Spring JdbcTemplate Tutorial

Spring **JdbcTemplate** is a powerful mechanism to connect to the database and execute SQL queries. It internally uses JDBC api, but eliminates a lot of problems of JDBC API.

Problems of JDBC API

The problems of JDBC API are as follows:

- 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

Spring JdbcTemplate eliminates all the above mentioned problems of JDBC API. It provides you methods to write the queries directly, so it saves a lot of work and time.

Spring Jdbc Approaches

Spring framework provides following approaches for JDBC database access:

- JdbcTemplate
- NamedParameterJdbcTemplate
- SimpleJdbcTemplate

- SimpleJdbcInsert and SimpleJdbcCall

JdbcTemplate class

It is the central class in the Spring JDBC support classes. It 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 excepion classes defined in the **org.springframework.dao** package.

We can perform all the database operations by the help of JdbcTemplate class such as insertion, updation, deletion and retrieval of the data from the database.

Let's see the methods of spring JdbcTemplate class.

No.	Method	Description
1)	public int update(String query)	is used to insert, update and delete records.
2)	public int update(String query,Object... args)	is used to insert, update and delete records using PreparedStatement using given arguments.
3)	public void execute(String query)	is used to execute DDL query.
4)	public T execute(String sql, PreparedStatementCallback action)	executes the query by using PreparedStatement callback.

5)	public T query(String sql, ResultSetExtractor rse)	is used to fetch records using ResultSetExtractor.
6)	public List query(String sql, RowMapper rse)	is used to fetch records using RowMapper.

Example of Spring JdbcTemplate

We are assuming that you have created the following table inside the Oracle10g database.

```
create table employee(
id number(10),
name varchar2(100),
salary number(10)
);
```

Employee.java

This class contains 3 properties with constructors and setter and getters.

```
package com.javatpoint;

public class Employee {
private int id;
private String name;
private float salary;
//no-arg and parameterized constructors
//getters and setters
```

```
}
```

EmployeeDao.java

It contains one property jdbcTemplate and three methods saveEmployee(), updateEmployee and deleteEmployee().

```
package com.javatpoint;
import org.springframework.jdbc.core.JdbcTemplate;

public class EmployeeDao {
private JdbcTemplate jdbcTemplate;

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
}

public int saveEmployee(Employee e){
    String query="insert into employee values(
    '"+e.getId()+"','"+e.getName()+"','"+e.getSalary()+"')";
    return jdbcTemplate.update(query);
}
public int updateEmployee(Employee e){
    String query="update employee set
    name='"+e.getName()+"',salary='"+e.getSalary()+"' where id='"+e.getId()+"' ";
    return jdbcTemplate.update(query);
}
public int deleteEmployee(Employee e){
    String query="delete from employee where id='"+e.getId()+"' ";
```

```
    return jdbcTemplate.update(query);
}


}
```

applicationContext.xml

The **DriverManagerDataSource** is used to contain the information about the database such as driver class name, connnection URL, username and password.

There are a property named **datasource** in the JdbcTemplate class of DriverManagerDataSource type. So, we need to provide the reference of DriverManagerDataSource object in the JdbcTemplate class for the datasource property.

Here, we are using the JdbcTemplate object in the EmployeeDao class, so we are passing it by the setter method but you can use constructor also.

```
<?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:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">


<bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSourc
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
<property name="username" value="system" />
```

```
<property name="password" value="oracle" />
</bean>

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

<bean id="edao" class="com.javatpoint.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>

</beans>
```

Test.java

This class gets the bean from the applicationContext.xml file and calls the saveEmployee() method. You can also call updateEmployee() and deleteEmployee() method by uncommenting the code as well.

```
package com.javatpoint;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Test {

public static void main(String[] args) {
    ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.

    EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
```

```
    int status=dao.saveEmployee(new Employee(102,"Amit",35000));
    System.out.println(status);

    /*int status=dao.updateEmployee(new Employee(102,"Sonoo",15000));
    System.out.println(status);
    */

    /*Employee e=new Employee();
    e.setId(102);
    int status=dao.deleteEmployee(e);
    System.out.println(status);*/

}

}
```