Using JdbcTemplate

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy/Spring5

Spring JdbcTemplate

- https://www.javatpoint.com/spring-JdbcTemplate-tutorial
- https://docs.spring.io/spring-framework/docs/5.3.10/javadoc-api/
- Spring **JdbcTemplate** is
  - A powerful mechanism to connect to the database and execute SQL queries.
  - **org.springframework.jdbc.core.JdbcTemplate**
  - Internally uses JDBC API, but eliminates a lot of problems of JDBC API.

public class JdbcTemplate extends JdbcAccessor implements JdbcOperations

Spring JdbcTemplate (Cont.)

- Problems of JDBC API
  - We need ***to write a lot of code*** before and after executing the query, such as creating connection, statement, closing resultset, connection etc.
  - We need ***to perform exception handling code*** on the database logic.
  - We need ***to handle transaction***.
  - ***Repetition of all these codes*** from one to another database logic is a time consuming task.
- Advantage of Spring **JdbcTemplate**
  - ***Eliminates*** all the above mentioned problems of JDBC API.
  - Provides methods to write the queries directly, so it ***saves*** a lot of work and time.

Spring JdbcTemplate (Cont.)

- Spring Jdbc Approaches
  - Spring framework provides following approaches for JDBC database access:
  - **JdbcTemplate**
    - Is the classic Spring JDBC approach and the most popular.
    - This ***lowest level*** approach and all others use a **JdbcTemplate** under the covers.
  - **NamedParameterJdbcTemplate**
    - Wraps a **JdbcTemplate** to provide named parameters instead of the traditional JDBC **?** placeholders.
    - This approach provides better documentation and ease of use when you have multiple parameters for an SQL statement.
    - **org.springframework.jdbc.core.namedparam** package

Spring JdbcTemplate (Cont.)

- Spring Jdbc Approaches (Cont.)
  - **SimpleJdbcInsert** and **SimpleJdbcCall**
    - Optimize database metadata to limit the amount of necessary configuration.
    - This approach simplifies coding so that you only need to provide the name of the table or procedure and provide a map of parameters matching the column names.
    - This only works if the database provides adequate metadata.
    - If the database doesn’t provide this metadata, you will have to provide explicit configuration of the parameters.
    - **org.springframework.jdbc.core.simple** package

Spring JdbcTemplate (Cont.)

- Spring Jdbc Approaches (Cont.)
  - RDBMS Objects including **MappingSqlQuery**, **SqlUpdate** and **StoredProcedure** requires you to create reusable and thread-safe objects during initialization of your data access layer.
  - This approach is modeled after JDO Query wherein you define your query string, declare parameters, and compile the query.
  - Once you do that, **execute** methods can be called multiple times with various parameter values passed in.

Spring JdbcTemplate (Cont.)

- **JdbcTemplate** class
  - Is the central class in the Spring JDBC support classes.
  - Takes care of creation and release of resources such as creating and closing of connection object etc.
  - So it will not lead to any problem if you forget to close the connection.
  - It handles the exception and provides the informative exception messages by the help of exception classes defined in the **org.springframework.dao** package.
  - We can perform all the database operations by the help of **JdbcTemplate** class such as insertion, updating, deletion and retrieval of the data from the database.

Spring JdbcTemplate (Cont.)

- **JdbcTemplate** class’s Methods
  - **public int update(String query)**
    - Is used to insert, update and delete records.
  - **public int update(String query, Object ... args)**
    - Is used to insert, update and delete records using **PreparedStatement** using given arguments.
  - **public void execute(String query)**
    - Is used to execute DDL query.
  - **public T execute(String sql, PreparedStatementCallback action)**
    - Executes the query by using **PreparedStatement** callback.
  - **public T query(String sql, ResultSetExtractor rse)**
    - Is used to fetch records using **ResultSetExtractor**.
  - **public List query(String sql, RowMapper rse)**
    - Is used to fetch records using **RowMapper**.

