Spring - JDBC Framework Overview

While working with the database using plain old JDBC, it becomes cumbersome to write unnecessary code to handle exceptions, opening and closing database connections, etc. However, Spring JDBC Framework takes care of all the low-level details starting from opening the connection, prepare and execute the SQL statement, process exceptions, handle transactions and finally close the connection.

So what you have to do is just define the connection parameters and specify the SQL statement to be executed and do the required work for each iteration while fetching data from the database.

Spring JDBC provides several approaches and correspondingly different classes to interface with the database. I'm going to take classic and the most popular approach which makes use of **JdbcTemplate** class of the framework. This is the central framework class that manages all the database communication and exception handling.

JdbcTemplate Class

The JDBC Template class executes SQL queries, updates statements, stores procedure calls, performs iteration over ResultSets, and extracts returned parameter values. It also catches JDBC exceptions and translates them to the generic, more informative, exception hierarchy defined in the org.springframework.dao package.

Instances of the *JdbcTemplate* class are *threadsafe* once configured. So you can configure a single instance of a *JdbcTemplate* and then safely inject this shared reference into multiple DAOs.

A common practice when using the JDBC Template class is to configure a *DataSource* in your Spring configuration file, and then dependency-inject that shared DataSource bean into your DAO classes, and the JdbcTemplate is created in the setter for the DataSource.

Configuring Data Source

Let us create a database table **Student** in our database **TEST**. We assume you are working with MySQL database, if you work with any other database then you can change your DDL and SQL queries accordingly.

```
CREATE TABLE Student(
   ID   INT NOT NULL AUTO_INCREMENT,
   NAME VARCHAR(20) NOT NULL,
   AGE  INT NOT NULL,
   PRIMARY KEY (ID)
);
```

Now we need to supply a DataSource to the JDBC Template so it can configure itself to get database access. You can configure the DataSource in the XML file with a piece of code as shown in the following code snippet −

```
<bean id = "dataSource"
   class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name = "driverClassName" value = "com.mysql.jdbc.Driver"/>
   <property name = "url" value = "jdbc:mysql://localhost:3306/TEST"/>
   <property name = "username" value = "root"/>
   <property name = "password" value = "password"/>
</bean>
```

Data Access Object (DAO)

DAO stands for Data Access Object, which is commonly used for database interaction. DAOs exist to provide a means to read and write data to the database and they should expose this functionality through an interface by which the rest of the application will access them.

The DAO support in Spring makes it easy to work with data access technologies like JDBC, Hibernate, JPA, or JDO in a consistent way.

Executing SQL statements

Let us see how we can perform CRUD (Create, Read, Update and Delete) operation on database tables using SQL and JDBC Template object.

Querying for an integer

```
String SQL = "select count(*) from Student";
int rowCount = jdbcTemplateObject.queryForInt( SQL );
```

Querying for a long

```
String SQL = "select count(*) from Student";
long rowCount = jdbcTemplateObject.queryForLong( SQL );
```

A simple query using a bind variable

```
String SQL = "select age from Student where id = ?";
int age = jdbcTemplateObject.queryForInt(SQL, new Object[]{10});
```

Querying for a String

```
String SQL = "select name from Student where id = ?";
String name = jdbcTemplateObject.queryForObject(SQL, new Object[]{10}, String.class
```

Querying and returning an object

```
String SQL = "select * from Student where id = ?";
Student student = jdbcTemplateObject.queryForObject(
   SQL, new Object[]{10}, new StudentMapper());

public class StudentMapper implements RowMapper<Student> {
   public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
      Student student = new Student();
      student.setID(rs.getInt("id"));
      student.setName(rs.getString("name"));
      student.setAge(rs.getInt("age"));

      return student;
   }
}
```

Querying and returning multiple objects

```
String SQL = "select * from Student";
List<Student> students = jdbcTemplateObject.query(
   SQL, new StudentMapper());

public class StudentMapper implements RowMapper<Student> {
   public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
      Student student = new Student();
      student.setID(rs.getInt("id"));
      student.setName(rs.getString("name"));
      student.setAge(rs.getInt("age"));

      return student;
   }
}
```

Inserting a row into the table

```
String SQL = "insert into Student (name, age) values (?, ?)";
jdbcTemplateObject.update( SQL, new Object[]{"Zara", 11} );
```

Updating a row into the table

```
String SQL = "update Student set name = ? where id = ?";
jdbcTemplateObject.update( SQL, new Object[]{"Zara", 10} );
```

Deleting a row from the table

```
String SQL = "delete Student where id = ?";
jdbcTemplateObject.update( SQL, new Object[]{20} );
```

Executing DDL Statements

You can use the **execute(..)** method from *jdbcTemplate* to execute any SQL statements or DDL statements. Following is an example to use CREATE statement to create a table −

```
String SQL = "CREATE TABLE Student( " +
   "ID   INT NOT NULL AUTO_INCREMENT, " +
   "NAME VARCHAR(20) NOT NULL, " +
   "AGE  INT NOT NULL, " +
   "PRIMARY KEY (ID));"

jdbcTemplateObject.execute( SQL );
```

Spring JDBC Framework Examples

Based on the above concepts, let us check few important examples which will help you in understanding usage of JDBC framework in Spring −

Sr.No.	Example & Description
1	Spring JDBC Example This example will explain how to write a simple JDBC-based Spring application.
2	SQL Stored Procedure in Spring Learn how to call SQL stored procedure while using JDBC in Spring.

Useful Video Courses