Training on

Micro services Spring boot and Docker

MOHAMMED LUQMAN SHAREEF

Spring Boot configuration

Default Properties file

.properties / .yaml

Reading Custom Property Value in Java code

Application.yaml

YAML – Yet Another Markup Language

- A human readable data serialization language
- Commonly used for configuration files (ex Docker, Kubernetes etc use for config)
- It is case sencistive
- It doesn’t allow tabs. Instead spaces are used.
- Structure is shown through indentation (one or more spaces). Sequence items are denoted by a dash, and key value pairs within a map are separated by a colon.

Lombok

Lombok is a Java library that helps developers reduce boilerplate code in their applications. It provides a set of annotations that automate the generation of commonly used methods like getters, setters, constructors, toString, equals, and hashCode. By using Lombok, developers can focus more on business logic rather than repetitive code.

Key Features of Lombok

- Annotations: Lombok provides a variety of annotations that automatically generate code at compile-time, which helps keep the code clean and maintainable.
- Compile-time Processing: Lombok modifies the abstract syntax tree of Java code during compilation, allowing for seamless integration without requiring manual code changes.
- IDE Support: Popular IDEs like IntelliJ IDEA and Eclipse support Lombok with plugins, making it easy to work with Lombok-annotated classes.

Data Validation

Validation Constraints

- @NotNull
- @NotBlank
- @Size(min, max)
- @Pattern(regex)
- @Email
- @Min
- @Max
- @Positive
- @PositiveOrZero
- @Negative
- @NegativeOrZero

Data Validation

@Valid ensures that the request body (or method parameters) are validated against the constraints defined in the entity class.

BindingResult captures validation errors, which can be checked and handled manually within the controller method.

Global Exception Handling: You can define global error handling using **@ControllerAdvice** and customize the error response for failed validations.

If the validation passes, the controller can proceed with saving the data or performing other operations, ensuring data integrity.

Global Exception Handler

A Global Exception Handler in Spring Boot is a centralized mechanism for handling exceptions across your entire application. Instead of writing try-catch blocks in every controller method, you can define a global handler that intercepts exceptions thrown by any part of your application. This leads to cleaner code, improved maintainability, and a consistent way of handling errors.

@ControllerAdvice: This annotation is used to define a global exception handler. It allows you to handle exceptions across all controllers.

@ExceptionHandler: This annotation is used within a class annotated with @ControllerAdvice to specify the type of exception to handle. You can create multiple methods within your global exception handler to handle different types of exceptions.

Java Persistence API (JPA)

JPA is a specification that provides a standard way for Java applications to interact with relational databases by mapping Java objects (entities) to database tables.

JPA defines the management of relational data in applications using Java, but it is not a framework itself. Instead, it is implemented by various ORM (Object-Relational Mapping) frameworks like Hibernate, EclipseLink, etc.

JPA helps bridge the gap between the object-oriented world of Java and the relational database systems by allowing developers to work with Java objects instead of writing SQL queries directly.

Key Concepts in JPA:

- **Entity**: A lightweight Java class mapped to a database table.
- **Entity Manager**: A component used to manage the lifecycle of entities (CRUD operations, transactions, etc.).
- **Persistence Context**: It keeps track of entity states (new, managed, detached, removed) and manages them during transactions.
- **Annotations**: JPA uses annotations like @Entity, @Table, @Id, @OneToMany, etc., to map Java classes and their fields to database tables and columns.

JPA vs.Hibernate

JPA:

- JPA is just a specification (a set of interfaces and annotations).
- It does not provide actual implementations of the ORM features; rather, it defines how ORM should be handled.
- JPA can be implemented by various frameworks like Hibernate, EclipseLink, or OpenJPA.

Hibernate:

- Hibernate is one of the most popular ORM frameworks that implements JPA. It provides a more advanced set of features for interacting with the database, in addition to what is defined by JPA.
- Hibernate offers features such as caching, lazy loading, batch fetching, and native SQL support.
- It uses JPA annotations but also introduces its own annotations for more advanced features.

Key Difference: Hibernate is a concrete implementation of the JPA specification, while JPA itself is just a set of guidelines.

Spring Data JPA

Spring Data JPA is a Spring module that abstracts and simplifies the use of JPA and ORM frameworks (like Hibernate). It sits on top of JPA and makes it easier to interact with the database by eliminating boilerplate code.

It introduces the concept of repositories (interfaces) and query methods, which auto-generate database queries without explicitly writing them.

