knóldus

knols . commitment . results

Presented By: Shashikant Tanti &
Krishna Jaiswal

Lack of etiquette and manners is a huge turn off.

KnolX Etiquettes

Punctuality

Join the session 5 minutes prior to the session start time. We start on time and conclude on time!

Feedback

Make sure to submit a constructive feedback for all sessions as it is very helpful for the presenter.

Silent Mode

Keep your mobile devices in silent mode, feel free to move out of session in case you need to attend an urgent call.

Avoid Disturbance

Avoid unwanted chit chat during the session.

Agenda

01 What Spring Data JPA Is?

02 Spring Data Repositories Interfaces

03 What Components Do We Need?

04 Getting the Required Dependencies

05 Query Methods

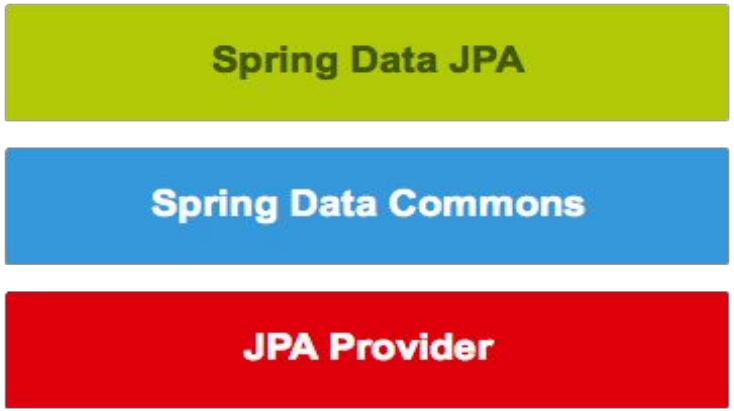
06 Reasons to use Spring Data JPA

07 Demo

1.What Spring Data JPA Is?

Spring Data JPA is used to reduce the amount of boilerplate code required to implement the data access object (DAO) layer.

Spring Data JPA is not a JPA provider. It is a library / framework that adds an extra layer of abstraction on the top of our JPA provider. If we decide to use Spring Data JPA, the repository layer of our application contains three layers that are described in the following figure :-



- **Spring Data JPA** :- It provides support for creating JPA repositories by extending the Spring Data repository interfaces.

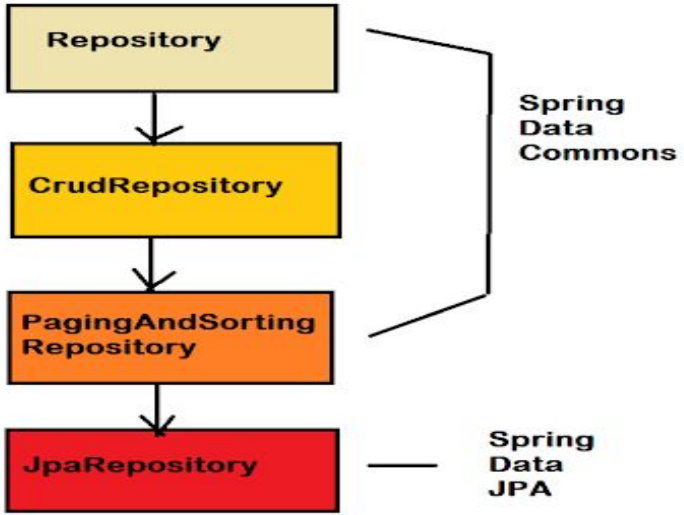
- **Spring Data Commons** :- It provides the infrastructure that is shared by the datastore specific Spring Data projects.

- **JPA Provider** :- The JPA Provider implements the Java Persistence API.

2. Spring Data Repositories Interfaces

The power of Spring Data JPA lies in the repository abstraction that is provided by the Spring Data Commons project and extended by the datastore specific sub projects.

We can use Spring Data JPA without paying any attention to the actual implementation of the repository abstraction, but we have to be familiar with the Spring Data repository interfaces. These interfaces are described in the following:



3.What Components Do We Need?

If we want to implement a persistence layer that uses Spring Data JPA, we need the following components:

- The **JDBC driver** provides a database specific implementation of the JDBC API. We use the MySQL database because it makes our example application easier to run.
- The **JPA Provider** implements the Java Persistence API. We use Hibernate because it is the most common JPA provider.
- **Spring Data JPA** hides the used JPA provider behind its repository abstraction.

4.Getting the Required Dependencies

We can get the required dependencies with Maven by using one of these options:

- We can manage our dependencies by using the Spring IO Platform.
- We can manage our dependencies "manually".
- Configure the required dependencies in the pom.xml file.

Dependencies

knóldus
knols . commitment . results

```
<!-- Database (MySQL) -->
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```

```
<!-- Spring Data JPA -->
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-jpa</artifactId>
</dependency>
```

5.Query Methods

Query methods are methods that find information from the database and are declared on the repository interface. For example, if we want to create a database query that finds the *Todo* object that has a specific id, we can create the query method by adding the findById() method to the TodoRepository interface. After we have done this, our repository interface looks as follows:

```
import org.springframework.data.repository.Repository;

interface TodoRepository extends Repository<Todo, Long> {

  //  This is a query method.

   Todo findById(Long id);

}
```

Creating the Properties File

Often we want to use a slightly different configuration in different environments. A good way to do this is move the configuration to a properties file and use a different properties file in different environments.

The *application.properties* file contains the configuration that is used to configure our example application. We can create this properties file by following these steps:

- Configure the database connection of our application. We need to configure the name of the JDBC driver class, the JDBC url, the username of the database user, and the password of the database user.
- Configure Hibernate by following these steps:
  - Configure the used database dialect.
  - Ensure that Hibernate creates the database when our application is started and drops it when our application is closed.
  - Configure the naming strategy that is used when Hibernate creates new database objects and schema elements.
  - Configure the Hibernate to NOT write the invoked SQL statements to the console.
  - Ensure that if Hibernate writes the SQL statements to the console, it will use prettyprint.

The *application.properties* file looks as follows:

knóldus
knols . commitment . results

```
#Database Configuration
```

1. spring.datasource.driver-class-name = com.mysql.jdbc.Driver
2. spring.datasource.username = ${dbName}
3. spring.datasource.password = ${dbPassword}
4. spring.jpa.show-sql = true
5. spring.datasource.url = jdbc:mysql://localhost:3306/user

```
#Hibernate Configuration
```

6. spring.jpa.hibernate.ddl-auto = create
7. spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

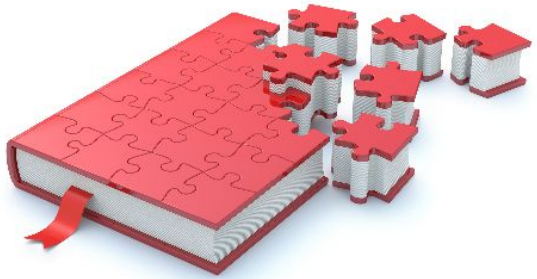
6.Reasons to use Spring Data JPA

knóldus
knols . commitment . results

- No-code Repositories
- Reduced boilerplate code
- Generated queries

References

- https://blog.knoldus.com/working-with-spring-data-jpa/
- https://www.youtube.com/watch?v=XdMCg6KssXQ



Demo

knóldus
knols . commitment . results

Thank You !

Get in touch with us:

Lorem Studio, Lord Building
D4456, LA, USA