LEARNING spring-data

#spring-data

Table of Contents

About

Chapter 1: Getting started with spring-data

Remarks

Modern software applications have the option of storing data in more than one type of data store. Whereas traditional data stores like Relational databases remain popular, NoSQL databases and Cloud-based storage have also become commonplace. Each of these types of data stores has its own strengths and is therefore suited for different types of business use cases. Complex business applications therefore end up utilizing more than one type of data store to make data storage, retrieval and presentation operations more efficient. This presents the challenge of application programmers having to deal with the complexity of understanding the API provided by multiple data stores and using these API appropriately in their business applications.

Spring Data is a project that aims at providing a consistent, easy-to-use API to application programmers, independent of the underlying data store used. It combines the power of the Spring framework with concepts from proven data access paradigms such as domain-driven design to provide a familiar and consistent foundation to application programmers for accessing different types of data stores, while still retaining the specifics of an underlying data store, where appropriate.

The Spring Data project consists of several subprojects that can be used as libraries for accessing specific types of data stores. The full set of data stores supported by Spring Data and its subprojects can be obtained from the main page of the project.

Examples

Installation or Setup

Spring Data is a project consisting of a number of subprojects. The most common ones are Spring Data JPA, Spring Data MongoDB, Spring Data Elasticsearch, Spring Data Neo4J, Spring Data Cassandra and Spring Data Redis.

Unless you are developing your own subproject based upon Spring Data, it is highly unlikely that you will need to use it directly in your application. See the individual subprojects for details on their installation and setup. If however, you do have the need to use Spring Data in your application directly, the following instructions will be helpful.

Using Maven

```
<dependencies>
  <dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-commons</artifactId>
    <version>[version-number]</version>
  </dependency>
</dependencies>
```

Using Gradle

```
dependencies {
  compile 'org.springframework.data:spring-data-commons:[version-number]'
}
```

Substitute *[version number]* with the Spring Data version you wish to use.

Read Getting started with spring-data online: https://riptutorial.com/spring-data/topic/5440/getting-started-with-spring-data

Chapter 2: Pagination with Spring Data

Introduction

Pagination by passing parmeter with custom query in spring data JPA

Examples

Pagination by passing parmeter with custom query in spring data JPA

I use Spring Boot 1.4.4.RELEASE , with MySQL as the Database and Spring Data JPA abstraction to work with MySQL. Indeed ,it is the Spring Data JPA module that makes it so easy to set up Pagination in a Spring boot app in the first place.

Scenario expose an endpoint /students/classroom/{id} . It will return a List of Students and other paging info(which we would see in a minute) based on the page and size parameters and classroomId that were passed along with it.

To begin with, i create a domain Student

```
@Entity
@Table(name = "student")
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name")
    private String name;

    @NotNull
    @Column(name = "rollnumber", nullable = false)
    private Integer rollnumber;

    @Column(name = "date_of_birth")
    private LocalDate dateOfBirth;

    @Column(name = "address")
    private String address;

    @ManyToOne(optional = false)
    @NotNull
    private Classroom classroom;

//getter and setter

}
```

Student relate with classroom

```
@Entity
@Table(name = "classroom")
public class Classroom  {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "standard")
    private String standard;

    @Column(name = "section")
    private String section;

    @Column(name = "year")
    private String year;

//getter && setter

}
```

we have RestController

```
@RestController
@RequestMapping("/api")
public class StudentResource {

private final StudentService studentService;

    public StudentResource(StudentService studentService) {
        this.studentService = studentService;
    }

 @GetMapping("/students/classroom/{id}")
    public ResponseEntity<Page<StudentDTO>> getAllStudentsBasedOnClassroom(@ApiParam Pageable
pageable,@PathVariable Long id)
        throws URISyntaxException {
        Page<StudentDTO> page = studentService.findByClassroomId(id, pageable);
        HttpHeaders headers = PaginationUtil.generatePaginationHttpHeaders(page,
"/api/students/classroom");
        return new ResponseEntity<Page<StudentDTO>>(page, headers, HttpStatus.OK);
    }

}
```

Notice that we haven’t passed RequestParams to our handler method . When the endpoint /students/classroom/1?page=0&size=3 is hit, Spring would automatically resolve the page and size parameters and create a Pageable instance . We would then pass this Pageable instance to the Service layer ,which would pass it to our Repository layer .

Service class

```
public interface StudentService {

  Page<StudentDTO> findByClassroomId(Long id,Pageable pageable);

}
```

service impl (here i user StudentMapper to convert Class to DTOby using mapStruct or we can do manualy)

```
@Service
@Transactional
public class StudentServiceImpl implements StudentService{

 private final StudentRepository studentRepository;

    private final StudentMapper studentMapper;

    public StudentServiceImpl(StudentRepository studentRepository, StudentMapper
studentMapper) {
        this.studentRepository = studentRepository;
        this.studentMapper = studentMapper;
    }
@Override
    public Page<StudentDTO> findByClassroomId(Long id, Pageable pageable) {
         log.debug("Request to get Students based on classroom : {}", id);
         Page<Student> result = studentRepository.findByClassroomId(id, pageable);
         return result.map(student -> studentMapper.studentToStudentDTO(student));
    }

}
```

this is mapper interface

```
@Mapper(componentModel = "spring", uses = {})
public interface StudentMapper{

  StudentDTO   studentToStudentDTO(Student student);
}
```

then in StudentRepository i worte custom method

```
public interface StudentRepository extends JpaRepository<Student,Long> {

    Page<Student> findByClassroomId(Long id, Pageable pageable);

}
```

then it will give us all below information with respective data

```
"last": false,
  "totalElements": 20,
  "totalPages": 7,
  "size": 3,
  "number": 0,
  "sort": null,
  "first": true,
  "numberOfElements": 3
```

Read Pagination with Spring Data online: https://riptutorial.com/spring-data/topic/9142/pagination-with-spring-data

Credits

S. No	Chapters	Contributors
1	Getting started with spring-data	Community, manish, sunkuet02
2	Pagination with Spring Data	Aman Tuladhar, VISHWANATH N P