spring JDBC

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

About the Tutorial

Spring JDBC Framework takes care of all the low-level details starting from opening the connection, preparing and executing the SQL statement, processing exceptions, handling transactions, and finally closing the connection.

This tutorial will take you through simple and practical approaches while learning JDBC framework provided by Spring.

Audience

This tutorial has been prepared for the beginners to help them understand the basic to advanced concepts related to JDBC framework of Spring.

Prerequisites

Before you start practicing the various types of examples given in this tutorial, we assume that you are already aware about computer programs and computer programming languages.

Copyright & Disclaimer

Table of Contents

Spring JDBC Basics

1. Spring JDBC – Overview

While working with 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, preparing and executing the SQL statement, processing exceptions, handling transactions, and finally closing the connection.

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. In this tutorial, we will take classic and the most popular approach which makes use of JDBC Template class of the framework. This is the central framework class that manages all the database communication and exception handling.

JDBC Template Class

JDBC Template class executes SQL queries, updates statements and stored procedure calls, performs iteration over ResultSets 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 JDBC Template class are threadsafe once configured. So, you can configure a single instance of a JDBC Template 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. The JDBC Template is created in the setter for the DataSource.

Data Access Object (DAO)

DAO stands for **Data Access Oject** 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 Data Access Object (DAO) support in Spring makes it easy to work with data access technologies such as JDBC, Hibernate, JPA, or JDO in a consistent way.

2. Spring JDBC – Environment Setup

This chapter takes you through the process of setting up Spring-AOP on Windows and Linux based systems. Spring AOP can be easily installed and integrated with your current Java environment and MAVEN by following a few simple steps without any complex setup procedures. User administration is required while installation.

System Requirements

JDK	Java SE 2 JDK 1.5 or above
Memory	1 GB RAM (recommended)
Disk Space	No minimum requirement
Operating System Version	Windows XP or above, Linux

Let us now proceed with the steps to install Spring AOP.

Step 1: Verify your Java Installation

First of all, you need to have Java Software Development Kit (SDK) installed on your system. To verify this, execute any of the following two commands depending on the platform you are working on.

If the Java installation has been done properly, then it will display the current version and specification of your Java installation. A sample output is given in the following table.

Platform	Command	Sample Output
Windows	Open command console and type: **\>java -version**	Java version "1.7.0_60" Java (TM) SE Run Time Environment (build 1.7.0_60-b19) Java Hotspot (TM) 64-bit Server VM (build 24.60-b09,mixed mode)
Linux	Open command terminal and type: **$java -version**	java version "1.7.0_25" Open JDK Runtime Environment (rhel-2.3.10.4.el6_4-x86_64) Open JDK 64-Bit Server VM (build 23.7-b01, mixed mode)

We assume the readers of this tutorial have Java SDK version 1.7.0_60 installed on their system. In case you do not have Java SDK, download its current version from http://www.oracle.com/technetwork/java/javase/downloads/index.html and have it installed.

Step 2: Set your Java Environment

Set the environment variable JAVA_HOME to point to the base directory location where Java is installed on your machine. For example,

Platform	Description
Windows	Set JAVA_HOME to C:\ProgramFiles\java\jdk1.7.0_60
Linux	Export JAVA_HOME=/usr/local/java-current

Append the full path of Java compiler location to the System Path.

Platform	Description
Windows	Append the String "C:\Program Files\Java\jdk1.7.0_60\bin" to the end of the system variable PATH.
Linux	Export PATH=$PATH:$JAVA_HOME/bin/

Execute the command **java -version** from the command prompt as explained above.

Step 3: Download Maven Archive

Download Maven 3.3.3 from http://maven.apache.org/download.cgi

OS	Archive name
Windows	apache-maven-3.3.3-bin.zip
Linux	apache-maven-3.3.3-bin.tar.gz
Mac	apache-maven-3.3.3-bin.tar.gz

Step 4: Extract the Maven Archive

Extract the archive to the directory you wish to install Maven 3.3.3. The subdirectory apache-maven-3.3.3 will be created from the archive.

OS	Location (can be different based on your installation)
Windows	C:\Program Files\Apache Software Foundation\apache-maven-3.3.3
Linux	/usr/local/apache-maven
Mac	/usr/local/apache-maven

Step 5: Set Maven environment variables

Add M2_HOME, M2, MAVEN_OPTS to environment variables.

