

There are also `Before`, `After` and `Between` options for numeric or date fields. Guess, for example, what tasks this method will find:

```
List<Task> findByPriorityBetween(int fromPriority, int toPriority);
```

Or with `GreaterThan/LessThan` option:

```
List<Task> findByPriorityGreaterThan(int priority);
```

With `And/Or` and `Not` options, you can combine different read or delete methods and create complex queries:

```

List<Task> findByPriorityAndSummaryNotContains(int fromPriority, String summary);
```

§12. Conclusion

In this topic, we explored how to make CrudRepository provide Create, Read, Update, and Delete operations and some options to make more complex read or delete operations. For example, find all entities from the database by provided field value or count them.

Report a typo

38 users liked this theory. **7** didn't like it. **What about you?**

Start practicing

Comments (1) Hints (1) Useful links (0) Show discussion