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 );

To understand the concepts related to Spring JDBC framework with JdbcTemplate class, let us write a simple example, which will implement all the CRUD operations on the following Student table.

CREATE TABLE Student(

ID INT NOT NULL AUTO\_INCREMENT,

NAME VARCHAR(20) NOT NULL,

AGE INT NOT NULL,

PRIMARY KEY (ID)

);

Before proceeding, let us have a working Eclipse IDE in place and take the following steps to create a Spring application −

|  |  |
| --- | --- |
| **Steps** | **Description** |
| 1 | Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Add Spring JDBC specific latest libraries **mysql-connector-java.jar**, **org.springframework.jdbc.jar** and**org.springframework.transaction.jar** in the project. You can download required libraries if you do not have them already. |
| 4 | Create DAO interface *StudentDAO* and list down all the required methods. Though it is not required and you can directly write *StudentJDBCTemplate*class, but as a good practice, let's do it. |
| 5 | Create other required Java classes *Student*, *StudentMapper*, *StudentJDBCTemplate* and *MainApp* under the *com.tutorialspoint*package. |
| 6 | Make sure you already created **Student** table in TEST database. Also make sure your MySQL server is working fine and you have read/write access on the database using the give username and password. |
| 7 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 8 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

Following is the content of the Data Access Object interface file **StudentDAO.java** −

package com.tutorialspoint;

import java.util.List;

import javax.sql.DataSource;

public interface StudentDAO {

/\*\*

\* This is the method to be used to initialize

\* database resources ie. connection.

\*/

public void setDataSource(DataSource ds);

/\*\*

\* This is the method to be used to create

\* a record in the Student table.

\*/

public void create(String name, Integer age);

/\*\*

\* This is the method to be used to list down

\* a record from the Student table corresponding

\* to a passed student id.

\*/

public Student getStudent(Integer id);

/\*\*

\* This is the method to be used to list down

\* all the records from the Student table.

\*/

public List<Student> listStudents();

/\*\*

\* This is the method to be used to delete

\* a record from the Student table corresponding

\* to a passed student id.

\*/

public void delete(Integer id);

/\*\*

\* This is the method to be used to update

\* a record into the Student table.

\*/

public void update(Integer id, Integer age);

}

Following is the content of the **Student.java** file

package com.tutorialspoint;

public class Student {

private Integer age;

private String name;

private Integer id;

public void setAge(Integer age) {

this.age = age;

}

public Integer getAge() {

return age;

}

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setId(Integer id) {

this.id = id;

}

public Integer getId() {

return id;

}

}

Following is the content of the **StudentMapper.java** file

package com.tutorialspoint;

import java.sql.ResultSet;

import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

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;

}

}

Following is the implementation class file **StudentJDBCTemplate.java** for the defined DAO interface StudentDAO.

package com.tutorialspoint;

import java.util.List;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;

public class StudentJDBCTemplate implements StudentDAO {

private DataSource dataSource;

private JdbcTemplate jdbcTemplateObject;

public void setDataSource(DataSource dataSource) {

this.dataSource = dataSource;

this.jdbcTemplateObject = new JdbcTemplate(dataSource);

}

public void create(String name, Integer age) {

String SQL = "insert into Student (name, age) values (?, ?)";

jdbcTemplateObject.update( SQL, name, age);

System.out.println("Created Record Name = " + name + " Age = " + age);

return;

}

public Student getStudent(Integer id) {

String SQL = "select \* from Student where id = ?";

Student student = jdbcTemplateObject.queryForObject(SQL,

new Object[]{id}, new StudentMapper());

return student;

}

public List<Student> listStudents() {

String SQL = "select \* from Student";

List <Student> students = jdbcTemplateObject.query(SQL, new StudentMapper());

return students;

}

public void delete(Integer id) {

String SQL = "delete from Student where id = ?";

jdbcTemplateObject.update(SQL, id);

System.out.println("Deleted Record with ID = " + id );

return;

}

public void update(Integer id, Integer age){

String SQL = "update Student set age = ? where id = ?";

jdbcTemplateObject.update(SQL, age, id);

System.out.println("Updated Record with ID = " + id );

return;

}

}

Following is the content of the **MainApp.java** file

package com.tutorialspoint;

import java.util.List;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.tutorialspoint.StudentJDBCTemplate;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

StudentJDBCTemplate studentJDBCTemplate =

(StudentJDBCTemplate)context.getBean("studentJDBCTemplate");

System.out.println("------Records Creation--------" );

studentJDBCTemplate.create("Zara", 11);

studentJDBCTemplate.create("Nuha", 2);

studentJDBCTemplate.create("Ayan", 15);

System.out.println("------Listing Multiple Records--------" );

List<Student> students = studentJDBCTemplate.listStudents();

for (Student record : students) {

System.out.print("ID : " + record.getId() );

System.out.print(", Name : " + record.getName() );

System.out.println(", Age : " + record.getAge());

}

System.out.println("----Updating Record with ID = 2 -----" );

studentJDBCTemplate.update(2, 20);

System.out.println("----Listing Record with ID = 2 -----" );

Student student = studentJDBCTemplate.getStudent(2);

System.out.print("ID : " + student.getId() );

System.out.print(", Name : " + student.getName() );

System.out.println(", Age : " + student.getAge());

}

}

Following is the configuration file **Beans.xml**

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd ">

<!-- Initialization for data source -->

<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>

<!-- Definition for studentJDBCTemplate bean -->

<bean id = "studentJDBCTemplate"

class = "com.tutorialspoint.StudentJDBCTemplate">

<property name = "dataSource" ref = "dataSource" />

</bean>

</beans>

Once you are done creating the source and bean configuration files, let us run the application. If everything is fine with your application, it will print the following message −

------Records Creation--------

Created Record Name = Zara Age = 11

Created Record Name = Nuha Age = 2

Created Record Name = Ayan Age = 15

------Listing Multiple Records--------

ID : 1, Name : Zara, Age : 11

ID : 2, Name : Nuha, Age : 2

ID : 3, Name : Ayan, Age : 15

----Updating Record with ID = 2 -----

Updated Record with ID = 2

----Listing Record with ID = 2 -----

ID : 2, Name : Nuha, Age : 20

You can try and delete the operation yourself, which we have not used in the example, but now you have one working application based on Spring JDBC framework, which you can extend to add sophisticated functionality based on your project requirements. There are other approaches to access the database where you will use **NamedParameterJdbcTemplate** and **SimpleJdbcTemplate** classes, so if you are interested in learning these classes then kindly check the reference manual for Spring Framework.