http://zetcode.com/articles/springbootdatasourcebuilder/

**Spring Boot DataSourceBuilder tutorial**

In Spring Boot DataSourceBuilder tutorial, we present the Spring DataSourceBuilder in a command line Spring Boot application. In addition, we show how to integrate HikariCP connection pool.

*DataSourceBuilder* is a Java convenience class to create a data source with common implementations and properties.

*H2* is an open source relational database management system created entirely in Java. It can be embedded in Java applications or run in the client-server mode. It is easy to deploy and install and has small footprint.

*Spring* is a Java application framework for developing Java enterprise applications. It also helps integrate various enterprise components. Spring Boot makes it easy to create Spring-powered, production-grade applications and services with minimum setup requirements.

**Application**

The following is a simple Spring Boot console application. It retrieves data from the H2 in-memory database and displays it in the terminal. To configure the datasource, we use the DataSourceBuilderclass.

$ tree

.

├── pom.xml

└── src

├── main

│ ├── java

│ │ └── com

│ │ └── zetcode

│ │ ├── bean

│ │ │ └── Car.java

│ │ ├── conf

│ │ │ └── AppConfig.java

│ │ └── main

│ │ ├── Application.java

│ │ └── MyRunner.java

│ └── resources

│ ├── application.yml

│ ├── data-h2.sql

│ ├── logback.xml

│ └── schema-h2.sql

└── test

└── java

This is the project structure.

pom.xml

<?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.zetcode</groupId>

<artifactId>SpringBootDataSourceBuilder</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.2.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

<exclusions>

<exclusion>

<groupId>org.apache.tomcat</groupId>

<artifactId>tomcat-jdbc</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>com.zaxxer</groupId>

<artifactId>HikariCP</artifactId>

</dependency>

</dependencies>

</project>

The Maven pom.xml file contains dependencies for H2 driver, HikariCP connection pool, and Spring framework.

<exclusions>

<exclusion>

<groupId>org.apache.tomcat</groupId>

<artifactId>tomcat-jdbc</artifactId>

</exclusion>

</exclusions>

We exclude the Tomcat connection pool. According to the Spring Boot documentation, if Tomcat, HikariCP, or Commons DBCP are on the classpath one of them will be selected (in that order with Tomcat first). So we remove Tomcat pool in order to use HikariCP.

Car.java

package com.zetcode.bean;

public class Car {

private Long id;

private String name;

private int price;

public Car() {}

public Car(Long id, String name, int price) {

this.id = id;

this.name = name;

this.price = price;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getPrice() {

return price;

}

public void setPrice(int price) {

this.price = price;

}

@Override

public String toString() {

return "Car{" + "id=" + id + ", name=" + name + ", price=" + price + '}';

}

}

This is Car bean class. It contains item ID, name, and price.

schema-h2.sql

CREATE TABLE Cars(ID BIGINT PRIMARY KEY AUTO\_INCREMENT,

NAME VARCHAR(100), PRICE INT);

This SQL script creates the Cars table.

data-h2.sql

INSERT INTO Cars(Name, Price) VALUES('Audi', 52642);

INSERT INTO Cars(Name, Price) VALUES('Mercedes', 57127);

INSERT INTO Cars(Name, Price) VALUES('Skoda', 9000);

INSERT INTO Cars(Name, Price) VALUES('Volvo', 29000);

INSERT INTO Cars(Name, Price) VALUES('Bentley', 350000);

INSERT INTO Cars(Name, Price) VALUES('Citroen', 21000);

INSERT INTO Cars(Name, Price) VALUES('Hummer', 41400);

INSERT INTO Cars(Name, Price) VALUES('Volkswagen', 21600);

This script fills the table with data. Both scripts are located in the root of the classpath.

logback.xml

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<include resource="org/springframework/boot/logging/logback/base.xml" />

<logger name="org.springframework" level="ERROR"/>

<logger name="com.zetcode" level="DEBUG"/>

<logger name="com.zaxxer.hikari" level="INFO"/>

</configuration>

In the logback.xml file, we configre the logging levels. We set the logging level for Spring framework to ERROR so that our output is not cluttered with unnecessary details.

**Java configuration**

In the first option, we configure the data source in the Java code.

application.yml

spring:

main:

banner-mode: "off"

datasource:

platform: h2

The main Spring Boot configuration file is called application.yml. With the banner-mode property we turn off Spring Boot banner. The platform value is used in SQL initialization scripts:schema-${platform}.sql and data-${platform}.sql.

