Spring Data JPA using Spring Boot Application

Posted by Dinesh Rajput

Hello friends !!! In this tutorial we are going to discuss how to use and configure **Spring Data JPA** with **Spring Boot Application** with minimal configuration as always with Spring Boot. Spring Data JPA and Hibernate (as JPA implementation) will be used to implement the data access layer.

Related Tutorials

1. Introduction to Spring Boot
2. Spring Boot Initializr Web Interface
3. External Configuration for Spring Boot Application
4. Spring MVC with Spring Boot
5. SQL Database with Spring Boot
6. MySQL Configuration with Spring Boot

Spring Data?
Spring Data is a high level project developed by Spring community aimed at simplifying the data access operations for the applications. If you are using Spring Data in your project, you are not going to write most of the low level data access operations like writing SQL query DAO classes etc. Spring Data will generate everything dynamically at run-time by creating the proxy instances of your abstract repositories and perform the required operations.

There are several sub projects provided by Spring Data like Spring Data JPA, Spring Data Solr, Spring Data MongoDB, Spring Data REST etc. Here we are going to discuss one of them Spring Data JPA.

Spring Data JPA?
The Spring Data JPA is the implementation of JPA for simplifying operation like data accessing, querying, pagination and removes lots of boilerplate codes. The Java Persistence API is a standard technology that allows you to 'map' objects to relational databases. The spring-boot-starter-data-jpa POM provides a quick way to get started. It provides the following key dependencies:

- **Hibernate** — One of the most popular JPA implementations.
- **Spring Data JPA** — Makes it easy to implement JPA-based repositories.
- **Spring ORMs** — Core ORM support from the Spring Framework

Like P
1 friend li

LABELS

- About My
- AJAX (4)
- Ant (11)
- cloud com
- collection
- Core JAV
- Core Java

Spring Data Repositories

Spring data provides the abstract repositories, which are implemented at run-time by the spring container and perform the CRUD operations. As a developer we have to just provide the abstract methods in the interface. This reduces the amount of boilerplate code required to write data access layers. There are following are base interfaces providing by Spring Data.

Repository

It is the central interface in the spring data repository abstraction. This is a marker interface.

CrudRepository

CrudRepository provides methods for the CRUD operations. This interface extends the Repository interface. If you are extending the CrudRepository, there is no need for implementing your own methods. Just extend this interface and leave it as blank.

JpaRepository

This is the special version of CrudRepository specific to the JPA technology. Recommended to use CrudRepository because it will not tie your application with any specific store implementations.

PagingAndSortingRepository

It is specialized version for the paging operations and extension of CrudRepository.

Spring Data JPA and Spring Boot Application

Here I am going make at REST API application for providing the booking information of train ticket and also we can the book the train ticket using API. Here I am using MySQL database in a Spring Boot web application, using less code and configurations as possible. First we have to create the project so I am using here Spring Boot Web Initializr for creating project with selecting maven build and required dependencies for this project like WEB, SPRING DATA JPA, and MYSQL. Now let’s see required part of the maven file for this project.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.dineshonjava.sdjpa</groupId>
 <artifactId>SpringBootJPASpringData</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>jar</packaging>

 <name>SpringBootJPASpringData</name>
 <description>SpringBootJPASpringData project for Spring Boot with Spring Data JPA implementation</description>

 <parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.4.0.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
 </parent>

 <properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
```

Follow

- Garbage (
- GitHub (3)
- Gradle (3)
- Hadoop (1
- Hibernate
- hibernate4
- interview q
- Java (5)
- Java Code
- JavaMail (
- JAXB (4)
- JDBC (1)
- JSP (29)
- JSTL (44)
- Linux (2)
- Logging (2
- maven (12
- Microservi
- migration
- mongodb
- motivation
- multithrea
- OCPJP (4
- REST (24
- Security (
- Servlet (2
- SOAP (9)
- Spring 5 (
- Spring AC
- Spring Ba
- Spring Ba
- Spring Bo
- Spring HA
- Spring Mc
- Spring MV
- Spring MV
- Spring RE
- Spring3.0
- Spring4 (3
- SpringSec
- String in J
- struts2 (49
- thymeleaf
- UDDI (1)
- WebServi
- WSDL (9)

9 1 2

MENU

- ► 2017 (74)
- ▼ 2016 (60)
  - November
  - October (2
  - Septembe
  - August (12
  - July (6)
  - June (7)
  - March (8)
  - February (
  - January (1
- ► 2015 (11)
- ► 2014 (89)
- ► 2013 (352
- ► 2012 (85)

Follow

