|| Date: 22-Aug-24 ||

 Spring Boot JDBC

 -------------------------

* 3 things to consider are,
1. data
2. medium
3. storage

 raw data---🡪 I/O streams ---🡪 File

 (data) (medium) (storage)

 Java object -🡪 serialization -🡪 File

 (data) (medium) (storage)

 raw data --🡪 JDBC ---🡪 Database

 (data) (medium) (storage)

 Java object --🡪 JDBC/ORM ----🡪 Database

 (data) (medium) (storage)

JDBC Vs Spring JDBC:

1. In JDBC, we have to write the boiler-plate code like loading driver, creating connection, creating statement, closing connection. But In Spring JDBC, spring will do this.
2. In JDBC, we must do exception handling. But in Spring JDBC, it is optional.
3. In JDBC, there is a chance of memory leak. But in Spring JDBC, no chance of memory leak.

 || DATE: 23-Aug-24 ||

JdbcTemplate class:

* This is the main class in Spring JDBC API.
* A spring bean, can perform CRUD operations on database, by using JdbcTemplate class.
* JdbcTemplate class depends on DataSource object,

to obtain a database connection.

* To obtain a databae connection, a java application has two options.
1. DriverManager class
2. DataSource object
* DataSource is an interface, it has multiple implentations, and the spring framework provided implementation is, DriverManagerDataSource class.
* Suppose, if we are creating a spring jdbc application, then the below changes are required.
* 1. add spring-jdbc dependency
* 2. add spring-context dependency
* 3. add jdbc driver dependency
* 4. configure/register DataSource object into the container
* 5. register JdbcTemplate object into the container.
* suppose, if we are creating a spring boot jdbc application, then the above configurations are not required.
* we need to add spring-boot-starter-jdbc dependency and jdbc driver dependency in pom.xml file.
* In application.properties file, we need to configure data source properties.
* Some methods of JdbcTemplate class, to perform a SQL operations are,
1. update(sql): performs insert/update/delete operation with the given static sql.
2. update(sql, args): performs insert/update/delete operation with the given dynamic sql, binding the args with the sql.
3. queryForMap(sql): performs select operation to select a single row, with the given static sql.
4. queryForMap(sql, args): performs select operation to select a single row, with the given dynamic sql, binding the args with the sql.
5. queryForList(sql): performs select operation to select mulitple rows, with the given static sql.
6. queryForList(sql, args): performs select operation to select multiple rows, with the given dynamic sql, binding the args with the sql.
* while selecting a single record with queryForMap() method, the selected record will be mapped to a Map object. So queryForMap() method returns a Map object.
* while selecting multiple records with queryForList() method, each record will be mapped to a Map object and the map objects are stored into a List. So queryForList() method returns a List object.

for ex:

Map<String,Object> map = jdbcTemplate.queryForMap(“SELECT \* FROM EMP WHERE EMPNO=7101”);

List<Map<String,Object>> lst = jdbcTemplate.queryForList(“SELECT \* FROM EMP”);

===============================================

 || DATE: 24-Aug-24 ||



Application.java

@SpringBootApplication

**public** **class** Application {

 **public** **static** **void** main(String[] args) {

 SpringApplication.*run*(Application.**class**, args);

 }

}

IEmployeeDAO.java

------------------

**package** com.ashokit.demo.dao;

**public** **interface** IEmployeeDAO {

 **void** createRow(**int** empno, String ename, **double** salary);

 **void** updateRow(**int** empno, **double** salary);

 **void** fetchByEmpno(**int** empno);

 **void** fetchAll();

}

EmployeeDAOImpl.java

---------------------

@Repository

**public** **class** EmployeeDAOImpl **implements** IEmployeeDAO {

 @Autowired

 JdbcTemplate jdbcTemplate;

 @Override

 **public** **void** createRow(**int** empno, String ename, **double** salary) {

 jdbcTemplate.update("INSERT INTO EMP VALUES(?, ?, ?)", empno, ename, salary);

 System.***out***.println("Row inserted.....");

 }

 @Override

 **public** **void** updateRow(**int** empno, **double** salary) {

 jdbcTemplate.update("UPDATE EMP SET SAL = ? WHERE EMPNO = ? ", salary, empno);

 System.***out***.println("Row updated......");

 }

 @Override

 **public** **void** fetchByEmpno(**int** empno) {

 Map<String, Object> empMap = jdbcTemplate.queryForMap("SELECT \* FROM EMP WHERE EMPNO = ?", empno);

 Set<Entry<String, Object>> entries = empMap.entrySet();

 entries.forEach( entry -> System.***out***.println(entry.getKey() + " - " + entry.getValue()));

 }

 @Override

 **public** **void** fetchAll() {

 List<Map<String, Object>> empList = jdbcTemplate.queryForList("SELECT \* FROM EMP");

 **for** ( Map<String, Object> empMap : empList) {

 Set<Entry<String, Object>> entries = empMap.entrySet();

 entries.forEach( entry -> System.***out***.println(entry.getKey() + " - " + entry.getValue()));