Spring Data JPA integrates seamlessly with Spring Boot, allowing rapid development with minimal configuration.

Key Difference: Spring Data JPA provides a higher level of abstraction over JPA, making database operations easier with pre-built repository patterns and query generation.

How Spring Data JPA Simplifies Interactions:

No Boilerplate Code:

- With Spring Data JPA, you only need to define repository interfaces that extend JpaRepository. The CRUD methods like save(), findById(), findAll(), and delete() are provided out of the box.

Auto Query Generation:

- Spring Data JPA automatically generates SQL queries based on method names in the repository. For example, a method named findByEmail() in a repository interface will generate a SQL query to fetch users based on their email address.

Paging and Sorting:

- Spring Data JPA offers built-in support for pagination and sorting via the Pageable interface.

Transaction Management:

- Spring Data JPA, in combination with Spring Framework, handles transaction management automatically. You can declare methods as transactional using the @Transactional annotation, and Spring takes care of the rest.

Integration with Spring Boot:

- Spring Data JPA integrates seamlessly with Spring Boot, which auto-configures the necessary beans and components. You can quickly start interacting with the database by adding minimal configurations.

Spring Data

The central interface in the Spring Data repository abstraction is Repository.

It takes these as type arguments

- the domain class
- the ID type of the domain class

The *CrudRepository* provides sophisticated CRUD functionality for the entity class that is being managed.

It also provide persistence technology-specific abstractions, such as *JpaRepository* or *MongoRepository*. Those interfaces extend *CrudRepository* and expose the capabilities of the underlying persistence technology.

Configuration MySQL/Postgres

<application.properties>

#spring.jpa.hibernate.ddl-auto=create / update

spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults=false

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

#spring.datasource.driver-class-name=org.postgresql.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/mysql

#spring.datasource.url=jdbc:postgresql://localhost:5432/mydb

spring.datasource.username=root

spring.datasource.password=mysqlroot

Spring Data - JPA

Entities

AutoGenerate IDs

Auto generate DDL

Validations

JPA Repository

- Built in methods
- Custom Queries

Query Builder

The query builder mechanism built into Spring Data repository infrastructure is useful for building constraining queries over entities of the repository.

The mechanism strips the prefixes *find…By, read…By, query…By, count…By,* and *get…By* from the method and starts parsing the rest of it.

The introducing clause can contain further expressions, such as a *Distinct* to set a distinct flag on the query to be created. The first *By* acts as delimiter to indicate the start of the actual criteria. You can define conditions on entity properties and concatenate them with *And* and *Or*.

```
interface PersonRepository extends Repository<User, Long> {

  List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```

Streaming query results

The results of query methods can be processed incrementally by using a Java 8 Stream<T> as return type.

```
Stream<User> readAllByFirstnameNotNull();
```

Limiting Query Results

The results of query methods can be limited by using the *first* or *top* keywords, which can be used interchangeably. An optional numeric value can be appended to *top* or *first* to specify the maximum result size to be returned.

```
User findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

Slice<User> findTop3ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);
```

Property Expressions

Property expressions can refer only to a direct property of the managed entity.

```
List<Person> findByAddressZipCode(ZipCode zipCode);
```

```
List<Person> findByAddress_ZipCode(ZipCode zipCode);
```

Special parameter handling

The infrastructure recognizes certain specific types like Pageable and Sort, to apply pagination and sorting to your queries dynamically.

```
Page<User> findByLastname(String lastname, Pageable pageable);

Slice<User> findByLastname(String lastname, Pageable pageable);

List<User> findByLastname(String lastname, Sort sort);

List<User> findByLastname(String lastname, Pageable pageable);
```

```java
public interface CrudRepository<T, ID extends Serializable>
  extends Repository<T, ID> {

  <S extends T> S save(S entity);

  Optional<T> findById(ID primaryKey);

  Iterable<T> findAll();

  long count();

  void delete(T entity);

  boolean existsById(ID primaryKey);

  // … more functionality omitted.
}
```

Pagination and Sorting

On top of the *CrudRepository*, there is a *PagingAndSortingRepository* abstraction that adds additional methods to ease paginated access to entities:

```
public interface PagingAndSortingRepository<T, ID extends Serializable>
  extends CrudRepository<T, ID> {

  Iterable<T> findAll(Sort sort);

  Page<T> findAll(Pageable pageable);
}
```

Service Integrations

Service Integrations

- Integration Patterns
- Sync Vs. Async

Sync Communication

- RestTemplate

Async Communications

- AMQP (RabbitMQ)
- Kafka