Spring4Session2
Spring JDBC Framework

Spring JDBC Framework

While working with database using plain old JDBC, it becomes cumbersome to write unnecessary code to handle exceptions, opening and closing database connections etc.

But **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 do is just define 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.

The most popular approach is using **JdbcTemplate** class of the framework.

This is the central framework class that manages all the database communication and exception handling.

JdbcTemplate Class

The **JdbcTemplate** class executes SQL queries, update statements and stored procedure calls, performs iteration over ResultSet and extraction of 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 JdbcTemplate 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

We need to supply a DataSource to the JdbcTemplate so it can configure itself to get database access.

We can configure the DataSource in the XML file with a piece of code as shown below:

```
<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="root"/>
</bean>
```

Executing SQL statements

Querying for an integer:

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

```
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);
```

Executing SQL statements

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;
}
}
```

Executing SQL statements

Querying and returning a 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;
       }
}
```

Executing SQL statements

Inserting a row into a table

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

jdbcTemplateObject.update( SQL, new Object[]{"Ravi", 11});
```

Updating a row into the table

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

Deleting row into the table

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

Executing DDL statements

```
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 );
```

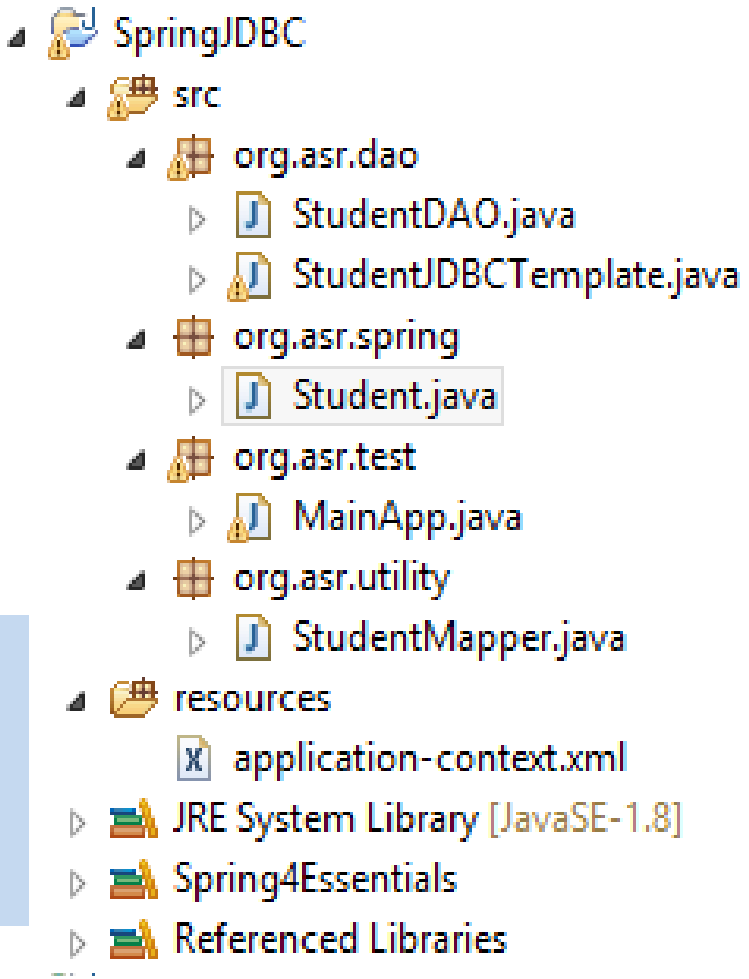
Spring JDBC Standalone Application

Include the following jar file into your library:

- spring-jdbc-4.0.6.RELEASE.jar
- spring-tx-4.0.6.RELEASE.jar

Note: spring-tx stands for Spring Transaction

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>4.2.6.RELEASE</version>
</dependency>
```



Student POJO

```
public class Student {
        private Integer id;
        private String name;
        private Integer age;

        //getter  and setter methods
}
```

application-context.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"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-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/csctest"/>
       <property name="username" value="root"/>
       <property name="password" value="root"/>
    </bean>

    <!-- Definition for studentJDBCTemplate bean -->
    <bean id="studentJDBCTemplate"   class="org.asr.dao.StudentJDBCTemplate">
       <property name="dataSource"  ref="dataSource" />
    </bean>
</beans>
```

IStudentDAO interface

```
public interface IStudentDAO {
/**
   * 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);

  }
```

StudentDAO Implementation

StudentJDBCTemplate.java

```
public class StudentJDBCTemplate implements IStudentDAO {

    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;
  }
```

Contd…

Contd…

StudentDAO Implementation

```
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;
   }
}
```

StudentMapper class

```
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;
   }
}
```

Tester class

```
public class MainApp {
public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("application-context.xml");
      StudentJDBCTemplate studentJDBCTemplate =
      (StudentJDBCTemplate)context.getBean("studentJDBCTemplate");
      System.out.println("------Records Creation--------" );
      studentJDBCTemplate.create("Ravi Kumar", 42);
      studentJDBCTemplate.create("Smith", 32);
      studentJDBCTemplate.create("Laxmi", 28);

      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());
}
}
```

Spring JDBC Web Application

SpringMVCDemoApp.zip

Spring JDBC Web Application

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
  <display-name>SpringMVCDemoApp</display-name>
  <servlet>
    <servlet-name>spring-jdbc</servlet-name>
    <servlet-class> org.springframework.web.servlet.DispatcherServlet </servlet-
class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>spring-jdbc</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

Spring XML Template

```
<?xml version="1.0" encoding="utf-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context.xsd
    http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd
    http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd">

                .....

</beans>
```

spring-jdbc-servlet.xml

```
<context:component-scan base-package="com.varaunited.trg"/>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
</bean>


  <!-- Simple implementation of the standard JDBC DataSource interface,
      configuring the plain old JDBC DriverManager via bean properties -->
  <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
    <property name="url" value="jdbc:oracle:thin:@//localhost:1521/myoracle" />
    <property name="username" value="scott" />
    <property name="password" value="tiger" />
  </bean>
….
```

spring-jdbc-servlet.xml

```
…..
    <!-- responsible for registering the necessary Spring components that power annotation-driven
transaction management;     such as when @Transactional methods are invoked -->
<!-- Enable Annotation based Declarative Transaction Management -->
<tx:annotation-driven proxy-target-class="true"
                transaction-manager="transactionManager" />

<!-- Creating TransactionManager Bean, since JDBC we are creating of type
                        DataSourceTransactionManager -->
<bean id="transactionManager"
                class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
                        <property name="dataSource" ref="dataSource" />
</bean>

    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
                        <property name="dataSource" ref="dataSource"/>
    </bean>

<mvc:annotation-driven />
</beans>
```

In Service class:

```
@Transactional
public class CustomerService{
}
```

In DAO class:

```
@Autowired
JdbcTemplate jdbcTemplate;
```

Thank You!