Pagination and Sorting With Spring Data JPA

Learn how to implement pagination and sorting with the Spring Data JPA.

 by

Amit Phaltankar

.

Feb. 20, 19 · Java Zone · Tutorial

Overview

While dealing with a large amount of data, lazy processing is often essential. Even if a service returns a huge amount of data, the consumer is less likely to use it. Consider a shopping website

where a customer searches for a product and the website has thousands of products to display. Fetching thousands of products and displaying them on a web page will be very time-consuming. In most cases, the customer may not even look at all of the products.

For such cases, a technique called Pagination is used. Only a small subset of products (page) is displayed at first and the customer can ask to see the next subset (page) and so on.

To learn the basics of JPA and Spring Data JPA, check out these links:

- Hands-on Spring Data JPA (A Spring Data JPA Learning Series)
- What is JPA, Spring Data and Spring Data JPA

Entity

For the sake of this tutorial, we will consider the simplest example of the Employee entity. Below is the `Employee` entity class.

```
1
2  public class Employee {
3      @Id private Long name;
4
5      private String firstName;
6      private String lastName;
7      private Date dateOfBirth;
8      private Integer age;
9      private String designation;
10     private double salary;
11     private Date dateOfJoining;
12
13     public Long getName() {
```

```
        return name;
    }

    public void setName(Long name) {
        this.name = name;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public Date getDateOfBirth() {
        return dateOfBirth;
    }

    public void setDateOfBirth(Date dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public String getDesignation() {
        return designation;
    }
```

```
    public void setDesignation(String designation) {
        this.designation = designation;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public Date getDateOfJoining() {
        return dateOfJoining;
    }

    public void setDateOfJoining(Date dateOfJoining) {
        this.dateOfJoining = dateOfJoining;
    }
}
```

Want to learn more about using the Java Persistence API (JPA) with Spring and Spring Boot?

Check out these additional links:

- Spring Boot with Spring Data JPA
- Spring Data JPA Composite Key with @EmbeddedId
- Spring Data JPA find by @EmbeddedId Partially
- Java Persistence API Guide
- Spring Data JPA Query Methods

Employee Repository

In the article Spring Data JPA Query Methods, we have already learned about Spring repository interfaces and query methods. Here, we need to learn Pagination, so we will use Spring's `PagingAndSortingRepository`.

```
1
2  public interface EmployeeRepository extends
   PagingAndSortingRepository<Employee, Long> {
3
4      Page<Employee> findAll(Pageable pageable);
5
6      Page<Employee> findByFirstName(String firstName, Pageable pageable);
7
8      Slice<Employee> findByFirstNameAndLastName(String firstName, String
   lastName, Pageable pageable);
9  }
```

Pagination

Have a look at the `EmployeeRepository`. The method accepts `Pageable` arguments. `Pageable` is an interface defined by Spring, which holds a`PageRequest`. Let's see how to create a `PageRequest`.

```
1
2  Page<Employee> page = employeeRepository.findAll(pageable);
```

In the first line, we created a `PageRequest`of 10 employees and asked for the first (0) page. The page request was passed to `findAll` to get a page of Employees as a response.

If we want to access the next set of subsequent pages, we can increase the page number every time.

```
PageRequest.of(2, 10);
PageRequest.of(3, 10);
...
```

Sorting

Spring Data JPA provides a `Sort` object in order to provide a sorting mechanism. Let's have a look at the ways of sorting.

```
employeeRepository.findAll(Sort.by("fistName").ascending().and(Sort.by("lastName").descending());
```

Obviously, the first one simply sorts by ‘firstName’ and the other one sorts by ‘firstName’ ascending and ‘lastName’ descending.

Pagination and Sort Together

```
Pageable pageable = PageRequest.of(0, 20,
Sort.by("fistName").ascending().and(Sort.by("lastName").descending());
```

Slice Vs. Page

In the`EmployeeRepository`, we saw one of the methods returns `Slice` and the other return `Page`. Both of them are **Spring Data JPA**, where `Page` is a sub-interface of `Slice`. Both of them are

used to hold and return a subset of data. Let’s have a look at them one by one

Slice

The `Slice` knows if it has content and if it is the first or last slice. It is also capable of returning the `Pageable`used in the current and previous slices. Let's have a look at some important methods of `Slice`.

```
Pageable getPageable(); // get current pageable

boolean hasContent();

boolean isFirst();

boolean isLast();

Pageable nextPageable(); // pageable of the next slice

Pageable previousPageable(); // pageable of the previous slice
```

Page

`Page` is a sub-interface of `Slice` and has a couple of additional methods. It knows the number of total pages in the table as well as the total number of records. Below are some important methods from `Page`.

```
long getTotalElements(); // number of total elements in the table

int totalPages() // number of total pages in the table
```

Summary

In this pagination and sorting example with Spring Data JPA, we learned why pagination is required. We also learned how to get paginated as well as sorted subsets of data. Lastly, we have also seen the `Slice` and `Page` interfaces and their differences.

Read the original post Pagination and Sorting with Spring Data JPA here.