```
  <java.version>1.8</java.version>
 </properties>

 <dependencies>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <!-- <dependency>
   <groupId>com.h2database</groupId>
   <artifactId>h2</artifactId>
   <scope>runtime</scope>
  </dependency> -->
  <dependency>
   <groupId>mysql</groupId>
   <artifactId>mysql-connector-java</artifactId>
   <scope>runtime</scope>
  </dependency>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-test</artifactId>
   <scope>test</scope>
  </dependency>
 </dependencies>

 <build>
  <plugins>
   <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
   </plugin>
  </plugins>
 </build>


</project>
```

If you look at the above maven dependencies.

- **spring-boot-starter-data-jpa** dependency will download the files required for spring data jpa.
- **spring-boot-starter-web** is required because it is web based application to expose the REST endpoints.
- **mysql-connector-java** dependency will download the files required for mysql database.

Configuration file application.properties

Follow

```
# DataSource settings: set here your own configurations for the database
# connection. In this example we have "dojsb" as database name and
# "root" as username and password.
spring.datasource.url = jdbc:mysql://localhost:3307/dojdb
spring.datasource.username = root
spring.datasource.password = root

# Keep the connection alive if idle for a long time (needed in production)
spring.datasource.testWhileIdle = true
spring.datasource.validationQuery = SELECT 1

# Show or not log for each sql query
spring.jpa.show-sql = true

# Hibernate ddl auto (create, create-drop, update)
spring.jpa.hibernate.ddl-auto = create

# Naming strategy
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy

# Use spring.jpa.properties.* for Hibernate native properties (the prefix is
# stripped before adding them to the entity manager)

# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

server.port = 8181
```

Using hibernate configuration **ddl-auto = update** the database schema will be automatically **created** (and **updated**), creating tables and columns, accordingly to java entities found in the project. For this application I am using server port is **8181**.

Entity file Booking.java
Create an entity class representing a table in your db. Here I am using **Booking.java** as entity files which map to the **booking** table in the database **dojdb**.

```
/**
 *
 */
package com.dineshonjava.sdjpa.models;

import java.io.Serializable;
import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
```

Follow

```
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

/**
 * @author Dinesh.Rajput
 *
 */
@Entity
@Table(name = "BOOKING")
public class Booking implements Serializable{

 /**
  *
  */
 private static final long serialVersionUID = 1L;
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 Long bookingId;
 @Column
 String psngrName;
 @Column
 String departure;
 @Column
 String destination;
 @Column
 Date travelDate;

 public Long getBookingId() {
  return bookingId;
 }

 public void setBookingId(Long bookingId) {
  this.bookingId = bookingId;
 }

 public String getPsngrName() {
  return psngrName;
 }

 public void setPsngrName(String psngrName) {
  this.psngrName = psngrName;
 }

 public String getDeparture() {
  return departure;
 }

```

Follow

```
 public void setDeparture(String departure) {
  this.departure = departure;
 }

 public String getDestination() {
  return destination;
 }

 public void setDestination(String destination) {
  this.destination = destination;
 }

 public Date getTravelDate() {
  return travelDate;
 }

 public void setTravelDate(Date travelDate) {
  this.travelDate = travelDate;
 }


}
```

The **@Entity** annotation marks this class as a JPA entity. The **@Table** annotation specifies the db table's name.

The Data Access Object (Repository): BookingRepository.java
Repository is needed to works with entities in database's table, with methods like ***save, delete, update,*** etc. With Spring Data JPA a DAO for your entity is simply created by extending the CrudRepository interface provided by Spring. The following methods are some of the ones available from such interface: ***save***, ***delete***, ***deleteAll***, ***findOne*** and ***findAll***.

```
package com.dineshonjava.sdjpa.models;

import org.springframework.data.repository.CrudRepository;
import org.springframework.transaction.annotation.Transactional;

@Transactional
public interface BookingRepository extends CrudRepository {

 /**
    * This method will find an Boooking instance in the database by its departure.
    * Note that this method is not implemented and its working code will be
    * automatically generated from its signature by Spring Data JPA.
    */
   public Booking findByDeparture(String departure);
}

```

Adding Rest Controller BookingController.java

Here adding one more file for accessing Rest APIs is RestController it provide me some URLs for CRUD operation the booking table.

