JaguarDB

Spring Boot With JaguarDB

Developer Guide

Spring Boot Framework

Spring Boot provides a platform for Java developers to develop a stand-alone and production-grade spring application. The following examples use Maven 3.9.3 (apache-maven-3.9.3) for Java project building, and Java 19.0.2 for compilation. The following pom.xml file is ready for Maven to build the project:

```
File  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.jaguardb</groupId>

    <artifactId>myproject</artifactId>

    <version>0.0.1-SNAPSHOT</version>


    <!-- Inherit defaults from Spring Boot -->

    <parent>

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

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

        <version>3.1.2</version>

    </parent>


    <!-- Add typical dependencies for a web application -->

    <dependencies>

        <dependency>

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

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

        </dependency>


        <dependency>

            <groupId>com.jaguardb</groupId>

            <artifactId>jaguar-jdbc</artifactId>

            <version>2.1</version>
    </dependency>
</dependencies>
    <!-- Package as an executable jar -->
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.11.0</version>
                <configuration>
                    <source>1.9</source>
                    <target>1.9</target>
                </configuration>
            </plugin>

        </plugins>
    </build>
    <!-- Add Spring repoistories -->
    <!-- (you don't need this if you are using a .RELEASE version) -->
    <repositories>
        <repository>
            <id>spring-snapshots</id>
            <url>https://repo.spring.io/snapshot</url>
            <snapshots><enabled>true</enabled></snapshots>
        </repository>
        <repository>
            <id>spring-milestones</id>
            <url>https://repo.spring.io/milestone</url>
        </repository>
    </repositories>
    <pluginRepositories>
```

```
        <pluginRepository>
            <id>spring-snapshots</id>
            <url>https://repo.spring.io/snapshot</url>
        </pluginRepository>
        <pluginRepository>
            <id>spring-milestones</id>
            <url>https://repo.spring.io/milestone</url>
        </pluginRepository>
    </pluginRepositories>
</project>
```

To invoke JaguarDB JDBC classed, the Jagua JDBC class jar file needs to be installed for Maven to find the classes with the following command:

```
JAR=$HOME/jaguar/lib/jaguar-jdbc-2.1.jar
mvn install:install-file -Dfile=$JAR -DgroupId=com.jaguardb \
    -DartifactId=jaguar-jdbc -Dversion=2.1 -Dpackaging=jar
```

In the same directory where the file pom.xml is located, you can create a directory src/main/java, and create a Java file for your project:

```
src/main/java/JaguarDBExample.java file:
```

```
import java.io.*;
import java.sql.DatabaseMetaData;
import java.sql.PreparedStatement;
import javax.sql.DataSource;
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.DriverManager;
import java.util.Random;
import com.jaguar.jdbc.JaguarDriver;

import com.jaguar.jdbc.JaguarDataSource;

import com.jaguar.jdbc.JaguarPreparedStatement;

import com.jaguar.jdbc.JaguarResultSetMetaData;


import org.springframework.boot.*;

import org.springframework.boot.autoconfigure.*;

import org.springframework.stereotype.*;

import org.springframework.web.bind.annotation.*;


@RestController

@EnableAutoConfiguration

public class JaguarDBExample {


    @RequestMapping("/")

    String home() {

        return "Hello World from JaguarDB SprintBoot!";

    }


    @RequestMapping("/create")

    String create() {

        try {

            Statement statement = connection_.createStatement();

            statement.executeUpdate("create table boot123 ( key: uid uuid, value: addr
char(32));" );

        } catch (SQLException e ) {

            return "create table exception";

        }


        return "Created table boot123 key: uid uuid, value: addr char(32)";

    }


    @RequestMapping("/insert")

    String insert() {

        StringBuffer sb = new StringBuffer();
        try {
            JaguarStatement jst;
            Statement statement = connection_.createStatement();
            statement.executeUpdate("insert into boot123 (addr ) values ( 'v1000')");
            jst = (JaguarStatement)statement;
            sb.append( jst.getLastUuid() + "<br>");

            statement.executeUpdate("insert into boot123 (addr ) values ( 'v2000')");
            jst = (JaguarStatement)statement;
            sb.append( jst.getLastUuid() + "<br>");

            statement.executeUpdate("insert into boot123 (addr ) values ( 'v3000')");
            jst = (JaguarStatement)statement;
            sb.append( jst.getLastUuid() + "<br>");

        } catch ( SQLException e ) {
            return "insert into table exception";
        }

        sb.append("<br>Inserted records into table boot123");
        return sb.toString();
    }

    @RequestMapping("/select")
    String select() {
        ResultSet rs;

        try {
            Statement statement = connection_.createStatement();
            rs = statement.executeQuery("select * from boot123");
        } catch ( SQLException e ) {
            return "select exception";
        }

        String key, val;
        StringBuffer sb = new StringBuffer();

        try {
            while(rs.next()) {
                key = rs.getString("uid");
                val = rs.getString("addr");

                sb.append( key );
                sb.append( ":" );
                sb.append( val );
                sb.append( "<br>" );
            }
        } catch ( SQLException e ) {
            return "select next() exception";
        }

        String hdr = "select result:<br><br>";
        String ret = hdr + sb.toString();
        return ret;
    }

    @RequestMapping("/drop")
    String drop() {
        try {
            Statement statement = connection_.createStatement();
            statement.executeUpdate("drop table if exists boot123");
        } catch (  SQLException e ) {
            return "drop exception";
        }

        return "Table boot123 is dropped";
    }

    public static void main(String[] args) throws Exception {
        System.loadLibrary("JaguarClient");
```

```
        ds_ = new JaguarDataSource( "127.0.0.1", 8888, "test");
        connection_ = ds_.getConnection("admin", "jaguarjaguarjaguar");

        SpringApplication.run(JaguarDBExample.class, args);
    }

    static private DataSource ds_;
    static private Connection connection_;
}
```

To build and run the Spring Boot application, you can execute the following command:

```
export LD_LIBRARY_PATH=$HOME/jaguar/lib
./mvnw spring-boot:run
```

The command mvnw is created in the current directory by the following command:

```
    mvn wrapper:wrapper
```

While the Sprint Boot application is up and running, you can open the following URL in your browser:

http://192.168.1.58:8080/

http://192.168.1.58:8080/create

http://192.168.1.58:8080/insert

http://192.168.1.58:8080/select

http://192.168.1.58:8080/drop

The IP address 192.168.1.58 represents the host where the Spring Boot application is running. The example provides a basic demonstration of how an application can interact with JaguarDB. However, developers have the potential to create much more sophisticated applications by leveraging the capabilities of JaguarDB and combining them with their expertise in Java programming.