AppConfig.java

package com.zetcode.conf;

import javax.sql.DataSource;

import org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Primary;

@Configuration

public class AppConfig {

@Bean

public DataSource primaryDataSource() {

return DataSourceBuilder.create()

.username("sa")

.password("")

.url("jdbc:h2:mem:testdb;DB\_CLOSE\_ON\_EXIT=FALSE")

.driverClassName("org.h2.Driver")

.build();

}

}

We build the datasource in Java code. Spring Boot automatically configures HikariCP connection pool.

**Externalized configuration**

In the second option, we externalize the datasource configuration into the application.yml file.

application.yml

datasource:

hikari:

minimum-idle: 1

maximum-pool-size: 20

jdbcUrl: jdbc:h2:mem:testdb;DB\_CLOSE\_ON\_EXIT=FALSE

username: sa

password:

driverClassName: org.h2.Driver

spring:

main:

banner-mode: "off"

datasource:

platform: h2

In this application.yml, we configure the HikariCP connection pool.

AppConfig.java

package com.zetcode.conf;

import javax.sql.DataSource;

import org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Primary;

@Configuration

public class AppConfig {

@Bean

@Primary

@ConfigurationProperties(prefix = "datasource")

public DataSource primaryDataSource() {

return DataSourceBuilder.create().build();

}

}

With the @ConfigurationProperties annotation, we externalize the configuration into the properties/yaml file.

MyRunner.java

package com.zetcode.main;

import com.zetcode.bean.Car;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.jdbc.core.BeanPropertyRowMapper;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Component;

@Component

public class MyRunner implements CommandLineRunner {

@Autowired

private JdbcTemplate jtm;

@Override

public void run(String... args) throws Exception {

String sql = "SELECT \* FROM Cars";

List<Car> cars = jtm.query(sql, new BeanPropertyRowMapper(Car.class));

for (Car car: cars) {

System.out.println(car);

}

}

}

MyRunner executes an SQL query and shows the output in the console.

@Autowired

private JdbcTemplate jtm;

JdbcTemplate is injected.

String sql = "SELECT \* FROM Cars";

This is SQL to be executed. We select all cars from the Cars table.

List<Car> cars = jtm.query(sql, new BeanPropertyRowMapper(Car.class));

BeanPropertyRowMapper converts a row into a new instance of the specified mapped target class.

for (Car car: cars) {

System.out.println(car);

}

We iterate over all car objects and print them to the console.

Application.java

package com.zetcode.main;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication(scanBasePackages="com.zetcode")

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

The Application sets up the Spring Boot application.

$ mvn -q spring-boot:run

2017-04-12 17:57:00.328 INFO 15065 --- [ main] com.zetcode.main.Application : Starting Application on t400 with PID 15065 (/home/janbodnar/NetBeansProjects/springboot/SpringBootDataSourceBuilder/target/classes started by janbodnar in /home/janbodnar/NetBeansProjects/springboot/SpringBootDataSourceBuilder)

2017-04-12 17:57:00.344 DEBUG 15065 --- [ main] com.zetcode.main.Application : Running with Spring Boot v1.5.2.RELEASE, Spring v4.3.7.RELEASE

2017-04-12 17:57:00.351 INFO 15065 --- [ main] com.zetcode.main.Application : No active profile set, falling back to default profiles: default

2017-04-12 17:57:01.465 INFO 15065 --- [ main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Started.

Car{id=1, name=Audi, price=52642}

Car{id=2, name=Mercedes, price=57127}

Car{id=3, name=Skoda, price=9000}

Car{id=4, name=Volvo, price=29000}

Car{id=5, name=Bentley, price=350000}

Car{id=6, name=Citroen, price=21000}

Car{id=7, name=Hummer, price=41400}

Car{id=8, name=Volkswagen, price=21600}

2017-04-12 17:57:02.370 INFO 15065 --- [ main] com.zetcode.main.Application : Started Application in 2.683 seconds (JVM running for 7.561)

2017-04-12 17:57:02.385 INFO 15065 --- [ Thread-2] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Close initiated...

2017-04-12 17:57:02.394 INFO 15065 --- [ Thread-2] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Closed.

We run the Spring Boot application. The eight cars are displayed. The output shows that we use HikariCP.

In this tutorial, we used DataSourceBuilder in a Spring Boot console application. We have configured the data source in Java code and in external configuration file. We used in-memory H2 database to store data.