```java
/**
 *
 */
package com.dineshonjava.sdjpa.controller;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.dineshonjava.sdjpa.models.Booking;
import com.dineshonjava.sdjpa.models.BookingRepository;

/**
 * @author Dinesh.Rajput
 *
 */
@RestController
@RequestMapping("/booking")
public class BookingController {

 @Autowired
 BookingRepository bookingRepository;
 /**
  * GET /create  --> Create a new booking and save it in the database.
  */
 @RequestMapping("/create")
 public Booking create(Booking booking) {
  booking.setTravelDate(new Date());
  booking = bookingRepository.save(booking);
     return booking;
 }

 /**
  * GET /read  --> Read a booking by booking id from the database.
  */
 @RequestMapping("/read")
 public Booking read(@RequestParam Long bookingId) {
  Booking booking = bookingRepository.findOne(bookingId);
     return booking;
 }

 /**
  * GET /update  --> Update a booking record and save it in the database.
```

Follow

```
  */
 @RequestMapping("/update")
 public Booking update(@RequestParam Long bookingId, @RequestParam String psngrName) {
  Booking booking = bookingRepository.findOne(bookingId);
  booking.setPsngrName(psngrName);
  booking = bookingRepository.save(booking);
     return booking;
 }

 /**
  * GET /delete  --> Delete a booking from the database.
  */
 @RequestMapping("/delete")
 public String delete(@RequestParam Long bookingId) {
  bookingRepository.delete(bookingId);
     return "booking #"+bookingId+" deleted successfully";
 }
}
```

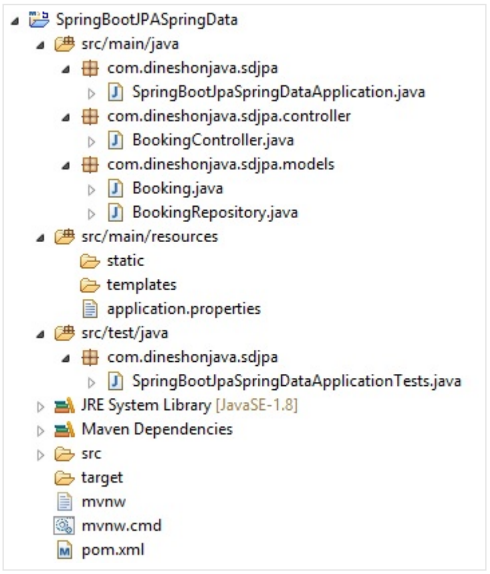
That's all! The connection with the database is done when we run the following file as Java Application or Spring Boot application.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootJpaSpringDataApplication {

public static void main(String[] args) {
SpringApplication.run(SpringBootJpaSpringDataApplication.class, args);
}
}
```

Project Structure:



```
SpringBootJPASpringData
  src/main/java
    com.dineshonjava.sdjpa
      SpringBootJpaSpringDataApplication.java
    com.dineshonjava.sdjpa.controller
      BookingController.java
    com.dineshonjava.sdjpa.models
      Booking.java
      BookingRepository.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
    com.dineshonjava.sdjpa
      SpringBootJpaSpringDataApplicationTests.java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  mvnw
  mvnw.cmd
  pom.xml
```

Follow

Test the controller launching the Spring Boot web application and using these urls:

Creating New Record:
/booking/create?psngrName=Dinesh&departure=Noida&destination=Pune create a new booking with an auto-generated.
For Example:
http://localhost:8181/booking/create?psngrName=Arnav&destination=USA&departure=Delhi

```
{
• bookingId: 4,
• psngrName: "Arnav",
• departure: "Delhi",
• destination: "USA",
• travelDate: 1470828738680
}
```

Reading a Record:
/booking/read?bookingId=[bookingId]: Read the booking with the passed bookingId.

For Example:
http://localhost:8181/booking/read?bookingId=1

```
{
• bookingId: 1,
• psngrName: "Dinesh",
• departure: "Noida",
• destination: "Pune",
• travelDate: 1470828563000
}
```

Updating a Record:
/booking/update?bookingId=[bookingId]&psngrName=Sweety: Update the booking for a given booking id.