OS	Output
Windows	Set the environment variables using system properties. *M2_HOME=C:\Program Files\Apache Software Foundation\apache-maven-3.3.3* *M2=%M2_HOME%\bin* *MAVEN_OPTS=-Xms256m -Xmx512m*
Linux	Open command terminal and set environment variables. *export M2_HOME=/usr/local/apache-maven/apache-maven-3.3.3* *export M2=$M2_HOME/bin* *export MAVEN_OPTS=-Xms256m -Xmx512m*
Mac	Open command terminal and set environment variables. *export M2_HOME=/usr/local/apache-maven/apache-maven-3.3.3* *export M2=$M2_HOME/bin* *export MAVEN_OPTS=-Xms256m -Xmx512m*

Step 6: Add Maven Bin Directory Location to System Path

Now append M2 variable to System Path.

OS	Output
Windows	Append the string ;%M2% to the end of the system variable, Path.
Linux	export PATH=$M2:$PATH
Mac	export PATH=$M2:$PATH

Step 7: Verify Maven installation

Now open console, execute the following **mvn** command.

OS	Task	Command
Windows	Open Command Console	c:\> mvn --version
Linux	Open Command Terminal	$ mvn --version
Mac	Open Terminal	machine:< joseph$ mvn --version

Finally, verify the output of the above commands, which should be something as follows:

OS	Output
Windows	Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T17:27:37+05:30) Maven home: C:\Program Files\Apache Software Foundation\apache-maven-3.3.3 Java version: 1.7.0_75, vendor: Oracle Corporation Java home: C:\Program Files\Java\jdk1.7.0_75\jre Default locale: en_US, platform encoding: Cp1252
Linux	Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T17:27:37+05:30) Maven home: /usr/local/apache-maven/apache-maven-3.3.3 Java version: 1.7.0_75, vendor: Oracle Corporation Java home: /usr/local/java-current/jdk1.7.0_75/jre
Mac	Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T17:27:37+05:30) Maven home: /usr/local/apache-maven/apache-maven-3.3.3 Java version: 1.7.0_75, vendor: Oracle Corporation Java home: /Library/Java/Home/jdk1.7.0_75/jre

Step 8: Setup Eclipse IDE

All the examples in this tutorial have been written using Eclipse IDE. So, I would suggest you should have the latest version of Eclipse installed on your machine.

To install Eclipse IDE, download the latest Eclipse binaries from http://www.eclipse.org/downloads/. Once you have downloaded the installation, unpack the binary distribution into a convenient location. For example, in C:\eclipse on Windows, or /usr/local/eclipse on Linux/Unix. Finally, set PATH variable appropriately.

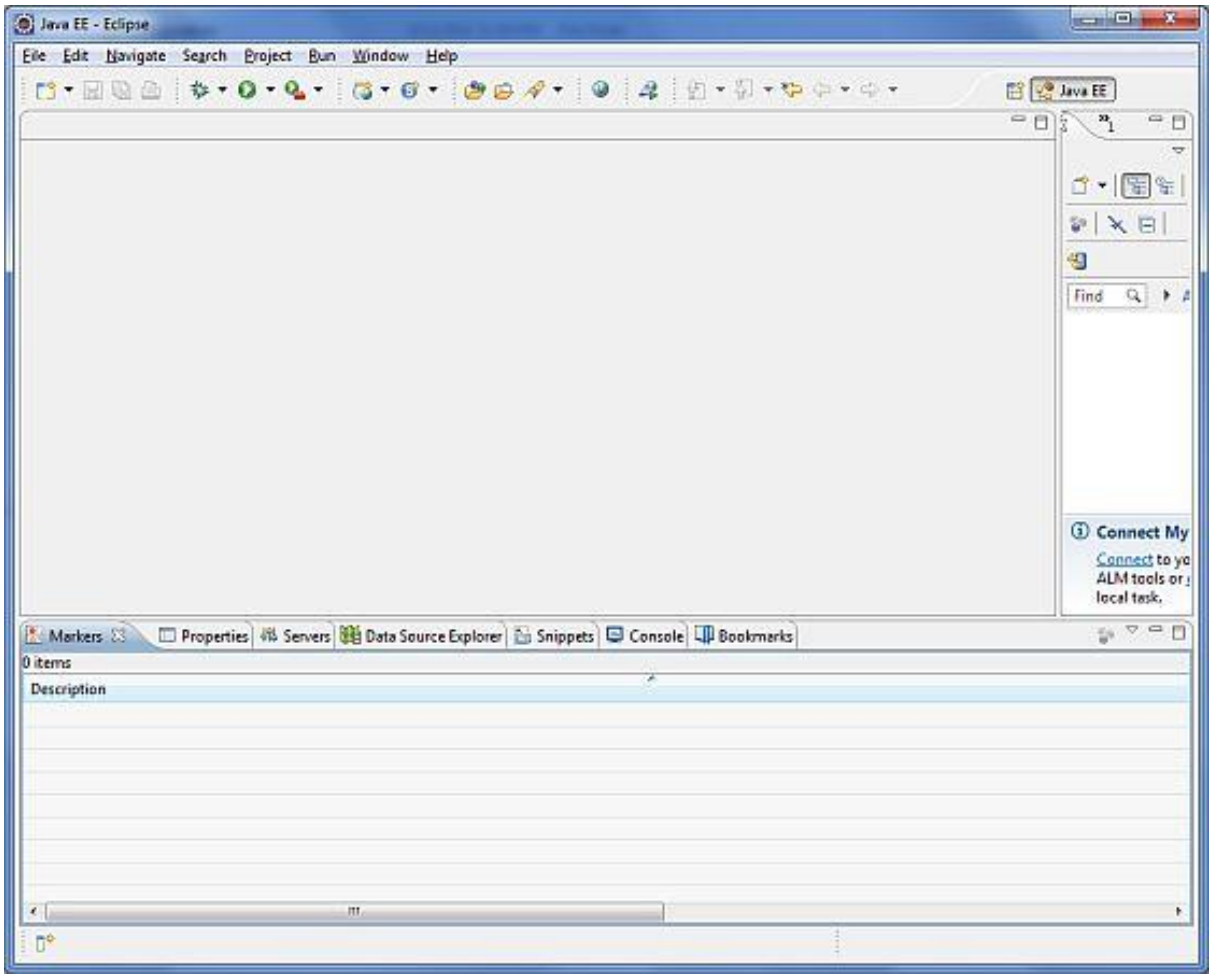
Eclipse can be started by executing the following commands on Windows machine, or you can simply double-click on eclipse.exe.

```
%C:\eclipse\eclipse.exe
```