THE LAB

Task 1. CRUD of Spring JdbcTemplate

THE LAB

Task 2. Example of Spring JdbcTemplate

PreparedStatement in Spring JdbcTemplate

- We can execute parameterized query using Spring **JdbcTemplate** by the help of **execute()** method of **JdbcTemplate** class.
- To use parameterized query, we pass the instance of **PreparedStatementCallback** in the **execute()** method.
- Syntax of execute method to use parameterized query

```
public T execute(String sql, PreparedStatementCallback<T>);
```

- **PreparedStatementCallback** interface
  - It processes the input parameters and output results.
  - In such case, you don't need to care about single and double quotes.

PreparedStatement in Spring JdbcTemplate (Cont.)

- Method of **PreparedStatementCallback** interface
  - It has only one method **doInPreparedStatement**.
  - Syntax of the method is given below:

```
@override
public T doInPreparedStatement(PreparedStatement ps)
        throws SQLException, DataAccessException
```

THE LAB

Task 3. Example of PreparedStatement in Spring JdbcTemplate

ResultSetExtractor Example | Fetching Records by Spring JdbcTemplate

- We can easily fetch the records from the database using **query()** method of **JdbcTemplate** class where we need to pass the instance of **ResultSetExtractor**.
- Syntax of query method using **ResultSetExtractor**

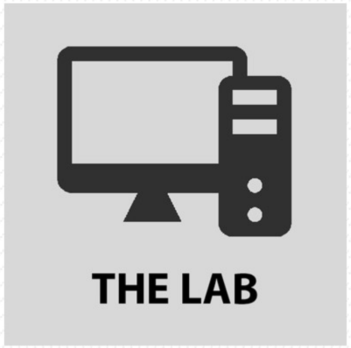
  public T query(String sql,ResultSetExtractor<T> rse)

- **ResultSetExtractor** Interface
  - **ResultSetExtractor** interface can be used to fetch records from the database.
  - It accepts a **ResultSet** and returns the list.

ResultSetExtractor Example | Fetching Records by Spring JdbcTemplate (Cont.)

- Method of **ResultSetExtractor** interface
  - It defines only one method **extractData** that accepts **ResultSet** instance as a parameter.
  - Syntax of the method is given below:

```
@Override
public T extractData(ResultSet rs)throws SQLException, DataAccessException
```



Task 4. ResultSetExtractor Example | Fetching Records by Spring JdbcTemplate

RowMapper Example | Fetching records by Spring JdbcTemplate

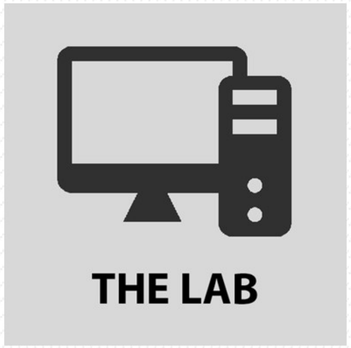
- Like **ResultSetExtractor**, we can use **RowMapper** interface to fetch the records from the database using **query()** method of **JdbcTemplate** class.
- In the execute of we need to pass the instance of **RowMapper** now.
- Syntax of query method using **RowMapper**

  public T query(String sql, RowMapper<T> rm)
- **RowMapper** Interface
  - **RowMapper** interface allows to map a row of the relations with the instance of user-defined class.
  - It iterates the **ResultSet** internally and adds it into the collection.
  - So we don't need to write a lot of code to fetch the records as **ResultSetExtractor**.

RowMapper Example | Fetching records by Spring JdbcTemplate (Cont.)

- Advantage of **RowMapper** over **ResultSetExtractor**
  - **RowMapper** saves a lot of code because it internally adds the data of **ResultSet** into the collection.
- Method of **RowMapper** interface
  - It defines only one method **mapRow** that accepts **ResultSet** instance and int as the parameter list.
  - Syntax of the method is given below:

```
@Override
public T mapRow(ResultSet rs, int rowNumber)throws SQLException
```



