Spring Jdbc

Java Database Connectivity (JDBC) is an application programming interface **(API)** that defines how a client may access a database. It is a data access technology used for Java database connectivity.

It provides methods to query and update data in a database and is oriented toward relational databases. JDBC offers a natural Java interface for working with **SQL**.

The disadvantages of JDBC API are as follows:

1. Writing a lot of codes before and after executing the query, such as creating connection, creating a statement, closing result-set, closing connection, etc.
2. Writing exception handling code on the database logic.
3. Repetition of these codes from one to database logic is time-consuming.

These problems of **JDBC API** are eliminated by **Spring JDBC-Template.**

Spring framework provides the following approaches for **JDBC** database access:

- JdbcTemplate
- NamedParameterJdbcTemplate
- SimpleJdbcTemplate
- SimpleJdbcInsert and SimpleJdbcCall

JdbcTemplate:

JdbcTemplate is a central class in the **JDBC** core package that simplifies the use of **JDBC** and helps to avoid common errors. It internally uses **JDBC API** and eliminates a lot of problems with **JDBC API**. It executes SQL queries or updates, initiating iteration over ResultSets and catching **JDBC** exceptions and translating them to the generic.

It executes core **JDBC** workflow, leaving application code to provide SQL and extract results.

It handles the exception and provides the informative exception messages with the help of exception classes defined in **org.springframework.dao** package.

The common methods of spring JdbcTemplate class.

Methods	Description
public int update(String query)	Used to insert, update and delete records.
public int update(String query, Object... args)	Used to insert, update and delete records using **PreparedStatement** using given arguments.
public T execute(String sql, PreparedStatementCallback action)	Executes the query by using **PreparedStatementCallback.**
public void execute(String query)	Used to execute **DDL** query.
public T query(String sql, ResultSetExtractor result)	Used to fetch records using **ResultSetExtractor**.

Note: Use jdbcTemplate.query for multiple rows or list

Use jdbcTemplate.queryForObject for single row or value

ResultsetExtractor:

ResultSetExtractor is an interface that is used to fetch the records from the database. It's a callback interface that is used by JDBC Template's query() method where we need to pass the instance of ResultSetExtractor in order to fetch the data

Syntax of query() method of ResultSetExtractor:

public T query(String sqlQuery, ResultSetExtractor<T> resultSetExtractor)

In order to fetch the data using ResultSetExtractor, we need to implement the ResultSetExtractor interface and provide the definition for its method.

It has only one method. i.e., extractData() which takes an instance of ResultSet as an argument and returns the list.

RowMapper:

RowMapper interface is used to fetch the records from the database using the query() method of the JdbcTemplate class.

Syntax for query() method of JdbcTemplate class:

public T query(String sqlQuery, RowMapper<T> rowMapper)

RowMapper is a callback interface that is called for each row and maps the row of relations with the instances to the model(user-defined) class. Unlike ResultSetExtractor the RowMapper iterates the ResultSet internally and adds the extracted data into a collection, And we do not need to write the code for collections as we do in ResultSetExtractor.

Syntax: public T mapRow(ResultSet resultSet, int rowNumber)throws SQLException

A RowMapper is usually a simpler choice for ResultSet processing, mapping one result object per row instead of one result object for the entire ResultSet.

ResultSetExtractor is suppose to extract the whole ResultSet (possibly multiple rows), while RowMapper is feeded with row at a time.

NamedParameterJdbcTemplate:

Spring provides another way to insert data by named parameter. In such way, we use names instead of ? (question mark).

Example:

String query = "insert into student values(:sid, :sname, :saddr)";

Where :sid, :sname, :saddr are named parameters for which we have to provide values.

In the case of NamedParameterJdbcTemplate, we can provide values to the named parameters in the following two approaches.

By Using Map directly.

By using SqlParameterSource interface.

1. By Using Map directly.

```
String query = "insert into student values(:sid, :sname, :saddr)";

Map map = new HashMap();

map.put("sid", "S-111");

map.put("sname", "Ashok Kumar");

map.put("saddr", "Bhimavaram");

namedParameterJdbcTemplate.update(query, map);
```

SimpleJdbcCall :

SimpleJdbcCall class in Spring is used to call a stored procedure and execute a function in database.

Example:

Basically, here are the steps to call a database stored procedure using SimpleJdbcCall in Spring:

1. Create a new instance of SimpleJdbcCall and specify the name of the procedure:

```
SimpleJdbcCall actor = new SimpleJdbcCall(dataSource).withProcedureName("name");
```

2. Specify some IN parameters if required by the procedure:

```
SqlParameterSource params = new MapSqlParameterSource();

params.addValue("in_param_1", "value1")

    .addValue("in_param_2", "value2");
```

3. Execute the procedure and get the values of the OUT parameters into a Map:

```
Map<String, Object> out = actor.execute(params);
```

4. Read out the values returned from the procedure:

```
String value1 = (String) out.get("out_param_1");

Integer value2 = (Integer) out.get("out_param_2");
```