 }

 }

}

MyApplicationRunner.java

@Component

**public** **class** MyApplicationRunner **implements** ApplicationRunner {

 @Autowired

 IEmployeeDAO empDao;

 @Override

 **public** **void** run(ApplicationArguments args) **throws** Exception {

 empDao.createRow(7655, "JOHN", 5000.0);

 System.***out***.println("====================================");

 empDao.updateRow(7788, 9000.0);

 System.***out***.println("====================================");

 empDao.fetchByEmpno(7201);

 System.***out***.println("====================================");

 empDao.fetchAll();

 }

}

application.properties

----------------------

spring.application.name=4-Application

# trun off the banner

spring.main.banner-mode=off

# data source properties

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

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

spring.datasource.username=root

spring.datasource.password=root

 || DATE: 26-Aug-24 ||



DatabaseFunctionCaller.java

---------------------------

**package** com.ashokit.demo.dao;

**public** **interface** DatabaseFunctionCaller {

 **double** callCalcBonus(**int** empno);

}

DatabaseFunctionCallerImpl.java

-------------------------------

@Repository

**public** **class** DatabaseFunctionCallerImpl **implements** DatabaseFunctionCaller {

 @Autowired

 SimpleJdbcCall simpleJdbcCall;

 @Override

 **public** **double** callCalcBonus(**int** empno) {

 //attach the database function name

 simpleJdbcCall.withFunctionName("calc\_bonus");

 Map<String, Integer> input = **new** HashMap<>();

 input.put("eno", empno); //function input parameter

 Map<String, Object> output = simpleJdbcCall.execute(input);

 Object obj = output.get("return");

 Double d = (Double) obj;

 **return** d.doubleValue();

 }

}

MyApplicationRunner.java

------------------------

@Component

**public** **class** MyApplicationRunner **implements** ApplicationRunner {

 @Autowired

 DatabaseFunctionCaller functionCaller;

 @Override

 **public** **void** run(ApplicationArguments args) **throws** Exception {

 **int** empno = Integer.*parseInt*(args.getOptionValues("empno").get(0));

 System.***out***.println("Bonus : " + functionCaller.callCalcBonus(empno));

 }

}

application.properties

-----------------------

spring.application.name=5-Application

spring.main.banner-mode=off

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

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

spring.datasource.username=root

spring.datasource.password=root

=======================================================



Employee.java

-------------

**package** com.ashokit.demo.model;

**public** **class** Employee {

 **private** **int** empno;

 **private** String ename;

 **private** **double** salary;

 **public** **int** getEmpno() {

 **return** empno;

 }

 **public** **void** setEmpno(**int** empno) {

 **this**.empno = empno;

 }

 **public** String getEname() {

 **return** ename;

 }

 **public** **void** setEname(String ename) {

 **this**.ename = ename;

 }

 **public** **double** getSalary() {

 **return** salary;

 }

 **public** **void** setSalary(**double** salary) {

 **this**.salary = salary;

 }

 @Override

 **public** String toString() {

 **return** "Employee [empno=" + empno + ", ename=" + ename + ", salary=" + salary + "]";

 }

}

EmployeeRowMapper.java

----------------------

/\*

 \* RowMapper is a functional interface,

 \* it maps a row of a ResultSet to a model.

 \* JdbcTemplate uses the RowMapper to map

 \* each row of ResultSet into a model.

 \*/

**public** **class** EmployeeRowMapper **implements** RowMapper<Employee> {

 @Override

 **public** Employee mapRow(ResultSet rs, **int** rowNum) **throws** SQLException {

 Employee employee = **new** Employee();

 employee.setEmpno(rs.getInt("EMPNO"));

 employee.setEname(rs.getString("ENAME"));

 employee.setSalary(rs.getDouble("SAL"));

 **return** employee;

 }

}

IEmployeeDAO.java

-----------------

**public** **interface** IEmployeeDAO {

 List<Employee> fetchAll();

}

EmployeeDAOImpl.java

--------------------

@Repository

**public** **class** EmployeeDAOImpl **implements** IEmployeeDAO {

 @Autowired

 JdbcTemplate jdbcTemplate;

 @Override

 **public** List<Employee> fetchAll() {

 List<Employee> empList = jdbcTemplate.query("SELECT \* FROM EMP", **new** EmployeeRowMapper());

 **return** empList;

 }

}

MyApplicationRunner.java

-------------------------

@Component

**public** **class** MyApplicationRunner **implements** ApplicationRunner {

 @Autowired

 IEmployeeDAO dao;

 @Override

 **public** **void** run(ApplicationArguments args) **throws** Exception {

 dao.fetchAll().forEach(employee -> System.***out***.println(employee) );

 }

}

application.properties

----------------------

spring.application.name=6-Application

spring.main.banner-mode=off

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

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

spring.datasource.username=root

spring.datasource.password=root