Task 5. RowMapper Example | Fetching records by Spring JdbcTemplate

Spring NamedParameterJdbcTemplate Example

- Spring provides another way to insert data by named parameter.
- In such way, we use names instead of **?**(question mark).
- So it is better to remember the data for the column.
- Simple example of named parameter query

  insert into employee values (:id, :name, :salary)
- Method of **NamedParameterJdbcTemplate** class
  - In this example, we are going to call only the execute method of **NamedParameterJdbcTemplate** class.
  - Syntax of the method is as follows:

pubic T execute(String sql,Map map, PreparedStatementCallback psc)

THE LAB

Task 6. Spring NamedParameterJdbcTemplate Example

Calling Stored Procedure in Spring JdbcTemplate

- **SimpleJdbcCall**
  - Is a multi-threaded, reusable object
  - Represents a call to a stored procedure or a stored function.
  - Provides meta-data processing to simplify the code needed to access basic stored procedures/functions.
  - Needs
    - The name of the procedure/function
    - A Map containing the parameters when you execute the call.
    - The names of the supplied parameters will be matched up with **in** and **out** parameters declared when the stored procedure was created.

Calling Stored Procedure in Spring JdbcTemplate

- **SimpleJdbcCall** (Cont.)
  - Constructor

    new SimpleJdbcCall(DataSource dataSource)

    new SimpleJdbcCall(JdbcTemplate jdbcTemplate)
  - Method

    withProcedureName(String procedureName)

    withFunctionName(String functionName)

    execute(SqlParameterSource parameterSource)

Calling Stored Procedure in Spring JdbcTemplate

- **SimpleJdbcCall** (Cont.)
  - CUD

```
SimpleJdbcCall jdbcCall = new SimpleJdbcCall(this.dataSource)
                                         .withProcedureName("sp_student_insert");
SqlParameterSource in = new MapSqlParameterSource()
                                         .addValue("v_name", name)
                                          .addValue("v_age", age);
jdbcCall.execute(in);
```

Calling Stored Procedure in Spring JdbcTemplate

- **SimpleJdbcCall** (Cont.)
  - READ

```
SimpleJdbcCall jdbcCall = new SimpleJdbcCall(this.dataSource)
                          .withProcedureName("sp_student_select")
                          .declareParameters(
                             new SqlOutParameter("v_name", Types.VARCHAR),
                             new SqlOutParameter("v_age", Types.INTEGER));
SqlParameterSource in = new MapSqlParameterSource().addValue("v_id", id);
Map<String, Object> map = jdbcCall.execute(in);
```

Calling Stored Procedure in Spring JdbcTemplate

- **CallableStatementCreator**
  - Creates a **CallableStatement** given a connection, provided by the **JdbcTemplate** class.
  - Implementations are responsible for providing SQL and any necessary parameters.

```
CallableStatement createCallableStatement(Connection conn)
                                                    throws SQLException
```

Calling Stored Procedure in Spring JdbcTemplate

- **CallableStatementCreator** (Cont.)
  - CUD

```
List<SqlParameter> parameters = Arrays.asList(
                                  new SqlParameter(Types.VARCHAR), new SqlParameter(Types.INTEGER));
this.jdbcTemplate.call(new CallableStatementCreator() {
      @Override
      public CallableStatement createCallableStatement(Connection conn) throws SQLException {
                CallableStatement cstmt = conn.prepareCall("{call sp_student_insert(?, ?)}");
                cstmt.setString(1, name);
                cstmt.setInt(2, age);
                return cstmt;
      }},
parameters);
```

Calling Stored Procedure in Spring JdbcTemplate

- **CallableStatementCreator** (Cont.)
  - READ

```
List<SqlParameter> parameter = Arrays.asList(new SqlParameter(Types.INTEGER),
                                             new SqlOutParameter("v_name", Types.VARCHAR),
                                             new SqlOutParameter("v_age", Types.INTEGER));
Map<String, Object> map = this.jdbcTemplate.call(new CallableStatementCreator() {
        @Override
        public CallableStatement createCallableStatement(Connection conn) throws SQLException {
                CallableStatement cstmt = conn.prepareCall("{call sp_student_select(?, ?, ?)}");
                cstmt.setInt(1, id);
                return cstmt;
        }},
parameter);
```

THE LAB

Task 7. Calling Stored Procedure