Eclipse can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine.

```
$/usr/local/eclipse/eclipse
```

After a successful startup, if everything is fine then it should display the following result.



Once you are done with this last step, you are ready to proceed for your first JDBC example which you will see in the next chapter.

3. Spring JDBC – Configure a Data Source

Let us create a database table **Student** in our database **TEST**. I 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 shown as follows:

```
<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="admin"/>
</bean>
```

In the next chapter, we'll write the first application using the database configured.

4. Spring JDBC – First Application

To understand the concepts related to Spring JDBC framework with JDBC Template class, let us write a simple example which will implement Insert and Read 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)
);
```

Let us proceed to write a simple console based Spring JDBC Application, which will demonstrate JDBC concepts.

Create Project

Let's open the command console, go the C:\MVN directory and execute the following **mvn** command.

```
C:\MVN>mvn archetype:generate -DgroupId=com.tutorialspoint -DartifactId=Student
      -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Maven will start processing and will create the complete Java application project structure.

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.4:generate (default-cli) > generate-sources
@ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.4:generate (default-cli) < generate-sources
@ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.4:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
Downloading:
https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/ma
```

```
ven-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.jar

Downloaded:
https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/mav

en-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.jar (5 KB at 1.1
KB/sec)

Downloading:
https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/ma

ven-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.pom

Downloaded:
https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/mav

en-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.pom (703 B at 1.2
KB/sec)

[INFO] ------------------------------------------------------------------------

[INFO] Using following parameters for creating project from Old (1.x)
Archetype: maven-archetype-quickstart:1.0

[INFO] ------------------------------------------------------------------------

[INFO] Parameter: groupId, Value: com.tutorialspoint

[INFO] Parameter: packageName, Value: com.tutorialspoint

[INFO] Parameter: package, Value: com.tutorialspoint

[INFO] Parameter: artifactId, Value: Student

[INFO] Parameter: basedir, Value: C:\MVN

[INFO] Parameter: version, Value: 1.0-SNAPSHOT

[INFO] project created from Old (1.x) Archetype in dir: C:\MVN\Student

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 01:17 min

[INFO] Finished at: 2017-02-19T21:11:14+05:30

[INFO] Final Memory: 15M/114M

[INFO] ------------------------------------------------------------------------
```

Now go to C:/MVN directory. You'll see a Java application project created named student (as specified in artifactId). Update the POM.xml to include Spring JDBC dependencies. Add Student.java, StudentMapper.java, MainApp.java, StudentDAO.java and StudentJDBCTemplate.java files.

POM.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.tutorialspoint</groupId>
   <artifactId>Student</artifactId>
   <packaging>jar</packaging>
   <version>1.0-SNAPSHOT</version>
   <name>Student</name>
   <url>http://maven.apache.org</url>
   <dependencies>
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>3.8.1</version>
         <scope>test</scope>
      </dependency>
      <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-jdbc</artifactId>
         <version>4.1.0.RELEASE</version>
      </dependency>
      <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-context</artifactId>
         <version>4.1.4.RELEASE</version>
      </dependency>
   </dependencies>
</project>
```

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);
   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();
}
```

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

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());
      }
   }
}
```

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="admin"/>
   </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
```

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**