For Example:
http://localhost:8181/booking/update?bookingId=5&psngrName=Suresh

```
{
• bookingId: 5,
• psngrName: "Suresh",
• departure: "Agra",
• destination: "Noida",
• travelDate: 1470828994000
}
```

Deleting a Record:

/booking/delete? bookingId=[bookingId]: Delete a booking for given booking id.

Follow

For Example:
http://localhost:8181/booking/delete?bookingId=5
booking #5 deleted successfully

For Whole Code of this application

https://github.com/DOJ-SoftwareConsultant/SpringBootJPASpringData

Summary
In this chapter I have explained how to use Spring Data JPA and Spring Boot frameworks. Also explained in short about the Spring Data Repositories and query methods.

Happy Spring Boot Learning!!!

Spring Boot Related Topics

5 Comments **dineshonjava** 1 **Login** Follow

Recommend 4 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Lenadro • 4 months ago
Hello, nice tutorial. Thanks
I have a question. What I should consider when using h2 DB? It is the same?
• Reply • Share ›

Dinesh Rajput Mod → Lenadro • 4 months ago
Hello, thanks for commenting.
Please have a look into this tutorial link. This has your answer.
http://www.dineshonjava.com...
• Reply • Share ›

Huang Binfang • 5 months ago
when I run it, I got the error:
UnsatisfiedDependencyException: Error creating bean with name 'bookingController': Unsatisfied dependency expressed through field 'bookingRepository';

Caused by: java.lang.IllegalArgumentException: Not a managed type: class java.lang.Object
• Reply • Share ›

Huang Binfang → Huang Binfang • 5 months ago
the issue has been fixed by change the repository to :

import org.springframework.data.repository.CrudRepository;

import javax.transaction.Transactional;

@Transactional
public interface BookingRepository extends CrudRepository<booking, long=""> {

}

Thanks.
• Reply • Share ›

Dinesh Rajput Mod → Huang Binfang • 5 months ago
Yes, It is good.
Thanks
• Reply • Share ›

ALSO ON **DINESHONJAVA**

Method injection with Spring using Lookup method property
2 comments • 6 months ago•
Dinesh Rajput — Thanks Naved.

Spring Boot Actuator A Complete Guide
5 comments • a year ago•
Dinesh Rajput — Thanks Mohan for your view.

RequestMapping Annotation in Spring MVC | DOJ Software Consultant
2 comments • a year ago•
Dinesh Rajput — Thanks for your feedback.

How to find all Pairs in Array of Integers whose Sum is equal to a given Number
2 comments • 7 months ago•
Dinesh Rajput — Could you please suggest best solution if you

Newer Post Home Older Post

POPULAR POSTS

Follow

Spring MVC Hibernate Integration CRUD Example Step by Step
In this example show how to write a simple web based application with CRUD operation using Spring3 MVC Framwork with Hibernate3 using Annot...

Core Java Tutorial-Learn Step by Step with Example
Hello Friends, Now we will focused on the Core Java tutorial, it is really a baby step to be a good, better and best Java ian :-) . I mea...

Spring Tutorial- Learn Framework Overview Step by Step
In this series of spring tutorial, it's provides many step by step examples and explanations on using Spring framework. The Spring framewor...

Spring Security Tutorial- Learn Step to Secure Web
In this spring security tutorial we will discuss about some of the security tips about the Spring Framework. Spring Security is a powerful...

How does java Hashmap work internally
What is Hashing? Hashing in its simplest form, is a way to assigning a unique code for any variable/object after applying any formula/algor...

REST and SOAP Web Service Interview Questions
In this interview questions tutorial we will explain most asking interviews questions on the web services like SOAP, REST etc and its protoc...

Spring Boot Tutorial Best Complete Introduction
Hello friends today I am going to discuss one of the latest innovation by the Spring Team (Pivotal Team) is Spring Boot, oops... sorry..fri...

Learn to Spring MVC Framework Tutorial with Simple Example
Model view controller is a software architecture design pattern. It provides solution to layer an application by separating three concerns...

NamedParameterJdbcTemplate in Spring with Example
The NamedParameterJdbcTemplate class helps you specify the named parameters inste a d of classic placeholder('?') argument. Named pa...

Spring Batch Tutorial - Introduction Get Best Examples
Hi In this spring batch tutorial I will discuss about one of the excellent feature of Spring Framework name Spring Batch. Spring Batch is a ...