(/)

Pagination in Spring Webflux and Spring Data Reactive

Last updated: September 28, 2023

Written by: Balamurugan Radhakrishnan (https://www.baeldung.com/author/balamuruganradhakrishnan)

Spring Data (https://www.baeldung.com/category/persistence/spring-persistence/spring-data)

Spring 5 (https://www.baeldung.com/tag/spring-5)

1. Introduction

In this article, we'll explore the significance of pagination for retrieving information, compare Spring Data Reactive pagination with Spring Data, and demonstrate how to implement pagination using an example.

2. Significance of Pagination

Pagination is an essential concept when dealing with endpoints that return large collections of resources. It allows for efficient retrieval and presentation of data by breaking it down into smaller, manageable chunks called “pages”.

Consider a UI page that shows product details, which could display anywhere from 10 to 10,000 records. Suppose the UI is designed to fetch and display the entire catalog from the backend. In that case, it’ll consume additional backend resources and result in a significant waiting time for the user.

Implementing a pagination system can significantly enhance the user experience. Rather than fetching the entire set of records at once, it would be more effective to retrieve a few records initially and provide an option to load the next set of records upon request.

Using pagination, the backend can return an initial response with a smaller subset, such as 10 records, and retrieve subsequent pages using an offset or a next-page link. This approach distributes the load of fetching and displaying records across multiple pages, improving the overall application experience.

3. Pagination in Spring Data vs. Spring Data Reactive

Spring Data (/spring-data) is a project within the larger Spring Framework ecosystem that aims to simplify and enhance data access in Java applications. Spring Data offers a set of common abstractions and functionalities that streamline the development process by reducing boilerplate code and promoting best practices.

As explained in the Spring Data Pagination example (/spring-data-jpa-pagination-sorting), the *PageRequest* (https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/PageRequest.html)object, which accepts *page*, *size*, and *sort* parameters, can be used to configure and request different pages. Spring Data offers *PagingAndSortingRepository, (https://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html)* which provides methods to retrieve entities using pagination and sorting abstraction. The repository methods accept

Pageable (https://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/domain/Pageable.html) and Sort (https://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/domain/Sort.html) objects, which can be used to configure the return *Page (https://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/domain/Page.html)* information. This *Page* object contains the *totalElements (https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Page.html#getTotalElements())* and *totalPages (https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Page.html#getTotalPages())* attributes which are populated by executing additional queries internally. This information can be used to request the subsequent pages of information.

Contrarily, Spring Data Reactive doesn't fully support pagination. The reason lies in Spring Reactive's support for asynchronous non-blocking. It must wait (or block) until it returns all data for a specific page size, which isn't very efficient. However, **Spring Data Reactive still supports *Pageable (https://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/domain/Pageable.html)*. We can configure it with a *PageRequest* (https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/PageRequest.html) object to retrieve specific chunks of data and add an explicit query to fetch the total count of records.**

We can get a *Flux* (https://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html) of responses as opposed to *Page* when using Spring Data which contains metadata about the records on the page.

4. Basic Application

4.1. Implementation of Pagination in Spring WebFlux and Spring Data Reactive

For this article, we'll use a simple Spring R2DBC application that exposes product information with pagination through GET /products.

Let's consider a simple Product Model:

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Table
public class Product {

    @Id
    @Getter
    private UUID id;

    @NotNull
    @Size(max = 255, message = "The property 'name' must be less
than or equal to 255 characters.")
    private String name;

    @NotNull
    private double price;
}
```

We can fetch a list of products from the Product Repository by passing a *Pageable (https://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/domain/Pageable.html)* object, which contains configurations like *Page* (https://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/domain/AbstractPageRequest.html#getPageNumber()) and Size: (https://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/domain/AbstractPageRequest.html#getPageSize())

```
@Repository
public interface ProductRepository extends
ReactiveSortingRepository<Product, UUID> {
    Flux<Product> findAllBy(Pageable pageable);
}
```

This query responds result set as a *Flux* as opposed to *Page* hence the total number of records needs to be queried separately to populate *Page* response.

Let's add a controller with a *PageRequest* (https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/PageRequest.html) object which also runs an additional query to fetch the total count of records. This is because our repository doesn't respond back with

Page (https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Page.html) information, and instead, it returns *Flux<Product>*:

```
@GetMapping("/products")
public Mono<Page<Product>> findAllProducts(Pageable pageable) {
    return this.productRepository.findAllBy(pageable)
      .collectList()
      .zipWith(this.productRepository.count())
      .map(p -> new PageImpl<>(p.getT1(), pageable, p.getT2()));
}
```

Finally, we must send both the query result sets and the originally received *Pageable* object to the *PageImpl* (https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/PageImpl.html). This class has helper methods that calculate the *Page* information, which includes metadata about the page to fetch the next set of records.

Now, when we try to access the endpoint, we should receive a list of products with page metadata:

```
{
  "content": [
    {
      "id": "cdc0c4e6-d4f6-406d-980c-b8c1f5d6d106",
      "name": "product_A",
      "price": 1
    },
    {
      "id": "699bc017-33e8-4feb-aee0-813b044db9fa",
      "name": "product_B",
      "price": 2
    },
    {
      "id": "8b8530dc-892b-475d-bcc0-ec46ba8767bc",
      "name": "product_C",
      "price": 3
    },
    {
      "id": "7a74499f-dafc-43fa-81e0-f4988af28c3e",
      "name": "product_D",
      "price": 4
    }
  ],
  "pageable": {
    "sort": {
      "sorted": false,
      "unsorted": true,
      "empty": true
    },
    "pageNumber": 0,
    "pageSize": 20,
    "offset": 0,
    "paged": true,
    "unpaged": false
  },
  "last": true,
  "totalElements": 4,
  "totalPages": 1,
  "first": true,
  "numberOfElements": 4,
  "size": 20,
  "number": 0,
  "sort": {
    "sorted": false,
    "unsorted": true,
    "empty": true
  },
  "empty": false
}
```

Like Spring Data, we navigate to different pages using certain query parameters (https://docs.spring.io/spring-data/r2dbc/docs/current/reference/html/#core.web.basic.paging-and-sorting), and by extending *WebMvcConfigurationSupport (https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/config/annotation/WebMvcConfigurationSupport.html)*, we configure the default attributes.

Let's change the default page size from 20 to 100 and also set the default page as 0 by overriding the *addArgumentResolvers* method:

```
@Configuration
public class CustomWebMvcConfigurationSupport extends
WebMvcConfigurationSupport {

    @Bean
    public PageRequest defaultPageRequest() {
        return PageRequest.of(0, 100);
    }

    @Override
    protected void
addArgumentResolvers(List<HandlerMethodArgumentResolver>
argumentResolvers) {
        SortHandlerMethodArgumentResolver argumentResolver = new
SortHandlerMethodArgumentResolver();
        argumentResolver.setSortParameter("sort");
        PageableHandlerMethodArgumentResolver resolver = new
PageableHandlerMethodArgumentResolver(argumentResolver);
        resolver.setFallbackPageable(defaultPageRequest());
        resolver.setPageParameterName("page");
        resolver.setSizeParameterName("size");
        argumentResolvers.add(resolver);
    }
}
```

Now, we can make the request starting from Page 0 with a size of max 100 records:

```
$ curl --location 'http://localhost:8080/products?
page=0&size=50&sort=price,DESC'
```

Without specifying page and size parameters, the default Page index is 0 with 100 records per page. But the request sets the page size to 50:

```
{                                   (/)
  "content": [
    ....
  ],
  "pageable": {
    "sort": {
      "sorted": false,
      "unsorted": true,
      "empty": true
    },
    "pageNumber": 0,
    "pageSize": 50,
    "offset": 0,
    "paged": true,
    "unpaged": false
  },
  "last": true,
  "totalElements": 4,
  "totalPages": 1,
  "first": true,
  "numberOfElements": 4,
  "size": 50,
  "number": 0,
  "sort": {
    "sorted": false,
    "unsorted": true,
    "empty": true
  },
  "empty": false
}
```

5. Conclusion

In this article, we understood the unique nature of Spring Data Reactive pagination. We also implemented an endpoint that returns a product list with pagination.

As always, the source code for the examples is available over on GitHub (https://github.com/eugenp/tutorials/tree/master/spring-reactive-modules/spring-reactive-data-2).

(/)

Comments are closed on this article!

ALL BULK COURSES (/ALL-BULK-COURSES)

ALL BULK TEAM COURSES (/ALL-BULK-TEAM-COURSES)

THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA “BACK TO BASICS” TUTORIAL (/JAVA-TUTORIAL)

JACKSON JSON TUTORIAL (/JACKSON)

APACHE HTTPCLIENT TUTORIAL (/HTTPCLIENT-GUIDE)

REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)

SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)

SECURITY WITH SPRING (/SECURITY-SPRING)

SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)

ABOUT

ABOUT BAELDUNG (/ABOUT)

THE FULL ARCHIVE (/FULL_ARCHIVE)

EDITORS (/EDITORS)

JOBS (/TAG/ACTIVE-JOB/)

OUR PARTNERS (/PARTNERS)

PARTNER WITH BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)

PRIVACY POLICY (/PRIVACY-POLICY)

COMPANY INFO (/BAELDUNG-COMPANY-INFO)

CONTACT (/CONTACT)